Karlsruhe Institute of Technology

On Multiscale Algorithms for Selected Applications in Molecular Mechanics

Zur Erlangung des akademischen Grades eines

DOKTORS DER NATURWISSENSCHAFTEN

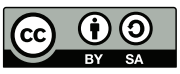
von der Fakultät für Mathematik des
Karlsruher Instituts für Technologie (KIT)
genehmigte

DISSERTATION

von
Lukas Fath
aus Weinheim

Tag der mündlichen Prüfung: 8. März 2017

1. Referentin:	Prof. Dr. Marlis Hochbruck
2. Referent:	PD Dr. Volker Grimm
3. Referent:	Prof. Dr. Wolfgang Wenzel



Acknowledgements

I would like to take this opportunity to express my gratitude to all those people who contributed, directly or indirectly, to this work.

First of all, I would like to thank my supervisor, Prof. Dr. Marlis Hochbruck, for giving me the opportunity to work on this research project. Her excellent supervision and constant support made this dissertation possible. I am also very grateful to PD Dr. Volker Grimm for being my second supervisor. He always had an open door for me, and I really enjoyed the countless hours of inspiring discussions. Prof. Dr. Wolfgang Wenzel deserves my honest thanks for serving as a referee and for guiding me to the exciting world of molecular mechanics.

I would also like to thank the DAAD for providing me with the opportunity to conduct research at the University of Toronto.

In my opinion, the pleasant and supportive working environment provided by my fellow colleagues is unmatched, and kept me motivated throughout the last years. A special thanks goes to the crew of TaLeLu. To David, I am grateful for giving me numerous valuable and constructive suggestions.

A big hug goes to Brianna, for proof-reading, English expertise, emotional support, and endless love.

Finally, I extend my sincere gratitude to my family for their invaluable and kind support.

Contents

1 Introduction

The rapid development of increasingly sophisticated computers has made it possible to simulate more complex physical processes. Compared to experiments, these simulations have the benefits of being cheaper and allowing us to easily change simulation parameters. They also let us study properties or phenomena not directly accessible in an experiment. In *molecular mechanics*, the study of molecular systems based on principles from classical mechanics, simulations are not derived from a continuum model but from a molecular or atomic one. The description of the model is fundamentally based on Newton's second law. Each atom is modeled as a point particle with assigned charge and mass, and behaves according to forces acting on it. As such, this model is a classical particle model and neglects quantum mechanical effects.

This thesis is divided into two parts and examines two prominent methods in molecular mechanics. The first part considers the problem of numerical time integration in a technique called molecular dynamics. The second part investigates and compares different numerical methods for normal mode analysis.

Molecular dynamics is centered around computing trajectories. A trajectory is the path of a molecular model following time. Relevant information is then extracted from that trajectory. In principle, the trajectory can be easily computed by solving the differential equation obtained from Newton's second law. Early papers conducting simulations of hard spheres appeared in the late '50s. In subsequent years, more sophisticated techniques were developed, which mainly applied to liquids, solids, and gases. In the '70s, the first simulations of macromolecules, such as BPTI, were carried out. The availability and rise of computers kept pushing the limits in both simulation size and simulation time. Nowadays, it is used in chemistry, engineering and medicine as a general purpose tool and large scale computations with hundreds of thousands CPU cores involving billions of atoms are successfully conducted.

Molecular dynamics simulations have a wide range of applications. In materials science, they are used to investigate the properties of materials at externally imposed physical conditions. This includes the detection of material failure such as crack propagation. Simulations can predict properties of new composite materials, such as composites with graphene, or the behavior of carbon nanotubes. In chemistry, molecular dynamics simulations allow us to investigate phase transitions or to find equilibrium states of structures. They are also used to refine three dimensional structures as obtained from X-ray crystallography or NMR spectroscopy. Many problems under examination are biophysical. With molecular dynamics, it is possible to observe phenomena on a macro-scale such as protein folding or protein mutation. This is highly interesting when trying to understand the mechanisms of biological processes, and can be useful in drug design.

For some general literature on molecular dynamics we refer to the monographs [1–5]. The difficulty in solving the simple differential equations for molecular dynamics is a combination of the following:

- stability over long simulation times and large simulation domains,
- preservation of geometric and physical properties [6–9],
- strongly nonlinear forces acting on vastly different time-scales [10–15], resulting in resonance issues with standard integrators [16–20].

The molecular models are large in dimension and the required computing time can be enormous. New methods improving existing ones by a few percent are worth considering.

In Chapter 2, we give a short introduction on the basics of molecular dynamics. In Chapter 3, we discuss standard integrators and show their limitations. Chapter 4 serves as an illustration and two small typical simulations are shown. In Chapter 5 and Chapter 6, we introduce a new family of filters for the mollified impulse method, called corotational filters. These filters allow us to avoid certain instabilities found in the impulse method and can thus improve the efficiency and multiscale character of the employed algorithm. We provide the general construction, illustration, and performance on small problems in Chapter 5. We consider two realistic examples in Chapter 6. The results of Chapter 5 and Chapter 6 have already been published in a paper by the author, together with M. Hochbruck and C. V. Singh [21].

In the second part of the thesis – covered in Chapter 7 and Chapter 8 – normal mode analysis is highlighted. *Normal mode analysis* tries to circumvent certain time limitations encountered in molecular dynamics by deducing large scale movements from a local harmonic approximation. The harmonic approximation provides vibrational frequencies with corresponding eigenvectors. The eigenvectors are also called modes, and describe in detail the motion associated with each frequency. Basically, a full decomposition into separate frequencies and modes can be computed. Again, the problem at hand is quite challenging due to the large dimension. Most methods found in the literature deal with this problem by a model approach: the complexity of the original problem is heavily reduced by replacing the model with a coarse grained description.

Normal mode analysis is a well established area [22,23]. In the ‘70s it was used for small molecules [24, 25], and its use was expanded in the ’80s with the development of protein normal mode analysis [26–28]. Nowadays, the techniques are found in many algorithms, see for example the reviews [29–32]. The most typical application is the investigation of functional motions in large biomolecules. However, it is also possible to combine the ideas from normal mode analysis with many other algorithms such as coarse graining [33] and molecular dynamics [34, 35]. It is an active area of research [36–43]. For a review of the different techniques and services available on the internet we refer to [44] and references therein.

In the second part of this thesis, the problem is investigated from a numerical perspective. Recent, complex algorithms designed for tackling such problems are tested and compared. We also develop a new multilevel algorithm which exploits the underlying structure of the physical problem. The method can be used as a standalone solver or as a preconditioner for multiple numerical methods. In Chapter 7, we present the theoretical requirements and available algorithms. Their performance is critically reviewed by applying them to six test problems of varying sizes in Chapter 8.

Goals of this thesis

The goals of the first part of this thesis are

- a numerical investigation of common stability issues in molecular dynamics,
- and to propose a new filter which removes linear resonances. This filter is employed in the mollified impulse method.

The goals of the second part of this thesis include

- comparing different methods available for normal mode analysis,
- and introducing a linear solver based on a multilevel hierarchy.

2 Basics of Molecular Dynamics Simulations

2.1 Hamiltonian Formulation

In molecular dynamics (MD), we try to predict the time evolution of a set of N atoms. We denote their position vector at time t by $q(t) \in \mathbb{R}^{3N}$ and their momentum vector by $p(t) \in \mathbb{R}^{3N}$. Their motion is described by a Hamiltonian system.

Definition 1. *(Hamiltonian system) For a given Hamiltonian function $H(q,p)$ with position coordinates q and momenta p the associated Hamiltonian system is*

$$\begin{aligned} \dot{q}(t) &= \nabla_p H(q,p), \\ \dot{p}(t) &= -\nabla_q H(q,p), \end{aligned} \tag{2.1}$$

for some initial values $q(t_0) = q_0,\ p(t_0) = p_0$.

If we abbreviate

$$X = \begin{bmatrix} q \\ p \end{bmatrix} \in \mathbb{R}^{6N} \text{ and } J = \begin{bmatrix} 0_{3N} & I_{3N} \\ -I_{3N} & 0_{3N} \end{bmatrix} \in \mathbb{R}^{6N \times 6N}$$

this shortens to

$$\dot{X} = J \nabla H(X), \qquad X(0) = X_0. \tag{2.2}$$

While the Hamiltonian framework allows for very general Hamiltonian functions, in molecular dynamics, we focus on a very specific, separable Hamiltonian of the form

$$H(q,p) = \frac{1}{2} p^T M^{-1} p + U(q). \tag{2.3}$$

The first part is the quadratic kinetic energy $T(p) = \frac{1}{2} p^T M^{-1} p$ with a (time invariant) diagonal mass matrix M, and the second part is the potential energy $U(q)$. The Hamiltonian itself thus represents the energy of the state (q,p). Throughout this thesis, we assume that the potential function $U(q)$ is at least twice continuously differentiable.

2.1.1 Invariants

Hamiltonian systems are known to have many interesting properties, a few of which we will discuss here. An essential property of a Hamiltonian system is that the Hamiltonian function itself is preserved along exact solutions of the Hamiltonian system, i.e. it is a *first integral* of the motion.

Lemma 2. *([6, Ex. IV.1.2]) The Hamiltonian H is an invariant along solutions of the corresponding Hamiltonian system.*

In molecular dynamics, most potentials are of the form

$$U(q) = \sum_{i,j=1}^{N} V_{ij}(||q_i - q_j||), \tag{2.4}$$

where $q_k \in \mathbb{R}^3$ denotes the position of atom k. V_{ij} is a potential describing the interaction between atoms i and j and its argument, $||q_i - q_j||$, is the distance between both atoms. Then it holds:

Lemma 3. *([6, Ex. IV.1.3]) For an N-body Hamiltonian system with potential of form (2.4) both linear ($\sum_i p_i$) and angular ($\sum_i q_i \times p_i$) momentum are conserved.*

The conservation of the Hamiltonian function and the conservation of both linear and angular momentum are intuitive, physical properties; the energy of a closed system does not change. If there are no external forces, the linear momentum, i.e. the center of mass velocity, and the angular momentum do not change.

2.1.2 Reversibility

Definition 4. *([6, Def. V.1.1]) Let ρ be an invertible linear transformation in the phase space of $\dot{y} = f(y)$. This differential equation and the vector field $f(y)$ are called ρ-reversible if*

$$\rho f(y) = -f(\rho y) \qquad \text{for all } y. \tag{2.5}$$

Hamiltonian systems with Hamiltonian (2.3) are reversible under the transformation $\rho(q, p) = (q, -p)$. If a vector field is reversible with respect to this specific transformation it is called *time-reversible*. So inverting the velocity vector does not change the trajectory, it just inverts the direction of motion. A deeper consequence of time-reversibility is that in MD we do not have a direction of time in the thermodynamics sense. This observation is known as *Loschmidt's paradox*. For more details see for example [4].

2.1.3 Symplecticity

There is one geometric property which characterizes Hamiltonian systems. But first, let us introduce the concept of a *flow*.

Definition 5. *The mapping*

$$\varphi_t(q_0, p_0) = X(t), \tag{2.6}$$

with

$$\dot{X}(t) = J\nabla H(X), \quad X(0) = \begin{bmatrix} q_0 \\ p_0 \end{bmatrix} \tag{2.7}$$

is called the flow φ of the Hamiltonian system H.

So $\varphi_t(q_0, p_0)$ denotes the solution at time t with initial values q_0, p_0. For Hamiltonian systems, the flow is symplectic.

Definition 6. *([6, Def. VI.2.2]) A mapping g is called symplectic, if its Jacobian g' is everywhere symplectic i.e. $g'(q,p)^T J g'(q,p) = J$.*

Theorem 7. *([6, Thm. VI.2.4], Poincaré 1899) Let H(q,p) be a twice continuously differentiable function on $\mathcal{S} \subset \mathbb{R}^{6N}$. Then, for each fixed t, the flow φ_t is a symplectic transformation wherever it is defined.*

It turns out that for Hamiltonian systems the symplectic flow is a characterizing feature. It also holds that if a flow is symplectic, then the corresponding ODE is at least locally Hamiltonian ([6, Thm. VI.2.6]).

2.1.4 Poisson Bracket

Frequently, one is interested in computing the time evolution of a certain property $a(X)$, defined on the phase space of all possible states $X = (q,p)$. A quick calculation reveals

$$\dot{a}(X) = \sum_{i=1}^{3N} \left(\frac{\partial a}{\partial p_i}\dot{p} + \frac{\partial a}{\partial q_i}\dot{q} \right) = \sum_{i=1}^{3N} \left(\frac{\partial a}{\partial q_i}\frac{\partial H}{\partial p_i} - \frac{\partial a}{\partial p_i}\frac{\partial H}{\partial q_i} \right) = \{a, H\}. \tag{2.8}$$

The last equality makes use of the Poisson bracket.

Definition 8. *(Poisson bracket) For two smooth scalar functions f, g defined on phase variables $(q,p) \in \mathbb{R}^{6N}$ the Poisson bracket is defined by*

$$\{f,g\} = \sum_{i=1}^{3N} \left(\frac{\partial f}{\partial q_i}\frac{\partial g}{\partial p_i} - \frac{\partial g}{\partial q_i}\frac{\partial f}{\partial p_i} \right) = \nabla f^T J \nabla g.$$

The Poisson bracket is bilinear, skew symmetric, and fulfills the Jacobi identity

$$\{\{g_1, g_2\}, g_3\} + \{\{g_2, g_3\}, g_1\} + \{\{g_3, g_1\}, g_2\} = 0$$

and the Leibniz' rule [6]

$$\{g_1 g_2, g_3\} = g_1\{g_2, g_3\} + g_2\{g_1, g_3\}.$$

In equation (2.8), the Poisson bracket of a Hamiltonian is related to the time derivative of $a(X)$. This is remarkable since the right hand side has no direct explicit time dependence anymore. Since $\{a, H\} = \dot{a}(X)$ all properties on the phase space with $\{a, H\} = 0$ are conserved. It is now easy to prove the conservation of total momentum $P = \sum_i p_i$. Note that with a potential of form (2.4) we have

$$\sum_i \frac{\partial H}{\partial q_i} = \sum_i \frac{\partial U}{\partial q_i} = 0$$

and thus

$$\dot{P} = \{P, H\} = \sum_i \{p_i, H\} = -\sum_i \frac{\partial H}{\partial q_i} = 0.$$

Finally, the Poisson bracket also allows us to express the Hamiltonian system (2.2) more concisely as

$$\dot{X} = \{X, H\}, \qquad X(0) = X_0. \tag{2.9}$$

2.1.5 Concepts of Numerical Integrators

In general, the Hamiltonian system (2.1) cannot be solved analytically. Instead, a selected algorithm (referred to as a *numerical integrator*) tries to provide an approximation to the exact solution with a given accuracy. Given an initial state $q_0 = q(t_0), p_0 = p(t_0)$, such algorithms deduce information about the exact solution by evaluating the right hand side of (2.1). Combing the collected information, they make an educated guess about the future $q_1 \approx q(t_0 + h), p_1 \approx p(t_0 + h)$ where h is a small positive number, the *time step*.

In the design of a suitable numerical integrator it is valuable to exploit any known structural properties of the exact solution. Indeed, it is possible to transfer the concepts of symplecticity and symmetry to numerical integrators.

Definition 9. *([6, Def. VI.3.1]) A numerical one-step method is called symplectic if the one-step map $y_1 = \Phi_h(y_0)$ is symplectic whenever the method is applied to a smooth Hamiltonian system.*

In the Hamiltonian context, we have a symplectic flow, so we are advised to use a symplectic integrator. Classical theorems from backward error analysis (e.g. [6, Thm. IX.8.1]) show that the numerical approximations from a symplectic integrator can be understood as the *exact* solution of a nearby Hamiltonian system over an exponentially long time (though care has to be taken not to overestimate this property, since the proofs of the results from backward error analysis are based on Taylor expansion. Here, however, $h\omega = const$ where ω is the highest frequency of the ODE [9]). This topic was actively researched some decades ago, and simple conditions for symplectic integrators have been found.

Sometimes, symplectic integrators cannot be formed, and good results are also achieved with symmetric methods.

Definition 10. *([6, Def. V.1.3]) A numerical one-step method Φ_h is called symmetric or time-reversible, if it satisfies*

$$\Phi_h \circ \Phi_{-h} = id \qquad \textit{or equivalently} \qquad \Phi_h = \Phi_{-h}^{-1}.$$

A symmetric numerical method applied to a reversible differential equation has a reversible flow ([6, Ch. V]). Since the Hamiltonian system with (2.3) has a reversible flow, conserving symmetry is preferable. Symplecticity and symmetry are two major geometric properties, but there are many other ideas in the community, such as projecting solutions back onto some energy manifold or employing constraints to preserve certain quantities.

2.2 Potentials

In MD, the potential function $U(q)$ tries to mimic the physical interaction between atoms. Designing such potential functions is its own branch of science; models are proposed and calibrated such that they reproduce certain observations obtained from experiments.

In biomolecular simulations there is a fast intra-molecular part U_{intra} which models bonded interactions within molecules. Short-range pair interactions U_{pair} describe local forces between molecules such as Van der Waals repulsion and attraction as well as short-range electrostatic forces. Long-range contributions U_{long} usually consist of the remaining long-range electrostatic contributions.

Hence, it is common to decompose the potential $U(q)$ into a sum of those components:

$$U(q) = U_{\text{intra}}(q) + U_{\text{pair}}(q) + U_{\text{long}}(q). \tag{2.10}$$

Let us give a few examples. For bonded interactions, a standard potential is the harmonic bond potential. If $r = ||q_i - q_j||$ denotes the distance between two bonded atoms, it is given by

$$U(r) = \frac{K}{2}(r - r_0)^2, \tag{2.11}$$

where K is a force constant and r_0 is the equilibrium length of that bond. A commonly used model for Van der Waals repulsion and attraction is the Lennard Jones potential

$$U(r) = 4\,\epsilon\left(\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right) \tag{2.12}$$

with suitably chosen constants ϵ and σ. It has a strong repulsive part if r is small, but a weak attraction part at intermediate distances. For r large, it quickly converges to zero. For electrostatic interactions the Coulomb potential

$$U(r) = \frac{1}{4\pi\epsilon_0}\frac{e_i e_j}{r} \tag{2.13}$$

with a constant ϵ_0, charges e_i, e_j and distance r between atoms i, j is used. Since it only decays as $\frac{1}{r}$, long-range interactions cannot be neglected and are vital for structural stability. In a regular setup, each atom is bonded through strong harmonic potentials to a few of its neighbors, usually within the same molecule. It then interacts through pair potentials with *all* of its neighbors within a certain cutoff radius, and through long-range potentials with all other atoms in the simulation.

Due to this potential structure, the resulting ODE is quite stiff. While the slow long-range interactions are smooth, the frequent collisions between atoms and high force constants in bonded interactions are responsible for the stiffness. Especially the Van der Waals repulsion with its $\frac{1}{r^{12}}$ singularity requires very small step sizes for most standard explicit integrators.

The three potentials presented here are simple examples. Many more potentials with

different potential functions exist. Besides pairwise potentials, such as the three potentials presented here, there are many potentials which include three, four or more atoms in their construction. Examples include potentials depending on specific angles between three connected atoms, or torsion potentials. Luckily, a number of so-called force fields exist which provide a consistent parametrization and a selection of potentials for a molecular structure. Well-known force fields for biomolecular simulations include AMBER [45], CHARMM [46–49], GROMOS [50] and OPLS-AA [51]. For more details and an overview we refer to Chapter 9 in [1] and references therein.

2.3 Time Scale and Number of Atoms

Atoms are very small. In tiny amounts of material there are many atoms. Avogadro's constant gives a hint of how many there are: 16 grams of oxygen would already consist of roughly $6.23 \cdot 10^{23}$ atoms, which is an unimaginably large number. More important, it is impossible to store position and momenta information for that magnitude of atoms, at least on any current (and probably near future) computer. As a consequence, we have to restrict the simulations to tiny fractions of these sizes. In a similar way, while interesting physical phenomena happen on time scales of seconds, microseconds or nanoseconds, the fastest oscillations in molecules happen on a scale of femtoseconds ($= 10^{-15}s$).

It is essential to find a compromise between the number of atoms - or molecules - and the time frame we want to conduct our investigations in. Usually, both the size and time frame is much smaller and shorter than what we can see and experience with our own eyes on a reasonable scale. Long, full atomistic simulations of large molecules consisting of millions of atoms have been conducted on the nano- or microsecond scale. For those computations serious computing power (tens of thousands of CPU cores for weeks or months) is essential. Some research groups even design and build their own specialized hardware for MD [52–54].

2.4 Simulation Cell

So far we have not made any assumptions on the simulation cell, that is the (physical) space in which the atoms with position $q_i,\ i = 1, ..., N$ are allowed to exist. We just assumed an infinite domain e.g. $q_i \in \mathbb{R}^3,\ i = 1, ..., N$. This turns out to be impractical for most problems. Much more interesting and realistic problems happen in a finite domain $\mathcal{D} \subset \mathbb{R}^3$ with corresponding boundary conditions. Note that in MD the term *boundary condition* differs from its usage in the context of PDEs. In MD, boundary conditions describe what happens when an atom reaches the boundary surface of $\mathcal{D}$. The restriction of the simulation cell to some domain $\mathcal{D} \subset \mathbb{R}^3$ is necessary due to practical issues and limited resources. In MD, interactions are happening on such a small scale that an atomistic simulation domain of a 1cm cube would be almost impossible. Furthermore, most phenomena do not happen in a vacuum. A proper background is very important, and long-range forces have

a significant impact on structural properties.

Multiple types of boundaries are used. Fixed boundaries simply keep the outer layers of atoms, which are far away from the area of interest, fixed. Wall boundaries let atoms reaching the boundary bounce back. Vacuum surroundings pretend nothing other than the few atoms of interest exists. Shrink-wrapped boundaries are automatically adjusted to only fit our atoms.

The most common type, though, are periodic boundaries. A finite cube, called a *unit cell*, is periodically repeated in every spatial direction. Thus, an infinite domain is generated (see Figure 2.1). This approach is very useful in applications where structure depends on long–range forces such as electrostatic interactions. It gives a reasonable background if the chosen cube is large enough. Also, this is very efficient in terms of memory requirements; only the information for the atoms inside the unit cell needs to be stored. Instead of using a cube as unit cell, other geometric structures are possible. For the simulation of a large biomolecule surrounded by a solute such as water, the cube can be replaced by a rhombohedron. Such a simulation cell minimizes the simulation of the unwanted solute while still keeping enough spatial distance between the periodically repeated copies of the large biomolecule.

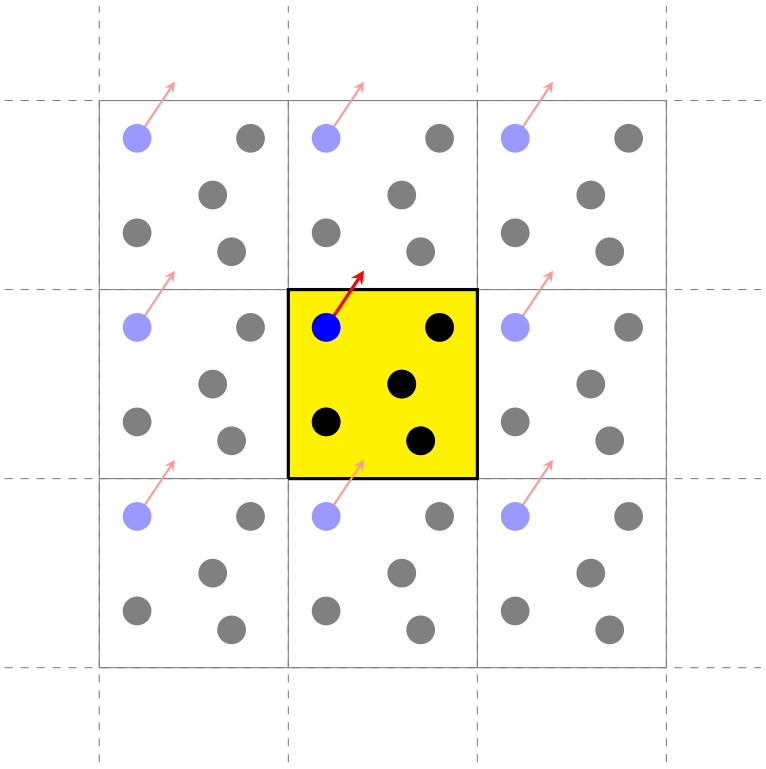


Figure 2.1: *Simple example of periodic boundaries in 2D. The unit cell containing 5 atoms is highlighted in yellow, repeated copies in x and y direction are drawn in opaque colors. If the blue atom moves in the indicated direction, so do all blue copies.*

All types of boundaries have their limitations. Some have no physical justification, others suffer from various spurious effects. For example, if too small of a periodic box is

chosen, periodic artifacts will pollute the results.

2.5 Ensembles and Targeted Simulation

In almost all simulations, one tries to investigate materials under certain (imposed) physical conditions such as a specific temperature or pressure. Therefore, we need to be able to control these basic conditions. It is common to abbreviate the following thermodynamic properties: N (number of atoms), V (volume of domain), E (energy), T (temperature) and P (pressure). Of those thermodynamic properties only three are independent, and can be chosen to be fixed. This is then called a simulation in a certain ensemble, where the capital letters indicate, which properties are kept fixed. A few examples are:

- Microcanonical ensemble (NVE): N, V, and E are kept fixed. There is no energy exchange with the surrounding, no volume change, and no atoms can enter or leave the domain.
- Canonical ensemble (NVT): temperature instead of energy is conserved. It corresponds to the simulation of a domain surrounded by a heat bath keeping the domain at a constant temperature. Energy can flow into or out of the domain.
- Isothermal-isobaric ensemble (NPT): temperature and pressure are fixed. In addition to the energy, the volume changes in order to maintain ambient pressure.

There are more ensembles, such as ensembles in which atoms can enter or leave the domain. For further details we refer to the literature, e.g. [55] and references therein. The Hamiltonian equations introduced in the previous chapter follow the microcanonical ensemble. There are (nontrivial) modifications to those such that they obey another ensemble.

A simple way of achieving a constant temperature is simply rescaling the velocity vector p every couple of steps. A popular version of rescaling is the variant of Berendsen [56]. However, there is some interest in maintaining the *ergodic property* which is the assumption that the limit of trajectory averages is equal to the phase space average, i.e. a system is said to be *ergodic*, if after a sufficiently long time, it visits all possible state space points [4]. It can be shown that velocity rescaling does not quite follow the canonical ensemble. Therefore, a more elaborate approach is to extend the Hamiltonian equations and add auxiliary variables in the form of so-called thermostats, which control the wanted thermodynamic properties. The most common thermostat is the Nosé-Hoover thermostat [57, 58], and a simple version for NVT with one auxiliary variable ξ reads:

$$\begin{aligned} \dot{q} &= M^{-1}p, \\ \dot{p} &= -\nabla U(q) - \xi p, \\ \mu\dot{\xi} &= p^T M^{-1} p - N_d \beta^{-1}. \end{aligned}$$

Here, ξ provides a negative feedback loop which controls the kinetic energy. When the average kinetic energy is higher than the target temperature β^{-1}, ξ increases and will damp p until it becomes negative again. N_d are the number of degrees of freedom and μ is a coupling parameter.

