GROW: A Gradient-based Optimization Workflow for the Automated Development of Molecular Models

Marco Hülsmann[a], Thorsten Köddermann[a], Jadran Vrabec[b], Dirk Reith[*,a]

[a] *Fraunhofer-Institute for Algorithms and Scientific Computing (SCAI), Schloß Birlinghoven, 53754 Sankt Augustin, Germany*

[b] *Institut für Thermodynamik und Energietechnik (ThEt), Universität Paderborn, Warburger Str. 100, 33098 Paderborn, Germany*

Abstract

The concept, issues of implementation and file formats of the **GR**adient-based **O**ptimization **W**orkflow for the Automated Development of Molecular Models 'GROW' (version 1.0) software tool are described. It enables users to perform automated optimizations of force field parameters for atomistic molecular simulations by an iterative, gradient-based optimization workflow. The modularly constructed tool consists of a main control script, specific implementations and secondary control scripts for each numerical algorithm, as well as analysis scripts. Taken together, this machinery is able to automatically optimize force fields and it is extensible by developers with regard to further optimization algorithms and simulation tools. Results on nitrogen are briefly reported as a proof of concept.

Key words: Force field design, Molecular simulation, Atomistic models, Numerical optimization, Gradient-based algorithms, Phase change properties
PACS: 34.20.Gj, 64.75.Gh

1. Introduction

Molecular simulation methods, most prominently molecular dynamics (MD) or Monte–Carlo (MC), are used nowadays in many scientific fields as powerful tools, e.g. to gain insight into microscopic processes that govern the macroscopic behavior of matter. Driven by the ongoing progressive growth in computational resources, it can be expected that these molecular methods will be even more useful in the next decades.
One key element that has to be specified as accurately as possible for a molecular simulation is the force field that describes the intra- and intermolecular interactions. Force fields comprise a semi-empirical ansatz to represent these interactions, e.g. a set of Lennard–Jones (LJ) sites [1] or point charges, and the associated parameters. While the model approach is usually straightforward, the parameterization of force fields is often tedious.
Numerous authors have worked over the past decades to develop force fields for a variety of areas, such as thermodynamic properties of fluids [2–8], mechanic properties of solids [9–11], phase change phenomena [12–14], transport processes in biologic tissue [15, 16], protein folding [17–19], transport processes in liquids [20–22], polymer properties by using different length scales

*Corresponding author
Email address: dirk.reith@scai.fraunhofer.de (Dirk Reith)

[23–26] or generic statistic properties of soft matter [27].
Quantum mechanical methods are useful to specify force field parameters, e.g. for geometry and electrostatics. However, weak short-range interactions like dispersion are hard to tackle by quantum mechanics, particularly when the regarded molecules are composed of more than few atoms.
Hence, the force field parameters for these weak interactions have to be fitted to experimental or theoretical target values. Thereby, a manual adjustment is usually not feasible or at best extremely time-consuming. Hence, an automated parameterization is essential, which can be achieved by iterative procedures. In Faller et al. [30], the Nelder–Mead simplex algorithm was applied, in order to optimize a variety of atomistic force fields. This algorithm is robust in finding a local optimum but its convergency is very slow, as it does not lead directly to the optimum, which implies numerous consecutive iterations associated with time-consuming molecular simulation runs. Furthermore, at some point, it hops around the minimum.
Ungerer and coworkers [32, 33] presented a direct gradient-based approach, resulting in the solution of a linear equation system (LES). In their studies on small molecules like olefins, this method delivered very accurate results. However, the drawback of that method is that the number of physical properties considered in the optimization must be greater than the number of force field parameters. Otherwise, the problem is underdetermined, the matrix of the LES is thus singular and the method is not applicable. However, this method enables the application of efficient gradient-based numerical algorithms to the automated development of force fields.
In recent years, some automatic parameterization software packages have been developed, in order to solve special optimization tasks on different kinds of granularity levels. On the quantum mechanical level, a tool named 'Parmscan' was published by [28, 29], which optimizes intramolecular force field parameters only, e.g. bond lengths and torsional angles, for the reasons mentioned above.
As an example on the coarse-grained level, a software package for a high-quality automated force field design named 'CG-OPT' has been developed by Reith et al. [31]. It maps a fully detailed atomistic model of a polymer system to a coarse-grained mesoscale model and parameterizes the associated force field automatically by the simplex algorithm.
In this work, an optimization workflow, based on efficient gradient-based numerical algorithms, for the automated development of atomistic force fields is presented. The main advantage of this kind of methods is that they converge faster and that the search is directed to the optimum because of the use of descent directions. Moreover, the algorithms can handle overdetermined and underdetermined optimization problems.
GROW is a tool kit of programs and scripts to facilitate the use of gradient-based numerical optimization of force field parameters. The components of the presented tool kit can be grouped as follows:

- Force field parameter optimizers,
- Simulation programs,
- Analysis scripts,
- Input/output handling,
- General computation utilities.

Most of the scripts of the tool are written in *python* (version 2.4.3). Some help scripts are written in *S+* (version 2.7.1), a scripting language originating from the statistics package *R*[1], and shell (tcsh). The reason for this choice is the fact that *python* is object-oriented and an interface to other tools, executed by shell commands, can be easily realized. Moreover, many computation utilities, e.g. the solution of a LES, can be performed by *R*-built-in functions. Therefore, all help scripts for such tasks are written in *S+*.

As a proof of concept, an automated parameter optimization for a force field for nitrogen was performed. In this case, fit functions developed by Stoll et al. [34] were used to compute the physical properties instead of running molecular simulations, as their use saves a lot of computational time and hence gives a fast insight into the quality of the behavior of the numerical optimization algorithms. Those fit functions, however, do not contain statistical noise so that a successful application to molecular simulations data is not guaranteed. The main problem of gradient-based algorithms is given by the fact that it is not predictable to what extent they can handle statistical noise. A detailed assessment of various numerical optimization algorithms, also with respect to noise, is given in [35]. Note that this approach allows us to clearly separate the basic technical issues from the more scientific questions of the optimization procedure.

2. Problem Definition

The central role in the optimization workflow is played by a quadratic loss function between the calculated (e.g. from simulation) and target (e.g. from experiment) data:

$$F(\boldsymbol{x}) = \sum_{i=1}^{n} w_i \left(\frac{f_i^{\mathrm{exp}} - f_i^{\mathrm{sim}}(\boldsymbol{x})}{f_i^{\mathrm{exp}}} \right)^2, \tag{1}$$

where $\boldsymbol{x} = (x_1, ..., x_N)^T \in \mathbb{R}^N$ is the force field parameter vector, N is the number of force field parameters, n is the number of physical properties considered in the optimization, $f_i^{\mathrm{sim}}(\boldsymbol{x})$ is the ith property dependent on the force field parameters, and f_i^{exp} is the respective target value. The weights w_i account for the fact that some properties may be easier to reproduce than others. This loss function has to be minimized with respect to $\boldsymbol{x}$.
In an ideal situation, some $\boldsymbol{x}^{\mathrm{opt}}$ can be found for which $F(\boldsymbol{x}^{\mathrm{opt}}) = 0$. However, the primary goal is to find a local minimum in an admissible compact parameter domain, where a range for each of the force field parameters has to be defined.
A smooth dependency of F on the force field parameters has to be guaranteed, which is usually the case. Hence, it is aimed to find a minimum at $\boldsymbol{x}^{\mathrm{opt}}$ for which

$$\nabla F(\boldsymbol{x}^{\mathrm{opt}}) = 0. \tag{2}$$

This goal can be achieved by gradient-based numerical optimization algorithms, several of which are presented in section 4. The gradient can be expressed in terms of the partial derivatives

$$\frac{\partial F}{\partial x_j}(\boldsymbol{x}) = -2 \sum_{i=1}^{n} w_i \frac{f_i^{\mathrm{exp}} - f_i^{\mathrm{sim}}(\boldsymbol{x})}{\left(f_i^{\mathrm{exp}}\right)^2} \frac{\partial f_i^{\mathrm{sim}}}{\partial x_j}(\boldsymbol{x}),$$

where $j = 1, ..., N$. Finally, the partial derivatives of the properties can be approximated numerically by

$$\frac{\partial f_i^{\mathrm{sim}}}{\partial x_j}(\boldsymbol{x}) = \frac{f_i^{\mathrm{sim}}(x_1, ..., x_j + h, ..., x_N) - f_i^{\mathrm{sim}}(\boldsymbol{x})}{h},$$

[1] `http://www.r-project.org`

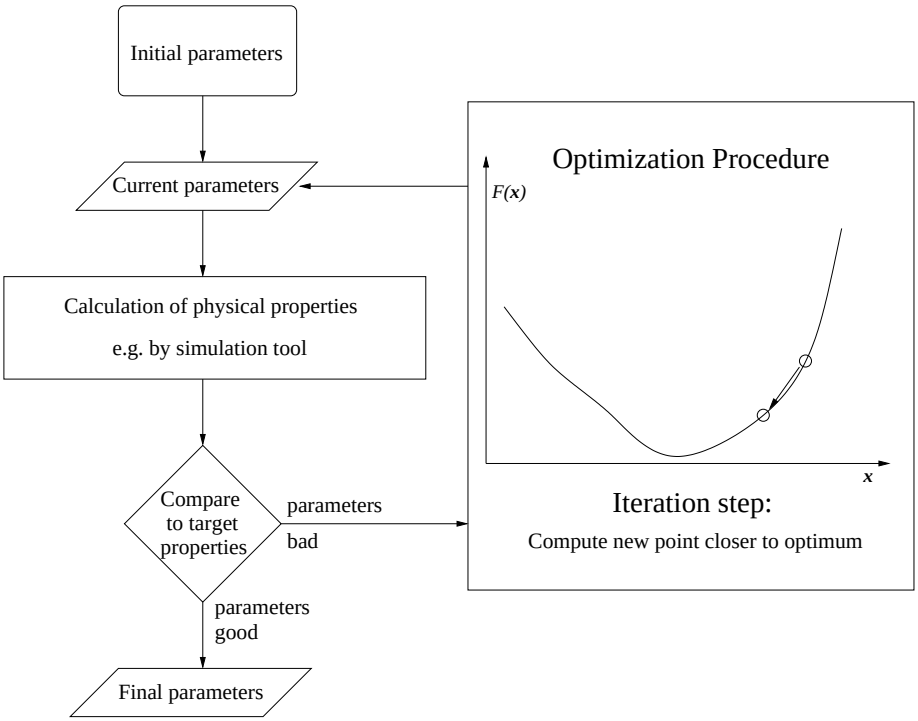

Figure 1: The optimization workflow: The physical properties are calculated for an initial guess of the force field parameters. If the calculated physical properties do not agree sufficiently well with the target properties, the optimization is executed, until a stopping criterion is fulfilled and the final parameters are found.

with $h > 0$.

