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

August 31, 2014
Version: 1.2.4 (867f09edac05)

Programs version: 44f48249bf70

www.votca.org

Contents

Chapter 1

Introduction

Versatile Object-oriented Toolkit for Coarse-graining Applications, or VOTCA, is a package which helps to systematically coarse-grain various systems [1]. 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 [2] 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.

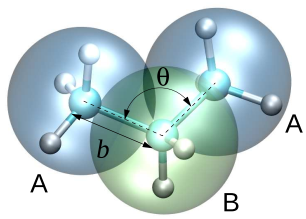

Figure 1.1: Three-bead coarse-grained model of propane.

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 [2] 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. [1]. 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 [4]. Following ref. [3], 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. [3].

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 [3]

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

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 [5]. 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 [5]. 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.

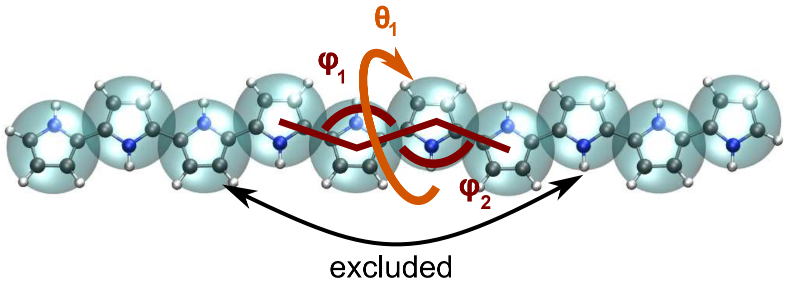

Figure 2.1: Example of excluded interactions.

[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.

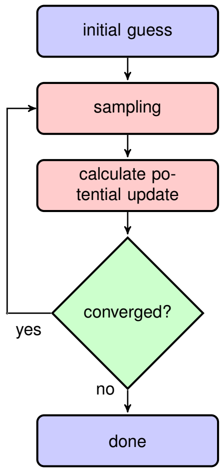

Figure 2.2: Block-scheme of an iterative method.

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 [6]

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

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

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, $U_{\mathrm{PMF}}^{(n)}$, 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. [7].

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

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

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.

2.6 Force Matching

Force matching (FM) is another approach to evaluate corse-grained potentials [8–10]. 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

$$\boldsymbol{F}_I^{\mathrm{ref}} = \sum_{j \in \mathcal{S}_{\mathcal{I}}} \frac{d_{Ii}}{c_{Ii}} \boldsymbol{f}_j(\boldsymbol{r}^n), \tag{2.19}$$

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 [3]. 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

$$\boldsymbol{F}_{Il}^{\mathrm{cg}}(g_1, \dots, g_M) = \boldsymbol{F}_{il}^{\mathrm{ref}},\ I = 1, \dots, N,\ l = 1, \dots, L\ . \tag{2.20}$$

Here $\boldsymbol{F}_{Il}^{\mathrm{ref}}$ is the force on the bead I and $\boldsymbol{F}_{Il}^{\mathrm{cg}}$ 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 2.20 is overdetermined and can be solved in a least-squares manner.

$\boldsymbol{F}_{il}^{\mathrm{cg}}$ 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 (2.20) become linear functions of $\{g_i\}$. This can be done using splines to describe the functional form of the forces [9]. Implementation details are discussed in ref. [1].

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.

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.

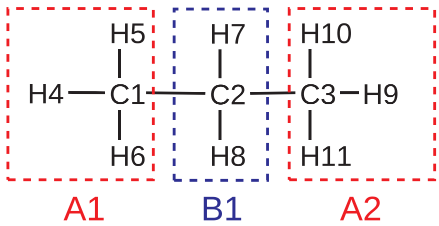

Figure 3.1: Atom labeling and mapping from an all-atom to a united atom representation of a propane molecule.

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 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>
```

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

```
<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.

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 and .gro 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

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.

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. 10.4.

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 $[\mathrm{min}, \mathrm{max}]$ 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>
  <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 \
  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 bonded table, run

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

It is also possible to use `angle` and `dihedral` as type as well.

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) expontially
- Extrapolate the right side (to `table_end` or 180) expontially (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 minium 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 [5]. The non-bonded potentials can then be obtained by using iterative methods or force matching.

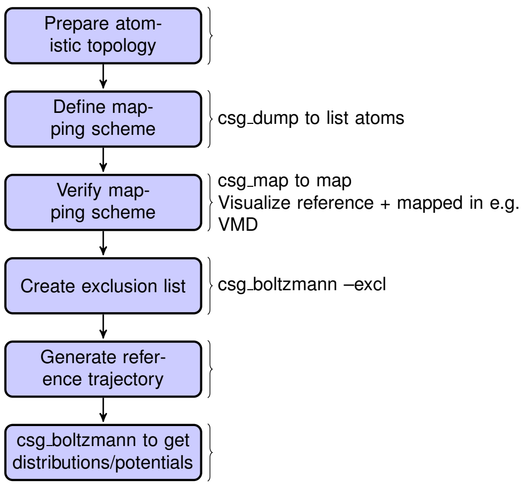

Figure 5.1: Flowchart demonstrating useful options of the tool.

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.

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 [1]. 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.

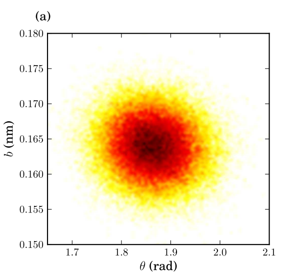

Figure 5.2: propane histogram

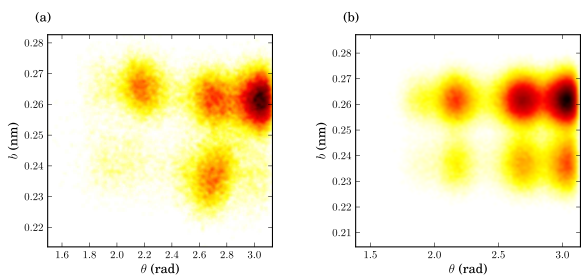

Figure 5.3: hexane histograms: before and after the coarse-grained run

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

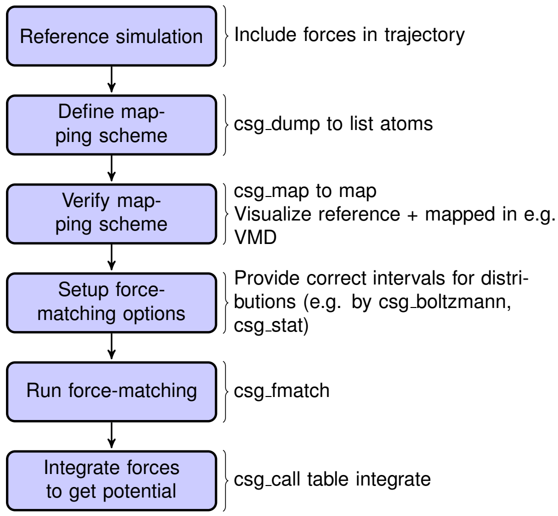

Figure 6.1: Flowchart to perform force matching.

`max` by using **csg_stat** is recommended (see sec. 3.5). A full description of all available options can be found in sec. 10.3.

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) and Inverse Monte Carlo (IMC).

In general, IBI and IMC are both 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 and IMC methods.