Similar extensions can be designed for the other ensembles. Integrators discussed in the following chapter naturally compute within a NVE ensemble. Again, we refer to the literature for information on the necessary modifications when using them in a different ensemble.

2.6 Setup and Running a Simulation

The following summarizes the steps necessary to conduct a molecular dynamics simulation. The entire workflow is depicted in Figure 2.2.

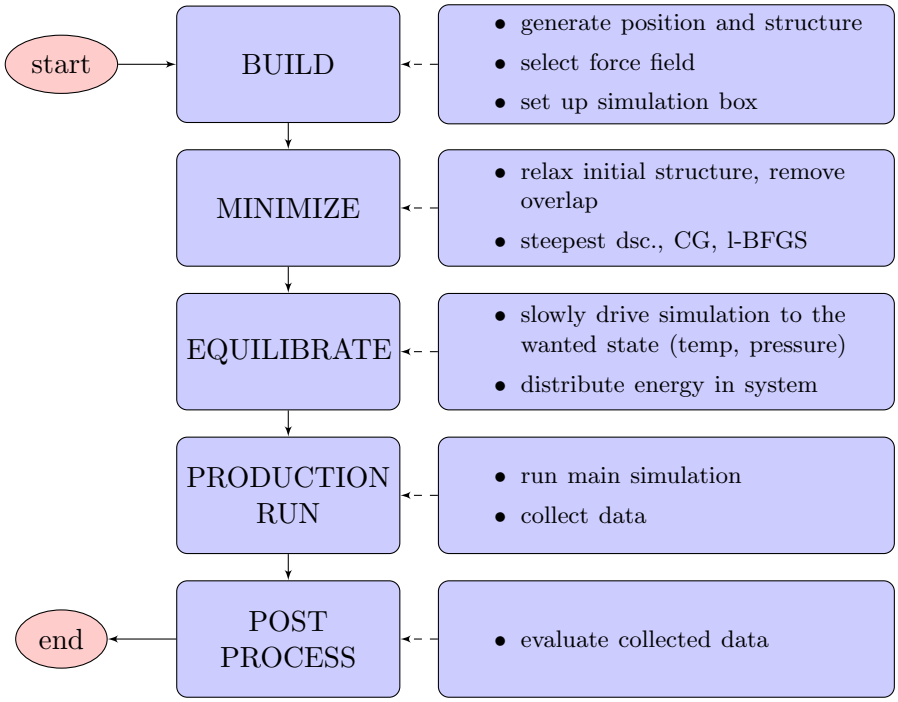


Figure 2.2: *Workflow for molecular dynamics*

First, position data for the molecular structure under investigation is selected. The structure is then equipped with a force field; that is, a full set of potential functions with parameters describing the potential $U(q)$ in (2.3). A suitable simulation box is identified. These choices go hand in hand. The simulation cell has to fit the molecular structure, and the boundary condition needs to be compatible to the force field and so on.

To avoid immediate blow ups in the simulation, it is common to first minimize the structure under the chosen potential function. A potential overlap of atoms is resolved. For this step, classical minimizers, such as steepest descent or a CG method, can be employed. More advanced algorithms include the limited-memory BFGS, a quasi-Newton

method.

The goal of equilibration is to distribute energy in the system. The simulation is started at a random velocity vector of low temperature, and then the simulation is slowly driven towards the target temperature and pressure. Depending on the sensitivity of the structure under investigation, this step can take a long time and requires a careful procedure.

Finally, the production run can be started. This is the part where the original investigation happens and data from the simulation is collected. Afterwards, the obtained raw data is evaluated, condensed and visualized.

2.7 Software for Molecular Dynamics

There is plenty of publicly available software for molecular dynamics. Many research groups develop their own code. Software packages are usually targeted and optimized towards a specific type of simulation. Here, we will only name software that we used in the simulations conducted for this thesis. For a more thorough overview we refer to the monographs [1–5] which include lists of available software.

For preparing and setting up a molecular structure we used moltemplate [59]. This tool allows us to setup geometric patterns and build large structures by combining smaller ones. We mostly used the set of force field parameters provided by CHARMM. For some tasks we required SwissParam [60] which can generate parameters for small molecules which are not included in the force fields. High-end software for running large simulations on clusters includes LAMMPS [61] and GROMACS [62]. The simulations for MD were computed with LAMMPS since its framework allows for simple modifications. Nevertheless, it is a very efficient software for running large scale simulations. GROMACS has very similar capabilities, however, since it also contains built-in tools for normal mode analysis, we mainly used it for the computations in Chapter 8. Postprocessing and visualization of the results can be done with VMD [63] and ovito [64]. Whereas ovito is a software mainly focused on visualizing, VMD contains many plug-ins which can provide in-depth analysis of simulation results.

3 Standard Integrators

As we have seen in the previous chapter, solutions of (2.1) with Hamiltonian (2.3) have a highly geometric nature with multiple invariants. Therefore, it is natural to devise numerical integrators mimicking (at least part of) those structures. Furthermore, we are interested in long-term simulations, often with millions of time steps. Short term accuracy of a method is of minor interest, since round-off errors will accumulate and model errors in the molecular models make high accuracy not feasible. Furthermore, methods such as the explicit Euler scheme, or even common high order Runge-Kutta methods will not work since they will provide approximations which drift off the energy landscape quickly. Hence, our main focus lies on topics such as stability, robustness, and efficiency [7].

3.1 The Verlet Method

In the MD community the most successful integrators are, without doubt, the Verlet method and its multiple time step version - the impulse method [65, 66]. Both are low order explicit integrators. The Verlet method is based on approximating the second order derivative in

$$M\ddot{q} = -\nabla U(q) \tag{3.1}$$

by a second-order central difference

$$\ddot{q}_n \approx \frac{q_{n+1} - 2q_n + q_{n-1}}{h^2}.$$

If we insert this second-order central difference in Newton's equation (3.1) we immediately get the following two step method (Stoermer-Verlet)

$$q_{n+1} - 2q_n + q_{n-1} = -h^2 M^{-1} \nabla U(q_n).$$

For MD though, the one step formulation in Algorithm 3.1 is more successful since it directly provides momenta. Here, there are different variations (leapfrog method, velocity-Verlet method, position-Verlet method). They are all fairly similar but differ in details. E.g. the leapfrog only provides velocities at half-steps, the velocity-Verlet method does a half-step with the velocities at the beginning and end, etc.

Algorithm 3.1: One step of the velocity-Verlet method

1 $p_{n+1/2} = p_n - \frac{h}{2}\nabla U(q_n)$

2 $q_{n+1} = q_n + hM^{-1}p_{n+1/2}$

3 $p_{n+1} = p_{n+1/2} - \frac{h}{2}\nabla U(q_{n+1})$

All of them have in common that they are explicit methods and only use a single force evaluation per time step. Note that for example in Algorithm 3.1, the force evaluated at the new position q_{n+1} is reused in the beginning of the next step. From now on whenever we refer to the Verlet method we mean the velocity-Verlet in Algorithm 3.1.

Theorem 11. *([6, Thm. VI.3.4.])*
The velocity-Verlet method given in Algorithm 3.1 is a symplectic and second order method when applied to a separable Hamiltonian.

The success of the Verlet method in molecular dynamics can be credited to its simple design while preserving both symmetry and symplecticity. The implementation is straight forward, since it is an explicit method with a single force evaluation. Furthermore, the Verlet method exhibits a beneficial long-time behavior [67].

The Verlet method can also be interpreted as a splitting method. This is based on the observation that when splitting the ODE in the following way

$$\begin{bmatrix} \dot{q} \\ \dot{p} \end{bmatrix} = \begin{bmatrix} M^{-1}p \\ -\nabla U(q) \end{bmatrix} = \begin{bmatrix} M^{-1}p \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ -\nabla U(q) \end{bmatrix}$$

the exact flows φ_h^T and φ_h^U corresponding to the kinetic part $T = \frac{1}{2}p^T M^{-1}p$ and the potential part $U(q)$ respectively can be easily computed. In fact, they are just simple translations and

$$\varphi_h^T(q_0, p_0) = \begin{bmatrix} q_0 + hM^{-1}p_0 \\ p_0 \end{bmatrix}, \qquad \varphi_h^U(q_0, p_0) = \begin{bmatrix} q_0 \\ p_0 - h\nabla U(q_0) \end{bmatrix}. \tag{3.2}$$

Therefore, the Verlet method with step size h can be expressed as

$$\Phi_h = \varphi_{h/2}^U \circ \varphi_h^T \circ \varphi_{h/2}^U.$$

Despite its simplicity and many good geometric properties, the Verlet method has a major drawback for simulations in MD: linear stability analysis reveals a step size restriction $h\omega < 2$ depending on the fastest frequency ω (see Section 3.5.1). For accurate results, step sizes for the Verlet method are commonly chosen around $h\omega \approx \frac{1}{2}$ in MD.

3.2 The Impulse Method

The major drawback of the Verlet method is the fact that the step size is restricted by the fastest frequency. Therefore, slow forces have to be computed more often than necessary. In most biomolecular simulations a step size below 0.5fs or 1.0fs is required to ensure long-time energy conservation.

Bonded forces oscillate faster than pair forces, and pair forces change more quickly than long-range electrostatic contributions, a fact to which the Verlet method is 'blind'. The impulse method is based on the Verlet method but it further splits the potential

allowing forces to be evaluated at different time scales. Assume we can split $U(q) = U_{\text{slow}}(q) + U_{\text{fast}}(q)$. The basic idea is to propagate U_{slow} only at the beginning and end of a time step (*impulse-like*), and then solve

$$\ddot{q} = -M^{-1}\nabla U_{\text{fast}}(q)$$

e.g. using a different integrator with a much smaller step size in the middle of Algorithm 3.2. Sometimes, line 1 and line 3 are referred to as the *kick* step, and line 2 as the *oscillate* step. Usually, the Verlet method is chosen for the inner integration loop, but any other method could be used.

Algorithm 3.2: One step of the impulse method with two stages, $U = U_{\text{fast}} + U_{\text{slow}}$

1 $p_n^+ = p_n - \frac{h}{2}\nabla U_{\text{slow}}(q_n)$

2 Obtain (q_{n+1}, p_{n+1}^-) by numerically solving the reduced Hamiltonian system with $\tilde{H} = T + U_{\text{fast}}$ for h time and starting at (q_n, p_n^+)

3 $p_{n+1} = p_{n+1}^- - \frac{h}{2}\nabla U_{\text{slow}}(q_{n+1})$

Clearly, this method is symmetric if the inner integration is done by a symmetric method. Furthermore, if we use a symplectic method in the inner loop, we obtain a symplectic method since we integrate $\dot{q} = 0,\ \dot{p} = -\nabla U_{\text{slow}}(q)$ exactly, (see [6, Ch. VIII.4.1]). So, if we choose the Verlet method in the inner stage, the impulse method becomes both symplectic and symmetric. From now on, whenever we refer to the impulse method, we refer to this combination with the Verlet method.

Similar to the Verlet method, we can express the impulse method as a splitting method with

$$\Phi_h = \varphi_{h/2}^{U_{\text{slow}}} \circ (\varphi_{h/2K}^{U_{\text{fast}}} \circ \varphi_{h/K}^{T} \circ \varphi_{h/2K}^{U_{\text{fast}}})^K \circ \varphi_{h/2}^{U_{\text{slow}}}.$$

Here, the inner integration is done by K steps of the Verlet method with step size h/K. This splitting approach can be easily extended to an arbitrary number of stages. Hence, potentials can be grouped into different time-scales and evaluated accordingly. The only limitation between the stages is that integer factors between the different step sizes are required. Especially if slow forces are computationally expensive, the impulse method can provide a significant speed-up. In MD simulations a common force splitting uses different stages for bond, pair and long-range forces. In the literature, a lot of research has been conducted to maximize the benefits of this method [10–14]. The studies give advice on how to choose the 'best' splitting and provide tips for implementation such as smooth cut-offs and other tuning parameters.

A good speed-up over the Verlet method can be obtained in biomolecular simulations, but the method is not free from problems. Resonances and instabilities already appear in simple linear models [15, 16]. For a further discussion see Section 3.5.2.

3.3 Simulations with Constraints: SHAKE

The family of SHAKE algorithms [68, 69] is a collection of standard methods for constraining intramolecular degrees of freedom within simulations. By removing the degrees of freedom which are associated with the fastest vibrations, larger step sizes are accessible. This rather drastic step is justified by the fact that the fastest vibrations are almost decoupled and do not significantly contribute to behavior on longer time scales. However, one has to carefully examine whether the impact of SHAKE does not actually interfere with the observations targeted. Freezing out the fastest degrees of freedom (such as with SHAKE/RATTLE) can lead to the wrong qualitative limit [70, 71]. Depending on the application, neglecting fast frequencies is questionable.

In most implementations the constraints are enforced using Lagrange multipliers. The resulting nonlinear equations are solved using a Newton or Gauss-Seidel method. Assume we want K constraints collected in a vector function $g(q) \in \mathbb{R}^K$ with $g(q) = 0$ to be satisfied. An additional constraint force such that the new positions fulfill the constraints is applied to the corresponding atoms. In this process we need to determine the Lagrange multipliers $\lambda \in \mathbb{R}^K$ in an iterative process. The constrained Hamiltonian system reads

$$\begin{aligned} \dot{q} &= M^{-1}p, \\ \dot{p} &= -\nabla U(q) - \nabla g(q)\lambda, \\ g(q) &= 0. \end{aligned}$$

The bottleneck of this procedure is obviously the solution of the nonlinear equations. As long as only small connected clusters (up to around 4 or 5 atoms) are constrained, there are very efficient ways to do so. However, constraining e.g. all backbone atoms of a long linear chain quickly becomes prohibitively expensive. There are multiple variations of this basic algorithm depending on its specific purpose and how the nonlinear equations are solved. These algorithms are known under names such as SETTLE, SHAKE, M-SHAKE, P-SHAKE, QSHAKE, SHAPE, LINCS, RATTLE, WIGGLE and many more ([5, Ch. 4.3.6], see also references in [72, Sec. II]).

3.4 The Implicit Midpoint Method

Implicit integrators are known to have better stability properties than explicit integrators. This makes them ideal candidates for the integration of stiff ODEs.

The prototype of an implicit, symplectic and symmetric method is the implicit midpoint method, and for $\dot{y} = f(y)$ it reads

$$y_{n+1} = y_n + hf(\frac{y_{n+1} + y_n}{2}).$$

Applied to the Hamiltonian system (2.1) with Hamiltonian (2.3), we have

$$q_{n+1} = q_n + hM^{-1}\left(\frac{p_{n+1} + p_n}{2}\right),$$
$$p_{n+1} = p_n - h\nabla U(\frac{q_{n+1} + q_n}{2}).$$

Inserting the second into the first equation we have to solve

$$q_{n+1} = q_n + hM^{-1}p_n - \frac{h^2}{2}M^{-1}\nabla U(\frac{q_{n+1} + q_n}{2}).$$

Algorithm 3.3: One step of the implicit midpoint method

1 $p_n^+ = p_n - \frac{h}{2}\nabla U(\frac{q_{n+1}+q_n}{2})$

2 $q_{n+1} = q_n + hM^{-1}p_n^+$

3 $p_{n+1} = p_n^+ - \frac{h}{2}\nabla U(\frac{q_{n+1}+q_n}{2})$

Written in a trusty leapfrog manner (see Algorithm 3.3), we have to solve the first two equations using a Newton method. One possibility is to solve for q_{n+1}, but it is also possible to solve for p_n^+ with a similar procedure. Each step of the Newton method to solve for $x = q_{n+1}$ applied to

$$0 \stackrel{!}{=} g(x) = x - q_n - hM^{-1}p_n + \frac{h^2}{2}M^{-1}\nabla U(\frac{q_n + x}{2})$$

then consists in solving the linear system

$$g'(x_n)(x_{n+1} - x_n) = -g(x_n),$$

with

$$g'(x) = I + \frac{h^2}{4}M^{-1}U_{qq}(\frac{q_n + x}{2}).$$

To avoid evaluating the Hessian in every iteration step, a simplified Newton method with fixed g' is usually used. However, solving this nonlinear system of equations in every time step is, in general, too expensive in MD. Even storing a full Hessian of the potential (as required for Newton-type methods) would be almost impossible for huge molecular systems. Yet for certain specific problems or subproblems with a very sparse or decoupled Hessian they might be worth the effort. If we apply the midpoint method only to small molecules such as water in the inner loop of an impulse method (only bond forces) and use the simplified Newton method, we could explicitly compute $(f')^{-1}(x)$. Each step then is reduced to computing the fast force and a small matrix vector multiplication. If pair forces are included, though, this quickly becomes a computationally challenging task. If only bonded forces are used, why go to all this trouble? The Verlet method with bonded forces is *extremely* cheap. Even if ten times more steps are used, it will still be competitive.

3.5 Limitations of Standard Integrators

Both the Verlet method and the impulse method suffer from severe step size restrictions. The implicit midpoint method is (linearly) unconditionally stable, yet the resulting computations are, in general, much too expensive.

3.5.1 Stability Restriction of the Verlet Method

The Verlet method has a severe step size restriction depending on the fastest frequency.

Lemma 12. *Applied to $\ddot{q} = -\omega^2 q$ the Verlet method has a step size restriction $h\omega < 2$.*

Proof. Applying Algorithm 3.1 to $\ddot{q} = -\omega^2 q$ we get following iteration:

$$\begin{pmatrix} q_{n+1} \\ p_{n+1} \end{pmatrix} = \begin{pmatrix} 1 - \frac{h^2\omega^2}{2} & h \\ \frac{h^3\omega^4}{4} - h\omega^2 & 1 - \frac{h^2\omega^2}{2} \end{pmatrix} \begin{pmatrix} q_n \\ p_n \end{pmatrix}. \tag{3.3}$$

This propagation matrix has eigenvalues $\omega_{1,2} = 1 - \frac{h^2\omega^2}{2} \pm \sqrt{h^2\omega^2(\frac{h^2\omega^2}{4} - 1)}$.

If $\frac{h^2\omega^2}{4} - 1 > 0$ the eigenvalues are real, and one has absolute value greater than 1. If $\frac{h^2\omega^2}{4} - 1 = 0 \Leftrightarrow h\omega = 2$ there is a double eigenvalue at -1. If $\frac{h^2\omega^2}{4} - 1 < 0$ we have one complex conjugate pair of eigenvalues with exact absolute value 1, and thus, we obtain the stability restriction $h\omega < 2$. □

This result is quite disappointing since evaluating long-range and pair forces is the main computational task, in most MD simulations. The fast oscillations resulting from stiff bonded interactions are very cheap to compute but force the Verlet method to use an overall small step size.

3.5.2 Resonances in the Impulse Method

"Resonance is a pronounced integrator-induced corruption of a system's dynamics." *Conclusions in [17]*

It is well known, that the impulse method suffers from strong linear resonances. Unfortunately, a very complex resonance behavior can also be observed on nonlinear examples. Let us have a look at an example with a slow potential frequently appearing in molecular dynamics: in 1D, a single atom with position q and momentum p is bonded to the origin with a fast harmonic potential and interacts with a fixed neighbor at $q = -2$ through a Coulomb potential. This can be modeled by the Hamiltonian

$$H(q,p) = \frac{1}{2}p^2 + \frac{\omega^2}{2}q^2 + \frac{1}{2+q}. \tag{3.4}$$

As initial conditions we set $(q_0, p_0) = (1, 0)$. Through the harmonic potential, we will get oscillations in q in the interval $(-1, 1)$ and forces F_h with $|F_h| \leq \omega^2$. The nonlinearity has

forces F_n with $F_n \in (\frac{1}{9}, 1)$ for $q \in (-1, 1)$. Note, that F_n is significantly less stiff than the harmonic part.

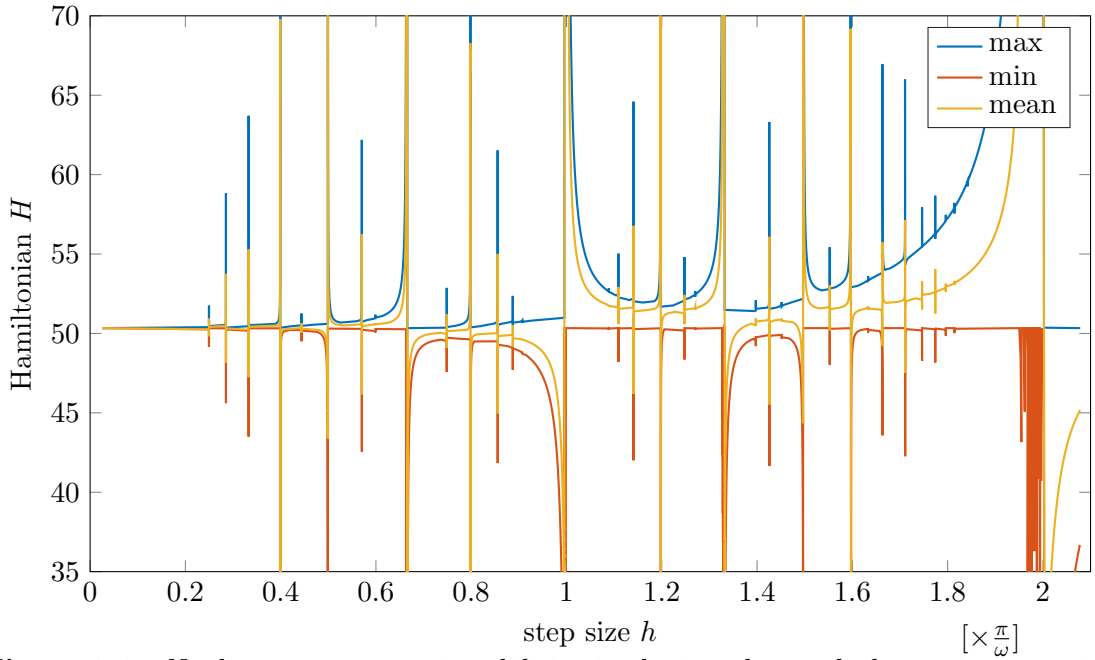


Figure 3.1: *Nonlinear resonance instabilities in the impulse method: maximum, arithmetic mean and minimum of Hamiltonian 3.4 with* $\omega = 10$ *over 10.000 integration steps and step size* h

For a fixed ω we now numerically integrate this problem for a fixed number of steps (10.000) with different step sizes. We use a 2-stage impulse method with exact inner integration of the stiff harmonic part. In Figure 3.1 we plot oscillations in the Hamiltonian function for $\omega = 10$. We plot arithmetic mean, minimum and maximum for each step size. In the ideal case, these would be close to each other, and certainly would not exhibit drastic changes when the step size h is slightly increased or decreased. Unfortunately, Figure 3.1 clearly reveals a very complex instability behavior. Besides the expected (linear) instabilities at $h = \frac{\pi}{\omega}$ (2:1), $h = \frac{2\pi}{3\omega}$ (3:1) and at $h = \frac{\pi}{2\omega}$ (4:1), there are a number of further instabilities of different magnitudes.

Resonances are a major drawback of the multiscale nature of the impulse method. This simple example with exact inner integration already shows severe instabilities and a local increase of oscillations in the Hamiltonian at step sizes as low as $h\omega \approx 2\pi/5$.

In practice, the fastest oscillations have a period of around 10fs, so the first strong linear resonance occurs at a step size of 5fs. This limit is called the *5fs-barrier* [19]. The resonances can be so strong that within a few time steps a completely non-physical behavior occurs. It usually manifests in an extreme build-up of energy in a specific mode and causes the numerical simulation to abort.

The problem of the impulse method with resonance artifacts is a well known topic in the MD community. It has been observed in many numerical experiments and theoretical explanations are available [16–20]. At the core of the problem lies the fact that the *kick*

step of the impulse method evaluates the slow potential at a single, possibly oscillatory, position. If the time step in between the kicks is large enough, this position might be a bad representation of what has happened in between the kicks. Imagine a harmonic oscillator perturbed by a slow nonlinear contribution such as in the example (3.4). When the slow force is always evaluated at half the period of the oscillation, it can happen that the *kick* step always evaluates at extreme positions of the underlying oscillation. Naturally, this can lead to a quick drift in energy.

3.5.3 Limitations of the Implicit Midpoint Method

The limitations of the implicit midpoint method are mainly practical in nature: solving the nonlinear equations quickly becomes a very expensive and challenging task. For large step sizes, the solution as obtained from a Newton-type method strongly depends on the initial guess. In addition, in large molecular problems, simply storing or even evaluating a full Hessian is a challenge on its own.

Even when ignoring all practical concerns, the implicit midpoint method does not perform well on nonlinear problems in MD. Simulations with simple nonlinear models [73] show that the implicit midpoint method also suffers from resonance issues revealing erratic energy fluctuations at certain step sizes. While for small step sizes it provides better approximations than the Verlet method with the same step size (although at a much, much higher cost), for larger time steps the quality quickly degrades, and frequencies are severely distorted [7, 74, 75].

There are approaches which try to combine implicit and explicit algorithms such as the IMEX method [76]. The IMEX method is basically an impulse method with a slow kick, but the inner fast part is computed with the implicit midpoint method. Thus, the nonlinear equations can be solved at fairly low cost, and the damping helps reducing resonances. However, with the slow outer kick, the method remains prone to resonance problems.

4 Two Small Illustrating Problems

In this section we illustrate practical difficulties on two small, but realistic examples. Resonance instabilities are not just a theoretical issue, and they are a real problem when performing molecular dynamics simulations. Also, we give computational timings for the Verlet and impulse method. There, we show that the computation of the long-range electrostatic forces poses a major bottleneck on large processor counts.

We abbreviate the two examples with (A) and (B). First, as problem (A), we choose the most common natural form of ice, that is ice Ih. Ice is very sensitive to changes and its structure is easily destroyed by a misperforming integrator. This makes it an challenging test problem. For the second problem (B), we choose a small peptide solvated in water. It has a variety of different fast vibrations from bonds, angles and torsions. For the time integration, we use the Verlet method and the impulse method. For the impulse method we split the potential as in (2.10) in three parts and use three stages in the impulse splitting.

Since descriptions for potentials, models and integrators can quickly become mired in technicalities, and we here tried to make this section as readable as possible, the reader is referred to the appendix for detailed explanations. In Appendix A the potentials can be found in more detail. Appendix B covers parameters of the different models, and in Appendix C we have a list of the time integration methods.

4.1 (A) - Ice Ih

Ice is a crystalline structure formed by water molecules. Around 20 different structures are currently known. Ice Ih is the natural phase of ice on earth. It is obtained by freezing water at atmospheric pressure, so almost all ice on earth has this structure. The name *Ih* stands for the first (I) discovered hexagonal (h) structure of ice. From a top view it looks astonishingly similar to honey comb in a bee hive.