The number of physical properties does not have to obey any constraints. Hence, $n > N$ is not required, as in the case of the method developed by Ungerer and coworkers [32, 33]. Any properties can be used. It is also possible to consider one or more properties at different temperatures or pressures.

Please note that whenever the simulated physical properties are affected by noise, it is always necessary to perform some statistical proofs concerning the quality of the resulting force field. This will be discussed in more detail in section 8.

In many cases, the force field parameters as well as the target data have physical units, which makes the results physically interpretable. Please note that this does not have any effect on the loss function, as the target data only differ by a constant factor, when units are changed. As relative errors are summed up, these factors are canceled out. The loss function itself does not have any unit but its gradient has units on its axes, namely the reciprocal value of the units of the respective force field parameter. In extreme cases, it might happen that some components of the gradient are much greater than others, leading to deformations of the loss function. This, in turn, can cause numerical problems in finding the minimum and the norm of the gradient cannot be computed in a straightforward way. To avoid this, one can simply divide each force field parameters by its unit, or by a certain physically meaningful reference value, which leads to a force field that is physically better defined. However, one can do this retroactively, i.e. by dividing the resulting optimal force field parameters by their reference values. It should be clear that the choice of reasonable units cannot be handled automatically and hence lies in the responsibility of the user.

3. Optimization Workflow

The complete optimization workflow is shown in Figure 1: All optimization algorithms require an initial parameter vector $\boldsymbol{x}^0$, which must be reasonably close to the minimum. The output of a simulation tool, i.e. the calculated physical properties, is inserted into the loss function (1). Then it is compared with the experimental or theoretical target data. If a specified stopping criterion is fulfilled, the parameters are final and the workflow ends. Otherwise, the current parameter vector is passed on to the optimization algorithm, which finds a new parameter vector with a lower loss function value via one of the different investigated gradient-based methods.
The stopping criterion depends on what can be expected from the molecular model and the optimization workflow. E.g., if the problem is overdetermined ($n > N$), the loss function cannot be expected to be zero for any parameter vector. This is usually the case, if the properties are fitted at different temperatures or pressures simultaneously. Furthermore, the calculated physical properties may contain statistical noise, and in that case, an exact prediction of the optimal parameters is not possible. If the error on a specific property is situated within the error bar of the simulated value due to noise, the result cannot be improved. If the noise is too high, gradient-based methods are not applicable.
The optimization procedure has to be performed within an admissible parameter domain, whenever the range of vailidity of force field parameters is limited. Moreover, if the problem is underdetermined, an infinite number of minima can be present and it has to be assured that a physically meaningful minimum is found.
In order to stay in the admissible domain, there are two possibilities: The first one is to use optimization algorithms with constraints but most of them use penalty functions, whose Lagrangian multipliers are determined by another iteration procedure. This leads, firstly, to an excessive complexity and, secondly, to penalty functions needing evaluations in the prohibited domain. This is not feasible in the present context, as simulation tools cannot be executed with 'forbidden' parameters. Another possibility is to use a step length control algorithm so that the iteration direction does not lead out of the admissible domain. This approach was chosen here but the main disadvantage is the fact that a fast convergency of the step length control algorithm, without significantly increasing the number of function evaluations, cannot be expected. Especially in a neighborhood of the minimum, the convergency will be quite slow, as another goal of the step length control is to find a new parameter vector with a lower loss function value, which becomes more difficult close to the minimum. At some point, the step length control is not capable anymore to find a significantly lower loss function value within a reasonable number of iterations and the resulting steps will be very small in relation to the norm of the descent direction. This leads to a slower convergency of the whole optimization algorithm. Additionally, the graph of the loss function may have the shape of a steep rain drain falling down slowly to the minimum. This shape cannot be handled well by gradient-based algorithms, as they only realize the functional decrease at the walls of the rain drain and not the much smaller one in the direction of rain drain itself. This is because the component in the direction of the walls is much greater than the other one and has therefore a much higher impact. This "rain drain phenomenon" has already been discussed in [36]. Hence, the stopping criterion

$$||\nabla F(\boldsymbol{x})|| \leq \tau,$$

where $\tau > 0$ is a small number, e.g. 10^{-3}, will not be fulfilled within a reasonable number of iteration steps. Hence, a stopping criterion was used, which only refers to the loss function value itself and not to its gradient. The employed stopping criterion was

$$F(\boldsymbol{x}) \leq \tau,$$

where $\tau = 10^{-4}$ or even $\tau = 10^{-5}$. Of course, the attainable value of τ depends on the loss function itself, the number of considered properties and the amount of noise. In GROW, the parameter τ has to be specified by the user.

4. Numerical Optimization Algorithms

We suppose to use gradient-based iteration procedures, i.e. it is started with an initial parameter vector $\boldsymbol{x}^0$ which is updated within the first iteration to $\boldsymbol{x}^1$. In general, optimization algorithms can be described by an updating formula

$$\boldsymbol{x}^{k+1} = g(\boldsymbol{x}^k).$$

The optimization algorithms should converge to the minimum $\boldsymbol{x}^{\text{opt}}$, e.g.

$$\exists_{C<1}\ \forall_{k\geq 0}\ ||\boldsymbol{x}^{k+1} - \boldsymbol{x}^{\text{opt}}|| \leq C||\boldsymbol{x}^k - \boldsymbol{x}^{\text{opt}}||^m,$$

for some $m > 0$ (convergency of mth order). For example, if $m = 1$, the convergency is termed linear, if $m = 2$, it is quadratic. The behavior and speed of convergency is different for each algorithm, which only differ by the choice of g. In this study, some state-of-the-art gradient-based methods were used, which can be found in [37, 38] and can be grouped into two general types:

1. Descent methods,
2. Trust Region methods.

This choice is due to the fact that these algorithms have very good convergency properties—most of them are at least superlinearly convergent—and that the search is directed to the optimum. The Descent methods consistently have the iteration instruction

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + t_k \boldsymbol{d}^k, \tag{3}$$

where $\boldsymbol{d}^k$ is a descent direction, i.e. $\langle \nabla F(\boldsymbol{x}^k), \boldsymbol{d}^k \rangle < 0$, and t_k is a step length, controlling how far the descent direction is followed. As mentioned in section 3, the boundary conditions for the parameter vector require an efficient step length control algorithm. Therefore, the descent direction was divided here by its norm so that

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \tilde{t}_k \frac{\boldsymbol{d}^k}{||\boldsymbol{d}^k||}.$$

Then, only the step length t_k controls the iteration procedure and not the length of the descent direction. Particularly in the initial iteration steps, $||\boldsymbol{d}^k||$ can be huge so that it may lead far out of the admissible domain. This, in turn, may lead to very small step lengths t_k because of a slow convergency of the step length control, which is not desired at the beginning, when the loss function is steep. Hence, in this case, large steps into the direction of the minimum are wanted. Of course, in a neighborhood of the minimum, dividing by the norm would falsify the algorithm, which may not converge further. Therefore, the descent direction was not normalized here, as soon as $||\nabla F|| \leq 1$.
Before the step length control is described in detail, the descent methods are briefly outlined. An important feature of most of the algorithms is their high convergency quality as they converge superlinearly

$$\forall_{k\geq 0}\ ||\boldsymbol{x}^{k+1} - \boldsymbol{x}^{\text{opt}}|| \leq c_k||\boldsymbol{x}^k - \boldsymbol{x}^{\text{opt}}||,$$

where $\lim_{k\to\infty} c_k = 0$.

- **Steepest Descent:** $\boldsymbol{d}^k = -\nabla F(\boldsymbol{x}^k)$. This is the simplest method, where $\boldsymbol{d}^k$ is the direction of the steepest descent. The main drawback is that the convergency is not guaranteed: If $\forall_k\, F(\boldsymbol{x}^{k+1}) < F(\boldsymbol{x}^k)$, then each accumulation point of the iteration sequence is a stationary point of F.