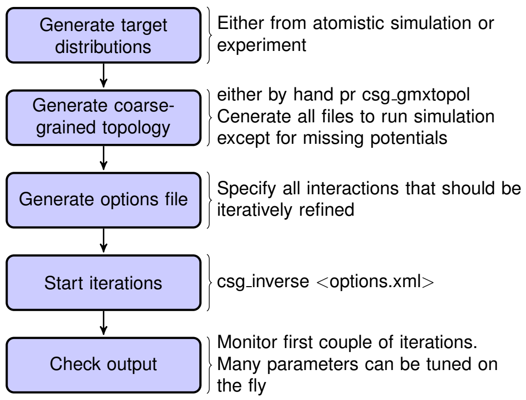

Figure 7.1: Flowchart to perform iterative Boltzmann inversion.

7.1 Iterative workflow control

Iterative workflow control is essential for the IBI and IMC 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.

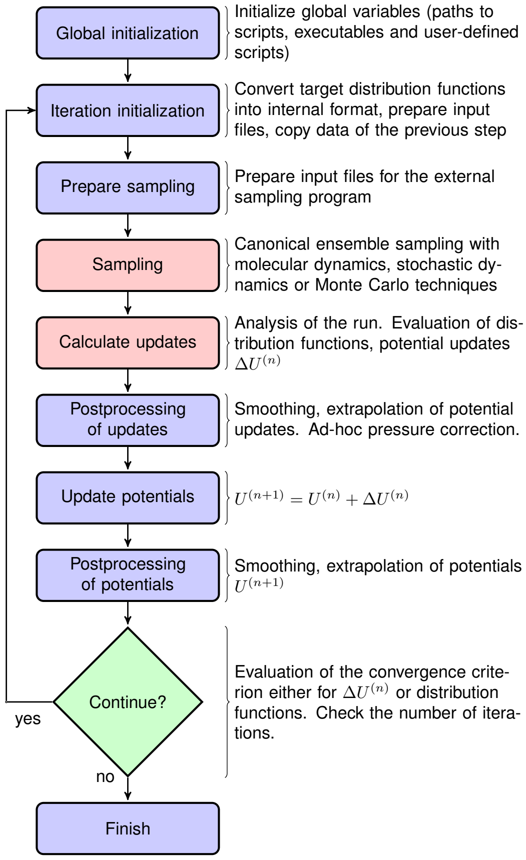

Figure 7.2: Block-scheme of the workflow control for the iterative methods. The most time-consuming parts are marked in red.

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. 10.3.

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

[width=7cm]

```xml
<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.

IBI so far only supports iterative refinement of non-bonded interactions. 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[1]. It is advisable to perform smoothing on the potential update. Smoothing can be performed as described in sec. 7.4.3. 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[1]. 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 Additional mapping for statistics

The program **csg_stat** is used for evaluating the IMC matrix. Although the matrix only acts on the coarse-grained system here, it still needs a mapping file to work. This will improve with one of the next releases to simplify the setup. The mapping file needs to be a one to one mapping of the coarse grained system, e.g. for coarse graining SPC/E water, the mapping file looks as follows:

```
</cg_molecule>
  <name>SOL</name>
  <ident>SOL</ident>
  <topology>
    <cg_beads>
      <cg_bead>
        <name>CG</name>
        <type>CG</type>
        <mapping>A</mapping>
        <beads>
          1:SOL:CG
        </beads>
      </cg_bead>
    </cg_beads>
  </topology>
  <maps>
    <map>
      <name>A</name>
      <weights>1</weights>
    </map>
  </maps>
</cg_molecule>
```

7.3.3 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>
```

7.4 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*.

7.4.1 Simple pressure correction

In ref.[6] a simple linear attractive potential was added to the coarse-grained potential

$$\Delta V(r) = A \left(1 - \frac{r}{r_{cutoff}}\right), \tag{7.1}$$

with prefactor A

$$A = -\operatorname{sgn}(\Delta P) 0.1 k_B T \min(1, |f \Delta P), \tag{7.2}$$

$\Delta p = P_i - P_{\text{target}}$, 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.

7.4.2 Advanced pressure correction

In [11] 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:

$$A = \left[\frac{-2\pi\rho^2}{3r_{cut}} \int_0^{r_{cut}} r^3 g_i(r) dr\right] A_i = \Delta P. \tag{7.3}$$

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

7.4.3 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.

7.5 Thermodynamic force

The thermodynamic force method is an iterative procedure to determine an external field that can correct for density variations. This has been prooven to be usefull for multi-scale simulations where all-atom and coarse-grained representations are simulated concurrently in one simulation. The AdresS simulation scheme provides a protocol for such simulations.
The thermodynamic force is updated from the density profile in each simulation step as:

$$\mathbf{F}_{\mathrm{th}}^{i+1}(\mathbf{r}) = \mathbf{F}_{\mathrm{th}}^{i}(\mathbf{r}) - \frac{m_{\mathrm{mol}}}{\rho_0^2 \kappa_T^{\mathrm{at}}} \nabla \rho_i(\mathbf{r}) \tag{7.4}$$

where $\rho_i(\mathbf{r})$ can be either a density along one of the box axis or a radial density, this is specified by the adress paramater adress_type in the gromacs mdp file. In order to use the thermodynamic force iteration, VOTCA must be used together with the gromacs-4.6.1 or later. To check whether your gromacs version support this type

```
mdrun -h
```

and look for the -tabletf option. A tutorial simulation set can be found in the tutorials (spce/tf) which performs the thermodynamic force iteration for spc/e water coupled to a coarse-grained spc/e water.
The method is selected by specifying

```
<method>tf</method>
```

in the inverse section. For each interaction type additional options have to be specified in the settings.xml file. To specify in which region the thermodynamic force should be nonzero, the min and max properties are used. A smoothing function proportional to $cos^2(r)$ is used to make the force go smoothly to zero at the region specified by min and max. Additonally a 'tf' section is needed for each interaction type

```
<non-bonded>
    <name>SOL</name>
    <min>1.4</min>
    <max>3.1</max>
   <step>0.01</step>
    <tf>
  <spline_start>0.9</spline_start>
  <spline_end>3.6</spline_end>
  <spline_step>0.4</spline_step>
  <molname>SOL</molname>
  <prefactor>0.01382</prefactor>
    </tf>
    <inverse>
  <target>dens.SOL.xvg</target>
  (...)
    </inverse>
</non-bonded>
```

Usually the density profile fluctuates too much to obtain a force directly from the gradient. Thus spline interpolation is used to smooth the force. To specify the spline interpolation range the spline_start and spline_end parameters are used. These can define a larger region than between min and max as it is sometimes usefull to extend the spline fit for numerical stability. The parameter spline_step sepcifies the bin width of the fit grid (see **csg_resample** for more). The field 'molname' specifies the molecule (as defined in the gromacs topology) used for calculating the density. The prefactor $\frac{m_{\mathrm{mol}}}{\rho_0^2 \kappa_T^{\mathrm{at}}}$ appearing in eq 7.4 is specified in the 'prefactor' field. Its units are kJ nm^3/(mol u) in the case of GROMACS due to the fact ρ is calculated in u/nm^3 and force units are kJ/(mol nm). If two representations of the fluid with different compressibilities are coupled, it is advisable to use the higher compressibility in order to stabilize the iteration. A target density file has to be specified for each interaction type, in most cases this will containt a flat density profile at the equilibrium density ρ_0.