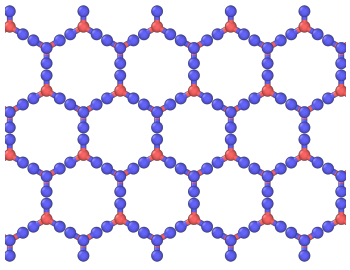


Figure 4.1: *(A) - ice Ih, crop of simulation cell, top view*

Each water molecule has hydrogen bonds to the four nearest neighbor molecules which are situated at the corners of a regular tetrahedron. Due to this structure, ice Ih is less

dense than water and has a higher volume than water. For more details, we refer to the literature, e.g. [77].

We set up a cubic simulation box with periodic boundaries containing 2880 molecules, arranged in a perfect hexagonal lattice (see [78] for lattice parameters). The simulation size is sufficiently large to get reliable results and obtain a computationally challenging problem (parallel efficiency is a major concern).

There are many atomistic models for water, each with different properties and each optimized for some special purpose. For a good overview see [79]. We conduct our investigation by choosing two different *flexible* models. As opposed to *rigid* models, *flexible* models provide intra-molecular degrees of freedom such as bond stretching and angle bending. First we choose the SPC/F model [80], which is based on the rigid three site SPC model [81] and uses harmonic potentials in bonds and angles to model intra-molecular forces. The second model we investigate is TIP4P/2005f [82], a flexible model based on the TIPNP topology. It additionally uses a massless fourth site, M, with a negative charge. Bond interactions are modeled using a Morse potential. The angle potential is again harmonic. Furthermore, both models have Lennard-Jones and long-range electrostatic potentials for intermolecular interactions.

The initial position is chosen as a perfect lattice structure, the initial velocities are randomly assigned such that the overall temperature is 150°K. We then equilibrate for 50ps (= 50,000fs) in NPT targeting 150°K and 1bar pressure. Afterwards we run for 50ps in NVE. We are not interested in any kind of order of the integrator (that is the behavior for small step sizes and short times). Instead we are interested in the behavior over the full 50ps after equilibration.

4.2 (B) - A Small Peptide

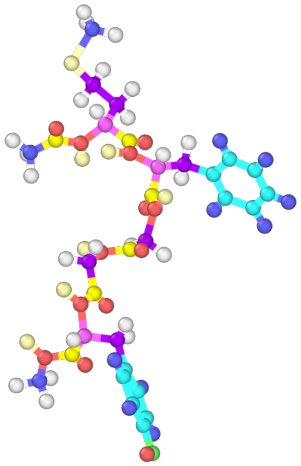


Figure 4.2: *(B) - a small peptide, without its surrounding water, drawn in a ball-stick representation*

We conduct simulations of a small 5-mer peptide solvated in water. The *mer* refers to the number of monomers or residues in the peptide, that is the number of 'building blocks' which are strung together in a chain to form the molecule. Initial positions and structure can be found in the LAMMPS example section. The peptide has 84 atoms and is solvated in 640 water molecules (total of 2004 atoms) in a periodic simulation cell. Potentials are modeled according to the CHARMM force field [46]. In CHARMM, the structure within the peptide is described by a multitude of different bond, angle and torsion potentials. This makes for a nice toy problem, and it is comparably cheap to compute due to the low dimension of this example.

Similar to example (A), we first equilibrate for 50ps in NVT at 250°K. Afterwards, we compute short 50ps trajectories in NVE for our analysis.

4.3 Frequency Plots

First of all, we identify the different time scales present in this simulation. We compute infrared spectra with VMD's IR Spectral Density Calculator Plugin by computing Fourier transformations of the velocity auto-correlation function. In MD, it is common to report such frequencies as the inverse of the wave length, called wave number, with the unit [1/cm]. This can be easily converted to a frequency f or oscillation period T through the relation

$$k = \frac{1}{\lambda} = \frac{f}{c} = \frac{1}{cT}$$

where k is the wave number, and $c \approx 3 \cdot 10^8 \frac{m}{s}$ the speed of light. Thus, the larger the wave number, the higher the frequency. A wave number of 3,330cm^{-1} corresponds to a vibration with a period of around 10fs. The results for examples (A) and (B) are plotted in Figure 4.3.

For simulation (A), it shows clear and separate spikes, since at around 150°K ice Ih stays in its fixed hexagonal structure, and there is almost no diffusion happening. The fastest vibrations (right-most peaks) correspond to the quick oscillations in the bonds. The second major peak at around 1,600cm^{-1} is due to angle motion. The remaining part then stems from all the other forces and slower motions. A closer look even allows us to distinguish both possible vibration modes in an triatomic molecule due to the bond forces: the peak is separated in a left and a right part. The left part originates in symmetric stretching of the two bonds in each molecules, whereas the right part indicates the slightly faster asymmetric stretching.

Compared to ice, example (B) provides a plot with much more interesting features. The peaks from the water solvent are clearly visible. Additionally, the other bonds and angles from the peptide are apparent. There is no gap between the fastest angle and slower forces anymore. This is due to torsion forces as well as all nonlinear interactions between faster bonds.

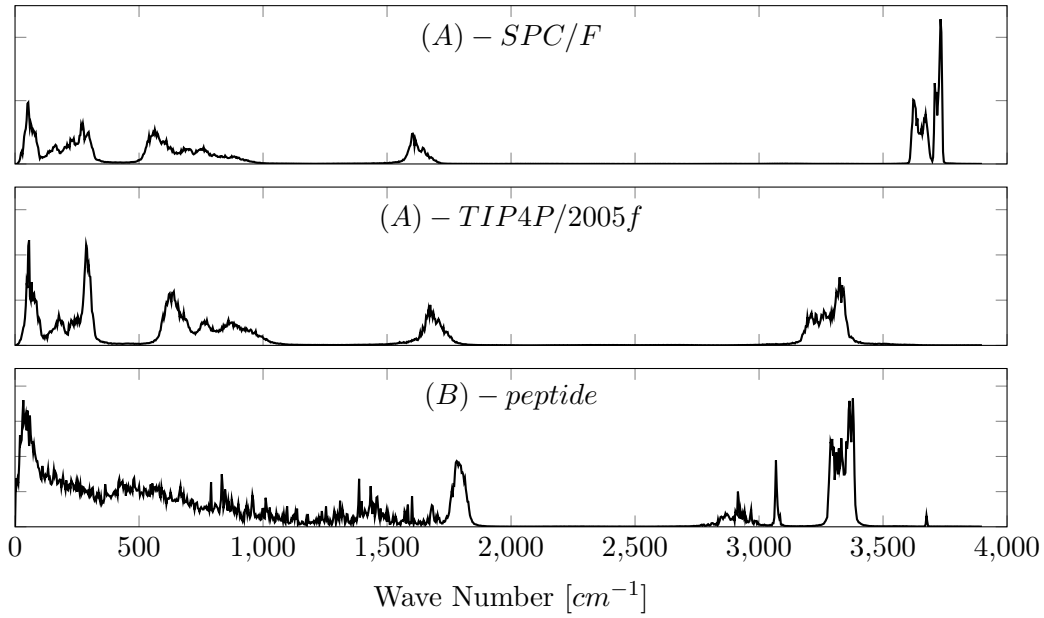


Figure 4.3: *IR spectra for (A) and (B), arbitrary intensity units vs wave number* $[cm^{-1}]$

4.4 Energy Drifts

We run both examples with the Verlet and impulse method at different step sizes. Furthermore, we show results for the implicit midpoint method, and the effect of combining both the Verlet and impulse method with the SHAKE algorithm. A simple way to estimate the quality of the results is measuring the drift in the Hamiltonian in the last 50ps of the simulation. To cancel out oscillations, we choose to average over the first and last 500fs in this interval.

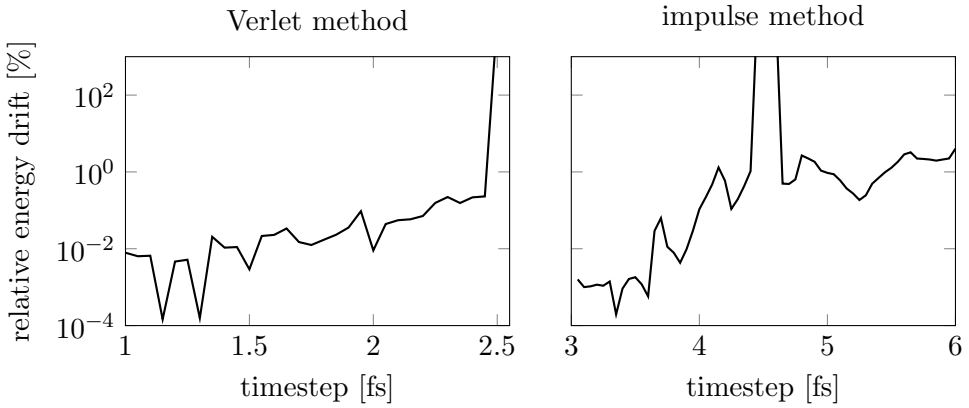


Figure 4.4: *(A) - SPC/F: energy drift [%] vs outer step size [fs]*

The Verlet method shows remarkable stability: at around 2.0fs (SPC/F) or 1.5fs (TIP4P/2005f) it is barely drifting. And together with SHAKE, the method stays stable for much larger step sizes. Yet it should be noted that despite showing almost no drift, at those step sizes, the quality of the computed trajectory quickly degrades. In fact, one is

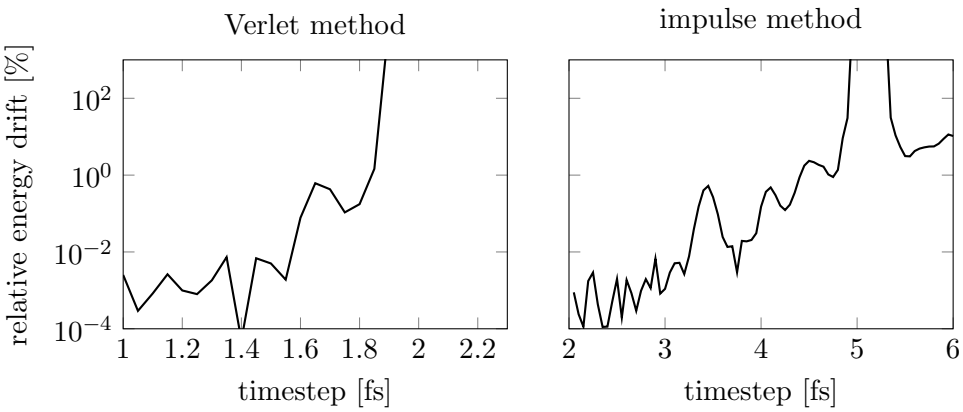


Figure 4.5: *(A) - TIP4P/2005f: energy drift [%] vs outer step size [fs]*

advised to use quite conservative choices (e.g below 0.5fs) for the Verlet method.

Figure 4.4 indicates that for ice simulations with the SPC/F model, the impulse method can easily handle outer step sizes as large as 4fs without significant drift. Resonance issues cause, as expected, a dramatic breakdown at step sizes near 4.5fs. It is a 2:1 resonance failure attributed to the fast bond oscillations. Due to a clear separation of frequency bands, this is a sharp spike. With larger time steps we can get stable results again. Note that at time steps between 4.4 and 4.6fs, the simulation actually 'explodes' and therefore aborts half way through. Furthermore, we can also detect nonlinear resonances other than the critical 2:1 breakdown. Following [18] we also have to expect 3:1 and 4:1 instabilities. Those are instabilities which happen not at half the period, but at a third or a fourth of the associated mode. The peaks at around 3.7fs and 4.2fs might stem from such higher order resonances, resulting from the angle motion. The situation is similar for the TIP4P/2005f, and the 2:1 instabilities occur near 5fs (see Figure 4.5). The peak at 3.4fs is clearly a 3:1 instability, the smaller peaks at 4.1fs and 4.6fs could originate in higher order resonances, resulting from the angle motion.

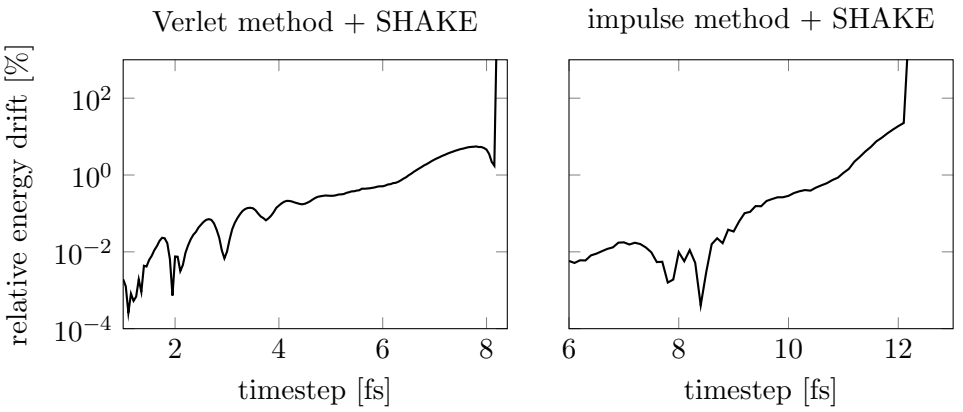


Figure 4.6: *(A) - TIP4P/2005f: energy drift [%] vs outer step size [fs]*

Applying SHAKE (see Figure 4.6) significantly increases the accessible time step for

both methods, yet at the cost of freezing out all bond and angle interactions. In an IR plot of a SHAKE-trajectory, the corresponding peaks would not appear.

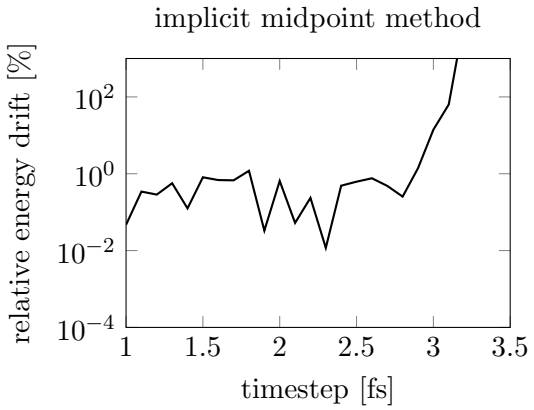


Figure 4.7: *(A) - SPC/F: energy drift [%] vs outer step size [fs]*

The implicit midpoint method - Figure 4.7 - performs surprisingly poorly. It is only stable for time steps up to 3fs, which is far too small to justify the huge costs of solving the implicit equations. For details concerning the implementation see Section C.3. Basically, it uses a simplified Newton iteration with a reduced Hessian. In Figure 4.7 we only give results for the simpler SPC/F model since we did not extend the code to include the special treatment of the electrostatics in the TIP4P/2005f model.

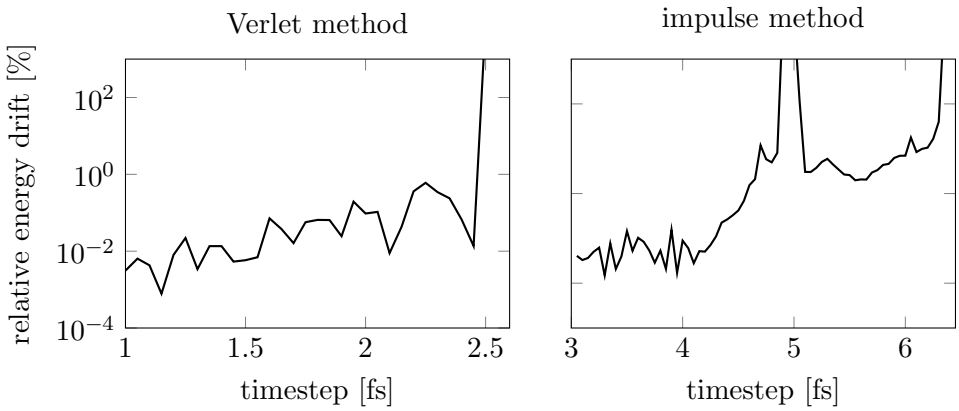


Figure 4.8: *(B) - peptide: energy drift [%] vs outer step size [fs]*

Due to the multitude of different frequencies, it is much harder to assign resonances to single eigenmodes of the peptide. In Figure 4.8 it is clear, though, that the first dramatic breakdown happens around 5fs for the impulse method. Other resonances are more smeared and difficult to identify, but clearly visible. For the Verlet method, the peaks in between 1.5 and 2.5fs have become much more pronounced compared to results for the SPC/F model in Figure 4.4.

Both methods with SHAKE exhibit more instabilities than in the previous example. Note that at a step size just above 4fs, the Verlet method with SHAKE just seems to hold

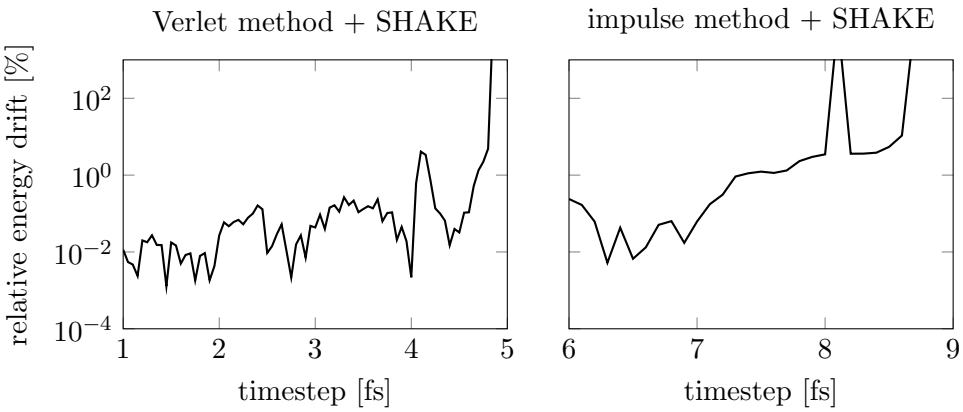


Figure 4.9: *(B) - peptide: energy drift [%] vs outer step size [fs]*

on (Figure 4.9). In a longer simulation, it certainly would result in a large energy drift.

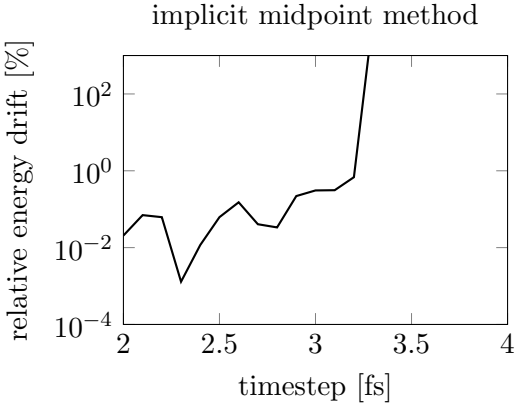


Figure 4.10: *(B) - peptide: energy drift [%] vs outer step size [fs]*

The implicit midpoint method (Figure 4.10) allows for larger step sizes than the Verlet method, yet the difference is not big enough to compensate for the higher computational cost.

4.5 Computational Cost

Timing results are obtained from short 50ps simulations after equilibration. The results are averaged over ten runs each. They were obtained on the bwUniCluster (a compute cluster at the Karlsruhe Institute of Technology) using 128 cores (8 nodes with 16 cores each). Since the results barely differ between (A) and (B), we only present timing information for example (A) here. Also, we focus on the Verlet method and impulse method (IM). Applying SHAKE incurs a slight additional overhead but does not significantly change the relative cost of force evaluations. We do not give results for the implicit midpoint method since the level of implementation of that method is by far not comparable to the other methods. For the Verlet method we use a step size of 1fs, and for the impulse method,

the step size of the outer stage is chosen at 4fs. Both are common and reasonable choices, and at these step sizes the integrators are expected to perform without any issues.

Categories in Figure 4.11 and Figure 4.12 are as follows: *Bond* contains the computation of intra-molecular forces, *Pair* the time spent on evaluating all forces between molecules within a certain cutoff. *Long-Range* is the time used calculating the long-range electrostatic contributions. *Comm* represents all time due to communication between processors. Remaining time is summarized in *Other*.

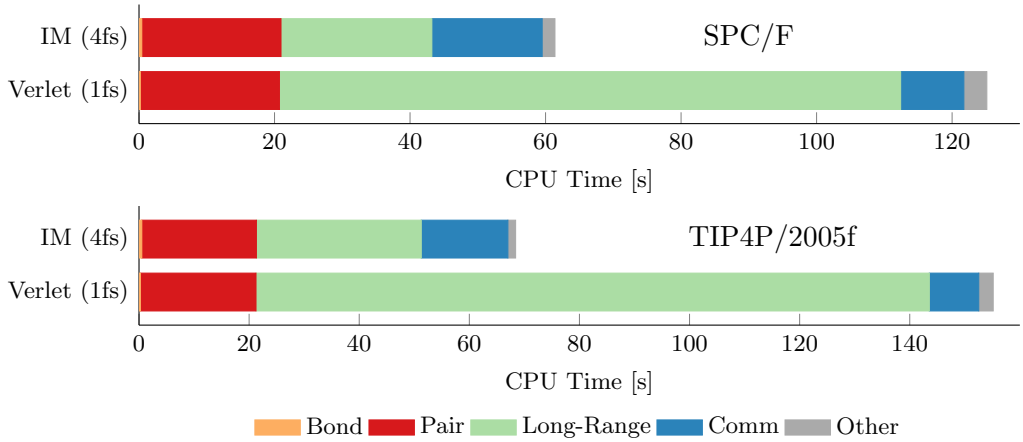


Figure 4.11: *(A) - absolute CPU time in seconds walltime for the SPC/F and TIP4P/2005f model, as obtained from short 50ps simulations using 128 cores with the Verlet method and impulse method (IM)*

First, it is essential to recognize that the explicit methods have barely any overhead. In all methods, force evaluations make up for more than 95% of computational time. Clearly, for the Verlet method the computing time is dominated by the long-range force evaluations. While pair forces still need roughly a sixth of the CPU time, the time spent on computing bond forces can be neglected. The impulse method evaluates the slow force four times less often than the Verlet method, which results in a significant speed-up. For pair forces, no difference is visible since they are still evaluated at the same step size. The TIP4P model behaves similarly, although the computation of long-range forces is a bit more complicated due to the additional M-site.

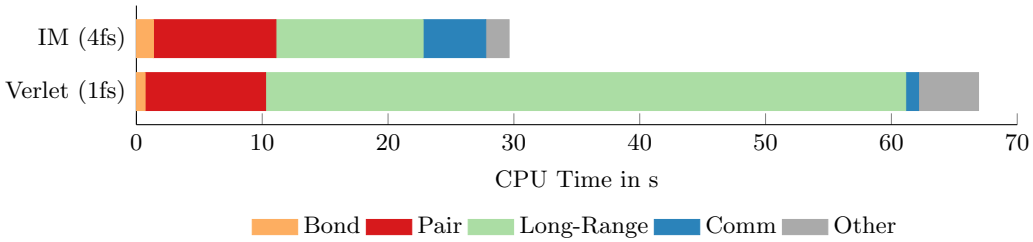


Figure 4.12: *(A) - SPC/F: absolute CPU time in seconds walltime, 6 cores + Tesla K20*

Another promising hardware configuration for MD simulations are GPUs. Some computations such as pair interactions can be calculated much more efficiently on a GPU

[83, 84] than on a CPU. In Figure 4.12, we show timing results obtained on SciNet's experimental GPU cluster (SciNet is Canada's largest supercomputer centre). 6 CPUs share one Tesla K20c GPU. We computed short 10ps trajectories, so the results cannot be compared directly to the previous plots, yet its relative proportions are clear. Unfortunately, the long-range solver is not well suited to run on a GPU, and its relative proportion in CPU time, despite only using 6 cores, is quite large.

Since problem sizes in molecular dynamics are usually very large, it is very important that algorithms scale very well with large processor counts. In MD, the most common strategy to distribute computation on a processor grid is based on a domain decomposition approach. The simulation cell is divided into smaller cubes which then are assigned, stored and handled on the individual processors. So every processor is responsible for a small part of the domain. Computations, which only require local information, can be quickly computed. Such are, for example, computing bond forces, or pair forces. If global information is needed, such as when computing long-range interactions, the processor grid needs extensive communication.

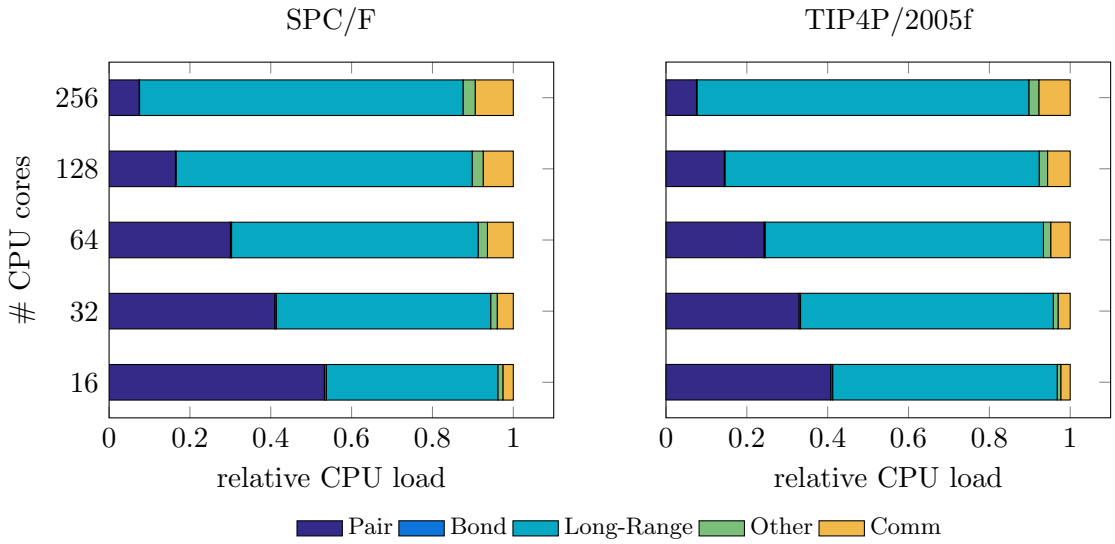


Figure 4.13: *(A) - relative CPU load with the Verlet method for 16 - 256 CPU cores*

In Figure 4.13, we plot relative CPU times for different processor counts, obtained with the Verlet method. They indicate the need for better methods: while pair forces scale very well, at higher processor counts the computation of long-range forces quickly dominates the computational effort.

A common concept to measure scalability of algorithms is calculating the *strong scaling* efficiency. If the amount of time to complete a work unit with 1 processing element is t_1, and the amount of time to complete the same unit of work with N processing elements is t_N, the strong scaling efficiency ϵ_{scale} is given as:

$$\epsilon_{\text{scale}} = \frac{t_1}{N t_N}.$$

Since, in general $t_N \geq \frac{t_1}{N}$ we have $\epsilon_{\text{scale}} \leq 1$. For values of ϵ_{scale} close to 1 an algorithm is said to be scaling well. For such an algorithm, doubling the number of processors results in roughly half the walltime until program completion.