- **Newton Raphson:** $\boldsymbol{d}^k = -(D^2F(\boldsymbol{x}^k))^{-1}\nabla F(\boldsymbol{x}^k)$. This is the multidimensional analogon of the popular root finding 1D-Newton method. If the Hessian D^2F is Lipschitz continuous in a neighborhood of $\boldsymbol{x}^{\text{opt}}$, then the convergency is quadratic. The complexity increases with the computation of the Hessian, but the gain in speed of convergency compensates this drawback. However, in practice, the Hessian can be singular. Therefore, if this is the case or if the descent is not steep enough, i.e.

  $$\langle \nabla F(\boldsymbol{x}^k), \boldsymbol{d}^k \rangle > -\rho ||\boldsymbol{d}^k||^p$$

  for some user-defined $\rho > 0,\ p \in \mathbb{N}^{>2}$, the steepest descent direction $-\nabla F(\boldsymbol{x}^k)$ was taken. There are two problems arising when using the Newton Raphson method:

  1. the Hessian must be computable within a reasonable time,
  2. the Hessian must be robust with respect to noise.

- **Quasi Newton**: $\boldsymbol{d}^k = -H_k^{-1}\nabla F(\boldsymbol{x}^k)$, where H_k is an approximation of the Hessian. The Quasi Newton methods are a heuristic solution to the two problems mentioned above. The approximation is updated at each iteration: $H_k \to H_{k+1}$, where $H_0 = D^2F(\boldsymbol{x}^0)$. Thereby, the so-called secant condition for the theorem of Dennis and Moré

  $$H_{k+1}\left(\boldsymbol{x}^{k+1} - \boldsymbol{x}^k\right) = \nabla F(\boldsymbol{x}^{k+1}) - \nabla F(\boldsymbol{x}^k)$$

  must be fulfilled. Three Quasi Newton methods were considered in this study, which only differ in the updating procedure of H_k:

  - Powell Symmetric Broyden (PSB): If H_k is symmetric, F is Lipschitz continuous in a neighborhood of $\boldsymbol{x}^{\text{opt}}$ and $\sum_{k=0}^{\infty} ||\boldsymbol{x}^k - \boldsymbol{x}^{\text{opt}}|| < \infty$, then the convergency is superlinear.
  - Davidon Fletcher Powell (DFP): If H_k is symmetric positive definite (spd), the convergency is superlinear under the same assumptions as in the case of PSB.
  - Broyden Fletcher Goldfarb Shanno (BFGS): Herein, the inverse Hessian is updated: $B_k \to B_{k+1}$, where $B_0 = (D^2F(\boldsymbol{x}^0))^{-1}$. If F is Lipschitz continuous in a neighborhood of $\boldsymbol{x}^{\text{opt}}$ and $||D^2F(\boldsymbol{x}^k)||_F$ is bounded for all $k \geq 0$, where $||\cdot||_F$ is the Frobenius norm for matrices, the convergency is superlinear as well.

- **Conjugate Gradients (CG):** $\boldsymbol{d}^{k+1} = -\nabla F(\boldsymbol{x}^{k+1}) + \beta_k \boldsymbol{d}^k,\ \boldsymbol{d}^0 = -\nabla F(\boldsymbol{x}^0)$. The descent direction is updated at each step by using the gradient in the following step, which has to be computed before. The method is a transfer of the CG method for LESs, where the residuals, which are the negative gradients of the respective quadratic form that has to be minimized, are conjugated, i.e.

  $$\forall_{k\geq 0}\ \langle \nabla F(\boldsymbol{x}^{k+1}), \nabla F(\boldsymbol{x}^k) \rangle = 0,$$

  to the multidimensional case. In general, the gradients are not conjugated anymore. However, the transfer works naturally with new convergency proofs. There are two main CG methods for optimization tasks, where the calculations of the β_k are different:

– Fletcher Reeves (FR): $\beta_k^{\mathrm{FR}} = ||\nabla F(\boldsymbol{x}^{k+1})||^2/||\nabla F(\boldsymbol{x}^k)||^2$. This is a direct transfer from the one-dimensional case. The convergency is guaranteed, if the level set $\mathcal{L}(\boldsymbol{x}^0) = \{\boldsymbol{x}|F(\boldsymbol{x}) < F(\boldsymbol{x}^0)\}$ is compact and if F is uniformly convex on $\mathcal{L}(\boldsymbol{x}^0)$. Those are conditions for global convergency. Therefore, it is not guaranteed that it converges to a local minimum in the case of non-convexity.

– Polak Ribière (PR): $\beta_k^{\mathrm{PR}} = \langle \nabla F(\boldsymbol{x}^{k+1}) - \nabla F(\boldsymbol{x}^k), \nabla F(\boldsymbol{x}^{k+1})\rangle/||\nabla F(\boldsymbol{x}^k)||^2$. In the one-dimensional case, where the successive residuals $\boldsymbol{r}^k$ are orthogonal, $||\boldsymbol{r}^{k+1}||^2/||\boldsymbol{r}^k||^2 = \langle \boldsymbol{r}^{k+1} - \boldsymbol{r}^k, \boldsymbol{r}^{k+1}\rangle/||\boldsymbol{r}^k||^2$, and therefore, also β_k^{PR} could be used instead of β_k^{FR} in the multidimensional case. This heuristic consideration leads to a good CG method with new convergency proofs. If the level set is compact and if ∇F is Lipschitz continuous in a ball containing the level set, the PR method converges.

All descent methods have the following assumptions in common: First of all, the initial parameter vector must be close to the minimum and in the case of the Quasi Newton methods, also the initial Hessian approximation H_0 must be in a neighborhood of $D^2F(\boldsymbol{x}^{\mathrm{opt}})$, with respect to some matrix norm, e.g. the Frobenius norm. Quite often, it is a challenge to find a good initial parameter vector which has to be specified by the user. There is no automatic procedure to solve this problem. Furthermore, a standard assumption is the existence of a global (or local) optimum, i.e. $\nabla F(\boldsymbol{x}^{\mathrm{opt}}) = 0$ and $D^2F(\boldsymbol{x}^{\mathrm{opt}})$ spd. Also the iteration matrices $D^2F(\boldsymbol{x}^k)$, H_k and B_k should be spd and regular. This is not always guaranteed, whereas all other assumptions like Lipschitz continuity etc. are natural. The positive definiteness of the Hessian is guaranteed, if the loss function is convex. But in the case of H_k and B_k, it is not. If they are not spd, the corresponding direction $\boldsymbol{d}^k$ is not a descent direction.
The step length control is essential for this kind of methods as well: At the beginning, when the function is still steep, large steps are preferred, in order to get as rapidly as possible close to the minimum. On the other hand, the steps must not be too large, as otherwise, it might be jumped beyond the minimum. A very heuristic approach is to simply make large steps far away and small steps near the minimum. These so-called *heuristic* step lengths are user-defined and finding good heuristic step lengths is a trial-and-error procedure, which is not very effective, because at some point, the algorithm will hop around the minimum. This is due to the fact that also the smaller steps will be too large then. Hence, it is aimed for adapted, so-called *efficient* step lengths t_k, which are defined by

$$\exists_{\theta\neq\theta\left(\boldsymbol{x}^k,\boldsymbol{d}^k\right),\ \theta>0}\ F(\boldsymbol{x}^k + t_k\boldsymbol{d}^k) \leq F(\boldsymbol{x}^k) - \theta\left(\frac{\langle\nabla F(\boldsymbol{x}^k), \boldsymbol{d}^k\rangle}{||\boldsymbol{x}^k||}\right)^2.$$

There are two types of step lengths which fulfill this criterion, namely the *Armijo* and the *Wolfe-Powell* step length. Both are again iterative procedures requiring function evaluations for $\boldsymbol{x}^k + t_k^\ell\boldsymbol{d}^k,\ \ell \geq 0$, with an initial step length t_k^0. The Wolfe-Powell algorithm is computationally too costly, as it also requires $\nabla F(\boldsymbol{x}^k + t_k^\ell\boldsymbol{d}^k)$ for each ℓ, and therefore, it is not suitable in the present case. Here, the Armijo step length control was used, which searches for step lengths of the form

$$t_A = \max\{\beta_A^\ell|\ell = 0, 1, ..., F(\boldsymbol{x} + \beta_A^\ell\boldsymbol{d}) \leq F(\boldsymbol{x}) + \zeta_A\beta_A^\ell\langle\nabla F(\boldsymbol{x}), \boldsymbol{d}\rangle\}, \tag{4}$$

with $0 < \beta_A, \zeta_A < 1$. The Armijo step length only exists, if $\boldsymbol{d}$ is a descent direction. Then, the convergency is guaranteed to be monotonous. Hence, the assumption $\forall_k\ F(\boldsymbol{x}^{k+1}) < F(\boldsymbol{x}^k)$ for the convergency of the Steepest Descent algorithm is fulfilled by the use of the Armijo step length control. Please note that there is a '+' instead of a '-', as $\langle\nabla F(\boldsymbol{x}), \boldsymbol{d}\rangle < 0$ is required. The only drawback, compared to the Wolfe-Powell step length, lies in the fact that the Armijo step lengths rapidly become too small. In particular, this is true if the compliance of the boundary conditions

is controlled by the step length. To achieve this, an admissible step length was calculated at each iteration so that $\boldsymbol{x}^{k+1}$ remains within the admissible domain: Let $[\min_i^k, \max_i^k]$ be the admissible interval of the ith component of $\boldsymbol{x}^k$ ($i = 1, ..., \dim(\boldsymbol{x}^k)$). Then, the admissible step length t_k^{adm} is given by

$$t_k^{\text{adm}} = \max\{t_k > 0 | \forall_{i=1,...,\dim(\boldsymbol{x}^k)} \ \min_i^k \leq x_i^k + t_k d_i^k \leq \max_i^k\},$$