Chapter 8

ESPResSo interface

WARNING: The ESPResSo interface only supports the Iterative Boltzmann Inversion scheme. It does not support Inverse Monte Carlo or Force Matching.

8.1 Running IBI with ESPResSo

While ESPResSo [12] is not capable of simulating atomistic systems, it is possible to coarse-grain molecules from existing radial distribution functions. In addition to the target RDFs, the user needs to provide two files:

- Blockfile
- XML settings file

The blockfile[1] contains all the initial ESPResSo parameters to start the first simulation step: time step, box size, temperature, friction coefficient of the thermostat, verlet skin, etc. It also includes the initial positions, velocities, particle types, masses, molecule IDs of all the particles. Including velocities is important to start at the correct temperature. Topology can be specified by including the bond descriptions between particles. Interactions need also to be present, as well as the thermostat itself. In this respect, the blockfile contains all the necessary information required to directly start the simulation: from ESPResSo variable to initial structure to topology to interactions. An example blockfile can easily be created by the following commands

```
set out [open "| gzip -c - > conf.esp.gz" w]
blockfile $out write variable all
blockfile $out write particles [list id type molecule mass pos v]
blockfile $out write interactions
blockfile $out write thermostat
blockfile $out write tclvariable [list list1]
close $out
```

where the first line opens the file for output (to a gzipped file), and the blockfile is generated by appending information blocks. The next to last line contains a special TCL variable that contains the list of particles to be taken into account during the RDF calculation. The blockfile itself can be ordered in any way and can contain as much information as the user needs. The script above represents the minimal amount of information that has to be supplied to VOTCA. For examples on generated blockfiles and on scripts to generate such blockfiles, see the Tutorials package:

```
tutorials/methanol/ibm_espresso/conf.esp.gz
tutorials/methanol/ibm_espresso/generate_esp_from_gro/
```

[1] For more information on ESPResSo blockfiles, see the ESPResSo user guide.

```
tutorials/propane/ibm_espresso/conf.esp.gz
tutorials/propane/ibm_espresso/generate_esp_from_gro/
```

The XML settings file contains several pieces of information specific to ESPResSo (entries that are common with GROMACS are not described here):

<**cg**><**non-bonded**><**inverse**><**espresso**><**index1**> provides the name of the TCL variable containing the list of type1 particle IDs involved in the type1-type2 RDF calculation

<**cg**><**non-bonded**><**inverse**><**espresso**><**index2**> same as previously for the list of type2 particle IDs.

<**cg**><**inverse**><**program**> should be “espresso”.

<**cg**><**inverse**><**espresso**> :

<**bin**> the name or path of the executable (*e.g.* Espresso_bin)

<**equi_snapshots**> trash so many snapshots before analyzing the data

<**table_bins**> bin size for table

<**table_end**> distance cutoff

<**blockfile**> input blockfile containing all simulation parameters (gzipped format)

<**n_steps**> number of MD steps to integrate between each snapshot

<**n_snapshots**> number of snaphsots before RDF calculation

See the Tutorials package for XML settings file examples:

```
tutorials/methanol/ibm_espresso/settings.xml
tutorials/propane/ibm_espresso/settings.xml
```

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. 10.4 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.scriptdir*) 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. 10.4. 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.scriptdir* in the setting XML file (see sec. 10.3).

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 Gnuplot

Get it from www.gnuplot.info

9.2.4 GNU Octave

Get it from www.gnu.org

9.2.5 Matlab

Get it from www.mathworks.com

9.2.6 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 ]` produce this help message
`--top arg` atomistic topology file
Mapping options:
`--cg arg` coarse graining mapping 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 exclusion list to file

Trajectory options:

`--trj arg` atomistic trajectory file
`--begin arg (=0)` skip frames before this time
`--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 for the iterative framework
Usage: `csg_call [OPTIONS] key1 key2`
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 CSGSHARE dir and exits
`--scriptdir DIR` Set the user script dir (Used if no optins xml file is given) Default: empty
`--simprog PROG` Set the simprog (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
`--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 ]` produce this help message
`--top arg` atomistic topology file
Mapping options:
`--cg arg` [OPTIONAL] coarse graining mapping definitions (xml-file). If no file is given, program acts on original trajectory

Specific options::

`--axis arg (=r)` [x|y|z|r] density axis (r=spherical)
`--bins arg (=50)` bins
`--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
`--first-frame arg (=0)` start with this frame
`--nframes arg` process the given number of frames

10.1.4 csg_dump

Print atoms that are read from topology file to help debugging atom naming.

Allowed options:

`-h [ --help ]` produce this help message
`--top arg` atomistic topology file
Mapping options:
`--cg arg` [OPTIONAL] coarse graining mapping definitions (xml-file). If no file is given, program acts on original trajectory

Specific options:

`--excl` display exclusion list instead of molecule list

10.1.5 csg_fmatch

Perform force matching (also called multiscale coarse-graining)

Allowed options:

`-h [ --help ]` produce this help message
`--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 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
`--first-frame arg (=0)` start with this frame
`--nframes arg` process the given number of frames

10.1.6 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 ]` produce this help message
`--top arg` atomistic topology file
`--out arg` output topology (will create .top and in future also .itp)
Mapping options:
`--cg arg` coarse graining mapping definitions (xml-file)
`--map-ignore arg` list of molecules to ignore separated by ;
`--no-map` disable mapping and act on original trajectory

10.1.7 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

10.1.8 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
`--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
```

10.1.9 csg_map

Map a reference trajectory to a coarse-grained trajectory. This program can be used to map a whole trajectory or to create an initial configuration for a coarse-grained run only.
Allowed options:
`-h [ --help ]` produce this help message
`--top arg` atomistic topology file
`--out arg` output file for coarse-grained trajectory
Mapping options:
`--cg arg` coarse graining mapping 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
`--first-frame arg (=0)` start with this frame
`--nframes arg` process the given number of frames

10.1.10 csg_part_dist

This program reads a topology and (set of) trajectory(ies). For every binned value of a chosen coordinate, it outputs the time-averaged number of particles, listed by particle types.
Allowed options:
`--top arg` topology file
`--trj arg` trajectory file
`--grid arg` output grid spacing (min:step:max)
`--out arg` output particle distribution table
`--ptypes arg` particle types to include in the analysis arg: file - particle types separated by space default: all particle types
`--first_frame arg` first frame considered for analysis
`--last_frame arg` last frame considered for analysis
`--coord arg` coordinate analyzed ('x', 'y', or 'z' (default))
`--shift_com` shift center of mass to zero
`--comment arg` store a comment in the output table
`--help` produce this help message

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_stat

Calculate all distributions (bonded and non-bonded) specified in options file. Optionally calculates update matrix 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 ]` produce this help message
`--top arg` atomistic topology file
Mapping options:
`--cg arg` [OPTIONAL] coarse graining mapping definitions (xml-file). If no file is given, program acts on original trajectory
Specific options:
`--options arg` options file for coarse graining
`--do-imc` write out Inverse Monte Carlo data
`--write-every arg` write after every block of this length, if --blocking is set, the averages are cleared after every output
`--do-blocks` write output for blocking analysis
Threading options:
`--nt arg (=1)` number of threads
Trajectory options:
`--trj arg` atomistic trajectory file
`--begin arg (=0)` skip frames before this time
`--first-frame arg (=0)` start with this frame
`--nframes arg` process the given number of frames

10.1.14 multi_g_rdf

This is a multiplexed version of g_rdf
Usage: `multi_g_rdf [OPTIONS] -- [g_rdf_options]`
Allowed options:

`-N, --NN` Number of tasks Default: 8
`-b TIME` Begin time Default: 0
`-e TIME` End time
`-n FILE` Name of the index file Default: index.ndx
`-o FILE.xvg` Name of the total output file Default: rdf.xvg
`--soutput FILE.xvg` Name of the single output files Default: rdf_NP.xvg (used trunc of name given by -o) (where NP is replaced later by the number of the process)
`--log FILE` Name of logfile Default: rdf_NP.log" (used trunc of name given by -o) (where NP is replaced later by the number of the process)
`--cmd CMD` Change the gromacs command to run Default: g_rdf
`--single` Run only one task at the time
`--debug` Enable debug output
`-q, --quiet` Be a little bit quiet
`-h, --help` Show this help

Examples:

```
multi_g_rdf -e 1
multi_g_rdf -e 1 -- -bin 0.05
```

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.

- **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 containing definition of coarse grained topology of molecule.
  - **cg_beads** Section defining coarse grained beads of molecule.
    - **cg_bead** Definition of a coarse grained 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.
      - **mapping** Mapping scheme to be used for this bead (specified in section mapping) to map from reference system.
      - **name** Name of coarse grained 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.

beads List of triples of beads that define a bond. Names specified in cg_beads, separated by commas.

name Name of the group.

bond Definition of a group of bonds.

beads List of pair of beads that define a bond. Names specified in cg_beads, separated by commas.

name Name of the group.

dihedral Definition of a group of dihedrals. Since the exact functional form does not matter, this combines proper as well as improper dihedrals.

beads List of quadruples of beads that define a bond. Names specified in cg_beads, separated by commas.

name Name of the group.

10.3 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 Head option, which contains all coarse-graining options

bonded Section for a bonded interaction. Most of the items in here are identical to items in cg.bonded, so they will be described in the same section.

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.

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

cleanlist these files are removed after each new run

convergence_check type of convergence check to do

convergence_check_options options for the convergence check

limit lower limit to stop

name_glob files to check for number (default *.conv)

espresso

blockfile Name of the original blockfile read by Espresso (default conf.esp.gz)

blockfile_out Name of the original outcome blockfile written by Espresso (default confout.esp.gz)

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

debug debug Espresso (yes/no)

exclusions Espresso stuff to exclude

first_frame rash the given number of frames at the beginning of trajectory

meta_cmd Espresso metadynamics command to call [experimental]

meta_min_sampling Espresso metadynamics minimal number of sampling [experimental]

n_snapshots number of snapshots. Total time = n_steps cginteraction.xml.t2t

cgoptions.xml.t2t config.t2t Makefile Makefile.incl Makefile.XMLS mapping.xml.t2t mapping.xml.tex xml2t2t.sh n_snapshots
n_steps number of steps to integrate before a snapshot
pressure_command Espresso command to run when calculating the pressure (name or absolute path or mpirun espresso..)
rdf_command Espresso command to run when calculating the rdf (name or absolute path or mpirun espresso..)
scriptdir overwrite ESPRESSO_SCRIPTS from environment with this dir
success File to create if Espresso simulation was successfull
table_bins grid for tabulated potentials
table_end Espresso end of table
traj Name of the output Espresso trajectory file

filelist these files are copied for each new run

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)
cutoff_check check interaction cutoffs against rvdw in mdp file: yes/no (default yes)
equi_time begin analysis after this time when using gromacs
first_frame trash the given number of frames at the beginning of trajectory
g_energy
g_energy.bin Name (or absolute path) of the g_energy binary
g_energy.opts Additional options to Gromacs g_rdf (e.g. -P 1)
g_energy.topol Gromacs g_rdf topol file to use, default topol.tpr
gmxrc GMXRC to source at the startup
g_rdf
g_rdf.bin Name (or absolute path) of the g_rdf binary
g_rdf.index Gromacs g_rdf index file to use, default index.ndx
g_rdf.opts Additional options for Gromacs g_rdf (e.g. -nopbc)
g_rdf.topol Gromacs g_rdf topol file to use, default topol.tpr
grompp
grompp.bin Name (or absolute path) of the grompp binary
grompp.index Gromacs grompp index file to use, default index.ndx
grompp.opts Additional options to Gromacs grompp (e.g. -maxwarn 1)
grompp.topol Text Gromacs toplogy file to use, default topol.top
mdp Gromacs mdp fie to use, 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..)
mdrun.opts Additional options to Gromacs mdrun (e.g. -nosum)
pot_max cut the potential at this value (gromacs bug)
rdf
rdf.topol Gromacs topol file to be used for csg_stat default topol.tpr
table_bins grid for table*.xvg !
table_end extend the tables to this value
temp_check check temperture against t_ref in mdp file: yes/no (default yes)
topol binary Gromacs topology file to use, default topol.tpr
traj_type Gromacs trajectory type (xtc/trr) file to use, default xtc

imc general imc specific options
matlab
matlab.bin Name (or absolute path) of the matlab binary

numpy
numpy.bin Name (or absolute path) of the python binary used by the numpy solver
octave
octave.bin Name (or absolute path) of the octave binary
solver solver for solving a linear equation system, can be octave or matlab
initial_configuration what initial configuration to use in every step: maindir/laststep (default laststep)
iterations_max do the given number of iterations (0=inf)
kBT kBT (300*0.00831451 gromacs units)
log_file write log to this file
method ibi: inverse boltzmann imc: inverse monte carlo
program simulation package to be used
restart_file Name of the restart file in case a step has to be resumed
scriptdir directory for user scripts (e.g. $PWD)
$sim_prog generic simulation program (e.g. GROMACS) options
equi_time begin analysis after this time
first_frame trash the given number of frames at the beginning of trajectory
simulation simulation options
background tell csg_inverse that simulation was send to the backgroud (default no)
tasks number of tasks (0/auto = automatic detect on linux)
nbsearch Grid search algorithm, simple (N square search) or grid (default is grid)
non-bonded Section for a non-bonded interaction. Most of the items in here are identical to items in cg.bonded, so they will be described in the same section.

10.3.1 Interaction options

This section contains all interaction option, which could be contained in the *non-bonded* or *bonded* section in sec. 10.3.

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.

bondtype Internal alias for non-bonded and bonded, set automatically
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 unsampled 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 update, then repeat.

espresso This section contains espresso specific options in case espresso is used as simulation program.

  index1 Index list of type1 -- Name of the Tcl variable containing all index1 particles that is contained in the espresso blockfile.

  index2 Index list of type2 -- Name of the Tcl variable containing all index2 particles that is contained in the espresso blockfile.

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

gromacs This section contains gromacs specific options in case gromacs is used as simulation program.

  grp1 Name of energy group of bead type1 using in the g_rdf index file.