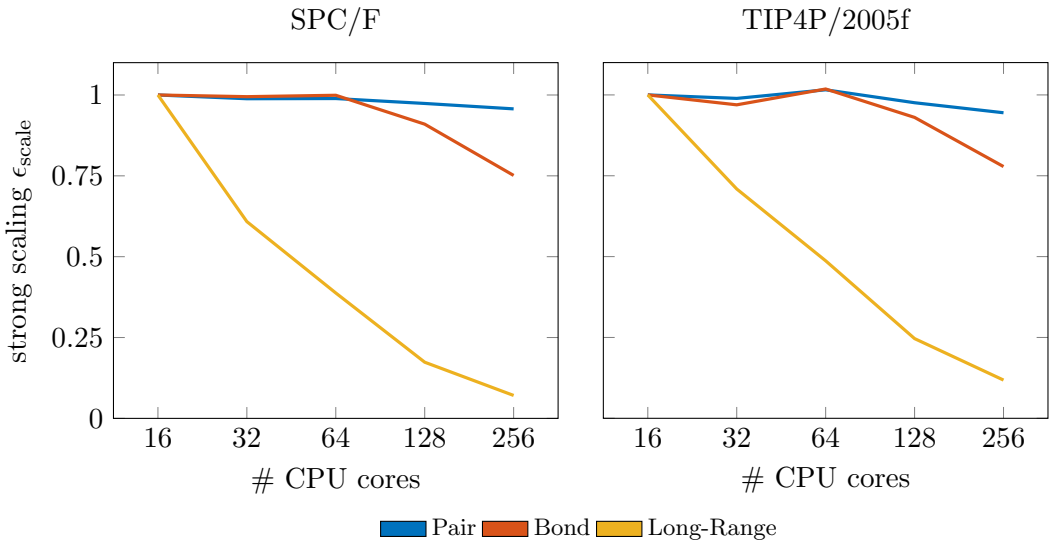


Figure 4.14: *(A) - strong scaling efficiency ϵ_{scale} vs # CPU cores*

As expected, Figure 4.14 reveals that while both bond and pair forces scale incredibly well, the long-range force interactions are not suited to run on large processor grids. Therefore, an appealing approach is to use a heterogeneous hardware structure suited for the different properties: a high-end GPU, which efficiently calculates pair forces, is combined with only a few processors, which then handle long-range computations.

4.6 A Word of Warning

We want to end this chapter with a word of warning. Judging the quality of an integrator just based on its behavior towards long term energy drift is misleading. It is only one out of many qualitative tests that have to be performed such as radial distribution functions, energy distributions, slow energy exchange etc. It can be very hard to judge this. It helps to keep our goal in mind: compute approximate trajectories where we can observe certain dynamic changes. The energy is just a warning flag. If an integrator is already highly unstable towards the Hamiltonian, it is doubtful it leads to good physical results. On the other hand, in extremely long trajectories every integrator will slightly drift, if only due to round off errors. Hence, they always require a coupling to a thermostat (even if it is very weak).

5 The Mollified Impulse Method

Resonances in the impulse method are caused by the instantaneous evaluation of the slow force at positions which do not represent the oscillatory nature of the molecule. Instead, it is advantageous to replace the potential of the slow force by a mollified version $U_{\text{slow}}(\Psi(q))$. Here, the slow force ∇U_{slow} is evaluated at a more representative filtered position $\Psi(q)$ and also post-filtered with the transposed Jacobian $\Psi_q(q)^T$:

$$\nabla U_{\text{slow}}(q) \to \Psi_q(q)^T \nabla U_{\text{slow}}(\Psi(q)). \tag{5.1}$$

The Jacobian Ψ_q^T removes contributions of the slow force in the direction of the fast vibrations. In combination with the impulse method, this is usually referred to as the mollified impulse method [85]. If the inner propagation is approximated by K steps with the Verlet method, the mollified impulse method reads

$$\phi_h^H \approx \varphi_{h/2}^{U_{\text{slow}}(\Psi(\cdot))} \circ \left(\varphi_{h/2K}^{U_{\text{fast}}} \circ \varphi_{h/K}^{T} \circ \varphi_{h/2K}^{U_{\text{fast}}}\right)^K \circ \varphi_{h/2}^{U_{\text{slow}}(\Psi(\cdot))}. \tag{5.2}$$

Note that removing the Jacobian Ψ_q in (5.1) destroys the symplecticity of the integrator. Indeed, if we omit the derivative we have to expect inherent energy drifts. For the same reason the filter must not depend on p.

Algorithm 5.1: One step of the mollified impulse method

1. $p_n^+ = p_n - \frac{h}{2}\Psi_q(q_n)^T \nabla U_{\text{slow}}(\Psi(q_n))$
2. Obtain (q_{n+1}, p_{n+1}^-) by numerically solving the reduced Hamiltonian system with $\tilde{H} = T + U_{\text{fast}}$ for h time and starting at (q_n, p_n^+)
3. $p_{n+1} = p_{n+1}^- - \frac{h}{2}\Psi_q(q_{n+1})^T \nabla U_{\text{slow}}(\Psi(q_{n+1}))$

For linear problems it can be shown that the mollified impulse method with a suitable filter does not suffer from linear resonances [85,86]. However, it does suffer from nonlinear resonances. We plot instabilities for the mollified impulse method for the Hamiltonian (3.4) in Figure 5.1. This plot shows a clear improvement over the unfiltered impulse method in Figure 3.1, but certain instabilities can be observed.

While the results with a mollified potential are very promising, one has to keep in mind that they come at the cost of tempering with the energy exchange between fast and slow modes. Mollifying a potential avoids resonances by removing force contributions in the direction of the fastest motions. We will have a closer look at this issue for our specific problem.

In the literature, two types of suitable filters have been identified. First, we have a short look at averaging techniques. Second, we describe equilibrium filters.

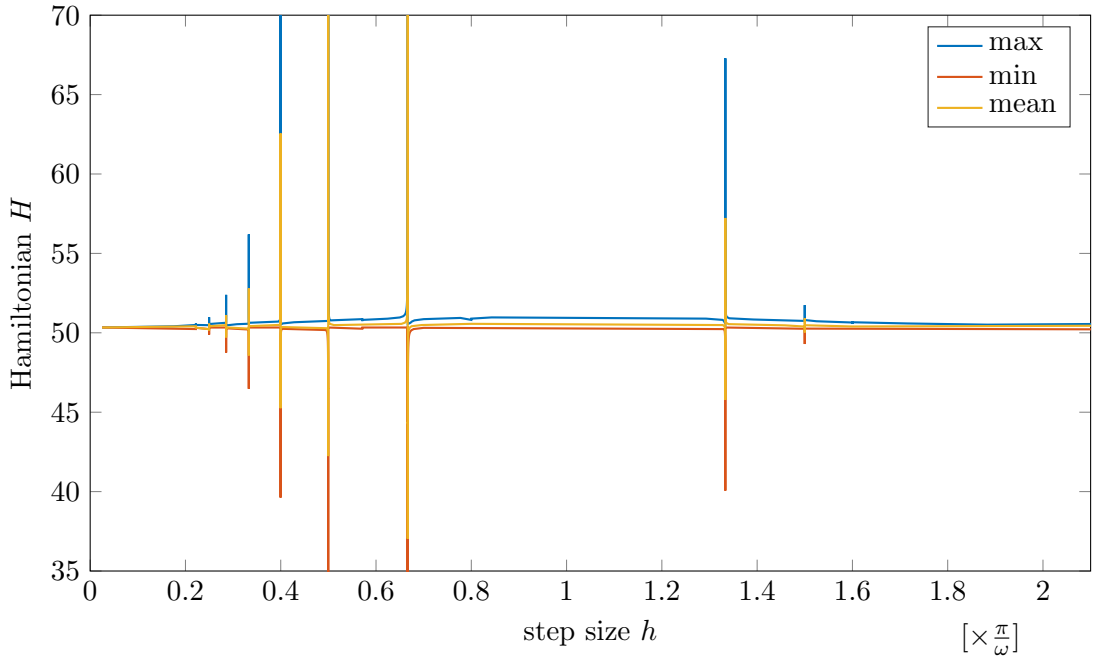


Figure 5.1: *Nonlinear resonance instabilities in the mollified impulse method: maximum, arithmetic mean and minimum of Hamiltonian* (3.4) *with* $\omega = 10$ *over 10.000 integration steps and step size* h

5.1 Averaging and Equilibrium Filters

With nonlinear force fields, it is not always easy to identify and remove components which are created by a fast and stiff potential. Averaging techniques take the current position, denoted by q_n, and the potential associated to unwanted frequencies U_{fast}, and then integrate over a significant time period (e.g. one period) forwards and backwards in time. A short trajectory, mostly exhibiting fast motion, is obtained. These positions are averaged using a weight function Φ:

$$\begin{aligned} &\Psi(q_n) = \int_{-\infty}^{\infty} \Phi(\tau) x(h\tau) d\tau \\ &\text{with} \quad \ddot{x} = -M^{-1} \nabla U_{\text{fast}}(x), \quad x(0) = q_n,\ \dot{x}(0) = 0. \end{aligned} \tag{5.3}$$

Of course, the integral is approximated by numerical integration with a suitable compact weight function Φ. In molecular dynamics, this can be implemented efficiently since the fast potential is cheap in computational cost. Multiple different weight functions are discussed in the literature [19, 85–87]. For application in the mollified impulse method we need the corresponding outer filter, which in case of conservative fast forces is obtained by integrating the variational equation

$$\dot{\psi} = J^{-1} \nabla^2 H(\varphi_t(X_0))\psi, \qquad X_0 = \begin{bmatrix} q_n \\ 0 \end{bmatrix}. \tag{5.4}$$

Recall, that the flow $\varphi_t(X_0)$ denotes the solution at time t with initial value X_0 (see Definition 5). An implementation is not at all straight forward in high end software. This process has significant additional storage requirements, since it needs an entire set of position and momenta coordinates plus integration with the (necessarily sparse) outer filter.

Fast intra-molecular motion can also be removed if the internal bonds and angles are set back to their equilibrium length [71, 88]. In this approach we define a manifold described by a constraint function g. The atomic position q_n is projected onto this manifold by solving

$$\begin{aligned} \Psi(q_n) &= q_n + M^{-1} g_q(q_n)^T \lambda, \\ g(\Psi(q_n)) &= 0. \end{aligned} \tag{5.5}$$

Note that for m constraints, $g : \mathbb{R}^{3N} \to \mathbb{R}^m$ is a vector function and $\lambda \in \mathbb{R}^m$ is a vector of Lagrange multipliers. $g_q \in \mathbb{R}^{m \times 3N}$ denotes the Jacobian with respect to the position vector q. At a first glance resetting internal degrees of freedom with g might seem too crude of an approximation, but if the outer time step is large enough this becomes a useful approximation since molecules oscillate around their equilibrium positions.

Due to the nonlinearity of the problem, (5.5) is usually solved with a Newton-type method. In Algorithm 5.2, the Newton iteration solves for λ when inserting the first equation into the second one in (5.5). In general, this is only feasible if we treat small unconnected clusters, otherwise it is computationally very expensive. Depending on the specific needs, the iteration can be tuned in multiple ways. Most importantly, the *simplified* Newton method, where the derivative is kept fixed during iteration, allows for much cheaper iterations at the cost of slower convergence.

Algorithm 5.2: Newton iteration for solving (5.5)

1 **for** *k=1,2,...* ***until*** *convergence* **do**
2 $\quad q_n^k = q_n + M^{-1} g_q(q_n)^T \lambda_k$
3 $\quad \lambda_{k+1} = \lambda_k - (g_q(q_n^k) M^{-1} g_q^T(q_n))^{-1} g(q_n^k)$

Similar to SHAKE-routines this is very competitive and fast, if unconnected clusters up to size three or four are used. Then the derivative is only 3-by-3 or 4-by-4 and can be analytically inverted. Again, for the mollified impulse method we need the derivative. This is straight forward if the approximation for λ is already computed.

With these filters, a time step roughly twice as large compared to a standard multiple time step method has been achieved. A reliable and stable example is the equilibrium method presented in [88] or a modified version in [71]. Unfortunately, both types of filters need significant computational effort. For every outer time step averaging filters need to integrate forward and backward (including the derivative!) with the fast forces. Equilibrium methods need to solve nonlinear systems resulting from the constraints.

In the next section, we introduce a new type of filter, which we call a *corotational*

filter. It is similar to equilibrium filters in the sense that *corotational filters* also reset certain degrees of freedom within molecules to their equilibrium positions. However, filtered molecules are obtained by an approach based on corotation. While this approach yields a similar quality of the filtering process, it is much cheaper in computational cost. Furthermore, this new filter can be implemented completely without computing trajectory averages or solving nonlinear systems. Instead, it is accessible via a simple and explicit algorithm, thus reducing algorithmic complexity in the filtering process.

Some of the following sections were already published in [21].

5.2 Corotational Filters

The basic idea of corotation is to decompose motion into rotational and translational parts plus deformations from flexibility in bonds and angles. Corotational filters then discard deformational contributions. However, it would be too much to ask that for an arbitrary molecule the structure could be well represented by a single rotation. Therefore, we decompose the (potentially very complex) molecular structure into disjunct clusters of much smaller size, e.g. 2-5 atoms.

5.2.1 Cluster Decomposition of a Biomolecular Structure

The goal of this decomposition is a division of the molecular structure into many but much smaller substructures. Much like in a 'divide and conquer' approach, the substructures are then treated independently, and the connection between substructures is neglected. We call the substructures *clusters*, and every atom should be in exactly one cluster. The construction of the clusters is such that for a fast bond, both atoms need to be part of the same cluster. Only atoms connected by a slower bond are allowed to be members of different clusters. Finally, we require that each cluster has a distinct central atom, which is usually the heaviest one.

Obviously, the classification of faster and slower bonds allows for some choice. Typically, we try to obtain small enough clusters (e.g. 2-5 atoms) such that the overall rotation of that cluster is still a good approximation of the local rotation for all its contained atoms, i.e. that there is no torsional degree of freedom inside a cluster. Of course we could also include trivial clusters containing only a single atom, however, this excludes that atom from the filtering process.

In biomolecular structures, the stretching modes of hydrogen bonds (e.g. H-O, H-N, H-C) are much faster than backbone connections (C-C, C-N, etc). Thus, most clusters consist of a central atom with fast hydrogen bonds within the cluster, and slower backbone bonds connecting it to other clusters. Let us take an alkane chain as an example (see Figure 5.2). It contains slow backbone bonds (C-C) which connect clusters of methyl (CH_3) and methanediyl (CH_2) groups: CH_3-CH_2-...-CH_2-CH_3. The cluster decomposition yields two terminal clusters of size 4, and then a chain of clusters of size 3.

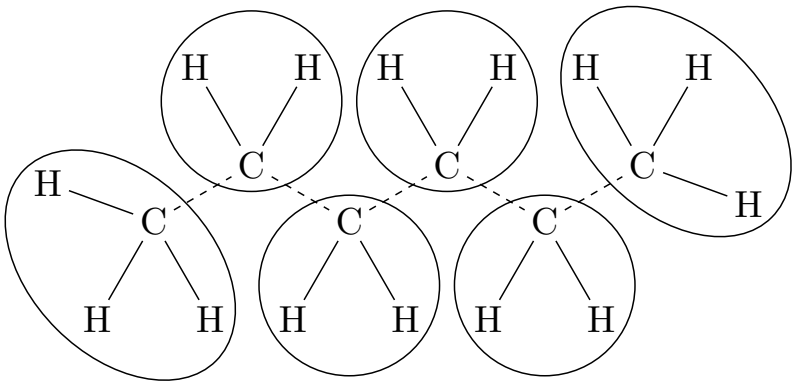


Figure 5.2: *Cluster decomposition of hexane, with two terminal clusters of size 4, and a chain of clusters of size 3. Clusters are indicated by circles, slow bonds are depicted by a dashed line, and fast bonds by a solid line.*

While for simple linear molecules, the choice of clusters is straight forward, it becomes a bit more complex in structures such as aromatic rings. In Figure 5.3, such a ring structure is decomposed in two different ways. Numerical experiments did not indicate a sensitivity towards the specific decomposition, as long as the fastest bonds were contained inside clusters. Indeed, even very simple decompositions (such as the one used for the peptide in the example section later) yield good results. There are (semi-) automatic

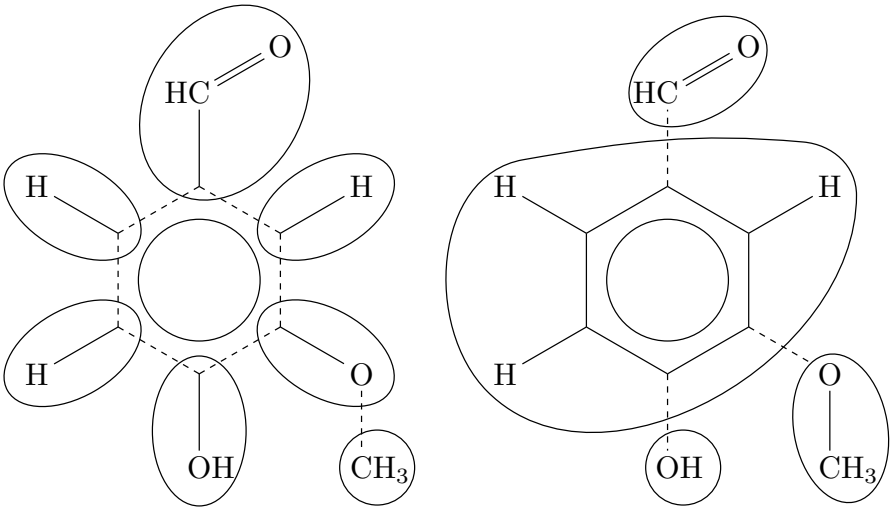


Figure 5.3: *Two possible cluster decompositions of vanillin. The first choice decomposes the ring structure with a few small clusters, the second approach uses the entire ring as a cluster, and then cuts off the attached groups. This works, since the aromatic ring can be considered quite stiff.*

algorithms to identify such cluster decompositions, and they are sometimes used in constraining algorithms such as the SHAKE algorithm implemented in LAMMPS, since such a decomposition allows for a very efficient implementation.

Similar decomposition approaches are used by many other methods in MD, most no-

tably by coarse-graining methods (see e.g. [89, 90] and references therein) and by internal coordinate molecular dynamics (ICMD) such as the GNEIMO method [91, 92].

5.2.2 The New Corotational Filter

After generating a cluster decomposition, the clusters are treated independently. We now illustrate the filter algorithm for a single cluster containing n atoms at position $q = [x_1^T, x_2^T, ..., x_n^T]^T \in \mathbb{R}^{3n}$.

A single cluster is a small structure, and can be well represented by its rotational orientation $R \in \mathbb{R}^{3\times 3}$ and center of mass $c \in \mathbb{R}^3$. To get rid of internal deformations, we construct a reference position $q_0 \in \mathbb{R}^{3n}$, which has the same structure as this cluster, but with all fast internal degrees of freedom (such as bonds and angles) set back to their corresponding equilibrium values. This reference configuration is then rotated and shifted to match the cluster's rotation R and center of mass c.

Definition 13. *(Corotational Filter) Using Kronecker products $\otimes$, the new corotational filter can be calculated as*

$$\Psi(q) = [1, \dots, 1]^T \otimes c + (I_n \otimes RR_0^T)q_0, \tag{5.6}$$

where we assume that q_0 has center of mass at $[0, 0, 0]^T$ and rotational orientation R_0. Note that this can be further simplified, if we choose q_0 such that the resulting R_0 is the identity matrix I_3.

It remains to give an algorithm which quickly approximates the rotational orientation. The given cluster has a central atom with position x_1 bonded to $n-1$ atoms with positions $x_2, ..., x_n$. Let us abbreviate $r_i = x_i - x_1$. A fast way of estimating the rotation between q and q_0 is described in [93]:

Definition 14. *We compute an orthonormal set of vectors $R = [n_1, n_2, n_3] \in \mathbb{R}^{3\times 3}$ with*

$$\begin{aligned} n_1 &= \sum_{i\in S_1} \frac{r_i}{||r_i||} \Bigg/ \left|\left| \sum_{i\in S_1} \frac{r_i}{||r_i||} \right|\right|, \\ n_2 &= \left(s - s^T n_1 n_1\right) / ||s - s^T n_1 n_1|| \,, \quad s = \sum_{i\in S_2} \frac{r_i}{||r_i||}, \\ n_3 &= (n_1 \times n_2) / ||n_1 \times n_2|| \,, \end{aligned} \tag{5.7}$$

where we choose $n_1 \in \mathbb{R}^3$ to be an averaged direction of some bonds $i \in S_1$ and for $n_2 \in \mathbb{R}^3$ we choose a different set S_2 of bonds. S_1 and S_2 are suitably chosen, and we illustrate a few common examples in a following section. The vectors n_1, n_2, n_3 form an orthonormal basis of $\mathbb{R}^3$. R_0 is computed in the same way at reference position q_0. Then RR_0^T is an estimate of the rotation from q_0 to q.

Under mild assumptions on S_1, S_2 and the structure of $x_1, ..., x_n$ (e.g. if $S_1, S_2 \neq \emptyset, s/||s|| \neq n_1$), the vectors n_1, n_2, n_3 form an orthonormal basis. In fact, for a rigid rotation this approach yields the exact rotation. In Figure 5.4, we sketch the filter algorithm applied to a cluster of three atoms.

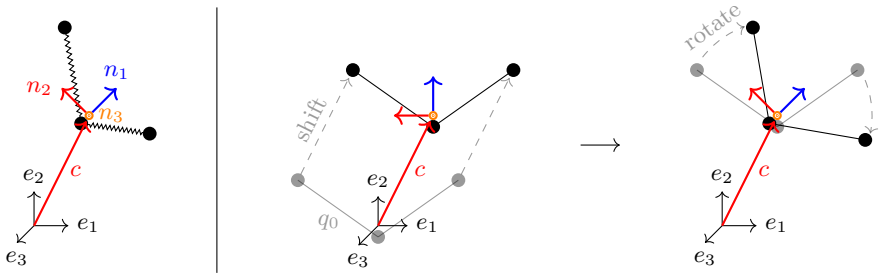


Figure 5.4: *Illustration of the corotational filter algorithm applied to a cluster with three atoms. First, the center of mass c and rotational orientation n_1, n_2, n_3 are computed. Then, a reference configuration q_0 is shifted and rotated to match the cluster.*

5.2.3 Properties of the Corotational Filter

We summarize some basic properties in the following lemma.

Lemma 15. *Properties of Corotational Filter*

By construction, the new corotational filter (5.6) has the following properties:

(a) If a cluster is already in its equilibrium position, the filter is the identity map.

(b) The filter is a small perturbation of the identity map.

(c) The filter is independent of rotations and translations in the following sense: if R describes a rotation around the center of mass of q and s is a translation of the entire cluster, then it holds that $\Psi(Rq + s) = R\Psi(q) + s$

(d) The filter conserves center of mass.

Proof. (a)-(d) directly by construction since these properties hold for each cluster. □

In (5.6), the filter depends only on the center of mass and rotational orientation of that cluster. Since the filter is supposed to smoothen the trajectory and remove highly oscillatory components to suppress resonances, the choice of S_1 and S_2 has a strong impact on the success of the filter. While the center of mass of a cluster is independent of fast internal motion, we need to ensure that the choice of S_1 and S_2 leads to a smooth rotational approximation. Constructing R from the direction of the bonds, the fast changes in bond length are already neglected. Furthermore, by a smart combination of neighboring bonds, it is usually possible to remove angle motions. However, not all fast modes can be removed entirely by a linear combination of bond directions, e.g. some asymmetric stretch modes

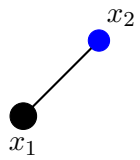
will remain, and we can only reduce the magnitude of those modes. Numerical experiments in Chapter 6 will show, though, that this does not pose a problem.

Note that the filter provides a general approach, and there is a lot of room for tailoring this filter to one's needs. First, the reference position is computed using the rigid body of that cluster. That is, all internal degrees of freedom are set to their equilibrium values. Since such a reference position does not change over time, it can be precomputed - and that is why we will use this approach in the numerical experiments. But it also works if only some faster bonds are reset, and the current angular position is kept. Second, the extraction of rotational information offers some freedom. The more smooth it behaves (under an oscillatory trajectory), the more effective the filter will be. In the given approach, the rotation is estimated using a linear combination of internal directions around a distinct, central atom. But in general, any other approach which gives a robust estimation can be used.

5.2.4 Examples of Sizes Two to Four

The filter presented in the compact notation in (5.6) and (5.7) is not very instructive. Here, we present examples for cluster sizes $n = 2, 3$ and 4. For $n = 2$ and 3, the filter can be expressed more succinctly.

In the simple case $n = 2$, two atoms x_1, x_2 with masses m_1, m_2 are connected via a single bond with equilibrium length r_0. Because of its symmetries, the cluster can be described by a single direction, i.e q_0 has no contribution in direction of n_2, n_3. We choose



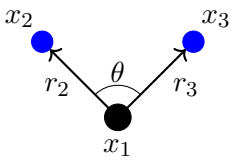
$$n_1 = \frac{x_2 - x_1}{||x_2 - x_1||},$$
$$n_2,\ n_3 \ \text{ s.t. they form an orthonormal basis.}$$

Therefore, the filter reads

$$\Psi(q) = \begin{pmatrix} c + a_1 n_1 \\ c + a_2 n_1 \end{pmatrix}, \qquad \text{with} \quad \begin{aligned} a_1 &= -\frac{m_2}{m_1 + m_2} r_0, \\ a_2 &= \frac{m_1}{m_1 + m_2} r_0. \end{aligned} \tag{5.8}$$

In the formalism of (5.6) and (5.7) this corresponds to $S_1 = \{2\}$ (since it only uses the bond $r_2 = x_2 - x_1$).

For a triatomic cluster ($n = 3$), we can reduce by one dimension if we choose the reference cluster $q_0 \in \mathbb{R}^9$ in the (x, y)-plane. Since the z-component is zero, we can omit the computation of n_3. We consider the following choice:



$$\begin{aligned} n_1 &= \left(\frac{r_2}{||r_2||} + \frac{r_3}{||r_3||} \right) \Big/ \left|\left| \frac{r_2}{||r_2||} + \frac{r_3}{||r_3||} \right|\right|, \\ n_2 &= \left(r_2 - r_2^T n_1 n_1 \right) / ||r_2 - r_2^T n_1 n_1||. \end{aligned} \tag{5.9}$$

This particular choice ($S_1 = \{2,3\}, S_2 = \{2\}$) has the advantage that vibrations due to angle bending and symmetric bond stretching are entirely removed by design; if r_2 changes as much in an opposite direction as r_3, then there is no influence on n_1. n_2 does not change either since r_2 stays in the same plane. This property holds for any triatomic symmetric molecule, most notably for flexible water models.