where $\beta_A = t_k^{\text{adm}}$ was set and the Armijo step length control was started at $\ell = 1$, where $\ell = 1$ is not allowed as a solution, as this would lead to the border of the admissible domain. Please note that $t_A > 1$ is also possible, if $\boldsymbol{d}^k$ is a descent direction. In this case, $t_A = s\beta^\ell$ for some $s > 1$, which is a generalization of the Armijo step length. In particular, this is important at points which are very close to the minimum, when ∇F does not lead out of the admissible domain anymore. Then, in the sense of Equation (3), t_A has to be increased to avoid too small step lengths. This is especially true, if $t_A = 1$ does not lead to a point beyond the minimum. However, the Armijo step length control will not converge within a reasonable number of iterations at some point before $t_A > 1$ is required. Furthermore, as $\beta_A^\ell \overset{\ell\to\infty}{\to} 0$ and β_A^1 will already be very small in relation to the range of the force field parameters due to the boundary conditions, the resulting step length will be too small so that no profit is drawn. Hence, it will not lead the whole optimization procedure to the stopping criterion $\nabla F(\boldsymbol{x}^{\text{opt}}) = 0$ within a reasonable number of iterations. Furthermore, the rain drain phenomenon mentioned in section 3 cannot be handled by gradient-based algorithms, even if the step lenghts are efficient. Hence, at some point, it will not be possible anymore to find a step length which leads to a significantly smaller function value, as the descent direction is not parallel to the rain drain direction. Therefore, the optimization was terminated in this work, if the step length control algorithm took more than a certain number of iterations, even if the stopping criterion was not fulfilled, and it was concluded that the optimization algorithm is not suitable for the regarded optimization task.

The second type of gradient-based optimization methods are the Trust Region methods. The step length control is contained in the algorithm itself. At each iteration k, a step length Δ_k is defined so that $B_{\Delta_k}(\boldsymbol{x}^k)$ is contained in the admissible domain. The compact ball $B_{\Delta_k}(\boldsymbol{x}^k)$ is called the *Trust Region* for $\boldsymbol{x}^k$. For each Δ_k, it is searched for a suitable direction $\boldsymbol{d}^k$ and $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \boldsymbol{d}^k$ is computed. This is done by an internal iteration procedure: While $\boldsymbol{d}^k$ is not suitable, Δ_k is decreased by a factor $0 < \gamma_1 < 1$. Otherwise, Δ_k is increased for reasons of precaution by a factor $\gamma_2 > 1$ under certain assumptions. But what does "suitable" mean in this case? The loss function is interpolated by a quadratic form stemming from a Taylor expansion around $\boldsymbol{x}^k$:

$$q_{\boldsymbol{x}^k}(\boldsymbol{d}^k) := F(\boldsymbol{x}^k + \boldsymbol{d}^k) \ddot{=} F(\boldsymbol{x}^k) + \langle \nabla F(\boldsymbol{x}^k), \boldsymbol{d}^k \rangle + \frac{1}{2}(\boldsymbol{d}^k)^T D^2 F(\boldsymbol{x}^k) \boldsymbol{d}^k. \tag{5}$$

Then $q_{\boldsymbol{x}^k}$ is minimized with respect to $||\boldsymbol{d}^k|| \leq \Delta_k$ and compared to the original loss function. This is done by evaluating the ratio

$$r^k := \frac{F(\boldsymbol{x}^{k+1}) - F(\boldsymbol{x}^k)}{q_{\boldsymbol{x}^k}(\boldsymbol{d}^k) - q_{\boldsymbol{x}^k}(0)} \overset{(5)}{=} \frac{F(\boldsymbol{x}^{k+1}) - F(\boldsymbol{x}^k)}{q_{\boldsymbol{x}^k}(\boldsymbol{d}^k) - F(\boldsymbol{x}^k)}.$$

If $r^k \approx 0$, the quadratic model does not coincide well with the original loss function and overestimates the decrease of F. On the other hand, if $r^k \approx 1$, the decrease of F is much higher than the decrease of the quadratic model and Δ_k can be increased. In practice, the user can define two parameters $0 < \eta_1 < \eta_2$, in order to decide, whether $\boldsymbol{d}^k$ is accepted and Δ_k is increased or

not. The direction is accepted, if $r^k \geq \eta_1$, and if even $r^k > \eta_2$, then Δ_k is increased.
The Trust Region methods require the computation of the Hessian but the quality of the convergency is much better than in the case of the descent methods: If D^2F is Lipschitz continuous in $\boldsymbol{x}^{\text{opt}}$, then

$$\forall_{t\in(0,1)}\ \forall_{k\geq 0}\ ||\boldsymbol{x}^k - \boldsymbol{x}^{\text{opt}}|| \leq c_k t^k$$

with $\lim_{k\to\infty} c_k = 0$.
The only difficulty lies in the minimization of $q_{\boldsymbol{x}^k}$, because it is a nonlinear constrained optimization problem. This so-called *Trust Region subproblem* can be solved in two ways: The first one is an approximate solution, which can be achieved by the *Double Dog Leg* algorithm (DD), a geometrically based method considering two descent directions. The main drawback of the DD method is the assumption that the Hessian is positive definite, which is often not fulfilled. The second way is to solve the Trust Region subproblem exactly. This is done by an eigenvalue decomposition of the Hessian. A solution is always obtained, even if the Hessian is not positive definite.
For details concerning the step length control and numerical optimization algorithms, cf. [37, 38].

5. Practical Aspects: Some Tricks of the Trade

Before the detailed description of the GROW optimization tool is given, some specific practical adaptations are elucidated:

1. The stopping criterion was chosen with the Steepest Descent method: Whenever the Armijo step length control did not converge within a reasonable number of iterations (usually 100), the actual iteration $\boldsymbol{x}^k$ was considered as optimal and the stopping criterion $F(\boldsymbol{x}) \leq \tau$ was used for all other algorithms, where τ is a small upper bound for $F(\boldsymbol{x}^k)$. The Steepest Descent method was used as a reference method because of a heuristic claim: If a method is not capable to find a parameter vector with a lower function value, following the *steepest* descent direction, another method using a different direction is not either. This claim turned out to be true in the case of the Newton and Quasi Newton methods, but not for the CG and Trust Region methods. They were terminated as well, whenever $F(\boldsymbol{x}) \leq \tau$ so that a meaningful comparison between the algorithms could be achieved. However, some more iterations leading closer to the minimum were performed in addition.
2. The convergency of the Armijo step length control can be manipulated by the control parameter ζ_A. The smaller ζ_A, the weaker is the acceptance constraint of the step length. This leads to lower function values in the next iteration but also to a faster convergency of the Armijo step length control. By default, $\zeta_A = 0.5$, but in some cases, the Armijo constraint has to be weakened so that $\zeta_A = 0.1$, and sometimes even drastically so that $\zeta_A = 0.001$.
   The same holds for the Trust Region control parameter η_1, which controls the quality of the quadratic model. By default, $\eta_1 = 0.7$, but in order to avoid a slow convergency of the Trust Region subproblem, its stopping criterion has to be weakened. Then, $\eta_1 = 0.5$ or even $\eta_1 = 0.2$ was specified.
3. In the case of noise, the discretization for the gradient and Hessian must be coarser: If the gradient is calculated too accurately, i.e. by $h = 10^{-5}$, the algorithm will get stuck in a local minimum formed by the oscillations due to noise. As this is not desirable, $h = 10^{-2}$ was used, in order to compute only the drift of the loss function so that the overall decrease was considered. Of course, this is not reasonable in a neighborhood of the minimum and setting h back to 10^{-5} would also be meaningless, because then the algorithm would get

stuck in an artificial minimum again. But on the other hand, in the case of noise, it suffices to reach a neighborhood of the minimum, as the target parameters are predictable only within some error bars around the minimum. Therefore, the stopping criterion was weakened as well, i.e. $F(\boldsymbol{x}) \leq \tilde{\tau}$ with $\tilde{\tau} > \tau$.

4. The size of the admissible domain is also an important criterion for the convergency. If it is too small, the global minimum can be situated at its boundary. In this case, a few iterations with a heuristic step length were performed, in order to reach a new initial point at the boundary, and a new admissible domain had to be defined.
5. The idea of normalizing $\boldsymbol{d}$ is heuristic and does not always lead to a better performance. In cases where $\boldsymbol{d}$ is not a descent direction, which is possible for the Quasi Newton methods, and where $\boldsymbol{d}$ does not lead out of the admissible domain, normalizing it can mean limiting the search for a lower loss function value, as $t_A \in (0, 1)$ in this case, which leads to a much slower convergency. But on the other hand, not normalizing can also mean that the Armijo step lengths t_A become rapidly too small. When $\boldsymbol{d}$ is not a descent direction, the only possibility to get a lower function value is the Armijo algorithm, which depends on $\boldsymbol{d}$ and its norm. Therefore, in the case of PSB, the descent direction was not normalized. Otherwise, the convergency was much slower and the value of the loss function did not change significantly.

6. GROW: Implementation Issues

The input of the program GROW only consists of a configuration file, where all further input files and program issues are defined by the user. All input/output files, except for trajectory files[2] associated the simulation tool used, are in ASCII format and intended to be human-readable. The optimization is started by the command

```
python grow.py <config-file>
```

where `config-file` is the name of the configuration file, e.g. `grow.cfg`.