  grp2 Name of energy group of bead type2 using in the g_rdf index file.

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

imc Section containing inverse monte carlo specific options.

  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.

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 This section contains all options for post add scripts.

  convergence

    weight weight factors for the convergence of the interaction, should be a list of same length as inverse.post_add_options.convergence.what (default 1)

    what list for what to calc the convergence: dist pot, .. (default dist)

  copyback

    filelist list of files to copy to the main dir

  overwrite Contains all options of the overwrite postadd scripts

    do pattern for overwrite postadd script (1 do, 0 do not).

  plot

    fd file descriptor to use (default 8), make it unique if you want to plot multiple things

    gnuplot_bin gnuplot binary to use (default gnuplot)

    gnuplot_opts extra options to give to gnuplot_bin (e.g. -persist, if one uses kill)

    kill kill all processes with that name before ploting (e.g. gnuplot_x11), this is more reliable than using multiplot

    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

  pressure Contains all options of the pressure correction scripts

    do pattern for pressure correction (1 do, 0 do not). To do pressure correction every third step specify "0 0 1", similar to inverse.do_update

    simple Contains all options of the simple pressure correction script

    simple.scale slope of the simple pressure correction

    type Pressure correction algoritm, can be simple or wjk

    wjk Contains all options of the wjk pressure correction script

wjk.scale extra scaling factor of pressure correction
scale scale factor for the update
smooth Contains all options of the smooth script
iterations number of iterations for triangular smooth
splinesmooth Contains all options of the spline smooth script
step grid spacing for spline fit when doing spline smoothing
p_target partial pressure of this species
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.
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.
tf Contains all information relevant to thermoforce iteration
cg_prefactor Second Prefactor for the thermoforce will be linear interpolated with tf.prefactor
molname Molecule name of this gropu used in gromacs topology
prefactor Prefactor for the thermoforce (f=-prefactor cginteraction.xml.t2t cgoptions.xml.t2t config.t2t Makefile Makefile.incl Makefile.XMLS mapping.xml.t2t xml2t2t.sh grad density)
spline_end End of the spline used to smooth the density
spline_start Start of the spline used to smooth the density
spline_step Grid of the spline used to smooth the density
type1 Only for non-bonded. **Bead** type 1 of non-bonded interaction.
type2 Only for non-bonded. **Bead** type 2 of non-bonded interaction.

10.4 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
function	common	functions_common.sh
csg	master	inverse.sh
prepare	ibi	prepare_generic.sh
prepare	imc	prepare_imc.sh
prepare	generic	prepare_generic.sh
prepare	tf	prepare_generic.sh
prepare_single	ibi	prepare_generic_single.sh
prepare_single	imc	prepare_generic_single.sh
prepare_single	tf	prepare_generic_single.sh
initstep	ibi	initialize_step_generic.sh
initstep	imc	initialize_step_generic.sh
initstep	tf	initialize_step_generic.sh
prepare	ibm	prepare_ibm.sh
update	ibm	update_ibm.sh
update	ibi	update_ibi.sh
update	imc	update_imc.sh
add_pot	ibi	add_pot_generic.sh
add_pot	imc	add_pot_generic.sh
add_pot	tf	add_pot_generic.sh
rdf	pot	RDF_to_POT.pl
post_update	ibi	post_update_generic.sh
post_update	imc	post_update_generic.sh
post_update	tf	dummy.sh
post_update_single	ibi	post_update_generic_single.sh
post_update_single	imc	post_update_generic_single.sh
postupd	scale	postupd_scale.sh
postupd	pressure	postupd_pressure.sh
postupd	splinesmooth	postupd_splinesmooth.sh
postupd	smooth	postupd_smooth.sh
postupd	shift	dpot_shift_nb.pl
postupd	dummy	postadd_dummy.sh
postupd	tag	tag_file.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	convergence	postadd_convergence.sh
postadd	acc_convergence	postadd_acc_convergence.sh
postadd	shift	dpot_shift_nb.pl
postadd	overwrite	postadd_overwrite.sh
postadd	plot	postadd_plot.sh
convergence_check	default	convergence_check_default.sh
dpot	shift_nonbonded	dpot_shift_nb.pl
pot	shift_nonbonded	dpot_shift_nb.pl
pot	shift_bonded	dpot_shift_bo.pl

resample	target	resample_target.sh
dpot	crop	dpot_crop.pl
update	ibi_single	update_ibi_single.sh
update	ibi_pot	update_ibi_pot.pl
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
update	tf	update_tf.sh
update	tf_single	update_tf_single.sh
calc	thermforce	calc_thermforce.sh
tf	apply_prefactor	apply_prefactor.pl
pressure_cor	simple	pressure_cor_simple.pl
pressure_cor	wjk	pressure_cor_wjk.pl
density	symmetrize	density_symmetrize.py
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	getsubset	table_getsubset.py
table	smooth_borders	table_smooth_borders.py
table	compare	table_compare.pl
configuration	compare	configuration_compare.py
tables	jackknife	tables_jackknife.pl
run	gromacs	run_gromacs.sh
pressure	gromacs	calc_pressure_gromacs.sh
rdf	gromacs	calc_rdf_generic.sh
imc_stat	gromacs	imc_stat_generic.sh
density	gromacs	calc_density_gromacs.sh
prepare_generic	gromacs	prepare_generic_gromacs.sh
initstep_generic	gromacs	initialize_step_generic_gromacs.sh
prepare_generic	espresso	prepare_generic_espresso.sh
initstep_generic	espresso	initialize_step_generic_espresso.sh
convert_potential	gromacs	potential_to_gromacs.sh
convert_potential	xvg	table_to_xvg.pl
functions	gromacs	functions_gromacs.sh
run	espresso	run_espresso.sh
pressure	espresso	calc_pressure_espresso.sh
rdf	espresso	calc_rdf_espresso.sh
convert_potential	espresso	potential_to_espresso.sh
convert_potential	tab	table_to_tab.pl
functions	espresso	functions_espresso.sh

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

10.4.1 add_pot_generic.sh

This script adds up the tables
Usage: `add_pot_generic.sh`
Used xml options:
    name

10.4.2 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: `add_POT.pl infile1 infile2 outfile`

10.4.3 apply_prefactor.pl

This script calculates the integral of a table
Usage: `apply_prefactor.pl [OPTIONS] <in> <out>`
Allowed options:
    `-h, --help` Show this help message

10.4.4 calc_density_gromacs.sh

This script calcs the density for gromacs for the AdResS therm force
Usage: `calc_density_gromacs.sh`
Used xml options:
    cg.inverse.gromacs.topol (default: topol.tpr)
    cg.inverse.gromacs.traj_type (default: xtc)
    cg.inverse.program
    cg.inverse.$sim_prog.equi_time (default: 0)
    cg.inverse.$sim_prog.first_frame (default: 0)
    name
    step
    tf.molname (default: *)
    tf.spline_end

10.4.5 calc_pressure_espresso.sh

This script calcs the pressure for espresso and writes it to outfile
Usage: `calc_pressure_espresso.sh outfile`
Used external packages: espresso
Used xml options:
    cg.inverse.espresso.blockfile (default: conf.esp.gz)
    cg.inverse.espresso.pressure_command (default: Espresso_bin)

10.4.6 calc_pressure_gromacs.sh