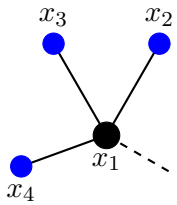
The filter then reads (q_0^i denotes the i-th entry of q_0)

$$\Psi(q) = \begin{pmatrix} c + q_0^1 n_1 + q_0^2 n_2 \\ c + q_0^4 n_1 + q_0^5 n_2 \\ c + q_0^7 n_1 + q_0^8 n_2 \end{pmatrix}. \tag{5.10}$$

However, we have to make one restriction: if θ is close or equal to 180° such as in thiocyanate (SCN) or hydrogen cyanide (HCN), $r_2/||r_2|| + r_3/||r_3||$ threatens to cancel. Thus, for such a cluster, it is much better to use a treatment similar to the case $n = 2$ with

$$n_1 = \left(\frac{r_2}{||r_2||} - \frac{r_3}{||r_3||} \right) \Big/ \left|\left| \frac{r_2}{||r_2||} - \frac{r_3}{||r_3||} \right|\right|. \tag{5.11}$$

Clusters with four atoms ($n = 4$) frequently happen with terminal CH_3 groups, where the carbon atom has a slow bond connecting it onwards to the next cluster.



Here, the best choice would be $S_1 = \{2,3,4\}$. For S_2 we choose any two of the 'satellites': $S_2 = \{2,3\}$ or $S_2 = \{3,4\}$ or $S_2 = \{2,4\}$. However, in a planar tetra-atomic structure (such as formaldehyde H_2CO) similar to the three atomic linear case, we face the danger that $r_2 + r_3 + r_4 \approx 0$. Instead, we then choose S_1 and S_2 such that n_1, n_2 reliably characterize the plane in which the cluster lives.

5.2.5 The Jacobian of the Corotational Filter

Application within the mollified impulse method requires the Jacobian $\Psi_q(q)$. In contrast to other filters in the literature, the derivative of the corotational filter is explicitly given and easy to compute. It does not require the solution of any nonlinear systems. Note that both filter and its derivative are spatially local properties, making a parallel implementation within a domain decomposition strategy fairly easy.

With notation as in (5.6) and (5.7) for a cluster of n atoms we have

$$\Psi(q) = [1, .., 1]^T \otimes c + (I_n \otimes R R_0^T) q_0$$

with

$$
\begin{aligned}
n_1 &= \sum_{i\in S_1} \frac{r_i}{||r_i||} \Bigg/ \left|\left| \sum_{i\in S_1} \frac{r_i}{||r_i||} \right|\right| =: \frac{u}{||u||}, \\
n_2 &= \left(s - s^T n_1 n_1\right) / ||s - s^T n_1 n_1|| =: \frac{v}{||v||}, \\
n_3 &= (n_1 \times n_2) / ||n_1 \times n_2|| =: \frac{w}{||w||}.
\end{aligned}
$$

We choose the reference cluster $q_0 = [q_0^1, q_0^2, \ldots, q_0^{3n}]^T \in \mathbb{R}^{3n}$ such that R_0 is the identity I_3. The derivative then simplifies to:

$$
\Psi_q(q) = [1, ..., 1]^T \otimes \frac{\partial}{\partial q} c + \begin{pmatrix} q_0^1 \frac{\partial n_1}{\partial q} + q_0^2 \frac{\partial n_2}{\partial q} + q_0^3 \frac{\partial n_3}{\partial q} \\ q_0^4 \frac{\partial n_1}{\partial q} + q_0^5 \frac{\partial n_2}{\partial q} + q_0^6 \frac{\partial n_3}{\partial q} \\ \ldots \end{pmatrix} \tag{5.12}
$$

with

$$
\begin{aligned}
\frac{\partial n_1}{\partial q} &= \frac{1}{||u||} \left(I_3 - n_1 n_1^T\right) \left(\sum_{i\in S1} \frac{1}{||r_i||} \left(I_3 - \frac{r_i}{||r_i||} \frac{r_i^T}{||r_i||} \right) \frac{\partial r_i}{\partial q} \right), \\
\frac{\partial n_2}{\partial q} &= \frac{1}{||v||} \left(I_3 - n_2 n_2^T\right) \left[\left(I_3 - n_1 n_1^T\right) \frac{\partial s}{\partial q} - \left(s^T n_1 I_3 + n_1 s^T\right) \frac{\partial n_1}{\partial q} \right], \\
\frac{\partial n_3}{\partial q} &= \frac{1}{||w||} \left(I_3 - n_3 n_3^T\right) \left(\frac{\partial n_1}{\partial q} \times n_2 + n_1 \times \frac{\partial n_2}{\partial q} \right).
\end{aligned} \tag{5.13}
$$

In the last row of (5.13) we slightly abuse the notation, and the cross products are meant column-wise. The derivative of the center of mass in (5.12) is given by

$$
\frac{\partial c}{\partial q} = \frac{1}{m_{\text{all}}} [m_1, m_2, \ldots, m_n] \otimes I_3, \qquad \text{with} \qquad m_{\text{all}} = \sum_{i=1}^{n} m_i. \tag{5.14}
$$

Note that in (5.13) we have $\partial r_i / \partial q = (e_i - e_1) \otimes I_3$ where e_k denotes the k-th canonical unit vector. Furthermore,

$$
\frac{\partial s}{\partial q} = \sum_{i\in S2} \frac{1}{||r_i||} \left(I_3 - \frac{r_i}{||r_i||} \frac{r_i^T}{||r_i||} \right) \frac{\partial r_i}{\partial q} \tag{5.15}
$$

and a similar expression, but for S_1 already appears in $\partial n_1 / \partial q$. Thus, if there is an overlap between S_1 and S_2, we can reuse part of that calculation. The derivative looks quite complicated, but one is reminded that most quantities are constructed from simple vectors of size 3, which already have been computed with the filter. Also, from an implementational viewpoint, the matrix Ψ_q does not need to be constructed explicitly, instead only its action on the force vector is required.

For the cases $n = 2, 3$ the derivative can be expressed much more concisely. In the

setting of (5.8), for a cluster with two atoms the derivative reads

$$\begin{aligned}\Psi_q(q) &= [1,1]^T \otimes \frac{\partial}{\partial q} c + \begin{pmatrix} a_1 \frac{\partial n_1}{\partial q} \\ a_2 \frac{\partial n_1}{\partial q} \end{pmatrix}, \\ \frac{\partial n_1}{\partial q} &= \frac{1}{||x_2 - x_1||}(I_3 - n_1 n_1^T)[-I_3, I_3].\end{aligned} \tag{5.16}$$

The derivative for a cluster with three atoms as obtained from (5.10) is

$$\Psi_q(q) = [1,1,1]^T \otimes \frac{\partial}{\partial q} c + \begin{pmatrix} q_0^1 \frac{\partial n_1}{\partial q} + q_0^2 \frac{\partial n_2}{\partial q} \\ q_0^4 \frac{\partial n_1}{\partial q} + q_0^5 \frac{\partial n_2}{\partial q} \\ q_0^7 \frac{\partial n_1}{\partial q} + q_0^8 \frac{\partial n_2}{\partial q} \end{pmatrix}. \tag{5.17}$$

5.2.6 Mechanism of Derivative

It is quite interesting to see the mechanism behind the derivative when applied to a force vector F. We denote the filtered force by $\tilde{F}$. Then we have

$$\begin{aligned}\tilde{F} = \Psi(q)_q^T F &= \left([1,...,1]^T \otimes \frac{\partial c}{\partial q} + \frac{\partial}{\partial q}(I_n \otimes R) q_0 \right)^T F \\ &= \underbrace{\left([1,...,1] \otimes \left(\frac{\partial c}{\partial q} \right)^T \right) F}_{\tilde{F}^1} + \underbrace{\left(\frac{\partial}{\partial q}(I_n \otimes R) q_0 \right)^T F}_{\tilde{F}^2}.\end{aligned}$$

The first part - $\tilde{F}^1$ - is a center of mass movement. Denoting by subscript i only the components for atom i (i.e. $F_i \in \mathbb{R}^3$ is the force on atom i), we have for the filtered value

$$\tilde{F}_i^1 = \frac{m_i}{m_{\text{all}}} \sum_{j=1}^{n} F_j, \qquad \text{with} \quad m_{\text{all}} = \sum_{i=1}^{n} m_i.$$

For the second part, let us first have a look at the i-th column of $\frac{\partial}{\partial q}(I_n \otimes R) q_0$, since this column's transpose will be multiplied with F to yield the i-th entry of $\tilde{F}^2$. The i-th column consists of the derivative of $(I_n \otimes R(q)) q_0$ with respect to q_i (where this time we mean the i-th entry of q, and not the position of the i-th atom):

$$\frac{\partial (I_n \otimes R(q)) q_0}{\partial q_i} = \left(I_n \otimes \frac{\partial R(q)}{\partial q_i} \right) q_0.$$

Since $R(q)$ is a rotation matrix, for its derivative we have

$$\frac{\partial R(q)}{\partial q_i} = S_\omega R(q)$$

with a skew-symmetric matrix S_ω, associated with a rotation around an axis $\omega \in \mathbb{R}^3$, and can also be written as a cross product (see e.g. [6, Ch. IV.6]). This means, that

$S_\omega R(q)v \perp R(q)v$ for any $v \in \mathbb{R}^3$, and therefore also

$$(I_n \otimes R(q))q_0 \perp \frac{\partial (I_n \otimes R(q))q_0}{\partial q_i}.$$

Most importantly, changing q_i generates a rotation of the cluster around the center of mass with axis ω. The filter now discards all components except force contributions in the direction of $(I_n \otimes S_\omega R(q))q_0$. Especially, components of F_i which are in the direction of $R(q)q_0^i$ will be removed.

Altogether, the second part is the rotational contribution of the force vector to the center of mass. Also, atoms in a cluster which do not contribute to either n_i will not get any rotational force. So, applying the derivative to a force vector only keeps the center of mass movement plus components which are necessarily perpendicular to the 'internal body coordinates'.

5.2.7 The Full Algorithm

Algorithm 5.3 summarizes the necessary steps as explained in the previous sections. First, a cluster decomposition of the molecular structure has to be determined and the individual reference configurations for each cluster need to be computed. Since both the cluster decomposition and the reference configurations only need to be computed once for every structure, they can be precomputed.

Now, whenever we need the filtered position and its derivative, for each cluster, we estimate the local rotation. By rotating and shifting the corresponding reference configuration we obtain the filtered position for each cluster. This way internal deformations, which are not represented by overall rotation or center of mass movement of a cluster, are discarded. The filtered position of the entire molecular system is then assembled by simply taking the filtered values from each individual cluster. In a similar manner, we proceed with the Jacobian of the filter.

Algorithm 5.3: Corotational filter algorithm

Input: position q

Output: filtered position $\Psi(q)$, Jacobian $\Psi_q(q)$

Precompute:
- generate a cluster decomposition
- compute a reference configuration for each cluster

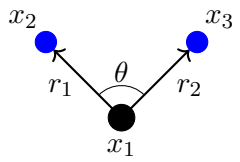
Filter:

for *each cluster* **do**
- estimate local rotation via (5.7)
- compute filtered position via (5.6)
- compute Jacobian via (5.12)

- assemble $\Psi(q), \Psi_q(q)$ from individual clusters

5.2.8 Corotational Filter for Water

Let us illustrate the mechanism behind the corotational filter when applied to a flexible water model. We consider the following choice (where $x_1, x_2, x_3 \in \mathbb{R}^3$ denote positions of the atoms, and $r_1 = x_2 - x_1, r_2 = x_3 - x_1$):



$$\begin{aligned} n_1 &= \left(\frac{r_1}{||r_1||} + \frac{r_2}{||r_2||}\right) \Big/ \left|\left|\frac{r_1}{||r_1||} + \frac{r_2}{||r_2||}\right|\right|, \\ n_2 &= \left(r_1 - r_1^T n_1 n_1\right) / ||r_1 - r_1^T n_1 n_1||, \\ n_3 &= (n_1 \times n_2) \,/\, ||n_1 \times n_2|| \,. \end{aligned} \tag{5.18}$$

This particular choice has the advantage that vibrations due to angle bending and symmetric bond stretching are entirely removed by design; if r_1 changes as much in an opposite direction as r_2, then there is no influence on n_1. n_2 and n_3 do not change either since r_1 stays in the same plane. This property holds for any triatomic symmetric molecule. Here, the term symmetry is understood not only for the geometry of the molecule, but also the masses and force field parameters.

In the case of water molecules, we only have a single cluster with three atoms per molecule. For flexible water we can directly compare the projection filter (5.5) to our new filter. Following [88] we choose

$$g(q) = \frac{1}{2} \begin{pmatrix} ||x_2 - x_1||^2 - r_0^2 \\ ||x_3 - x_1||^2 - r_0^2 \\ ||x_3 - x_2||^2 - 2r_0^2(1 - \cos\theta_0) \end{pmatrix}$$

for water molecules at position $q = [x_1^T, x_2^T, x_3^T]^T \in \mathbb{R}^9$ with equilibrium bond length r_0 and equilibrium angle θ_0. As before, x_1 is the position of the central oxygen atom. Hence, for every molecule, a small 3-by-3 nonlinear constraint system has to be solved e.g. with a simplified Newton method. Since the molecules are close enough to an equilibrium position, this method usually converges within a few steps. Nevertheless, it is more expensive than the corotational filter since the latter only costs roughly as much as a single Newton step.

Lemma 16. *For water molecules (or any symmetric triatomic molecules) equilibrium (5.5) and corotational filter ((5.6) with (5.18)) are close in the sense that they are exactly equal whenever the two bonds have equal length.*

Proof. A water molecule in its equilibrium position is uniquely identified by its center of mass and rotation matrix R containing the vectors n_i, $i = 1, 2, 3$. Both filters maintain center of mass. For the corotational filter, by design, the vectors n_i do not change during filtering. So we only need to check the vectors n_i for the equilibrium filter.

It is, in fact, sufficient to check n_1 since both filters do not alter the plane in which the molecule lives. Denoting filtered values with tilde, easy calculations reveal that after

filtering we have

$$\frac{\tilde{r}_1}{||\tilde{r}_1||} + \frac{\tilde{r}_2}{||\tilde{r}_2||} = \frac{r_1}{r_0}\left(1 + \frac{\lambda_1}{m_2} + 2\frac{\lambda_1}{m_1}\right) + \frac{r_2}{r_0}\left(1 + \frac{\lambda_2}{m_3} + 2\frac{\lambda_2}{m_1}\right)$$

where m_i is the mass associated with x_i. If we would have $m_2 = m_3$ and $\lambda_1 = \lambda_2$ then we would have

$$\frac{\tilde{r}_1}{||\tilde{r}_1||} + \frac{\tilde{r}_2}{||\tilde{r}_2||} = c\left(\frac{r_1}{||r_1||} + \frac{r_2}{||r_2||}\right), \quad c \in \mathbb{R}$$

and hence n_1 would not change during filtering for the equilibrium method.

Obviously for water we have $m_2 = m_3$. Furthermore, by assumption, both bonds have the same length and there is a solution with $\lambda_1 = \lambda_2$. Since $\frac{\partial g(\Psi(q))}{\partial \lambda}$ has full rank (note that r_1 is not parallel to r_2), this solution is also unique by the implicit function theorem. □

5.2.9 Slow Energy Exchange

Filtering tempers with the energy flow. But how much does filtering effect the results? The slow potential in the mollified impulse method becomes independent of the fastest motions. Slow force contributions in the direction of the fastest modes are filtered. On the other hand, in a regular three stage splitting, the intermediate stage might be responsible for transferring most energy flow anyways, so how significant is the influence?

There is both theoretical and numerical evidence that the mollified impulse method applied to the Fermi-Pasta-Ulam problem has a major impact on the energy exchange between stiff springs [6, 94]. For MD, it is difficult to find a good testing problem. First, we try a two stage splitting (bond — pair) on three SPC/F water molecules without periodic boundary conditions, that is, we test the mollified impulse method with

$$\phi_h^H \approx \varphi_{h/2}^{U_{\text{pair}}(\Psi(\cdot))} \circ \left(\varphi_{h/2L}^{U_{\text{intra}}} \circ \varphi_{h/L}^{T} \circ \varphi_{h/2L}^{U_{\text{intra}}}\right)^L \circ \varphi_{h/2}^{U_{\text{pair}}(\Psi(\cdot))}. \tag{5.19}$$

The initial position of the three water molecules is perfectly minimized forming a triangular position. Then we give one water molecule a small force kick on one of its hydrogen atoms. This creates a vibration in that molecule, which is then passed on to its neighbors via pair interactions. We plot the time evolution of oscillatory energies $I_i = T_i + U_{\text{bond+angle}}$, $i = 1, 2, 3$ for a small step size Verlet method, the standard impulse method, and the mollified impulse method with filtered pair stage.

Figure 5.5 shows that both Verlet and the standard impulse method seem to capture the energy exchange well. The energy quickly transfers between the molecules. The mollified integrator, however, barely reproduces this behavior. The filter 'traps' the energy in the single molecule and impedes the energy exchange.

More mathematically, the oscillatory energy in each molecule is given by

$$I_i = \frac{1}{2} p_i^T M_i^{-1} p_i + \frac{k_b}{2} \sum_j (r_j - r_0)^2 + \frac{k_a}{2} (\varphi_i - \varphi_0)^2$$

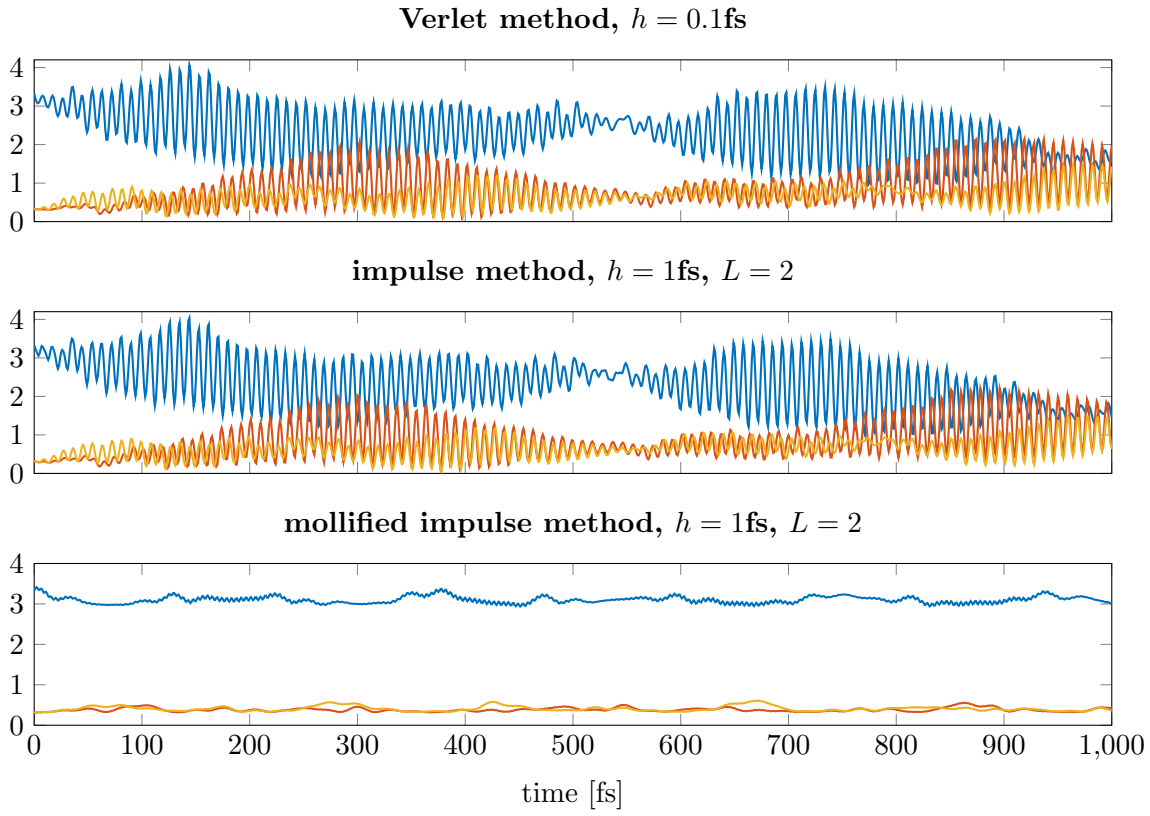


Figure 5.5: *Oscillatory energies - that is bond+angle+kinetic energy - per molecule on a 1ps trajectory for three different integrators*

where $q_i, p_i \in \mathbb{R}^9$ stores the position and momentum of molecule $i = 1, 2, 3$, and M_i is the associated mass matrix. The sum over j runs over the two hydrogen bonds with distance r_j. φ_i denotes the angle in the i-th molecule. k_a, k_b, r_0 and φ_0 are parameters of the corresponding harmonic force field.

For the leading term in the exact energy exchange with the other components we have (∇_{q_i} denotes differentiation only with respect to the position variables q_i in molecule i):

$$\dot{I}_i = \{I_i, H\} = \{I_i, I_1 + I_2 + I_3 + U_{\text{pair}}\} = \{I_i, U_{\text{pair}}\} = -p_i^T M_i^{-1} \nabla_{q_i} U_{\text{pair}}.$$

On the other hand, with a mollified pair potential, we have

$$\dot{I}_i = -p_i^T M_i^{-1} \nabla(U_{\text{pair}}(\Psi q)) = -p_i^T M_i^{-1} \Psi_{q_i}^T \nabla_{q_i} U_{\text{pair}}(\Psi q)$$

and the derivative $\Psi_{q_i}^T$ – as discussed in Section 5.2.6 – removes components in the direction of the fastest motions. All remaining components correspond to either a center of mass movement or an overall rotation of the molecule. Hence, the energy flow is significantly impeded.

While this simple example serves as a warning to be careful when using a filter, in more realistic simulations - such as the following discussion of examples (A) and (B) -

there is no need to filter the pair stage. The computational bottleneck is the long-range electrostatic contributions. Therefore, in realistic simulations with long-range forces we employ a three stage splitting:

$$\phi_h^H \approx \varphi_{h/2}^{U_{\text{long}}(\Psi(\cdot))} \circ \left(\varphi_{h/2K}^{U_{\text{pair}}} \circ \left(\varphi_{h/2KL}^{U_{\text{intra}}} \circ \varphi_{h/KL}^{T} \circ \varphi_{h/2KL}^{U_{\text{intra}}} \right)^L \circ \varphi_{h/2K}^{U_{\text{pair}}} \right)^K \circ \varphi_{h/2}^{U_{\text{long}}(\Psi(\cdot))} \tag{5.20}$$

Here, we only filter the outer stage with respect to the fast oscillations of the inner stage. The intermediate stage remains unfiltered. Using this intermediate stage allows for an undisturbed energy flow between the short-range pair potential and the intra-molecular potential. See Algorithm 5.4 for an algorithmic description.

Algorithm 5.4: One step of the mollified impulse method applied to (2.1) with (2.3) and (2.10)

Input: position q, momentum p, step size h, stage factors K, L, filter Ψ
Output: new position q, new momentum p

$p \hookleftarrow p - \frac{h}{2}\Psi_q^T(q)\nabla U_{\text{long}}(\Psi(q))$
for $i = 1, \ldots, K$ **do**
  $p \hookleftarrow p - \frac{h}{2K}\nabla U_{\text{pair}}(q)$
  for $j = 1 \ldots, L$ **do**
    $p \hookleftarrow p - \frac{h}{2KL}\nabla U_{\text{intra}}(q)$
    $q \hookleftarrow q + \frac{h}{KL}M^{-1}p$
    $p \hookleftarrow p - \frac{h}{2KL}\nabla U_{\text{intra}}(q)$
  $p \hookleftarrow p - \frac{h}{2K}\nabla U_{\text{pair}}(q)$
$p \hookleftarrow p - \frac{h}{2}\Psi_q^T(q)\nabla U_{\text{long}}(\Psi(q))$

For the more realistic tests, we use problems (A) and (B). Following oscillatory energies of single molecules does not make sense here. Instead we use structural properties.

5.3 Performance of the Mollified Impulse Method

We test the mollified impulse method on examples (A) and (B). Since it already has been shown that averaging filters are less stable than equilibrium filters, we focus on the projection and corotational filter. Actually, we are more interested in the corotational filter, since it promises to be cheaper. If the corotational filter introduced in Section 5.2 is used in the mollified impulse method, we refer to this method as the corotational impulse method (CIM). When the outer stage is filtered with the equilibrium filter in (5.5) we call it equilibrium impulse method (EIM). More details can be found in Appendix C.

For water, the cluster decomposition is natural, each molecule is a cluster of size three. The reference configuration then is simply a water molecule with equilibrium bond length and angle as specified in the corresponding water model. The decomposition of the peptide is chosen such that all bonds containing a hydrogen and carbonyl groups (C=O) are filtered. Then, the peptide decomposes in clusters mostly of size two and

three. However, there are three terminal CH_3 groups which form clusters of size four, and four single atoms remain, which we choose not to filter. Of the four unfiltered atoms, three are carbon atoms with slow onwards connections in the aromatic rings, and the last one is a sulfur atom connected with two slow bonds to a CH_3 and CH_2 group. Note that those four atoms can be easily included in a neighboring cluster. However, as the numerical results indicate, this is not necessary due to their slow bonds. In total, the decomposition has 25 clusters of size 2, 646 of size 3 and 3 of size 4.

5.3.1 Efficiency of Filter in Removing Fast Motions

The corotational filter is supposed to smoothen a trajectory. First we check that the filter is actually reliable. We take the trajectory we computed for calculating the infrared spectra in Figure 4.3. We now apply the corotational filter to each time step and then compute the infrared spectrum of the filtered positions.

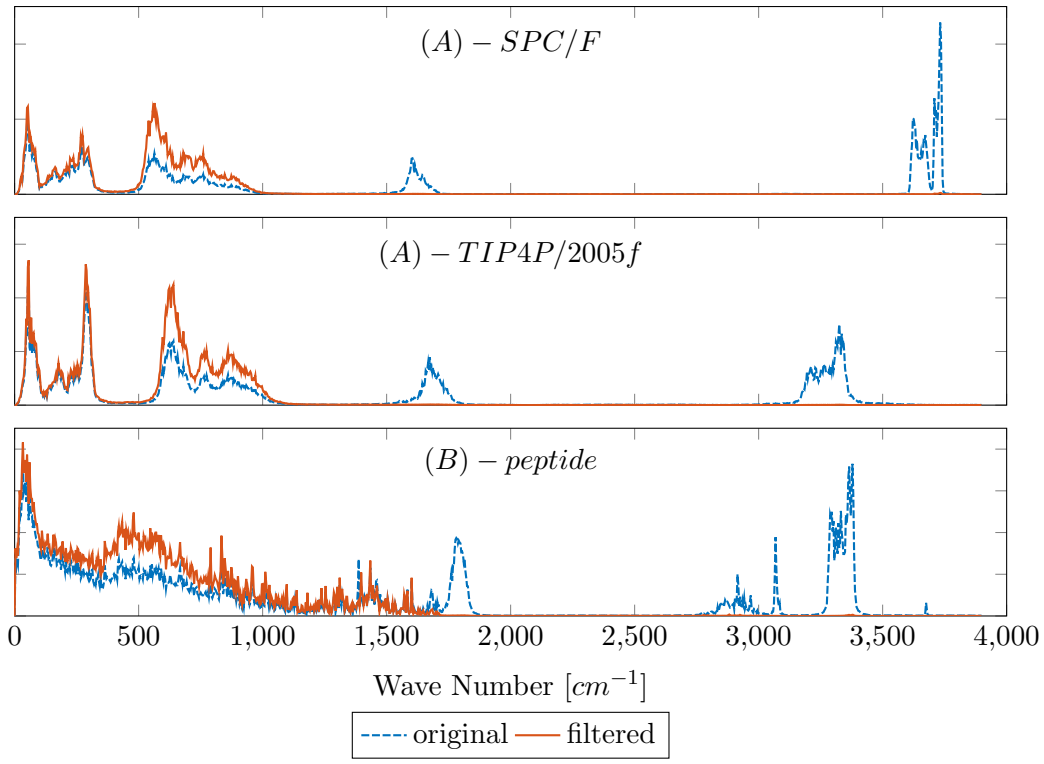


Figure 5.6: *IR spectra: wave number* $[cm^{-1}]$ *vs intensity*

In Figure 5.6, we compare the unfiltered with the filtered values, and indeed, the corotational filter removes all frequencies associated with bond and angle motion. That also seems to work very well in a heterogeneous molecule, such as the small peptide with its many different bonds and angles. Note that this filtering only refers to the position when evaluating the slow force, and the mollified impulse method, in contrast to SHAKE, retains the ability to fully resolve the fast oscillations.