6.1. Configuration File

The configuration files comprises all input files required for the optimization workflow. Moreover, all workflow options have to be defined, e.g. the numerical optimization algorithm used, the step length control, and some corresponding input parameters. The file consists of two sections, 'SIM' and 'OPT': The first section contains all input files and parameters for the molecular simulation and the output directories. The second section contains specific adjustments concerning the optimization algorithms. A typical configuration file has the following form:

```
[SIM]
program: gromacs                                # simulation tool
bin_path: /home/user/local/gromacs-4.0.2/bin/   # binary path for simulation tool
topology: ../input/topology.top                 # topology file of simulation tool
coordinates: /home/user/input/start.pdb         # coordinate file of simulation tool
sim_in: /home/user/input/sim.mdp                # input file of simulation tool
target: ../input/exp_properties.target          # file containing the target properties
```

[2]The trajectory of a molecular simulation describes the temporal progression of motion states within the total system. The files containing such trajectories are usually huge and not human-readable.

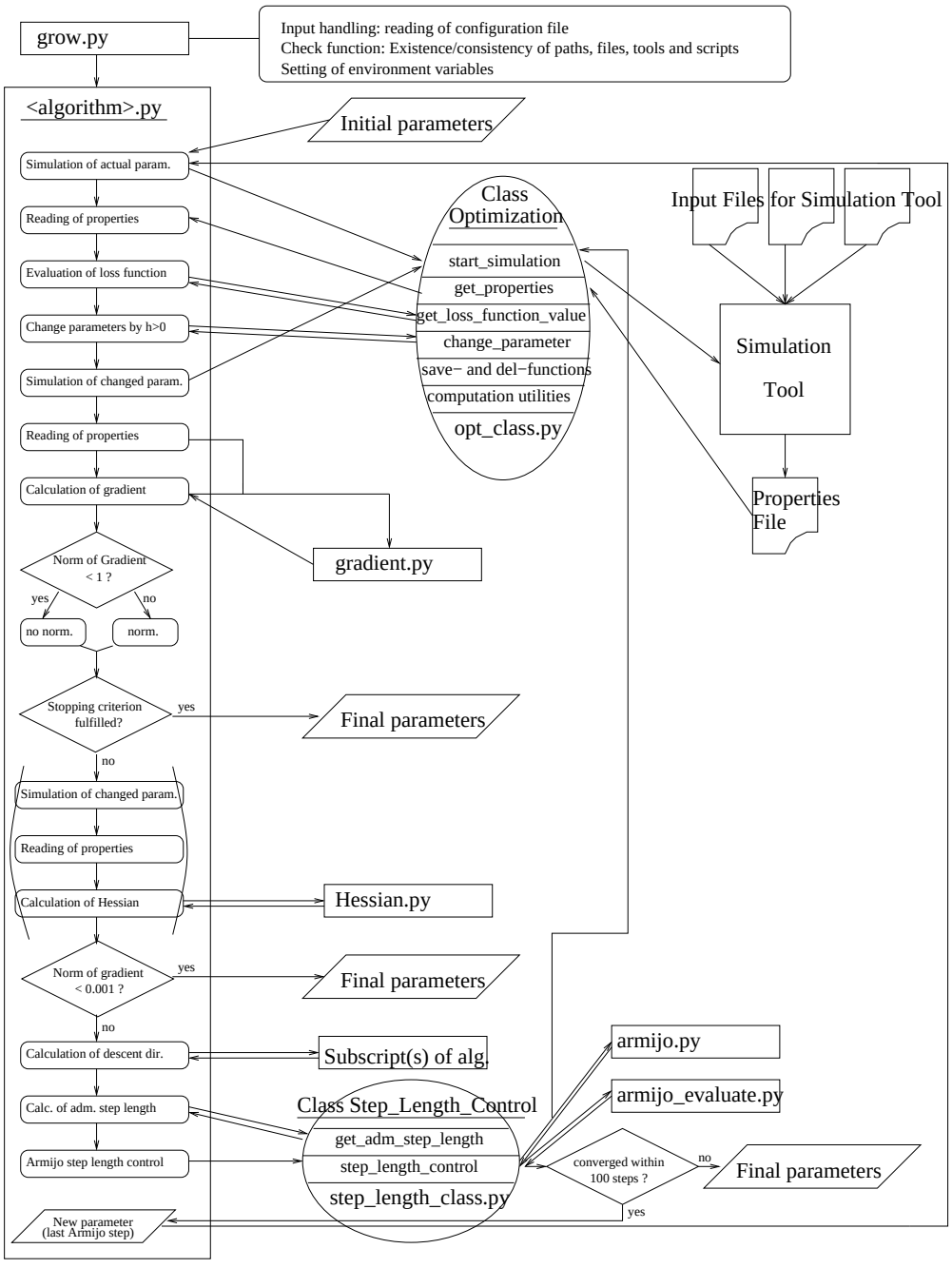

Figure 2: Program Structure of GROW: The main control script `grow.py` calls—after some input handling and checking routines—the algorithm control script, named `<algorithm>.py`. This script executes the main algorithm specific workflow, calling different general subscripts like `gradient.py` or `Hessian.py`. Furthermore, it calls functions implemented in the *python* classes `Optimization` and `Step_Length_Control`. The class `Optimization` contains a function named `start_simulation`, which starts a simulation tool and handles all computation utilities for the optimization itself. The class `Step_Length_Control` executes the Armijo step length control algorithm, calling two subscripts named `armijo.py` and `armijo_evaluate.py`. For each optimization algorithm, one or more help subscripts have been implemented, which calculate the specific descent direction.

```
parameters: ../input/initial.para               # file containing the initial force field parameters
par_file_name: ../input/parameters.para         # file used by the optimization tool for all intermediate parameters
substance: IL                                   # text field: name of chemical system to be simulated
optoutpath: steepest_descent/armijo/final_out   # outpath containing the results of the optimization workflow
tmppath: steepest_descent/armijo/tmp            # temporary path containing intermediate help files
outpath: steepest_descent/armijo/out            # outpath of the simulation tool
delete: n                                       # delete tmppath and outpath?

[OPT]
method: conjugate_gradient                      # numerical optimization method
cg: fletcher_reeves                             # CG algorithm
gradient: numerical                             # how to calculate the gradient (GROW version 1.0: only numerically)
sl_method: armijo                               # step length control (heuristic/armijo)
zeta: 0.5                                       # regularization parameter of Armijo step length control
h: 0.00001                                      # discretization parameter for gradient and Hessian
sigma_bound: 10                                 # LJ size parameter is not changed by more than 10% (adm. domain)
epsilon_bound: 40                               # LJ energy parameter is not changed by more than 40% (adm. domain)
limit: 0.0001                                   # stopping criterion
```

In this example, GROMACS[3] (version 4.0.2) was chosen as molecular simulation tool and the optimization was performed by the Fletcher Reeves method.

6.2. Program Structure

Figure 2 shows the structure of GROW. The main control script `grow.py` executes the input handling and reads the configuration file. Then, it checks the existence and consistency of the paths, files and scripts. Furthermore, it defines environment variables.
The main control script of the numerical optimization algorithm, e.g. `conjugate_gradient.py`, regulates the optimization itself. It creates an object of the *python* class `Optimization`, contained in a file named `opt_class.py`. This class contains all functions handling the optimization. E.g., it changes the force field parameters for the gradient and Hessian calculation, starts the simulation tool, reads the resulting physical properties and computes the loss function. Moreover, all necessary computation utilities, e.g. geometrical and statistical calculations as well as reading and writing routines, are implemented in this class. Please note that in Computer Science, the functions of a class are called *methods*. But for obvious reasons, this term is not used in this work.
The gradient calculation is performed by a script called `gradient.py` and the Hessian calculation by a script named `Hessian.py`. As long as $||\nabla F(\boldsymbol{x}^k)|| \geq 1$, the descent direction is normalized, except in the case of PSB. The evaluation of the actual iteration is performed by the control script of the optimization algorithm, i.e. it is checked whether the norm of the gradient is smaller than a meaningful threshold (usually 0.001) or whether the stopping criterion for the loss function is fulfilled. If this is not the case, the descent direction is calculated by subscripts of the corresponding algorithm and the step length control is executed by a script named `step_length_class.py` containing the *python* class `Step_Length_Control`. This class contains all necessary functions for this task, e.g. the computation of the admissible step length and a function executing the step length control algorithm itself. In the case of the Armijo step length control, two subscripts are called: Firstly, the script `armijo.py` calculates an Armijo step length β_A^ℓ and secondly, the script `armijo_evaluate.py` checks if this step length is accepted in the sense of constraint (4), or not. The last iteration of the Armijo step length control is the new iteration of the whole optimization algorithm. Then, a new simulation is performed with the new force field parameters, and the algorithm continues. If the Armijo step length control does not converge within a reasonable number of iterations (usually 100), the optimization algorithm

[3] `http://www.gromacs.org/`

is interrupted and the actual parameters are taken as final parameters.
The algorithm terminates, if the stopping criterion is fulfilled.