This script calcs the pressure for gromacs and writes it to outfile
Usage: `calc_pressure_gromacs.sh outfile`
Used external packages: gromacs
Used xml options:
 cg.inverse.gromacs.g_energy.bin (default: g_energy)
 cg.inverse.gromacs.g_energy.opts (default: empty)
 cg.inverse.gromacs.g_energy.topol (default: topol.tpr)

10.4.7 calc_rdf_espresso.sh

This script calcs the rdf for espresso
Usage: `calc_rdf_espresso.sh`
Used external packages: espresso
Used xml options:
 cg.inverse.espresso.blockfile (default: conf.esp.gz)
 cg.inverse.espresso.first_frame (default: 0)
 cg.inverse.espresso.rdf_command (default: Espresso_bin)
 cg.inverse.espresso.traj (default: top_traj.esp)
 inverse.espresso.index1
 inverse.espresso.index2
 max
 min
 name
 step
 type1
 type2

10.4.8 calc_rdf_generic.sh

This script implemtents statistical analysis for the iterative Boltzmann inversion using generic csg tools (csg_stat)
Usage: `calc_rdf_generic.sh`
Used xml options:
 cg.inverse.gromacs.rdf.topol (default: topol.tpr)
 cg.inverse.gromacs.traj_type (default: xtc)
 cg.inverse.program
 cg.inverse.$sim_prog.equi_time (default: 0)
 cg.inverse.$sim_prog.first_frame (default: 0)

10.4.9 calc_thermforce.sh

This script calcs the thermoforce out of gromacs density for the AdResS therm force
Usage: `calc_thermforce.sh infile outfile`
Used xml options:
 cg.inverse.gromacs.mdp (default: grompp.mdp)

max
min
name
step
tf.cg_prefactor (default: empty)
tf.prefactor
tf.spline_end
tf.spline_start
tf.spline_step

10.4.10 configuration_compare.py

Usage: `configuration_compare.py [options] conf1 conf2`
Options:
`-h, --help` show this help message and exit
`--eps=EPS` tolerance for mismatch

10.4.11 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: `convergence_check_default.sh`
Used xml options:
cg.inverse.convergence_check_options.limit
cg.inverse.convergence_check_options.name_glob (default: *.conv)

10.4.12 density_symmetrize.py

This script symmetrizes the density around --adressc for thermodynamic force iteration
Usage: `density_symmetrize.py`
Allowed options:
`--adressc X.X` center of the adress zone (x-value)
`--infile FILE` input file
`--outfile FILE` output file

10.4.13 dpot_crop.pl

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

10.4.14 dpot_shift_bo.pl

This script shifts the whole potential to minimum, like it is normally done for bonded potentials.
Usage: `dpot_shift_bo.pl infile outfile`

10.4.15 dpot_shift_nb.pl

This script shifts the whole potential to the last value, like it is normally done for non-bonded potentials.
Usage: `dpot_shift_nb.pl infile outfile`

10.4.16 dummy.sh

dummy script (does nothing), useful to overwrite default by nothing
Usage: `dummy.sh`

10.4.17 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
`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
`check_for_duplicated_interactions` -- checks for duplicated interactions
`csg_get_interaction_property` -- gets an interaction property from the xml file, should only be called from inside a for_all loop
`csg_get_property` -- get an property from the xml file
`trim_all` -- strips white space from beginning and the end of all args
`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
`int_check` -- checks if 1st argument is a integer or calls die with error message (2nd argument)
`num_check` -- checks if 1st argument is a number or calls die with error message (2nd argument)
`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_number_tasks` -- get the number of possible tasks from the xml file or determine it automatically under linux
`get_table_comment` -- get comment lines from a table and add common information, which include the hgid and other information

`csg_inverse_clean` -- clean out the main directory
`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: inverse.log)
cg.inverse.restart_file (default: restart_points.log)
cg.inverse.simulation.tasks (default: auto)
cg.non-bonded.name
name

10.4.18 functions_espresso.sh

Useful functions for espresso:
`simulation_finish` -- checks if simulation is finished
`checkpoint_exist` -- check if a checkpoint exists
`get_simulation_setting` -- check if a checkpoint exists

Used external packages: espresso

Used xml options:
cg.inverse.espresso.blockfile_out (default: confout.esp.gz)
cg.inverse.espresso.scriptdir (default: empty)
cg.inverse.espresso.success (default: success.esp)
cg.inverse.espresso.traj (default: top_traj.esp)

10.4.19 functions_gromacs.sh

Useful functions for gromacs:
`get_simulation_setting` -- gets a parameter (1st argument) from gromacs mdp file (2nd parameter)
`check_cutoff` -- compared current interactions cutoff vs rvdw,
`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

Used external packages: gromacs
Used xml options:

cg.inverse.gromacs.conf_out (default: confout.gro)
cg.inverse.gromacs.cutoff_check (default: yes)
cg.inverse.gromacs.equi_time (default: 0)
cg.inverse.gromacs.first_frame (default: 0)
cg.inverse.gromacs.gmxrc (default: empty)
cg.inverse.gromacs.mdp (default: grompp.mdp)
cg.inverse.gromacs.mdrun.checkpoint (default: state.cpt)
cg.inverse.gromacs.temp_check (default: yes)
cg.inverse.gromacs.traj_type (default: xtc)
cg.inverse.kBT
max

10.4.20 imc_purify.sh

This scripts cleans up the dpot tables for each interaction when using IMC
Usage: `imc_purify.sh`
Used xml options:

cg.inverse.kBT
inverse.do_potential (default: 1)
max
min
name
step

10.4.21 imc_stat_generic.sh

This script implemtents statistical analysis for the Inverse Monte Carlo Method using generic csg tools (csg_stat)
Usage: `imc_stat_generic.sh`
Used xml options:

cg.inverse.gromacs.topol (default: topol.tpr)
cg.inverse.gromacs.traj_type (default: xtc)
cg.inverse.program
cg.inverse.$sim_prog.equi_time (default: 0)
cg.inverse.$sim_prog.first_frame (default: 0)

10.4.22 initialize_step_generic_espresso.sh

This script initializes an espresso simulation
Usage: `initialize_step_generic_espresso.sh`
Used xml options:

cg.inverse.espresso.blockfile (default: conf.esp.gz)
cg.inverse.espresso.blockfile_out (default: confout.esp.gz)
cg.inverse.initial_configuration (default: laststep)

10.4.23 initialize_step_generic_gromacs.sh

This script implemtents the function initialize
Usage: `initialize_step_generic_gromacs.sh`
Used external packages: gromacs
Used xml options:

- cg.inverse.gromacs.conf (default: conf.gro)
- cg.inverse.gromacs.conf_out (default: confout.gro)
- cg.inverse.initial_configuration (default: laststep)
- cg.inverse.method

10.4.24 initialize_step_generic.sh

This script implements the initialization for every step in a generic way
Usage: `initialize_step_generic.sh`
Used xml options:

- cg.inverse.method
- cg.inverse.program
- name

10.4.25 inverse.sh

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

- `-h, --help` show this help
- `-N, --do-iterations N` only do N iterations
- `--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.cleanlist (default: empty)
- cg.inverse.convergence_check (default: none)
- cg.inverse.filelist (default: empty)
- cg.inverse.iterations_max
- cg.inverse.method
- cg.inverse.program
- cg.inverse.scriptdir (default: empty)