5.3.2 Energy Drifts

Energy drifts are computed as before, except that for filtered versions we compute the energy at the filtered slow potential $U_{\text{long}}(\Psi(q))$ (since this can be done on the fly).

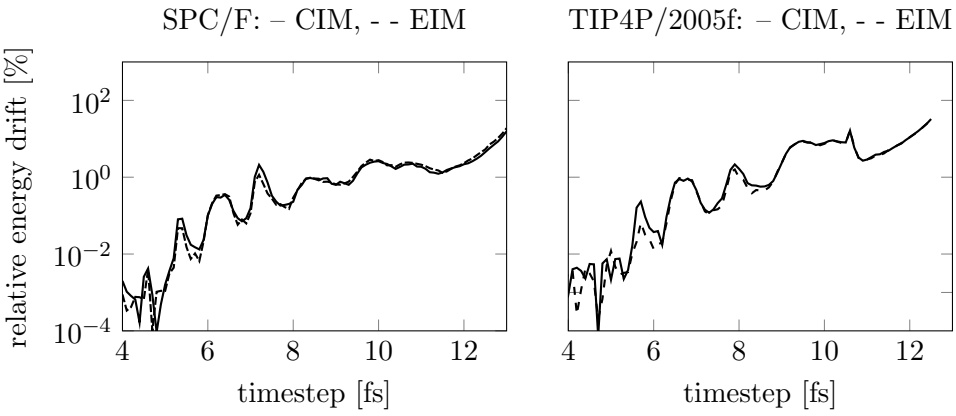


Figure 5.7: *(A) - SPC/F, TIP4P/2005f: energy drift [%] vs outer step size [fs]*

Severe resonance instabilities, as observed with the unfiltered impulse method (IM), do not seem to appear in the filtered integrators at all. In fact, step sizes as large as 10 fs (or around 9fs for the TIP4P model) can be used if we are willing to accept a drift of less than 1% on a 50ps run with 2880 molecules. This should not be a problem if we are running very long simulations within a NVT/NPT ensemble with a very weak coupling to a target temperature and/or pressure. As for the impulse method, there are artifacts of nonlinear resonances visible for both the CIM and EIM. We suspect that they originate from modes associated with angle vibrations at around 20.8fs (SPC/F) and 20.2fs (TIP4P/2005f). The peaks around 10fs are then 2:1 instabilities, those just below 7fs are 3:1 resonances, and the remaining peaks are probably a mix of 3:1 and 4:1 resonances from different modes.

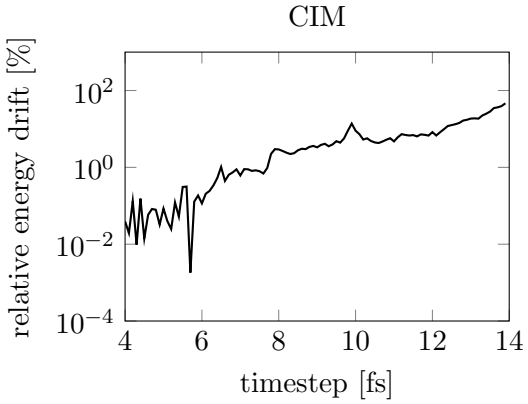


Figure 5.8: *(B) - peptide: energy drift [%] vs outer step size [fs]*

5.3.3 Structural and Energy Analysis

A radial distribution function (RDF) is a structural fingerprint. It describes how density varies as a function of distance. It is computed by calculating the distance of all atom pairs of a given type and binning them into a histogram. Thus, it provides a measure of the probability of finding a given atom type at a certain distance away from a reference atom.

For problem (A) we investigate the RDF between oxygen atoms obtained after 250ps equilibration in NPT at 180°K and 1bar with each method.

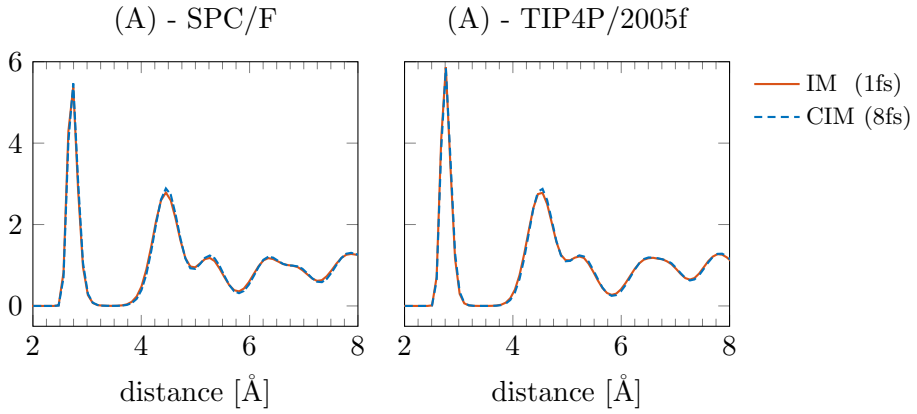


Figure 5.9: *(A) - RDFs of oxygen-oxygen distribution*

In Figure 5.9 we compare RDFs obtained with a large step CIM at 8fs to a reference solution using the IM with a small outer step size of 1fs. Bond and pair forces are computed every 0.25fs. The data is collected and averaged over 10ps. The results are almost indistinguishable, and we have peaks at exactly the same positions. There are small differences in height for both models, with a slight emphasis on the low energy positions.

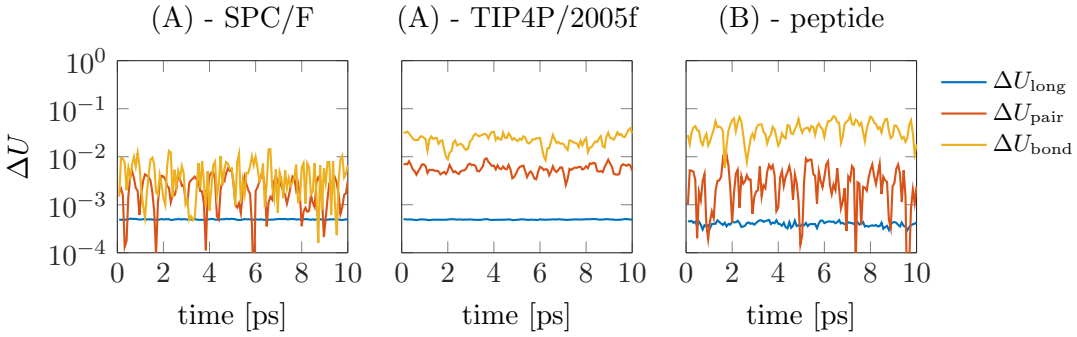


Figure 5.10: *(A) and (B): relative energy difference of CIM*

Filtering long-range force contributions successfully removes resonance instabilities which originate from the fastest vibrations. However, filtering interferes with the slow energy exchange between fast and slow modes. We compare differences in the potential energy of U_{bond}, U_{pair} and U_{long}. After 250ps equilibration (A: NPT at 180°K and 1bar, B:

NPT at 250°K and 1bar) with each method, we collect these values for 10ps. Figure 5.10 plots the relative difference $\Delta U = |U_{\mathrm{CIM}} - U_{\mathrm{IM}}|/|U_{\mathrm{IM}}|$, where we use the CIM with an 8fs outer time step and the IM as a reference solution with outer time step of 1fs.

These plots indicate that there is a slight difference, primarily in bonded interactions. This is somewhat expected since we mollify the long-range force contributions exactly in the direction of these interactions. Otherwise we have no indication that filtering manipulates the energy flow.

5.3.4 Computational Cost

Similar to Section 4.5, timing results in Figure 5.11 are once more obtained from short 50ps simulations after equilibration. The results are averaged over ten runs each. They are obtained on the bwUniCluster using 128 cores (8 nodes with 16 cores each).

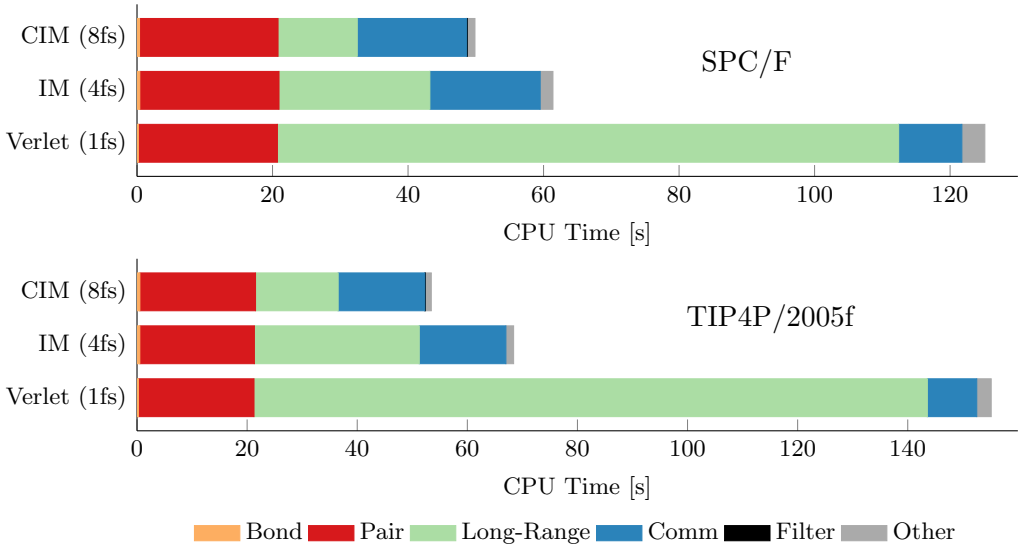


Figure 5.11: *(A) - absolute CPU time in seconds walltime for the SPC/F and TIP4P/2005f model, as obtained from short 50ps simulations using 128 cores. For the CIM, the time spent on the filter is below 0.1s for both models.*

The improvement of CIM over a regular Verlet method is about eight times less time spent on long-range computations. Compared to a standard IM, the CIM saves half the computational cost of long-range interactions. The additional computation of the filter itself is very cheap and costs less than 0.1% overall CPU time. Therefore, in Figure 5.11, it is represented by a very narrow black line between *Comm* and *Other*. Obviously, we also timed the EIM, but do not give separate results. Qualitatively, it is the same as the CIM, and only differs in the filtering time, where it tends to be 50% more expensive. However, the overall variation between the ten runs each is of a similar magnitude.

In general, at least $15 - 20\%$ overall speed up can be expected by using the filtered CIM compared to a standard IM; and $> 50\%$ gains over the traditional Verlet algorithms can be accomplished.

6 Long-term Performance of the Mollified Impulse Method

The following two realistic problems show that the mollified impulse method combined with the corotational filter can be used in the type of simulations which frequently appear in molecular dynamics. Both problems investigate phenomena which happen at a much slower time scale, and therefore, long simulations are necessary to observe them. First, we investigate the process of ice friction. Due to the complexity of phase transformation, the friction force takes a long time to converge. We compute 17ns trajectories. Second, we highlight slow protein folding on a classical example, namely, bovine pancreatic trypsin inhibitor. Folding is a very slow process, and usually occurs at time scales of the order of microseconds or slower. We neither have the hardware nor the cluster time for such simulations, however, we observe the initiation of a folding process in a 80ns trajectory.

6.1 (C) - Ice-Ice Friction

In problem (C) friction processes between two layers of ice Ih are simulated. It has been known for a long time that the creation of a liquid layer in between ice interfaces plays a significant role. The investigation of the dynamics and mechanisms in this liquid layer is an active area of research from both theoretical/computational and experimental groups [95–99].

The interesting and challenging part here is the phase transformation at the interface: in contrast to the crystal ice, the liquid layer has a higher temperature and density. This layer is more active, and the diffusion is much higher than in the ice.

Similar to [97], we create two slabs of ice separated by a vacuum in z-direction and bring them slowly into contact. The motion of the two slabs is controlled by harmonically restricting the movement of a layer of water molecules in each slab (indicated by yellow coloring in Figure 6.1). This allows us to control friction velocity and contact pressure. Each slab contains 2880 molecules, in total we have to keep track of 17280 atoms. We choose the TIP4P/2005f model for this investigation.

The following simulations are computed with the CIM using an outer step size of 8fs (8fs steps with the long-range forces, pair forces are evaluated every fs and bond forces every 0.5fs).

6.1.1 Simulation Protocol

First, an initial equilibration is performed for 800ps in NPT targeting 1.0bar and slowly heating up to 180°K. After equilibration, the simulation switches to NVE except for the constrained layers. Here, temperature is controlled by a Nose-Hoover thermostat.

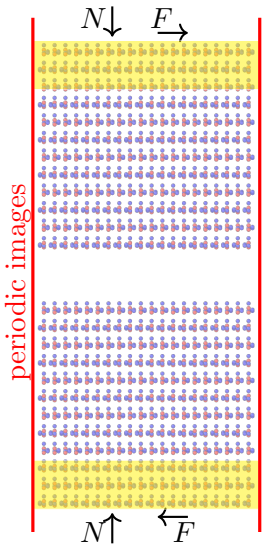


Figure 6.1: *(C) - friction model: yellow coloring indicates areas where we impose friction and normal force to control velocity and contact pressure*

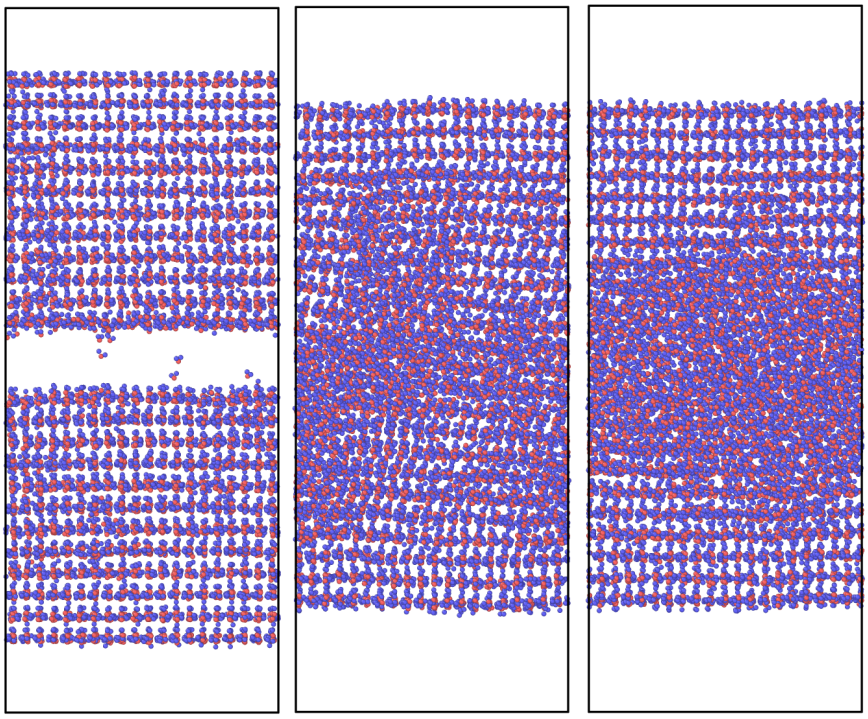


Figure 6.2: *(C) - snapshots after 800ps, 1200ps and 9200ps, 5m/s friction velocity*

Throughout the simulation, friction causes heat to be created at the interface. Therefore, the temperature control of the constrained layers models heat dissipation out of the domain.

Each slab is set to an opposing velocity and we start applying a normal force to the constrained layers of molecules. Within the next 8.4ns of simulation time we slowly increase the target temperature of the constrained layers to 210°K. Afterwards, the temperature

of the constrained layers is kept fixed at 210°K and the simulation continues for another 8ns.

The friction velocity is chosen ranging from 0.01m/s to 10m/s. For the imposed normal force we select 10 and 100kcal/mol-Å, half of each is applied to the top layer, the other half is applied in the opposite direction to the bottom layer.

Total simulation time for *each* run is 17.2ns. Most runs were computed on 64 cores (4 nodes with 16 cores each) at bwunicluster. Depending on the exact configuration 8ns of simulation cost roughly 10-12h walltime. So each combination of friction velocity and normal load requires roughly 1 day of computing time on 64 cores. On average, long-range forces make for around 20% of the computing time, pair forces for around 40% and bond forces for around 1%. Communication between processors and nodes takes around 30-35%, filter induced communication needs less than 0.5%. Filter computation is almost neglectable with below 0.3%. In Figure 6.2 three pictures show the domain after the initial equilibration, then just after both slabs come into contact, and finally, after the liquid layer has formed.

6.1.2 Friction Force

Previous experimental studies [98, 99] found that friction forces are rather sensitive to a variety of parameters such as velocity, normal force and temperature. Unfortunately, it takes a long simulation to get from static friction to kinetic friction. Figure 6.3 shows one simulation run, plotting the average friction force over time (at 100kcal/mol-Å normal force and 5m/s velocity). We can see that as soon as the slabs come into contact, due to static friction there is a high peak in friction. The strong connections subsequently begin to break down, and we observe sliding between the layers. The force of friction drops quickly in the beginning, but much slower afterwards. It takes a long time to converge. This behaviour can be spotted to some extent in Figure 6.2: the middle picture indicates that the ice slabs connect just after contact, and then bend while creating an enormous friction force.

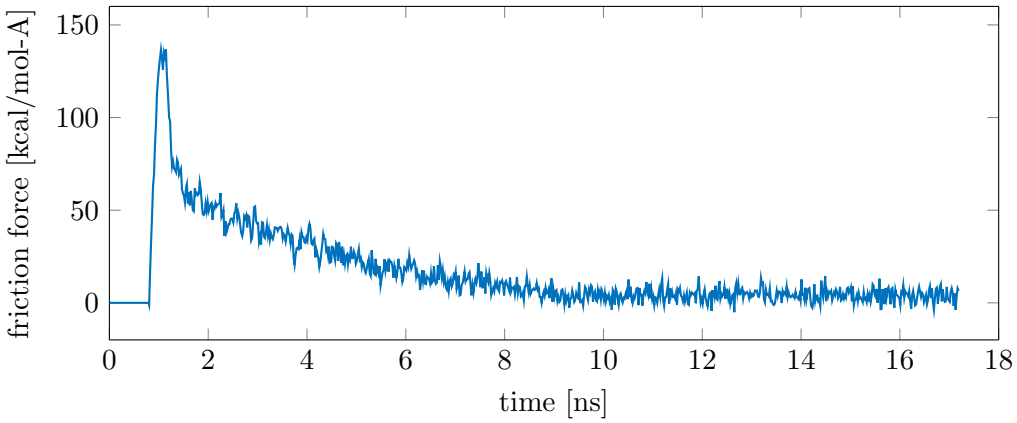


Figure 6.3: *(C) - friction force at 5m/s, 100 kcal/mol-Å normal force*

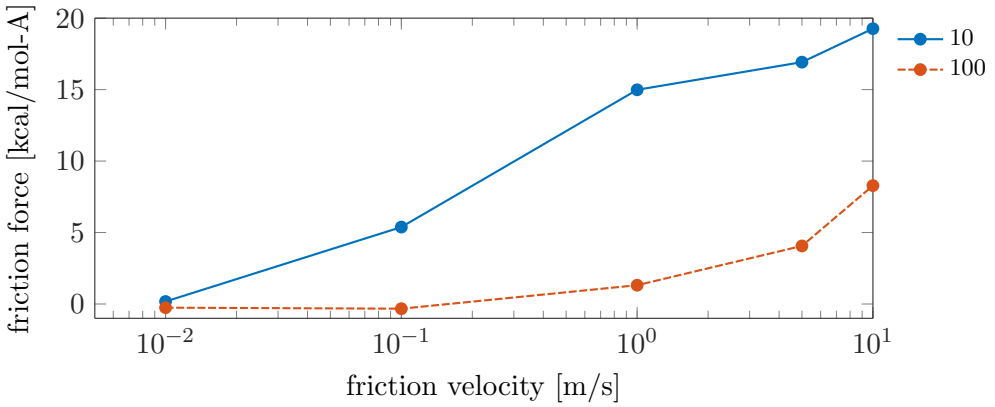


Figure 6.4: *(C) - average final friction force for 10 and 100 kcal/mol-Å normal force and different friction velocities*

Results obtained by varying velocity and normal force are plotted in Figure 6.4. Force values are averaged over the last 4ns of the 17.2ns runs. They seem to indicate two relations: first, the higher the velocity, the larger the friction, and second, higher normal force, as compared to lower normal force, is correlated with significantly lower friction.

6.1.3 Temperature Distribution and Liquid Layer

In the simulation, friction generates heat and the temperature increases near the interface. A liquid layer is formed at the interface which has a higher density. The order parameter σ [100] is an indicator for the structure of ice Ih. It can be computed for each molecule, and is a measure of how close the local structure is to a tetrahedron. This is estimated by computing the angles to its four nearest neighbor molecules. For molecule i with nearest neighbors numbered $1-4$, it reads

$$\sigma_i = 1 - \frac{3}{8}\sum_{j=1}^{3}\sum_{k=j+1}^{4}\left(\cos\theta_{ijk} + \frac{1}{3}\right)^2,$$

where θ_{ijk} denotes the angle between the oxygen atoms in molecule i, j and k.

In a perfect tetrahedral network, $\sigma_i = 1$. If the arrangement of molecules is random, such as in an ideal gas, the mean value of σ_i is zero. As noted in [97], in practice we can expect it to be around 0.95 in ice, and around 0.5 to 0.85 in liquid water.

In Figure 6.5, we plot order parameter and temperature along the z-axis for three different friction velocities. As expected, it indicates that a liquid layer forms in the middle, and also that the temperature is significantly higher at the friction interface than in the exterior of the domain.

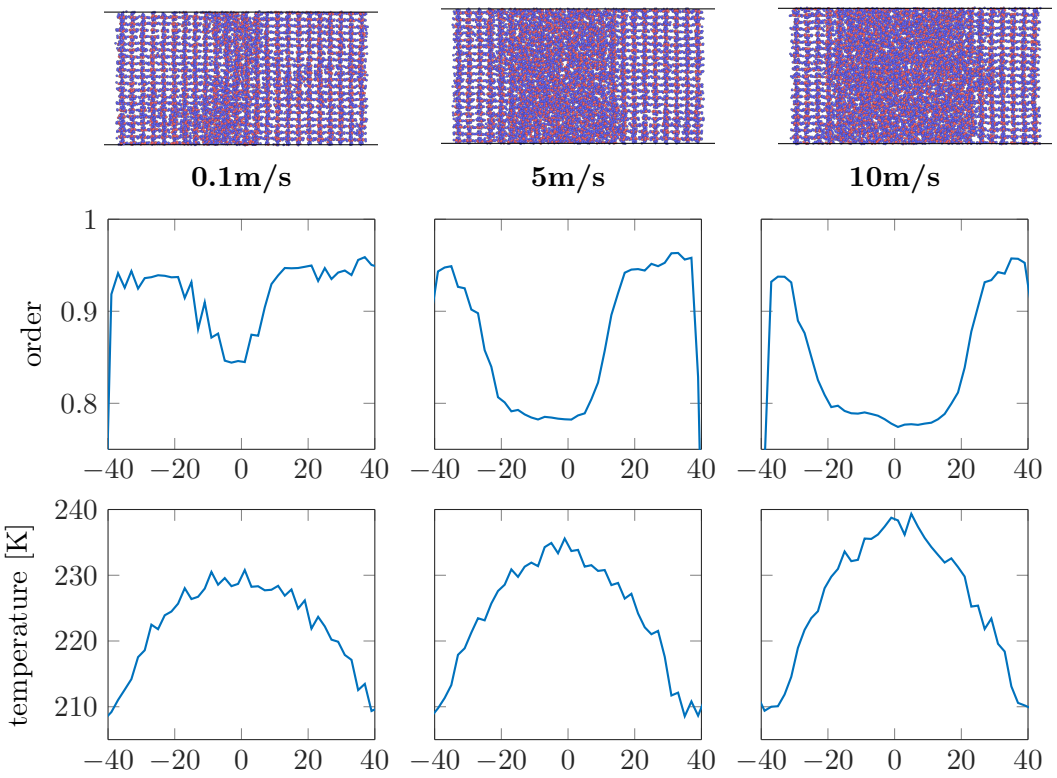


Figure 6.5: *(C) - thickness of melted layer at 10 kcal/mol-Å for different friction velocities, order/temperature [K] vs spatial [Å]*

6.2 (D) - Bovine Pancreatic Trypsin Inhibitor

Bovine pancreatic trypsin inhibitor (BPTI) is a small protein which can be found in bovine lung tissue. Its function is suppression of protein digestion and it acts as a competitive inhibitor. For example, it can be used to reduce bleeding during heart surgery. Well studied since the 70s/80s, it exhibits complex folding pathways.

Structural information was obtained from the RCSB database (ID: 4PTI [101]). BPTI has 892 atoms, and we solvate it in 3165 explicit water molecules (10387 atoms in total). Potentials are modeled according to the CHARMM force field [46]. We use our corotational filter with 163 clusters of size 2, 3263 of 3 and 25 of 4.

In Figure 6.6 the structure of BPTI is plotted. The representation of BPTI in the right figure follows the cartoon style common for proteins, also known as a ribbon diagram. In larger molecules, explicit knowledge of the position of each atom and bond is not of interest. Instead, the underlying structure receives more attention and such plots give information of the local structure of the main backbone chain. Coiled ribbons represent α-helices, arrows indicate β-strands. For more details on the structure of proteins and their schematic representation we refer to [1] and references therein.