6.3. Subscripts needed by the optimization algorithms

In the following, all scripts and subscripts supporting the optimization workflow are indicated and explained in alphabetical order. All *python* scripts end with '.py', all scripts written in *S+*, related to the statistics package *R*, end with '.R' and all *shell* (tcsh) scripts end with '.sh'.
Please note that GROW version 1.0 is a generic implementation but only contains concrete interfaces to the molecular dynamics simulation tools GROMACS (version 4.0.2) and YASP[4]. Moreover, the calculations of specific vapor-liquid equilibrium (VLE) properties from force field parameters can be realized by the fit functions developed by [34]. However, this is only possible for two-center LJ particles with a dipolar or a quadrupolar moment.

armijo.py
Calculates the Armijo step lengths $t_A = \beta_A^\ell,\ \ell = 2, 3, 4, ...$ Note that β_A^1 is computed in the class *Step_Length_Class* by the function *get_admissible_step_length*. It is the admissible step length within the admissible compact domain.

armijo_evaluate.py
Evaluates the Armijo step length calculated by *armijo.py*. If the Armijo constraint (4) is met, the actual step length is taken and the step length control ends.

bfgs.R
Performs the update $H_k \to H_{k+1}$ of the Hessian approximation in the case of the BFGS method.

bfgs_evaluate.py
Calculates the descent direction in the case of the BFGS method. Within all other Quasi Newton methods, the script *newton_evalutate.py* is taken for this task. In the case of BFGS, no LES has to be solved but only a matrix-vector multiplication has to be performed.

calculate_Hessian_inverse.R
Calculates the inverse of the Hessian, which can be performed by the *R*-built-in function *solve*.

calculate_newton_descent.R
Calculates the descent direction in the case of the Newton Raphson and Quasi Newton methods (except BFGS). This is done by the solution of a LES, which can be performed by the *R*-built-in function *solve*.

cholesky.R
Performs a Cholesky decomposition of an spd matrix, which can be performed by the *R*-built-in function *chol*. This is required, when the Trust Region subproblem is solved exactly by an eigenvalue decomposition of the Hessian.

conjugate_gradient.py

[4] `http://www.theo.chemie.tu-darmstadt.de/group/services/yaspdoc/yaspdoc.html`

Control script of the CG algorithms.

determinant.R
Calculates the determinant of a matrix, which can be performed by the *R*-built-in function *determinant*. By this script, it can be decided if a matrix is singular or not.

dfp.R
Performs the update $H_k \to H_{k+1}$ of the Hessian approximation in the case of the DFP method.

double_dog.py
Solves the Trust Region subproblem by the Double Dog Leg algorithm and returns the minimum of the quadratic form (5).

exact_TR_solution.py
Solves the Trust Region subproblem exactly by an eigenvalue decomposition of the Hessian and returns the minimum of the quadratic form (5).

fletcher_reeves_evaluate.py
Calculates the descent direction in the case of the Fletcher Reeves method.

grow.py
Main control script. Reads the configuration file and checks, whether all input files exist and are consistent with each other. Then, it starts the control script of the corresponding numerical optimization algorithm.

gradient.py
Computes a numerical approximation of the gradient. The gradient of the quadratic loss function itself is determined analytically and the partial derivatives of the physical properties are approximated numerically by finite differences.

Hessian.py
Computes the Hessian by finite differences of the loss function related to its second partial derivatives.

korrmD.R
Calculates specific VLE properties by fit functions [34] from force field parameters. The considered molecules must be two-center LJ particles with a dipolar moment.

korrmQ.R
Calculates specific VLE properties by fit functions [34] from force field parameters. The considered molecules must be two-center LJ particles with a quadrupolar moment.

mkjobequigromacs.sh
Control shell script for GROMACS, which writes the actual force field parameters into the GROMACS topology file and performs equilibration runs. When the system is equilibrated, a production run is executed in order to calculate the physical properties as thermodynamic averages.

mkjobequiyasp.sh

Control shell script for YASP having the same features as *mkjobequigromacs.sh*.

newton.py
Control script of the Newton Raphson algorithm.

newton_evaluate.py
Calculates the descent direction in the case of the Newton Raphson and Quasi Newton methods (except BFGS).

opt_class.py
Contains the main *python* class of the optimization workflow. In this class, named *Optimization*, all functions for the optimization itself are implemented. The most important function is *start_simulation*, which starts a molecular simulation tool via shell script. Furthermore, the class contains computation utilities required for the optimization.

polak_ribiere_evaluate.py
Calculates the descent direction in the case of the Polak Ribière method.

positive_definiteness_test.R
Checks whether a matrix is positive definite or not. This can be realized by the *R*-built-in function *eddcomp*, which calculates the eigenvalues of a matrix.

psb.R
Performs the update $H_k \to H_{k+1}$ of the Hessian approximation in the case of the PSB method.

quasi_newton.py
Control script of the Quasi Newton methods.

solve_LES.R
Solves a LES, which can be performed by the *R*-built-in function *solve*.

steepest_descent.py
Control script of the Steepest Descent method.

steepest_descent_evaluate.py
Calculates the descent direction in the case of the Steepest Descent method.

step_length_class.py
Contains the *python* class *Step_Length_Class*, where all functions for the step length control are implemented. The admissible step length is computed and the Armijo step length control is performed by calling the subscripts *armijo.py* and *armijo_evaluate.py*.

trust_region.py
Control script of the Trust Region method.

6.4. Summary result file generated during the iterations

During the iterations, a summary result file is automatically created for human use. It is written into the directory `optoutpath`, specified in the configuration file. It contains the actual parameter vector, the gradient and its norm, the calculated physical properties at each temperature,

the mean average percental error (MAPE) for each property over the temperature range, and the value of the loss function.
Finally, the optimal parameter vector, the optimal loss function value, the number of iterations and function evaluations, i.e. how often the simulation tool had to be started, and the gradient at the optimal parameter vector together with its norm are written.
An example of a summary file is given below, generated by a steepest descent optimization workflow using the fit functions [34] in order to calculate the physical properties of the quadrupolar two-center LJ model for nitrogen. The considered properties were the enthalpy for vaporization (`vapor`) and the saturated liquid density (`density`) at the temperatures 65, 75, 85, 95, 105, and 115 K. The stopping criterion was $F(\boldsymbol{x}) \leq 0.01$.

```
Optimization Workflow 2009/3/23 10:52:38
Program: fits, Method: steepest_descent
Configuration file: /home/user/nitrogen/steepest_descent/armijo/T_range/vap/grow.cfg

Values for x0:
x0 = 0.3101 0.331 0.02073 0.10464
gradient = 28.17 4.56 15.38 -47.90
norm of gradient = 57.84
T = 65.0: calc vapor = 6.3367, exp vapor = 5.9800, abs error = 0.3567, rel error = 0.0596
T = 65.0: calc density = 886.2005, exp density = 859.6000, abs error = 26.6005, rel error = 0.0309
T = 75.0: calc vapor = 6.2757, exp vapor = 5.6600, abs error = 0.6157, rel error = 0.1088
T = 75.0: calc density = 845.1734, exp density = 816.6700, abs error = 28.5034, rel error = 0.0349
T = 85.0: calc vapor = 6.2172, exp vapor = 5.2800, abs error = 0.9372, rel error = 0.1775
T = 85.0: calc density = 801.4362, exp density = 770.1300, abs error = 31.3062, rel error = 0.0407
T = 95.0: calc vapor = 6.2888, exp vapor = 4.8000, abs error = 1.4888, rel error = 0.3102
T = 95.0: calc density = 754.0496, exp density = 718.2600, abs error = 35.7896, rel error = 0.0498
T = 105.0: calc vapor = 5.3340, exp vapor = 4.1700, abs error = 1.1640, rel error = 0.2791
T = 105.0: calc density = 701.3797, exp density = 657.5200, abs error = 43.8597, rel error = 0.0667
T = 115.0: calc vapor = 4.4874, exp vapor = 3.2600, abs error = 1.2274, rel error = 0.3765
T = 115.0: calc density = 640.0994, exp density = 578.7000, abs error = 61.3994, rel error = 0.1061
MAPE on Density = 5.49
MAPE on Vapor = 21.86
loss = 0.384781

Values for x1:
x1 = 0.308706 0.330774 0.019966 0.107011
gradient = 21.70 4.15 11.56 -35.97
norm of gradient = 43.76
T = 65.0: calc vapor = 6.2347, exp vapor = 5.9800, abs error = 0.2547, rel error = 0.0426
T = 65.0: calc density = 876.4636, exp density = 859.6000, abs error = 16.8636, rel error = 0.0196
T = 75.0: calc vapor = 6.1731, exp vapor = 5.6600, abs error = 0.5131, rel error = 0.0907
T = 75.0: calc density = 834.7286, exp density = 816.6700, abs error = 18.0586, rel error = 0.0221
T = 85.0: calc vapor = 6.1118, exp vapor = 5.2800, abs error = 0.8318, rel error = 0.1575
T = 85.0: calc density = 790.1106, exp density = 770.1300, abs error = 19.9806, rel error = 0.0259
T = 95.0: calc vapor = 5.9631, exp vapor = 4.8000, abs error = 1.1631, rel error = 0.2423
T = 95.0: calc density = 741.5581, exp density = 718.2600, abs error = 23.2981, rel error = 0.0324
T = 105.0: calc vapor = 5.0845, exp vapor = 4.1700, abs error = 0.9145, rel error = 0.2193
T = 105.0: calc density = 687.1901, exp density = 657.5200, abs error = 29.6701, rel error = 0.0451
T = 115.0: calc vapor = 4.2219, exp vapor = 3.2600, abs error = 0.9619, rel error = 0.2951
T = 115.0: calc density = 622.9822, exp density = 578.7000, abs error = 44.2822, rel error = 0.0765
MAPE on Density = 3.70
MAPE on Vapor = 17.46
loss = 0.239212

...

Values for x50:
x50 = 0.304200 0.326315 0.011931 0.114235
gradient = -0.29 0.12 0.06 0.42
norm of gradient = 0.53
T = 65.0: calc vapor = 5.8284, exp vapor = 5.9800, abs error = -0.1516, rel error = -0.0254
T = 65.0: calc density = 866.5656, exp density = 859.6000, abs error = 6.9656, rel error = 0.0081
T = 75.0: calc vapor = 5.7651, exp vapor = 5.6600, abs error = 0.1051, rel error = 0.0186
T = 75.0: calc density = 820.6438, exp density = 816.6700, abs error = 3.9738, rel error = 0.0049
T = 85.0: calc vapor = 5.6910, exp vapor = 5.2800, abs error = 0.4110, rel error = 0.0778
```

```
T = 85.0: calc density = 770.9424, exp density = 770.1300, abs error = 0.8124, rel error = 0.0011
T = 95.0: calc vapor = 4.9639, exp vapor = 4.8000, abs error = 0.1639, rel error = 0.0342
T = 95.0: calc density = 715.7594, exp density = 718.2600, abs error = -2.5006, rel error = -0.0035
T = 105.0: calc vapor = 4.1979, exp vapor = 4.1700, abs error = 0.0279, rel error = 0.0067
T = 105.0: calc density = 651.6152, exp density = 657.5200, abs error = -5.9048, rel error = -0.0090
T = 115.0: calc vapor = 3.1471, exp vapor = 3.2600, abs error = -0.1129, rel error = -0.0346
T = 115.0: calc density = 568.8182, exp density = 578.7000, abs error = -9.8818, rel error = -0.0171
MAPE on Density = 0.73
MAPE on Vapor = 3.29
loss = 0.009933

Optimal set of parameters: 0.304200 0.326315 0.011931 0.114235
Value of loss function: 0.009933
Number of iterations: 50
Number of function evaluations: 346
Gradient: -0.29 0.12 0.06 0.42
Norm of gradient: 0.53
```