- cg.inverse.simulation.background (default: no)

10.4.26 linsolve.m

This script has no help

10.4.27 linsolve.octave

This script has no help

10.4.28 linsolve.py

This script has no help

10.4.29 merge_tables.pl

Merge two tables
Usage: `merge_tables.pl [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`

10.4.30 postadd_acc_convergence.sh

postadd accumulate convergence script: accumulate ${name}.conv of all steps
Usage: `postadd_acc_convergence.sh infile outfile`
Used xml options:

name

10.4.31 postadd_convergence.sh

postadd convergence script, calcs int of (${name}.DIST.tgt-${name}.DIST.new)**2 and saves it to ${name}.conv. DIST is dist, but changed by onvergence.what option
usage: postadd_convergence.sh infile outfile
Used xml options:

inverse.post_add_options.convergence.weight (default: 1)
inverse.post_add_options.convergence.what (default: dist)
max
min
name
step

10.4.32 postadd_copyback.sh

postadd copyback script, copies files back to the maindir, use ${name} in filename as replacement for the interaction name
Usage: `postadd_copyback.sh infile outfile`
Used xml options:

inverse.post_add_options.copyback.filelist (default: empty)
name

10.4.33 postadd_dummy.sh

postadd dummy script (does nothing), useful to overwrite default by nothing
Usage: `postadd_dummy.sh infile outfile`

10.4.34 postadd_overwrite.sh

postadd overwrite script, overwrites potential of all other interactions with this one
Usage: `postadd_overwrite.sh infile outfile`
Used xml options:

inverse.post_add
inverse.post_add_options.overwrite.do (default: 1)
cg.non-bonded.name
name

10.4.35 postadd_plot.sh

postadd plot script, send a certain plot script to gnuplot
Usage: `postadd_plot.sh infile outfile`
Used external packages: gnuplot
Used xml options:

inverse.post_add_options.plot.fd (default: 8)
inverse.post_add_options.plot.gnuplot_bin (default: gnuplot)
inverse.post_add_options.plot.gnuplot_opts (default: empty)
inverse.post_add_options.plot.kill (default: empty)
inverse.post_add_options.plot.script

10.4.36 post_add.sh

This script makes all the post update
Usage: `post_add.sh`

10.4.37 post_add_single.sh

This script makes all the post update with backup for single pairs
Usage: `post_add_single.sh`
Used xml options:

inverse.post_add (default: empty)
name

10.4.38 post_update_generic.sh

This script makes all the post update
Usage: `post_update_generic.sh`
Used xml options:

cg.inverse.method

10.4.39 post_update_generic_single.sh

This script makes all the post update with backup for single pairs incl. backups
Usage: `post_update_generic_single.sh`
Used xml options:

inverse.post_update (default: empty)
name

10.4.40 postupd_pressure.sh

This script implements the pressure update
Usage: `postupd_pressure.sh infile outfile`
Used xml options:

cg.inverse.program
inverse.post_update_options.pressure.do (default: 1)
inverse.post_update_options.pressure.type (default: simple)
max
min
name
step

10.4.41 postupd_scale.sh

This script implements scaling of the potential update (.dpot)
Usage: `postupd_scale.sh infile outfile`
Used xml options:

inverse.post_update_options.scale (default: 1.0)
name

10.4.42 postupd_smooth.sh

This script implements smoothing of the potential update (.dpot)
Usage: `postupd_smooth.sh infile outfile`
Used xml options:

inverse.post_update_options.smooth.iterations (default: 1)
name

10.4.43 postupd_splinesmooth.sh

This script implements smoothing of the potential update (.dpot)
Usage: `postupd_splinesmooth.sh infile outfile`
Used xml options:

inverse.post_update_options.splinesmooth.step
max
min
name
step

10.4.44 potential_to_espresso.sh

This script is a wrapper to convert a potential to espresso
Usage: `potential_to_espresso.sh`
Used xml options:

cg.inverse.espresso.table_bins
inverse.espresso.table
max
name

10.4.45 potential_to_gromacs.sh

This script is a wrapper to convert a potential to gromacs
Usage: `potential_to_gromacs.sh [input] [output]`
Used xml options:

cg.inverse.gromacs.mdp (default: grompp.mdp)
cg.inverse.gromacs.pot_max (default: empty)
cg.inverse.gromacs.table_bins
cg.inverse.gromacs.table_end
cg.inverse.gromacs.table_end (default: empty)
cg.inverse.method (default: empty)
bondtype
inverse.gromacs.table
name

10.4.46 prepare_generic_espresso.sh

This script implements the prepare step for espresso
Usage: `prepare_generic_espresso.sh`
Used xml options:

cg.inverse.espresso.blockfile (default: conf.esp.gz)
cg.inverse.espresso.blockfile_out (default: confout.esp.gz)

10.4.47 prepare_generic_gromacs.sh

This script does the prepare step for gromacs
Usage: `prepare_generic_gromacs.sh`
Used xml options:

cg.inverse.gromacs.conf (default: conf.gro)
cg.inverse.gromacs.conf_out (default: confout.gro)

10.4.48 prepare_generic.sh

This script prepares potentials in a generic way
Usage: `prepare_generic.sh`
Used xml options:

cg.inverse.method
cg.inverse.program

10.4.49 prepare_generic_single.sh

This script implements the prepares the potential in step 0, using pot.in or by resampling the target distribution
Usage: `prepare_generic_single.sh`
Used xml options:

cg.inverse.method
bondtype
inverse.target
max
min
name
step

10.4.50 prepare_ibm.sh

Informs users that ibm was renamed to ibi.
Usage: `prepare_ibm.sh`

10.4.51 prepare_imc.sh

This script initializes potentials for imc
Usage: `prepare_imc.sh`

10.4.52 pressure_cor_simple.pl

This script calls the pressure corrections $dU = A * (1 - r/r\_c)$, where $A = -0.1k\_BT * \max(1, |p\_cur - p\_target| * scale) * \mathrm{sgn}(p\_cur - p\_target)$
Usage: `pressure_cor_simple.pl p_cur outfile`
Used xml options:

- cg.inverse.kBT
- inverse.post_update_options.pressure.simple.scale
- inverse.p_target
- max
- min
- step

10.4.53 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: `pressure_cor_wjk.pl p_cur outfile`
Used xml options:

- cg.inverse.kBT
- inverse.particle_dens
- inverse.post_update_options.pressure.wjk.scale (default: 1.0)
- inverse.p_target
- max
- min
- name
- step

10.4.54 RDF_to_POT.pl

This script converts rdf to pot of mean force ($F(r) = -k\_BT \ln g(r)$)
In addtion, it does some magic tricks:

- do not crash when calc log(0)
- extrapolate the beginning of pot
- the maximum to interpolate is pot_max (see xml)
- bigger value will be set to that max
- shift the potential, so that it is zero at the cutoff
- set all values to zero after the cutoff

Usage: `RDF_to_POT.pl infile outfile`
Used xml options:

- cg.inverse.kBT

max

10.4.55 resample_target.sh

This script resamples target distribution to grid spacing of the setting xml file
Usage: `resample_target.sh`
Used xml options:

- bondtype
- inverse.target
- max
- min
- name
- step

10.4.56 run_espresso.sh

This script runs espresso for the Inverse Boltzmann Method
Usage: `run_espresso.sh`
Used external packages: espresso
Used xml options:

- cg.inverse.espresso.blockfile (default: conf.esp.gz)
- cg.inverse.espresso.blockfile_out (default: confout.esp.gz)
- cg.inverse.espresso.command (default: Espresso_bin)
- cg.inverse.espresso.debug (default: no)
- cg.inverse.espresso.exclusions (default: 0)
- cg.inverse.espresso.n_snapshots
- cg.inverse.espresso.n_steps