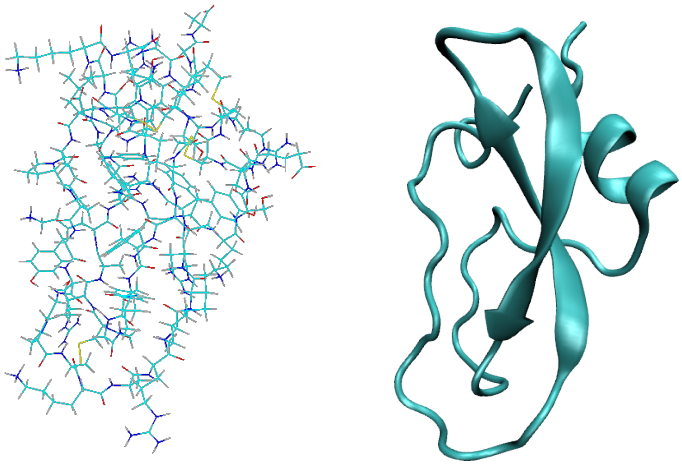


Figure 6.6: *(D) - BPTI. The left image is a full atomistic view of BPTI, the right picture is a schematic representation of the secondary structure as a ribbon diagram*

6.2.1 Simulation Protocol

We equilibrate BPTI to 300°K at 1bar (NPT) for 210ps using a regular IM with an outer step size of 4fs. Afterwards, we switch to the CIM with an outer step size of 8fs and run at 300°K (NVT) for roughly 10 million steps, achieving a simulation time of 80ns. 1 million steps, so 8ns of simulation time required around 6.5h on 64cores. The load distribution is a bit different than the ice-friction simulations: almost 60% time spent on *pair*, 1% on *bond*, 32% on *comm*. Due to less expensive potential in water only around 4% were spent on long-range interactions. Since *pair* usually scales very well, more cores would have helped. However, it was much easier to get a cluster allocation for 64 cores than for 128 or 256 cores.

This is still a very short trajectory, and major transitions happen on larger time scales [53,54]. And yet, we could still observe small changes using the root-mean-square deviation (RMSD) as a reaction coordinate.

6.2.2 RMSD Indicates Two Configurations

The *root-mean-square deviation* is a measure of the average distance between atoms of a given state compared to a reference model. The overall rotation and translation between the two states is removed before the RMSD is computed. Usually, only backbone atoms are included in the computation.

In protein folding, it is common to use the RMSD as a *reaction coordinate* which quantifies the progress of transitioning between different folding states. If a protein does not structurally change and just oscillates more or less around its current configuration, the RMSD does not change significantly. That happens most times during a simulation. However, when a folding process occurs, the RMSD will quickly drift since the new state will not match the reference model anymore.

The RMSD is plotted in Figure 6.7 for backbone atoms of residues 5 to 54 versus time.

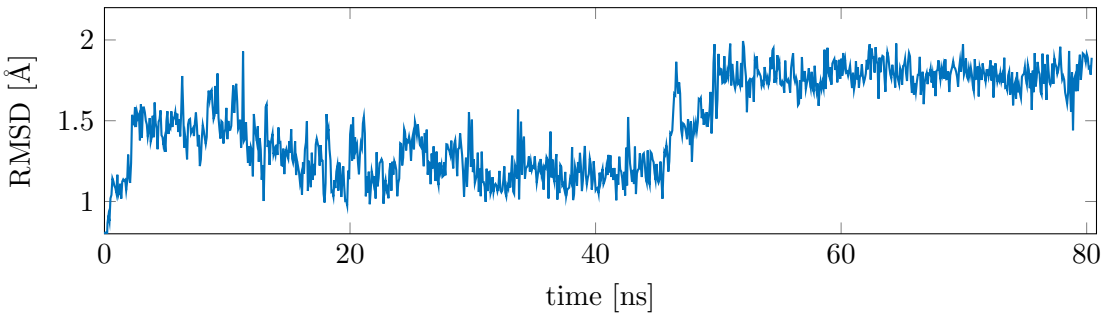


Figure 6.7: *(D) - RMSD evolution for residues 5 to 54, backbone atoms only*

After an initial equilibration, it seems to settle down just above 1Å until around 45ns. It then quickly moves to just below 2Å to find equilibrium again. In Figure 6.8, we plot the initial structure and snapshots after 39ns and 80ns. They indicate the origin of this behaviour: the lower left part of the backbone partially rotates. This is also supported by Figure 6.9, which shows the RMSD for each residue during those time frames. After 39ns, the structure is still relatively unchanged, as compared to the original. It is no surprise that the end tail clearly moves. There is an additional small peak at id 38 which indicates changes near the middle of the protein chain. Towards 80ns, the start tail also unfolds. More interestingly, residues 10 to 15 flip and move away from the center.

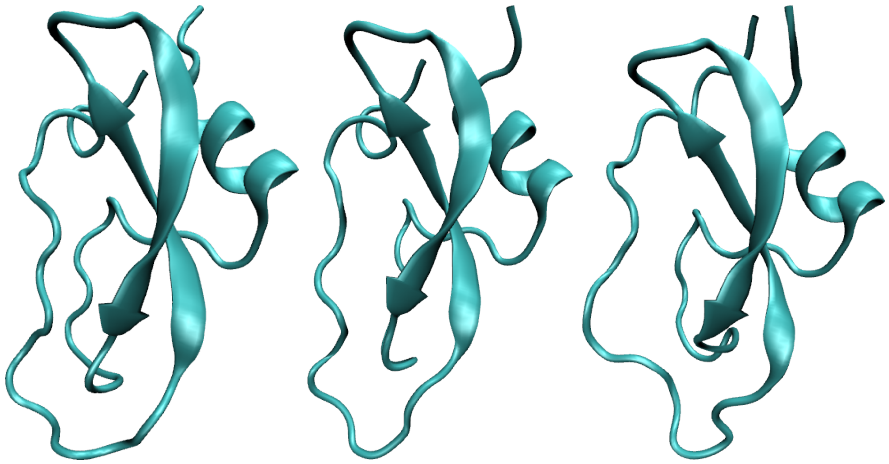


Figure 6.8: *(D) - BPTI at approx. 0ns, 39ns and 80ns*

The obtained results are consistent with [53, 54]. The backbone RMSD there shows similar movements in the folding process. Of course, since the trajectory here is shorter by several orders of magnitude only two and not 5 states were found.

The simulations conducted in examples (C) and (D) show that the mollified impulse method with the corotational filter is a competitive method which can be used for the time integration of everyday tasks in molecular dynamics. The corotational filter is robust and stable in long-term simulations. By removing force contributions which are responsible for severe resonance instabilities, this method can use a twice as large outer time step as a comparable unfiltered impulse method. While examples (A) and (B) show that the method

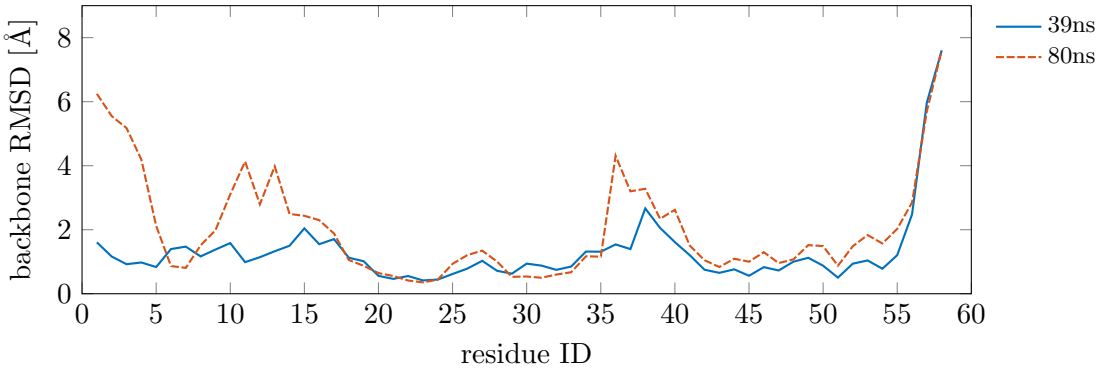


Figure 6.9: *(D) - RMSD per residue averaged over 800ps*

behaves as expected and has favorable computational properties such as scalability, ease in implementation and very low cost, the examples (C) and (D) indicate that the slow energy exchange is not significantly destroyed. In fact, the simulation of the protein BPTI shows a similar folding behaviour as was obtained in other computational studies.

7 Normal Mode Analysis

Example (D) in the previous chapter pointed to some limitations of molecular dynamics simulations. In many biomolecular applications, such as the folding mechanism of large molecules, we are not interested in the highly oscillatory and fast motions. Instead, we are trying to capture motion associated with slow, yet large, conformational moves of a molecule. One way to approach this is normal mode analysis (NMA). Here, the harmonic approximation of the motion near an equilibrium is investigated. This simplification allows us to identify (instantaneous) vibrational frequencies with its corresponding motion [22]. It is an essentially different approach than MD. Whereas in MD a brute force approach through fully resolving and integrating all motion is taken, NMA focuses on the slow modes of a local harmonic approximation.

One of the most important applications of NMA is the prediction of functionally relevant motion of large macromolecules. It is expected that low frequency modes make dominant contributions to conformational fluctuations at thermal equilibrium [102]. See also the nice introduction and critical review [29]. There, the authors call "... the relative success of normal mode analysis ... surprising and intriguing." And, they state that

> [a]lthough some studies found partial overlap between some of the lowest frequency modes and the functional mode determined from two X-ray conformers, in general it would be fanciful to expect anything more than a moderate correspondence to individual normal modes. However, the expectation that there exists a larger subspace spanned by the first M lowest frequency normal modes (where M may be between 10 and 20% of $3N$) that is also spanned by the modes with the largest fluctuation in the real molecule is more realistic. *(quoted from [29])*

7.1 Standard Normal Mode Analysis

Similar to molecular dynamics, we consider a molecular model and refer to the ODE

$$M\ddot{q} = -\nabla_q U(q) \tag{7.1}$$

with a potential $U(q)$ and associated energy

$$H(q,p) = \frac{1}{2}p^T M^{-1} p + U(q). \tag{7.2}$$

However, in contrast to MD, we search for certain motions in a local harmonic approximation. That is, normal mode analysis is based on linearizing

$$\nabla U(q) \approx \nabla U(q_0) + \mathcal{H}(q_0)(q - q_0),$$

where $\mathcal{H}$ is the Hessian matrix of U. Assuming an equilibrium position q_0, the force vector $\nabla U(q_0)$ vanishes, and thus

$$M\ddot{q} + \mathcal{H}(q - q_0) = 0. \tag{7.3}$$

Solutions of (7.3) can be found by solving the equivalent generalized eigenvalue problem

$$\mathcal{H}\, v = \lambda M v. \tag{7.4}$$

Assume (λ, v) is an eigenpair to $(\mathcal{H}, M)$ with $\lambda \in \mathbb{R},\ v \in \mathbb{R}^{3N}$. Then $q(t) = q_0 + \cos(\sqrt{\lambda}t)v$ solves $M\ddot{q} + \mathcal{H}(q - q_0) = 0$.

Therefore, normal mode analysis consists of the following steps:

1. compute an equilibrium position q_0 of a given molecular structure (e.g. some initial position and a force field),
2. for Hamiltonian (7.2) compute a number of eigenpairs (Λ, X) corresponding to the lowest frequencies from the generalized eigenvalue problem $\mathcal{H}\, X = M X \Lambda$, where $\mathcal{H} = \mathcal{H}(q_0) = U''(q_0)$ and M is the mass matrix ,
3. eigenvectors v are so-called normal modes, and the corresponding eigenvalues λ are the squares of vibrational frequencies $\sqrt{\lambda}$.

So, with normal mode analysis, we obtain eigenpairs indicating both the direction and the vibrational frequency of each mode. Such information should not be used without thought: normal mode analysis is based on a harmonic approximation around a single minimum, and all anharmonic effects are neglected. Especially, the modes might not reflect all possible directions/pathways possible [32].

In the following, we will refer to full atomistic normal mode analysis in Cartesian coordinates, that is, M is a diagonal matrix, and when using an appropriate cutoff in the force field, $\mathcal{H}$ is a sparse and symmetric matrix. There are quite a few approaches which reduce the coordinate space or simplify the potential structure (or both). Such strategies include normal mode analysis of coarse grained molecular models [33] and the elastic network model [103]. The reduced problems are of much smaller dimension and complexity, but possess the same structure (M diagonal, $\mathcal{H}$ sparse symmetric). Thus, the algorithms discussed in the following can also be applied to those problems. However, there are a few methods where this is not the case. Most notably, for methods built on employing internal coordinates [104,105] the resulting matrices $M, \mathcal{H}$ are, in general, dense matrices.

7.1.1 Generalized Eigenvalue Problem

The Hamiltonian (7.2) leads to the generalized eigenvalue problem

$$\mathcal{H}\, x = \lambda M x,$$

where $\mathcal{H}$ is the Hessian of the potential function U evaluated at a local minimum q_0. Recall that $\mathcal{H}$ is symmetric, and M is a diagonal matrix. This can be transformed to

$$M^{-1}\, \mathcal{H}\, x = \lambda x$$

which has the advantage of being a 'regular' eigenvalue problem but unfortunately $M^{-1}\, \mathcal{H}$ is, in general, not symmetric. Instead, it is common to transform to mass weighted coordinates $y = M^{1/2}x$ and solve the equivalent problem

$$M^{-1/2}\, \mathcal{H}\, M^{-1/2} y = \lambda y, \qquad \text{with } y = M^{1/2}x.$$

This retains the symmetry of the problem and can be easily computed since M is a diagonal matrix.

It is well known, that a symmetric matrix has an eigendecomposition. Hence, there exists a full orthonormal basis denoted by V with a diagonal matrix Λ containing the corresponding eigenvalues λ_i s.t.

$$M^{-1/2}\, \mathcal{H}\, M^{-1/2} V = V\Lambda.$$

I.e. v_i, the i-th column of V, is an eigenvector corresponding to the eigenvalue λ_i. Note that $VV^T = V^TV = I$. Transforming v_i back to the original coordinates, that is $x_i = M^{-1/2}v_i$, we see that each x_i fulfills $\mathcal{H}\, x_i = \lambda_i M x_i$. However, the x_i are not orthonormal with respect to the standard scalar product. Instead, since

$$x_i^T M x_j = v_i^T v_j = \delta_{ij} \tag{7.5}$$

they are orthonormal with respect to M-weighted scalar product $\langle u, v\rangle_M = u^T M v$.

The core task of normal mode analysis is thus computing a (partial) eigendecomposition of the mass-weighted Hessian. This sounds deceptively easy and straight forward, however, we again face the curse of dimensionality. With increasing dimension of $\mathcal{H}$, a full decomposition quickly becomes unfeasible, and further approximations have to be introduced.

7.1.2 Characterization of Eigenmodes

There is a connection between a molecular model and the resulting eigenvectors from normal mode analysis. The mass-weighted Hessian should have six eigenvalues very close to/or exactly zero. Three eigenvectors of eigenvalue zero correspond to the translation

invariance of radial potentials. The remaining three correspond to a rotation of the entire model. All other eigenvalues should be positive since the Hessian should have been evaluated at a (local) minimum. Negative eigenvalues are a sign of insufficient equilibration during computation of q_0. However, it can be quite challenging to find such a minimum. Positive, small eigenvalues are associated with conformational, large scale moves. Large eigenmodes are usually very localized and associated with highly oscillatory motion.

The interest in slow modes comes from the hypothesis that these modes are responsible for the biggest contributions to large fluctuations within a molecule. And in fact, this claim is backed up by both empirical and theoretical evidence. In [102], a theorem is presented which relates the mass-weighted means-square displacement of a normal mode to the inverse of the corresponding eigenvalue, i.e. (see equation (1) in [102])

$$\sum_{a=1}^{N} m_a \langle \Delta r_a^2 \rangle = k_B T \sum_{j=1}^{3N-6} \frac{1}{\omega_j^2}$$

where m_a and Δr_a are the mass and displacement vector, ω_j is the frequency of the j-th mode, T the absolute temperature and k_B is Boltzmann's constant. $\langle \cdots \rangle$ denotes the time average. Hence, the lowest eigenmodes contribute the most to the overall fluctuation. The influence of eigenmodes with larger eigenfrequencies decays with the square inverse.

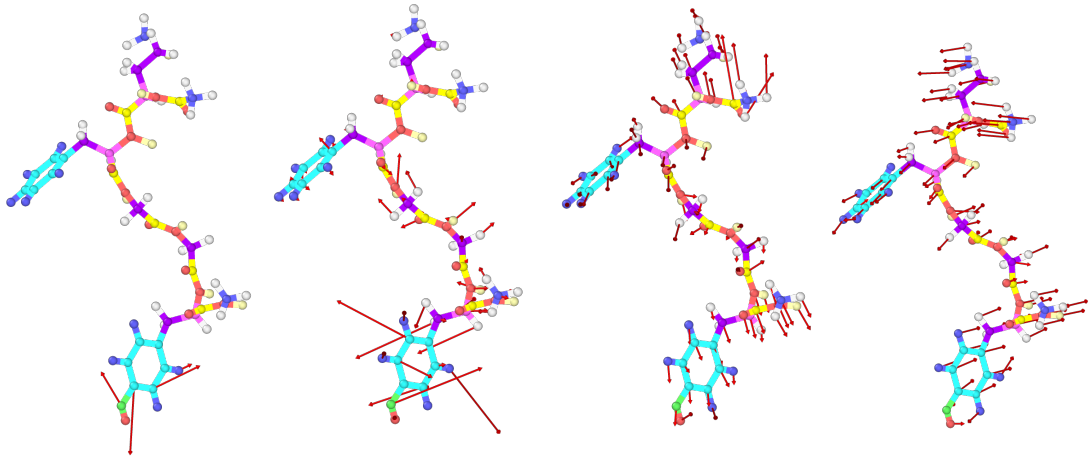


Figure 7.1: *Four normal modes of the small peptide from example (B): the directions of the normal mode are drawn as displacement vectors. The two modes on the left correspond to fast vibrations whereas the two modes on the right are slow modes.*

As an early example we computed the normal modes of the small peptide used as example (B) in Chapter 4. We plot two fast modes on the left and two slow modes on the right in Figure 7.1. Clearly visible is the qualitative difference: the fast modes are highly localized whereas the slow modes correspond to large fluctuations withing the entire molecule.

7.1.3 Approaches in the Literature

In general, due to the large dimension of the problem, a direct solution is not available and instead iterative methods prevail. The approaches in the literature can be categorized into two types: first, an algorithmic approach, and second, a model approach.

In the algorithmic approach, the problem at hand is faced with high end numerics. For example, the normal mode routines in GROMACS, NOMAD and MMTK use the eigensolver from ARPACK, that is an implicitly restarted Arnoldi/Lanczos method. We only found 'explicit' algorithms; that is, no inverse iterations, or shift-and-invert strategies were employed. Instead, the ARPACK is called in a symmetric standard mode where matrix vector products with the (mass-weighted) Hessians are provided by the user. Closely related, in [106] a version of the Block Lanczos [107] is shown to converge rapidly when creating a subspace based on a shift-and-invert technique. Other approaches include the *Component Synthesis Method* (CSM) [39,40] or the *Method of Diagonalization in a Mixed Basis* (DIMB) [42] (DIMB is available in the CHARMM program) where eigenvectors of subblocks of the Hessians are computed and then combined and refined (see also [108–111]).

In the model approach the basic assumption is that the model is too large and provides too much negligible information. Thus, the model is simplified following a certain strategy or set of assumptions. One obtains a much smaller and directly solvable model, or one which is much easier to handle. Examples of such a strategy include coarse grained approximations [33] or the restriction to internal coordinates [104,105,112]. In [38], residues are restricted to rotation and translation vectors, and eigenvectors in the resulting subspace are computed. Both the *Elastic Network Model* (ENM) [103] and the *Gaussian Network Model* (GNM) remove the original force field altogether and replace it with a simple spring model (ENM) or a description of the inter-residue contact topology (GNM).

For further information on the vast amount of methods and literature, we refer the interested reader to one of the following reviews and references therein [29–32,44].

7.2 Numerical Approximations of Eigenpairs

In this section we present numerical algorithms for approximating eigenpairs of the symmetric eigenvalue problem

$$Ax = \lambda x, \qquad \text{with } A = M^{-1/2}\,\mathcal{H}\,M^{-1/2} \in \mathbb{R}^{n\times n},$$

where $A = A^T$ is the mass-weighted Hessian with $n = 3N$.

For large eigenvalue problems, iterative methods attempt to approximate a subset of eigenpairs. In normal mode analysis, the subset of interest usually consists of a fixed number of the slowest modes. Iterative algorithms face restrictions from multiple directions due to the large scale nature of the problem. Of course, the runtime on a computer or compute cluster until an acceptable approximation has been reached is of considerable interest. The best we could hope for would be a complexity which is linear in the dimension

n, that is, a complexity of $\mathcal{O}(n)$. However, a sparse matrix-vector multiplication is already in $\mathcal{O}(n)$. Thus, if the main computational task of an algorithm consists in computing matrix-vector multiplications, then the number of steps required until convergence needs to be independent of size of the matrix. The computing time is, to some extent, also connected to the question of how well the computational task can be computed in parallel. Storage can become a limiting factor. Storage is required not only for the sparse large matrix itself but also for a search space and for the approximate eigenvectors. Whenever dealing with large scale problems, some concern should be given to the general stability of an algorithm and the effect of roundoff errors. A very common issue is, for example, the loss of orthogonality in a Lanczos process, as we will see later.

7.2.1 Rayleigh-Ritz Method

We follow Chapter 2.2 in [113]. The standard approach for approximating eigenpairs of a large and sparse matrix $A \in \mathbb{R}^{n\times n}$ is the *Rayleigh-Ritz method*.

Definition 17. *Given a subspace $\mathcal{S} \subseteq \mathbb{R}^n$, $\theta_k \in \mathbb{R}$ is a **Ritz value** of A with respect to $\mathcal{S}$ with **Ritz vector** u_k if (θ_k, u_k) satisfies the **Galerkin condition***

$$u_k \in \mathcal{S}, \quad u_k \neq 0, \qquad Au_k - \theta_k u_k \perp \mathcal{S}. \tag{7.6}$$

*The pair (θ_k, u_k) is called a **Rayleigh-Ritz approximation**.*

Exact eigenpairs (also called *Ritz pairs*) of the projection of A onto the subspace $\mathcal{S}$ serve as approximations to the eigenpairs of A.

Theorem 18. *Let $\mathcal{S} \subseteq \mathbb{R}^n$ be an m-dimensional subspace and $V \in \mathbb{R}^{n\times m}$ be any matrix such that $\mathcal{R}(V) = \mathcal{S}$. Then (θ_k, u_k) is a Rayleigh-Ritz approximation if and only if θ_k is an eigenvalue of*

$$H = (V^T V)^{-1} V^T A V \tag{7.7}$$

and $u_k = V y_k$, where y_k is an eigenvector of H corresponding to θ_k.

Note that if the columns of V are orthonormal, then $V^T V = I_m$, and the problem reduces to computing eigenpairs of the small matrix $H = V^T A V$. If the subspace is suitably chosen, and of much smaller dimension than the original problem, useful approximations at low cost can be obtained. Usually, this strategy is employed in an iterative manner. In order to improve the approximation, the subspace is gradually extended by some heuristic.

This method is known to approximate outer eigenpairs much faster than inner ones. However, iterating with A^{-1} has a much higher associated cost. The concept of *harmonic Ritz values* offers an alternative.

Definition 19. *Given a subspace $S \subseteq \mathbb{R}^n$, $\theta_k \neq 0$ is a **harmonic Ritz value** of A with respect to S if θ_k^{-1} is a Ritz value of A^{-1} with respect to S.*

Surprisingly, the actual usage of A^{-1} can be avoided.

Theorem 20. *Let $\mathcal{V} \subseteq \mathbb{R}^n$ be a subspace with dim $\mathcal{V} = m$, and orthonormal basis $V \in \mathbb{R}^{n\times m}$. Assume $\mathcal{W} = A\mathcal{V}$ has dimension m. Then θ_k is a harmonic Ritz value of A with respect to $\mathcal{W}$ if and only if*

$$Au_k - \theta_k u_k \perp \mathcal{W} \qquad \text{for some} \qquad u_k \in \mathcal{V}, \quad u_k \neq 0. \tag{7.8}$$

Moreover, if $W \in \mathbb{R}^{n,m}$ is a matrix with $\mathcal{R}(W) = \mathcal{W}$, then (7.8) is equivalent to

$$(W^T AV)y_k = \theta_k (W^T V)y_k, \qquad u_k = Vy_k. \tag{7.9}$$

The pair (θ_k, u_k) is called a **harmonic Rayleigh-Ritz approximation**.

Corollary 21. *If (θ_k, u_k) is a harmonic Rayleigh-Ritz approximation of A, and $w_k = Au_k$, then*

$$\theta_k^{-1} = \frac{w_k^T A^{-1} w_k}{w_k^T w_k} = \frac{u_k^T A u_k}{u_k^T A^T A u_k} \tag{7.10}$$

is the Rayleigh quotient of Au_k with respect to A^{-1}.

Both Rayleigh-Ritz approximations have in common that the large matrix A is projected onto a subspace of much smaller dimension. They differ in how the information is processed, though. Harmonic Ritz values are known to give better approximations of interior eigenvalues.

7.2.2 Lanczos Method

In the following, we refer to the Lanczos method as the (symmetric) Lanczos process for the construction of the Krylov basis combined with a Rayleigh-Ritz method.

Definition 22. *For $A \in \mathbb{R}^{n\times n}$ and $0 \neq v \in \mathbb{R}^n$*

$$\mathcal{K}_m(A, v) = \text{span}\{v, Av, A^2 v, ..., A^{m-1} v\} \tag{7.11}$$

*is called the mth **Krylov subspace** with respect to A and v.*

It directly follows that for $x \in \mathcal{K}_m(A, v)$, we have $x = p(A)v$ where $p \in \mathbb{P}_{m-1}$ denotes a polynomial of degree at most $m-1$. Therefore, the m-th Krylov space is equivalent to the space of all polynomials in A of degree at most $m-1$ applied to v:

$$\mathcal{K}_m(A, v) = \{p(A)v \mid p \in \mathbb{P}_{m-1}\}.$$

Furthermore, a Krylov space is invariant to shifts of the matrix A and it holds

$$\mathcal{K}_m(A, v) = \mathcal{K}_m(A + \sigma I, v), \qquad \forall \sigma \in \mathbb{R}\,.$$

When dealing with a symmetric matrix, a basis of a Krylov space can be computed by the Lanczos iteration. It uses a modified Gram–Schmidt process to produce an orthonormal basis $V_m = [v_1, ..., v_m]$ of $\mathcal{K}_m$. This can be written as a recurrence of the form

$$AV_m = V_{m+1}\tilde{H}_m = V_m H_m + h_{m+1,m} v_{m+1} e_m^T \tag{7.12}$$

where $V_m \in \mathbb{R}^{n\times m}$ has orthonormal columns, $V_m^T v_{m+1} = 0$ and $H_m \in \mathbb{R}^{m\times m}$ is symmetric and tridiagonal. The Lanczos iteration with start vector v_1 can also be written as a simple three-term recursion ($v_0 = 0, \beta_1 = 0$):

$$v_{m+1} = w_{m+1}/\beta_{m+1}, \quad w_{m+1} = Av_m - \alpha_m v_m - \beta_m v_{m-1}, \qquad m > 0 \tag{7.13}$$

$$\text{with } \alpha_m = v_m^T A v_m, \quad \beta_m = ||w_m||. \tag{7.14}$$

The coefficients α_i and β_i form the diagonal and superdiagonal entries of the tridiagonal matrix H_m. Finally, the eigenpairs (θ_k, y_k) of H_m are used to compute the Ritz pairs $(\theta_k, u_k = V_m y_k)$ of A.