6.5. Extension Possibilities for Developers

GROW is a generic implementation and can be easily extended by developers: An interface to a new simulation tool can be realized within the function `start_simulation` from the class `Optimization`. This function must call a new script handling the equilibration and production runs of the new simulation tool, comparable to `mkjobequigromacs.sh` and `mkjobequiyasp.sh`. The physical properties must be written into ASCII files, containing tables. The first column must contain the time steps and the second column must consist of the properties at each time step. These tables have to be read with the function `get_properties` from the class `Optimization`.
For the application of a new numerical optimization algorithm, a new control script for this algorithm has to be written with specific subscripts. If it is gradient-based, it can use the subscript `gradient.py` and if it requires the calculation of a Hessian, it can use `Hessian.py`. It can also perform a step length control by calling the class `Step_Length_Class`. Of course, it can use all computation utilities implemented in the class `Optimization`. All necessary parameters for the new optimization method must be indicated in the configuration file and read by the main control script `grow.py`.

7. Proof of Concept: Case Study for Nitrogen

As a proof of concept, we consider the optimization of the four state-independent parameters of the quadrupolar two-center LJ (2CLJQ) model [34] for nitrogen. The 2CLJQ model consists of two identical LJ sites, separated by a constant elongation L and a point quadrupole site at the center of mass with a moment Q, which is oriented along the molecular axis. The pair potential u_{2CLJQ} is given by

$$\begin{aligned} u_{\mathrm{2CLJQ}}(\boldsymbol{r}_{ij}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_j, L, Q^2) &= \sum_{a=1}^{2}\sum_{b=1}^{2} 4\epsilon \left[\left(\frac{\sigma}{r_{ab}}\right)^{12} - \left(\frac{\sigma}{r_{ab}}\right)^{6} \right] \\ &+ \frac{3}{4}\frac{Q^2}{||\boldsymbol{r}_{ij}||^5}[1 - 5(\cos^2\theta_i + \cos^2\theta_j) - 15\cos^2\theta_i\cos^2\theta_j \\ &+ 2(\sin\theta_i \sin\theta_j \cos\phi_{ij} - 4\cos\theta_i\cos\theta_j)^2]. \end{aligned}$$

Therein, $\boldsymbol{r}_{ij}$ is the center-center distance vector of two molecules i and j, r_{ab} is one of the four LJ site-site distances, where a refers to the two sites of molecule i and b to the two sites of molecule j. Furthermore, $\boldsymbol{\omega}_i$ and $\boldsymbol{\omega}_j$ represent the orientations of the two molecules, θ_i the azimuthal

angle between the axis of the molecule i and the center-center connection line, and ϕ_{ij} the angle between the axes of molecules i and j. For more details, cf. [34].
Critical values of temperature and density, as well as the saturated liquid density, saturated vapor density, and vapor pressure are available as fit functions of the four force field parameters [34]. In the cited work, the VLE data was determined by molecular simulations for 30 individual 2CLJQ fluids. The calculated properties thereof were fitted by nonlinear regression functions, whose coefficients were determined individually for each property, over a range of temperatures. In order to obtain VLE data for the whole range of $Q^2/(\epsilon\sigma^5)$, L/σ and Tk_B/ϵ, where k_B is the Boltzmann constant, the simulation data was globally fitted. In this regard, the critical data $T_c(\epsilon, \sigma, Q^2, L)$ and $\rho_c(\epsilon, \sigma, Q^2, L)$, the saturated liquid density $\rho_l(\epsilon, \sigma, Q^2, L, T)$, the saturated vapor density $\rho_v(\epsilon, \sigma, Q^2, L, T)$, as well as the vapor pressure $p_\sigma(\epsilon, \sigma, Q^2, L, T)$ were considered to be the key VLE data for an adjustment to real fluids.
The force field parameter vector of the 2CLJQ model is the four-dimensional vector $(\epsilon, \sigma, Q^2, L)^T$. The considered physical properties were the saturated liquid density ρ_l and the enthalpy of vaporization Δh_v at $T = 75$ K. The latter was calculated from the vapor pressure and the saturated densities via the Clausius-Clapeyron equation. The initial parameter vector was taken from the literature [39] but modified a little so that the optimization workflow did not start with an optimal parameter vector already. The admissible domain was defined as follows: The range of validity for Q^2 is given by $Q^2/(\epsilon\sigma^5) \in [0, 4]$ and for L, it is given by $L/\sigma \in [0, 0.8]$. The LJ parameters ϵ and σ are not changed by more than 40% and 10%, respectively. The stopping criterion was $F(\boldsymbol{x}) \leq 10^{-5}$.

Table 1 shows the results of all considered numerical optimization algorithms. It contains the following details:

- Optimization method (Algorithm): One of the above mentioned numerical optimization methods.
- Number of Iterations (# Iter.): Those are the iteration steps required, in order to fulfill the stopping criterion. Hence, it is the $P \in \mathbb{N}$, so that $\boldsymbol{x}^P = \boldsymbol{x}^{\text{opt}}$, starting from $\boldsymbol{x}^0$. The number of iterations indicates the quality of the convergency.
- Number of Function Evaluations (# Eval.): This number indicates how often the loss function or its partial derivatives had to be evaluated, i.e. how often the physical properties had to be calculated or a molecular simulation would have had to be performed, until the stopping criterion was fulfilled. This number increases with the computation of the gradient and the Hessian as well as the number of iterations required for the Armijo step length control. If the algorithm was interrupted for some reasons, before the stopping criterion was fulfilled, the column contains '$\boldsymbol{x}^{\tilde{P}} =$', where $\tilde{P}$ is the number of iterations so far, followed by the parameter $\boldsymbol{x}^{\tilde{P}}$ itself in the next column.
- Final Parameter ($\boldsymbol{x}^{\text{opt}}$): The final parameter, either the optimal one, for which the stopping criterion is fulfilled, or the last one before the optimization was interrupted.
- Error on Liquid Density (ρ_l): The percental error on saturated liquid density.
- Error on Enthalpy of Vaporization (Δh_v): The percental error on enthalpy of vaporization.
- Value of Loss Function ($F(\boldsymbol{x}^{\text{opt}})$): The value of the loss function for the final or last parameter. Whenever the stopping criterion is fulfilled, this value is less than the upper bound contained in the criterion. The loss function should be as close to zero as possible.

Density and Enthalpy of Vaporization, $T = 75$ K, VLE Fit Functions							
Algorithm	# Iter.	# Eval.	$\boldsymbol{x}^{\mathrm{opt}}$	ρ_l	Δh_v	$F(\boldsymbol{x}^{\mathrm{opt}})$	$\|\|\nabla F(\boldsymbol{x}^{\mathrm{opt}})\|\|$
Steepest Descent	9	62	0.300120 0.325710 0.013278 0.115343	-0.03%	0.19%	$4{\cdot}10^{-6}$	0.02
Newton Raphson	9	152	0.300120 0.325710 0.013278 0.115343	-0.03%	0.19%	$4{\cdot}10^{-6}$	0.02
PSB (Quasi Newton)	$2^\star$	$\boldsymbol{x}^2 =$	0.301444 0.327477 0.014151 0.114339	-0.58%	1.46%	$2.5{\cdot}10^{-5}$	0.17
DFP (Quasi Newton)	$4^\star$	$\boldsymbol{x}^4 =$	0.301445 0.327476 0.014145 0.114337	-0.58%	1.46%	$2.5{\cdot}10^{-5}$	0.17
BFGS (Quasi Newton)	$8^\star$	$\boldsymbol{x}^8 =$	0.301446 0.327473 0.014145 0.114338	-0.83%	0.35%	$8.1{\cdot}10^{-5}$	7.21
Fletcher Reeves (CG)	36	235	0.300137 0.325656 0.013175 0.115355	-0.01%	0.16%	$3{\cdot}10^{-3}$	0.02
Polak Ribière (CG)	6	41	0.300215 0.325784 0.013409 0.115276	-0.03%	0.29%	$8{\cdot}10^{-6}$	0.03
Trust Region (DD)	No solution of Trust Region partial problem found: Hessian not spd						
Trust Region (Exact)	6	105	0.299802 0.325402 0.013067 0.115660	-0.03%	-0.14%	$2{\cdot}10^{-6}$	0.02

Table 1: Optimization results for the loss function containing the saturated liquid density and the enthalpy of vaporization at $T = 75$ K, using VLE fit functions [34] instead of molecular simulations. The stopping criterion was $F(\boldsymbol{x}) \leq 10^{-5}$. The asterisk indicates that the stopping criterion was not fulfilled and that the Armijo step length search did not converge within 100 steps. In the case of PSB, the convergency was too slow. Therefore, the descent direction was not normalized which led to better values in much less time. The Newton Raphson results are exactly concordant with the Steepest Descent results, as the Newton direction was never accepted. The Trust Region constraint had to be weakened. Therefore, $\eta_1 = 0.2$ was chosen.