- cg.inverse.espresso.success (default: success.esp)
- cg.inverse.espresso.traj (default: top_traj.esp)
- cg.inverse.method
- cg.non-bonded.inverse.espresso.index1
- cg.non-bonded.inverse.espresso.index2

10.4.57 run_gromacs.sh

This script runs a gromacs simulation
Usage: `run_gromacs.sh`
Used external packages: gromacs
Used xml options:

- cg.inverse.gromacs.conf (default: conf.gro)
- cg.inverse.gromacs.conf_out (default: confout.gro)
- cg.inverse.gromacs.grompp.bin (default: grompp)
- cg.inverse.gromacs.grompp.index (default: index.ndx)
- cg.inverse.gromacs.grompp.opts (default: empty)
- cg.inverse.gromacs.grompp.topol (default: topol.top)
- cg.inverse.gromacs.mdp (default: grompp.mdp)
- cg.inverse.gromacs.mdrun.checkpoint (default: state.cpt)
- cg.inverse.gromacs.mdrun.command (default: mdrun)

cg.inverse.gromacs.mdrun.opts (default: empty)
cg.inverse.gromacs.topol (default: topol.tpr)
cg.inverse.gromacs.traj_type (default: xtc)

10.4.58 solve_matlab.sh

This script solves a linear equation system from imc using matlab
Usage: `solve_matlab.sh <group> <outfile>`
Used external packages: matlab
Used xml options:
cg.inverse.imc.matlab.bin (default: matlab)

10.4.59 solve_numpy.sh

This script solves a linear equation system from imc using numpy
Usage: `solve_numpy.sh <group> <outfile>`
Used external packages: numpy
Used xml options:
cg.inverse.imc.numpy.bin (default: python2)

10.4.60 solve_octave.sh

This script solves a linear equation system from imc using octave
Usage: `solve_octave.sh <group> <outfile>`
Used external packages: octave
Used xml options:
cg.inverse.imc.octave.bin (default: octave)

10.4.61 table_compare.pl

This script compares two tables
Usage: `table_compare.pl infile1 infile2`

10.4.62 table_dummy.sh

This script creates a dummy table with grid min:step:max
Usage: `table_dummy.sh min:step:max outfile`

10.4.63 table_extrapolate.pl

This script extrapolates a table
Usage: `table_extrapolate.pl [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: $y = C * (x + a)^2 + b \;\; a = m/(2 * C) - x0 \;\; b = y\_0 - m^2/(4 * C)$

10.4.64 table_getsubset.py

This script get the a subset of a table
Usage: `table_getsubset.py`
Allowed options:
 `--xstart` X.X x value where the subset starts
 `--xstop` X.X x value where the subset stops
 `--infile` FILE input file
 `--outfile` FILE output file

10.4.65 table_get_value.pl

This script print the y value of x, which is closest to X.
Usage: `table_get_value.pl [OPTIONS] X infile`
Allowed options:
 `-h, --help` Show this help message

10.4.66 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: `table_integrate.pl [OPTIONS] <in> <out>`
Allowed options:
 `--with-errors` calculate error
 `--with-S` Add entropic contribution to force $2k\_BT/r$
 `--kbT NUMBER` use NUMBER as $k\_B * T$ for the entropic part
 `-h, --help` Show this help message
Examples:

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

10.4.67 table_linearop.pl

This script performs a linear operation on the y values: $y\_new = a * y\_old + b$
Usage: `table_linearop.pl [OPTIONS] <in> <out> <a> <b>`
Allowed options:
`-h, --help` Show this help message
`--withflag` only change entries with specific flag in src
`--with-errors` also read and calculate errors
Examples:

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

10.4.68 tables_jackknife.pl

This script has no help

10.4.69 table_smooth_borders.py

This script smooths the border for thermodynamic force iteration
Usage: `table_smooth_borders.py`
Allowed options:
`--xstart` X.X where the smoothing starts
`--xstop` X.X where the smoothing stops
`--infile` FILE input file
`--outfile` FILE output file

10.4.70 table_smooth.pl

This script smoothes a table
Usage: `table_smooth.pl infile outfile`

10.4.71 table_to_tab.pl

This script converts csg potential files to the tab format (as read by espresso). Potential is copied in the C12 column.
In addition, it does some magic tricks:
shift the potential, so that it is zero at the cutoff
set all values to zero after the cutoff
Usage: `table_to_tab.pl in_pot in_deriv_pot outfile`
Used xml options:
cg.inverse.espresso.table_bins
cg.inverse.espresso.table_end

10.4.72 table_to_xvg.pl

This script converts csg potential files to the xvg format.
Allowed options:
`-v, --version` print version
`-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, thermforce, C12, C6
Examples:
`table_to_xvg.pl --type bond table.in table_b0.xvg`

10.4.73 tag_file.sh

Add table_comment to the head of a file
Usage: `tag_file.sh input output`

10.4.74 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: `update_ibi_pot.pl new_rdf target_rdf cur_pot outfile`
Used xml options:
cg.inverse.kBT

10.4.75 update_ibi.sh

This script implements the function update for the Inverse Boltzmann Method
Usage: `update_ibi.sh`
Used xml options:
cg.inverse.program

10.4.76 update_ibi_single.sh

This script implemtents the function update for a single pair for the Inverse Boltzmann Method
Usage: `update_ibi_single.sh`
Used xml options:
inverse.do_potential (default: 1)
max
min
name
step

10.4.77 update_ibm.sh

Informs users that ibm was renamed to ibi.
Usage: `update_ibm.sh`

10.4.78 update_imc.sh

This script implements the function update for the Inverse Monte Carlo Method
Usage: `update_imc.sh`
Used xml options:

- cg.inverse.imc.solver
- cg.inverse.program
- cg.non-bonded.inverse.imc.group

10.4.79 update_tf.sh

This script implements the function update for the thermodynamic force interation
Usage: `update_tf.sh`

10.4.80 update_tf_single.sh

This script implemtents the function update of a single interaction for the thermodynamics force iteration
Usage: `update_tf_single.sh`
Used xml options:

- cg.inverse.program
- inverse.do_potential (default: 1)
- max
- min
- name
- step

Bibliography

[1] V. Rühle, C. Junghans, A. Lukyanov, K. Kremer, and D. Andrienko. Versatile object-oriented toolkit for coarse-graining applications. *J. Chem. Theor. Comp.*, page accepted, 2009. 1, 2, 7, 21, 29

[2] 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. 1, 2

[3] 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. 3, 4, 7

[4] 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. 3

[5] 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. 4, 5, 19

[6] D Reith, M Pütz, and F Müller-Plathe. Deriving effective mesoscale potentials from atomistic simulations. *J. Comp. Chem.*, 24(13):1624–1636, 2003. 6, 31

[7] 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. 6

[8] F. Ercolessi and J. B. Adams. Interatomic potentials from 1st-principles calculations - the force-matching method. *Europhys. Lett.*, 26(8):583–588, 1994. 7

[9] S Izvekov and GA Voth. Multiscale coarse graining of liquid-state systems. *J. Chem. Phys.*, 123(13):134105, OCT 1 2005. 7

[10] 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. 7

[11] 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. 31

[12] Hans-Jörg Limbach, Axel Arnold, Bernward A. Mann, and Christian Holm. ESPResSo – an extensible simulation package for research on soft matter systems. *Computer Physics Communications*, 174(9):704–727, May 2006. 35