- Norm of the Gradient ($||\nabla F(\boldsymbol{x}^{\mathrm{opt}})||$): The norm of the gradient for the final iteration. This value should be close to zero as well, but it cannot be expected that it gets as small as $F(\boldsymbol{x}^{\mathrm{opt}})$ for the reasons mentioned in section 3.

In this example, the Trust Region method gives the lowest percental errors on ρ_l and Δh_v within only six iterations. However, the number of molecular simulations which would have had to be performed was quite high in general. The best method in this regard was the Polak Ribière method, which would have required 41 simulations only. In comparison, the simplex algorithm [30] needed approximately 70–100 molecular simulations for comparable optimization tasks.

Please note that the Quasi Newton methods are not suitable for this kind of optimization tasks. In many cases, they did not reach an iteration where the stopping criterion was fulfilled. In the original Quasi Newton algorithms, there is no step length control. Hence, $\forall_k\ t_k = 1$, and if H_k is spd, then H_{k+1} is spd as well. Otherwise, it is not guaranteed that H_k is spd, and therefore d^k is not always a descent direction. This can lead to iterations located far away from the minimum, and the algorithm does not converge. Furthermore, the Armijo step length control does not converge either, as it is not possible to find a lower loss function value anymore. However, if the calculated physical properties are affected by noise, the Quasi Newton methods converged in some cases. This is because in the case of noise, it is more likely for the Armijo step length control to find a lower loss function value. Afterwards, at the subsequent iteration, the Hessian approximation H_k may be spd again and a much lower loss function value can be found. Therefore, the Quasi Newton methods performed better, when the properties were noisy but they are not robust with respect to noise.

8. GROW—A New Basis for the Development of High–Quality Force Fields

The usage of the fit functions makes the loss function smooth. The question how noise influences the behavior of the optimization algorithms is treated and discussed thoroughly in [35]. It has been studied by introducing artificial statistical noise on the simulated physical properties and retesting the convergency behavior. By way of doing so, the pursuit is followed to clearly separate the various problem classes of the optimization process as far as possible: the influence of noise can be studied in a controlled fashion without dealing with molecular simulation, the final and most important step of our challenge.
As specifically tested for a simple nitrogen model, the Steepest Descent method, the Conjugate Gradient methods, and the Trust Region method with an exact solution of the subproblem turned out to be most suitable in the case of noise. Please note that the resulting quality of the force field alone will justify the success of the methods. It has only been proven that the loss function falls robustly below some specified upper bound using the methods mentioned above. The Conjugate Gradient methods and the exact Trust Region method were then used to achieve even better results. When applying the algorithms to molecular simulations, it must always be considered how long the simulations have to be in order to reduce the statistical uncertainties and how the noise can be handled. Moreover, the resulting force field must be assessed with respect to the uncertainties on the simulated and the target data. For example, this can be achieved by choosing $w_i = 1/s_i^2$ in equation (1), where s_i are the relative standard deviations of the target data or the square root of the sum of the relative variances of target and simulation data, according to the maximum likelihood principle.
However, for the assessment of the numerical optimization algorithms performed in [35] using the

fit functions instead of molecular simulation, it was sufficient to prove that good or satisfactory force fields are achieved robustly and that even better force fields can also be obtained by some of the numerical methods with more iterations. This proof has been performed in a practical way, i.e. by some replications of each optimization workflow. Thereby, the assessment was executed in a general way, assuming uniformly distributed instead of normally distributed errors on the simulation data. Furthermore, it was set $\forall_i \; w_i = 1$, because within the assessment of the algorithms, all properties should be treated equally.
The present work is only a presentation of a high–performance methodology and its technical implementation. It does not consist of the realization of high–quality force fields but it is the basis for a lot of subsequent work to achieve this goal by the usage of molecular simulations.

9. Acknowledgements

We are grateful to Astrid Maaß and Florian Müller-Plathe for valuable discussions and appreciate their intellectual support to our work. Furthermore, we would like to thank Axel Arnold, Christoph M. Friedrich, and Roman Klinger for their technical aid. Marco Hülsmann acknowledges the financial support for his scholarship of the University of Cologne (Germany).

References

[1] F. Jensen, Introduction to Computational Chemistry (John Wiley & Sons, New York, 1999)

[2] S.J. Singer, G.L. Nicolson, Science 175 (1972) 720.

[3] M.P. Allen, D.J. Tildesley, Computer Simulations of Liquids (Oxford Science, Oxford, 1987)

[4] Y. Zhou, G. Stell, J. Chem. Phys. 96 (1992) 1504

[5] J.I. Siepmann, S. Karaborni, B. Smit, Nature 365 (1993) 330

[6] S. O'Connell, P.A. Thompson, Phys. Rev. E 52 (1995) 5792

[7] J. Kolafa, I. Nezbeda, M. Lisal, Mol. Phys. 99 (2001) 1751

[8] R. Valiullin, S. Naumov, P. Galvosas, J. Kärger, H.-J. Woo, F. Porcheron, P.A. Monson, Nature 443 (2006) 965

[9] I.P. Batra, B.I. Bennett, F. Herman, Phys. Rev. B 11 (1975) 4972

[10] T.P. Fehlner, J. Solid State Chem. 154 (2000) 110

[11] C.N. Della, S. Dongwei, Diffus. Defect Data Pt. B Solid State Phenom. 136 (2008) 45

[12] S.-T. Lin, M. Blanco, W.A. Goddard III, J. Chem. Phys. 119 (2003) 11792

[13] D.E. Bien, V.A. Chiriac, in: Proceedings of the 9th Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (IEEE, New Jersey, 2004) p. 748

[14] J. Vrabec, J. Gross, J. Phys. Chem. B 112 (2008) 51

[15] A. Hodgkin, A. Huxley, J. Physiol. 117 (1952) 500

[16] B.J. Barkla, O. Pantoja, Annu. Rev. Plant Physiol. Plant Mol. Biol. 47 (1996) 159

[17] M. Levitt, A. Warshe I, Nature 253 (1975) 694

[18] J. Gsponer, A. Caflisch, in: Proceedings of the National Academy of Sciences (PNAS, 2002) p. 6719

[19] C.D. Snow, E.J. Sorin, Y.M. Rhee, V.S. Pandel, Annu. Rev. Biophys. Biomol. Struct. 34 (2005) 43

[20] F. Müller-Plathe, D. Reith, Comput. Theor. Polymer Sci. 9 (1999) 203

[21] P. Bordat, D. Reith, F. Müller-Plathe, J. Chem. Phys. 115 (2001) 8978

[22] G. Guevara-Carrion, C. Nieto-Draghi, J. Vrabec, H. Hasse, J. Phys. Chem. B 112 (2008) 16664

[23] G.S. Grest, K. Kremer, Phys. Rev. A 33 (1986) 3628

[24] F. Müller-Plathe, Acta Polymer 45 (1994) 259

[25] K. Binder, Monte Carlo and Molecular Dynamics Simulations in Polymer Science (Oxford University Press, Oxford, 1995)

[26] K. Kremer, F. Müller-Plathe, Mol. Sim. 28 (2002) 729

[27] M. Praprotnik, C. Junghans, L. Delle Site, K. Kremer, Comput. Phys. Commun. 179 (2008) 51.

[28] J. Wang, P.A. Kollman, J. Comp. Chem. 22 (2001) 1219

[29] J. Wang, R.M. Wolf, J.W. Caldwell, P.A. Kollman, D.A. Case, J. Comp. Chem. 25 (2004) 1157

[30] R. Faller, H. Schmitz, O. Biermann, F. Müller-Plathe, J. Comp. Chem. 20 (1999) 1009

[31] D. Reith, H. Meyer, F. Müller-Plathe, Comput. Phys. Commun. 148 (2002) 299

[32] P. Ungerer, C. Beauvais, J. Delhommelle, A. Boutin, B. Rousseau, A.H. Fuchs, J. Phys. Chem. 112 (2000) 5499

[33] E. Bourasseau, M. Haboudou, A. Boutin, A.H. Fuchs, P. Ungerer, J. Chem. Phys. 118 (2003) 3020

[34] J. Stoll, J. Vrabec, H. Hasse, J. Fischer, Fluid Phase Eq. 179 (2001) 339

[35] M. Hülsmann, J. Vrabec, A. Maaß, D. Reith, Submitted to: Comput. Phys. Commun. (2009)

[36] S. Roweis, Levenberg-Marquardt Optimization (University of Toronto, 1996) `http://www.cs.toronto.edu/~roweis/notes.html`

[37] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical Recipes in C (Cambridge University Press, Cambridge, 1992)

[38] J. Nocedal, S.J. Wright, Numerical Optimization (Springer-Verlag, New York, 1999)

[39] J. Vrabec, J. Stoll, H. Hasse, J. Phys. Chem. B 105 (2001) 12126