The following lemma allows for easy error estimation during the Lanczos iteration.

Lemma 23. *Let $(\theta_k, u_k), u_k = V_m y_k$ be Rayleigh-Ritz approximations of A corresponding to $\mathcal{K}_m$. Then the residual is given by*

$$Au_k - \theta_k u_k = h_{m+1,m}(e_m^T y_k) v_{m+1} \tag{7.15}$$

and thus $||Au_k - \theta_k u_k|| = h_{m+1,m}|e_m^T y_k|$.

Altogether, for a symmetric matrix A we obtain the resulting algorithm (Algorithm 7.1). The Lanczos method is known to approximate exterior eigenvalues quite quickly and well. Its main advantage is the efficiency of the algorithm.

Algorithm 7.1: Lanczos method with symmetric matrix A, start vector v, see [114, 115]

```
r = v, β_1 = ||v||, v_0 = 0
for k = 1, 2, ... until convergence do
    v_k = r/β_k
    r = A v_k
    r = r − β_k v_{k−1}
    α_k = v_k^T r
    r = r − α_k v_k
    β_{k+1} = ||r||_2
    compute approximate eigenvalues H_k = Y Θ^(k) Y^T
    test for convergence
compute approximate eigenvectors U = V_k Y
```

More specifically, assume $\lambda_1 \geq ... \geq \lambda_n$ are the exact eigenvalues of A, then convergence

theorems of the Lanczos method [116, Thm. 10.1.2.] show that the Ritz value θ_1 (largest eigenvalue of H_m) is bounded by

$$\lambda_1 \geq \theta_1 \geq \lambda_1 - (\lambda_1 - \lambda_n)\left(\frac{\tan\phi_1}{c_{m-1}(1+2\rho_1)}\right)^2, \qquad \text{with} \quad \rho_1 = \frac{\lambda_1 - \lambda_2}{\lambda_2 - \lambda_n}.$$

Here, c_m denotes the m-th Chebyshev polynomial and $\cos(\phi_1) = |q_1^T z_1|$ where z_1 is the start vector of the Lanczos method and q_1 the exact eigenvector to λ_1. Hence, the larger ρ_1, the quicker the convergence. It is clear that ρ_1 is large for a well separated eigenvalue. A similar result is obtained for λ_n by applying the bounds to $-A$.

Convergence of the Lanczos method can thus be accelerated if the original problem can be manipulated in such a way that the eigenvalues of interest are well separated, exterior eigenvalues of the transformed problem. To this end, *polynomial filtering* can be used. The basic idea is based on the fact that for a polynomial p, $p(A)$ has the same eigenvectors as the original A and eigenvalues $p(\lambda_i)$. However, the approximations are still in the same Krylov subspace as when iterating with A, and it is not clear if the approximations can thus be improved. More of this technique's advantages and problems can be found in the literature [116, 117].

For computing harmonic Ritz values, the Arnoldi iteration (7.12) is also useful. Using a Krylov basis V_m, the generalized eigenvalue problem reads

$$(W_m^T A V_m) y_k = \theta_k (W_m^T V_m) y_k. \tag{7.16}$$

Since $W_m = AV_m$ this is equivalent to

$$(\tilde{H}_m^T \tilde{H}_m) y_k = \theta_k H_m^T y_k. \tag{7.17}$$

However, this should not be solved directly, and the computation of $\tilde{H}_m^T \tilde{H}_m$ should be avoided. Instead, a QR factorization of $\tilde{H}_m$ avoids stability problems. This way only the condition number of $\tilde{H}_m$ enters, not its square [113, Ch. 2.4].

7.2.3 Shift-and-Invert Iteration

An appealing approach for computing slow modes is inverse iteration. The regular Lanczos iteration approximates large and exterior eigenvalues very quickly. We can exploit this property by iterating with $\tilde{A} = (A - \sigma I)^{-1}$ instead of A. This corresponds to a spectral transformation since the eigenvalue problem $Ax = \lambda x$ is transformed to

$$\tilde{A}x = (A - \sigma I)^{-1} x = \frac{1}{\lambda - \sigma} x.$$

The transformed eigenvalue problem has the same eigenvectors, but eigenvalues $\mu = 1/(\lambda - \sigma)$. However, the largest eigenvalues of $\tilde{A}$ are now the eigenvalues of A which are closest to the shift σ. Hence, we can expect quick convergence of eigenpairs close to the shift σ.

Using $\tilde{A}$ in the Lanczos method is called the shift-and-invert Lanczos method, here, we will simply call it *shift-and-invert iteration*.

Obviously, the matrix $\tilde{A}$ is not computed explicitly. Instead, the resulting linear system has to be solved in each step of the Lanczos method. While this algorithm promises less iteration steps (and with it less memory for storing the basis vectors), the bottleneck will be solving the linear system. If we find a quick and efficient way to solve sparse linear systems with the mass weighted Hessian, this method has great potential.

Usually, in each step of the shift-and-invert iteration, the linear system will be solved itself by an iterative method. Such a concept is known as *inner-outer iteration*. The outer iteration is the application of the subspace method. For each outer step, an inner iteration is performed to (inexactly) solve the linear problem.

In contrast to the Lanczos iteration, the basis vectors of the Krylov space with $(A - \sigma I)^{-1}$ are obtained with an (inexact) iterative method, and therefore do not build a Krylov space anymore. However, it has been observed by many numerical investigations that, despite using inexact solutions, the methods are still reliable. Even with a low tolerance in the solver, convergence can still be achieved [118–121].

7.2.4 The Implicitly Restarted Arnoldi Method

The implicitly restarted Arnoldi method [122,123] (IRAM), or when applied to a symmetric problem, the implicitly restarted Lanczos method for sparse systems addresses some numerical difficulties encountered in the regular Arnoldi or Lanczos iteration.

One major disadvantage of the Arnoldi method is that with an increasing number of steps, more and more computer memory is required to store the basis vectors V_m. For large problems, since $V_m \in \mathbb{R}^{n\times m}$ is dense, this is a severe restriction. This is not directly true for the Lanczos iteration. It is based on a three-term recursion and does not need to store the full basis V_m. However, we are interested in approximations to the eigenvectors and thus either need to store the full basis V_m or compute a second run.

In the IRAM, an efficient restarting process is implemented. Whenever a certain number of basis vectors, let us say m vectors, is reached, p unwanted directions are identified and then discarded. This is accomplished by an implicitly shifted QR algorithm which compresses the Arnoldi factorization of length m into a factorization of length $k = m - p$. The reduced factorization can then be extended by another p steps to again yield a length m factorization. Thus, the Arnoldi space is limited to a fixed number of vectors.

Assume we have arrived at an Arnoldi factorization of length m with

$$AV_m = V_m H_m + f_m e_m^T, \qquad \text{with} \qquad f_m = h_{m+1,m} v_{m+1}.$$

Then p shifts $\nu_1, \ldots, \nu_p$ are selected, which usually correspond to unwanted eigenvalues of

H_m. The associated directions are removed by computing a QR decomposition of

$$\prod_{i=1}^{p}(H_m - \nu_i I) = Q_m R_m$$

and then setting

$$AV_m Q_m = V_m Q_m Q_m^T H_m Q_m + f_m e_m^T Q_m.$$

Denoting $Q_m^T H_m Q_m = H_m^+$ and $V_m^+ = V_m Q_m$ we obtain

$$AV_m^+ = V_m^+ H_m^+ + f_m e_m^T Q_m.$$

It can be shown that the first $k-1$ components of $e_m^T Q_m$ are zero, and thus, by truncating after the k-th column an Arnoldi decomposition of order k is obtained

$$AV_k^+ = V_k^+ H_k^+ + f_k^+ e_k^T, \qquad \text{with} \qquad f_k^+ = h_{k+1,k}^+ V_m^+ e_{k+1} + f_m q_{m,k}.$$

Here, $h_{k+1,k}^+$ refers to the $(k+1,k)$-th entry of H_m^+, and $q_{m,k}$ is the (m,k)-th entry of Q_m. The crucial point is now that V_k^+ and H_k^+ are the matrices which would have been obtained by running k steps of the original Arnoldi process, but with a modified starting vector

$$v_1^+ = \frac{1}{c}\prod_{i=1}^{p}(A - \nu_i I)v_1,$$

where c is some normalization factor. Hence, restarting corresponds to applying a polynomial filter to the initial vector v_1 in a very efficient way without applying the filter in each step.

The resulting algorithm is summarized in Algorithm 7.2. Using such an algorithm is a complex task. The performance is influenced by many parameters (e.g. number of basis vectors or search space, starting vector, selection criterion, tolerance).

Obviously, this method can be 'upgraded' with shift-and-invert ideas if the linear systems can be solved at a reasonable cost. Actually, the implementation in ARPACK [124] uses a reverse communication interface, that is, the program returns the control to the user whenever interaction with the matrix A is required. Depending on the mode (standard, inverse, or shift-and-invert), the user then returns the required information (e.g. a matrix-vector multiplication or the solution of a linear system).

GROMACS built-in normal mode tools use LAPACK for the dense eigensolver, and ARPACKs implicitly restarted Arnoldi method in standard mode for sparse systems.

7.2.5 Jacobi-Davidson Method

The Jacobi-Davidson method [125] was developed in the '90s. It can be seen as an improvement of Davidson's method, which is a cheap version of the shift-and-invert method

Algorithm 7.2: Implicitly restarted Arnoldi method in ARPACK (see [124])

- *Start:* Build a length m Arnoldi factorization $AV_m = V_m H_m + f_m e_m^T$ with the starting vector v_1.
- *Iteration:* Until convergence
  1. Compute the eigenvalues $\{\lambda_j : j = 1, 2, ..., m\}$ of H_m. Sort these eigenvalues according to the user selection criterion into a wanted set $\{\lambda_j : j = 1, 2, ..., k\}$ and an unwanted set $\{\lambda_j : j = k+1, k+2, ..., m\}$.
  2. Perform $m - k = p$ steps of the QR iteration with the unwanted eigenvalues $\{\lambda_j : j = k+1, k+2, ..., m\}$ as shifts to obtain $H_m Q_m = Q_m H_m^+$.
  3. *Restart:* Postmultiply the length m Arnoldi factorization with the matrix Q_k consisting of the leading k columns of Q_m to obtain the length k Arnoldi factorization $AV_m Q_k = V_m Q_k H_k^+ + f_k^+ e_k^T$, where H_k^+ is the leading principal submatrix of order k for H_m^+. Set $V_k \leftarrow V_m Q_k$.
  4. Extend the length k Arnoldi factorization to a length m factorization.

where the occurring linear systems are solved very approximately.

A weak point of the Lanczos method is that the Krylov space does not use currently available information to improve the subspace. The Jacobi-Davidson method uses a different set of subspaces, and then employs the Rayleigh-Ritz method. The expansion of the subspace is more elaborate than a simple matrix-vector multiplication. A suitable direction is identified by (approximately) solving a linear system.

The idea is as follows. Assume we want to expand the current space $\mathcal{S}_m$ by a vector $t_m \perp \mathcal{S}_m$. If we already had a Rayleigh-Ritz approximation (θ, u), then the optimal choice of t would be

$$A(u+t) = \lambda(u+t) \qquad \text{i.e.} \qquad (A - \lambda I)t = -(Au - \lambda u). \tag{7.18}$$

However, t can not be computed since λ is not known. Hence it is replaced by the best available approximation, namely θ. Also, since t is used to expand the search space, t is restricted to be orthogonal to u. The resulting linear system reads (where $r = Au - \theta u$)

$$(I - uu^T)(A - \theta I)(I - uu^T)t = -r, \qquad t \perp u. \tag{7.19}$$

This equation is called the *Jacobi-Davidson correction equation.* It is solved approximately, and its approximate solution is taken for the expansion of the subspace. This is the main and fundamental difference to Krylov subspace methods. The Jacobi-Davidson method has no Krylov space structure (e.g. powers of the matrix or a shifted inverse applied to a starting vector). Instead, the inverse operator is applied to the residual vector r of a current guess. The correction equation is approximately solved by some steps of a usually preconditioned iterative method, e.g. see Section 7.3.2.

Algorithm 7.3: Basic Jacobi-Davidson method for $\lambda_{max}(A)$ for Hermitian eigenvalue problem [114, 126]

start with $t = v_0$, starting guess
for $m = 1, 2, \ldots$ **do**
  for $i = 1, \ldots, m-1$ **do**
    $t = t - (v_i^T t) v_i$
  $v_m = t/||t||_2, v_m^A = A v_m$
  for $i = 1, \ldots, m$ **do**
    $M_{i,m} = v_i^T v_m^A$
  compute the largest eigenpair (θ, s) of M $(||s||_2 = 1)$
  $u = Vs$ // $V \in \mathbb{R}^{n \times m}$ with columns v_j
  $r = Au - \theta u$
  if $||r||_2 \leq \epsilon$ **then**
    $\tilde{\lambda} = \theta, \tilde{x} = u$
    stop
  solve (approximately) a $t \perp u$ from
  $(I - uu^T)(A - \theta I)(I - uu^T)t = -r$
$(\tilde{\lambda}, \tilde{x})$ is approximation to largest eigenvalue $\lambda_{\max}$ with corresponding eigenvector

Algorithm 7.3 is a very simple version of the Jacobi-Davidson method for the computation of a single exterior eigenpair. If restart and deflation techniques are added (see [127]), then this algorithm is called JDQR, see also [128] for computing interior eigenvalues. Since the Jacobi-Davidson method involves solving a linear system, this method can be easily preconditioned. Also, since t is 'only' used for expanding the space, a very accurate solution of the linear system is not necessarily required.

We refer to [125, 129] for more details. In [130], the connection between the Jacobi-Davidson method and inexact Newton iterations is shown. Convergence proofs for the Hermitian eigenvalue problem are given in [131].

7.2.6 Contour Integral Methods

Contour integral methods for computing eigenpairs are a combination of approximating a contour integral and a Rayleigh-Ritz approach (CIRR method [132, 133]). These methods are a relatively new development (e.g. FEAST algorithm [134, 135] from around 2009). Contour integration techniques were mainly developed for quantum mechanics. They are built around a Rayleigh-Ritz approach; however, the subspace is chosen in a very different manner. As the author puts it in [134], "[t]he technique deviates fundamentally from the [...] traditional techniques".

In our notation, for real, symmetric A and resolvent $G(z) = (zI - A)^{-1}$ a contour

integral is computed

$$XX^T = -\frac{1}{2\pi i}\int_{\mathcal{C}} G(z)\ dz \tag{7.20}$$

where $X \in \mathbb{R}^{n\times m}$ contains m eigenvectors to the m eigenvalues inside the contour $\mathcal{C}$. Postmultiplying by an arbitrary matrix $Y \in \mathbb{R}^{n,m}$ with linear independent columns yields

$$X(X^TY) = -\frac{1}{2\pi i}\int_{\mathcal{C}} G(z)Y\ dz \tag{7.21}$$

$$\Rightarrow \qquad XW = -\frac{1}{2\pi i}\int_{\mathcal{C}} G(z)Y\ dz \tag{7.22}$$

where $W \in \mathbb{R}^{m\times m}$, and thus each column of XW is a linear combination of eigenvectors in X. XW is now used in a Rayleigh-Ritz procedure to obtain the eigenpairs inside the contour $\mathcal{C}$. The integral is approximated by numerical integration. The original paper [134] uses an N_e-point Gauss-Legendre quadrature on the positive half circle. We denote the quadrature nodes and weights by $(x_e, \omega_e), e = 1, ..., N_e$. A basic version of this algorithm for a search interval $[\lambda_{\min}, \lambda_{\max}]$ is shown in Algorithm 7.4.

Algorithm 7.4: Basic FEAST algorithm (see [134])

1 Select $M_0 > M$ random vectors in $Y \in \mathbb{R}^{n\times m_0}$
2 **while** *not converged* **do**
3 Set $Q = 0$ with $Q \in \mathbb{R}^{n,m_0}$, $r = (\lambda_{max} - \lambda_{min})/2$
4 **for** $e = 1, \ldots, N_e$ **do**
5 compute $\theta_e = -(\pi/2)(x_e - 1)$
6 compute $Z_e = (\lambda_{max} + \lambda_{min})/2 + r\exp(i\theta_e)$
7 solve $(Z_eI - A)Q_e = Y$ to obtain $Q_e \in \mathbb{C}^{n\times m_0}$
8 compute $Q = Q - (\omega_e/2)\Re(r\exp(i\theta_e))Q_e$ `// ℜ denotes the real part`
9 Form $A_Q = Q^TAQ$
10 Solve $A_Q\Phi = \lambda\Phi$ to obtain the M_0 eigenvalues λ and eigenvectors Φ
11 If $\lambda_i \in [\lambda_{min}, \lambda_{max}]$, λ_i is an eigenvalue with eigenvector in the i-th column of $X = Q\Phi$
12 Check convergence for the trace of eigenvalues. If iterative refinement is needed, set $Y = X$

The performance of this approach depends vastly on the ability to solve the occurring linear systems with $(zI - A)$. If direct solvers are not available, iterative solvers with modest accuracy can be employed. Furthermore, this method has potential for parallelism. For example, in the inner **for**-loop in Algorithm 7.4, each iteration is independent and thus can be computed in parallel. Then, each occurring linear system in line 7 has to be solved for multiple right-hand sides. Depending on the solver, this can either be computed in parallel, or with significant savings by reusing information from a previous step.

Efficient implementations of this algorithm are available. A blackbox version of this

algorithm has been implemented in the high-end Intel MKL library. As a linear solver it uses the PARDISO direct solver [136–138].

7.3 Efficient Approximation of Shifted Linear Systems

All previous methods (except the regular Lanczos method) rely heavily on being able to solve a linear system $(A - \sigma I)x = b$ with $A = M^{-1/2}\,\mathcal{H}\,M^{-1/2}$ and shift σ, which constitutes the main computational task.

The matrix A results from a numerical computation, and in theory, since it is the Hessian evaluated at a local minimum, it should be symmetric, semi-positive definite with exactly six zero eigenvalues with eigenvectors corresponding to translation and rotation of the entire domain. However, due to rounding errors on large computations, or simply the fact that the matrix was evaluated very close to (but not exactly at) the minimum, a few negative eigenvalues of small magnitude might be present. The shift σ is usually required to be close to the eigenvalues of interest (e.g. σ small, positive), and hence the resulting linear systems are symmetric and indefinite. For simplicity of notation, in this section we present algorithms for solving the linear system

$$Ax = b, \qquad \text{with} \qquad A = M^{-1/2}\,\mathcal{H}\,M^{-1/2} - \sigma I.$$

Thus, A is a symmetric and indefinite matrix.

7.3.1 Direct Methods

Direct methods should not be ruled out without further consideration. Since the eigensolvers usually require multiple solutions of the same linear system, some initial effort can pay off.

The standard approach for a direct method for sparse matrices is to precompute some triangular factorization, and then solve each system with a backward and forward triangular solve. Since we are dealing with sparse systems, one would hope that this sparsity could also be present in the triangular factorization. This is in general not easy to achieve. For symmetric, banded matrices, though, it is well known that an LDL^T decomposition will keep the band structure in its triangular factors. Here, L is a lower triangular matrix, and D is diagonal. Fill-in in L only happens within the bandwidth of the original matrix A.

Therefore, a common strategy is the following: first, the sparse matrix is reordered to reduce the bandwidth and limit the fill-in. Reordering algorithms such as the reverse Cuthill-McKee algorithm [139,140] or the symmetric approximate minimum degree permutation [141,142] can be used for this task. For a recent survey on reducing the bandwidth of a matrix, see [143]. Then, the sparse LDL^T factors are computed and used for repeatedly solving the linear system with multiple right hand sides through forward and backward solves with the LDL^T factors.

We want to mention that the LDL^T decomposition necessarily needs to include a pivoting strategy for symmetric indefinite problems. For stability, it is also advisable to use a more general form of D, where D is block diagonal with blocks of sizes 1 or 2. For reasons and details we refer to Chapter 4.4 in [116] and references therein.

The following table lists the approximate computational cost for dense and symmetric banded matrices with upper and lower bandwidth p (see [116], Chapter 4).

#	dense matrix	band matrix
compute LDL^T	$n^3/3$	np^2
solve LDL^T	twice $n^2/2$	twice $2np$

Table 7.2: *Computational cost of an LDL^T decomposition for dense and banded matrices*

While quick in expected computational time (each solve is technically in $\mathcal{O}(n)$), there are two expected limitations: first, due to the unavoidable fill-in, direct methods based on a decomposition are quite memory consuming, and second, they require the shifted Hessian to be sorted to a banded matrix with minimal bandwidth. However, this might not be possible, or the bandwidth might be large depending on the molecular structure. If there are no loops etc, the Hessian can be reordered to a band structure, and the fill-in stays bounded.

Unfortunately, as we will see in the next chapter, for most molecules the Hessian does not sort well, and the fill-in is significant, and is often in an order which makes a factorization unfeasible. In such situations, an iterative solver may be employed.

7.3.2 Iterative Methods

Iterative solvers are commonly used when solving large and sparse linear systems. They are usually based on matrix-vector multiplications with the original sparse matrix, and thus exploit the sparsity to their advantage. The topic itself can easily fill multiple monographs on its own [144–148]. Here, however, we are interested in specific iterative methods which are able to solve sparse symmetric indefinite systems. We will highlight two Krylov subspace methods, but we note that many more methods exist.

Krylov subspace methods are iterative methods where the approximations x_m are chosen such that $x_m \in x_0 + \mathcal{K}_m(A, r_0)$ where x_0 is an initial approximation to $Ax = b$ and $r_0 = b - Ax_0$ is the initial residual. We then have

$$x_m = x_0 + p_{m-1}(A)r_0,$$

where $p_{m-1} \in \mathbb{P}_{m-1}$ denotes a polynomial of degree at most $m-1$. For ease in notation we will assume $x_0 = 0$ and thus have $x_m \in \mathcal{K}_m(A, b)$. Note that this is not a restriction, since for a given start vector x_0, we have $A(x - x_0) = b - Ax_0$ which is of the desired form.

There are a number of methods available for symmetric indefinite problems. First, the

MINRES method [149], which is based on the *minimal residual* condition

$$||r_m|| = ||b - Ax_m|| = \min_{x \in \mathcal{K}_m} ||b - Ax||,$$

provides the m-th approximation as

$$x_m = \operatorname*{argmin}_{x \in \mathcal{K}_m} ||b - Ax||.$$

Let V_m be a basis of $\mathcal{K}_m$ obtained from the Lanczos process and $AV_m = V_{m+1}\tilde{H}_m$. Then $x_m = V_m y_m$ for some $y_m \in \mathbb{R}^m$ and with $\beta = ||b||$ we have

$$\begin{aligned} r_m &= b - Ax_m = b - AV_m y_m \\ &= V_{m+1}(\beta e_1 - \tilde{H}_m y_m). \end{aligned}$$

Thus, by minimizing

$$||r_m|| = ||\beta e_1 - \tilde{H}_m y||, \qquad y \in \mathbb{R}^m,$$

the iterate $x_m = V_m y_m$ is obtained from solving an $(m+1) \times m$ dimensional least squares problem (recall V_{m+1} is orthogonal).

Hence, the MINRES is basically the GMRES method applied to a symmetric matrix. Instead of the Arnoldi process in GMRES, the Lanczos iteration generates a basis of $\mathcal{K}_m$. The obtained algorithm can be expressed by a short three term recurrence without actually having to store the entire basis of $\mathcal{K}_m$. Instead, only two vectors are required. For details of this algorithms, see for example [145, 146].

The MINRES yields a general convergence result [144, Prop. 6.32] (since A is diagonalizable with eigenvalues $\lambda_i,\ i = 1, \ldots, n$)

$$||r_m||_2 \leq \min_{p \in P_m, p(0)=1} \max_{i=1,\ldots,n} |p(\lambda_i)|\ ||r_0||_2.$$

If A is symmetric indefinite with eigenvalues contained in $I^- \cup I^+ = [a, b] \cup [c, d]$ with $a \leq b < 0 < c \leq d$ the following error bound is obtained [145]

$$\begin{aligned} \min_{p \in \mathbb{P}_m, p(0)=1} \max_{i=1,\ldots,n} |p(\lambda_i)| &\leq \min_{p \in \mathbb{P}_m, p(0)=1} \max_{z \in I^- \cup I^+} |p(z)| \\ &\leq 2 \left(\frac{\sqrt{|ad|} - \sqrt{|bc|}}{\sqrt{|ad|} + \sqrt{|bc|}} \right)^{[\frac{m}{2}]} \end{aligned}$$

and $[\cdot]$ denotes the integer part. In our application, d will be a large number, and a comparably small. Also, b, c might be close to 0, and the convergence might be very slow.

The second method we want to highlight is the symmetric QMR (quasi-minimal residual) method. It is a special case of the quasi-minimal residual (QMR) method [150].

The general QMR uses two sequences of Lanczos vectors V_m, W_m that span the Krylov subspaces $\mathcal{K}_m(A, b)$ and $\mathcal{K}_m(A^T, b)$ with relations $AV_m = V_{m+1}\tilde{H}_m$, $A^T W_m = W_{m+1}\tilde{T}_m$ with tridiagonal $\tilde{H}_m, \tilde{T}_m$. Although our problem is symmetric, this flexibility allows the use of indefinite preconditioners. For a non-symmetric matrix A the resulting V_m, W_m are not orthogonal matrices anymore. Instead, they are constructed to be biorthogonal ($V_m^T W_m = I_m$). The iterates $x_m = V_m z_m$ are then characterized by a *quasi-minimal residual* property

$$\min_{z \in \mathbb{R}^m} ||\beta e_1 - \tilde{H}_m z||,$$

and the residual vector is given by

$$b - Ax_m = V_{m+1}(\beta e_1 - \tilde{H}_m z_m).$$

Algorithm 7.5: Symmetric QMR algorithm for $Ax = b$ with preconditioner $B = B_1 B_2$ (see [151])

Choose $x_0 \in \mathbb{R}^N$
Set $r_0 = b - Ax_0, t = B_1^{-1} r_0, \tau_0 = ||t||_2, q_0 = B_2^{-1} t, \theta_0 = 0$ and $\rho_0 = r_0^T q_0$
for $n = 1, 2, \ldots$ **do**
    Compute $t = Aq_{n-1}$ and $\sigma_{n-1} = q_{n-1}^T t$.
    if $\sigma_{n-1} = 0$ **then**
        stop
    $\alpha_{n-1} = \frac{\rho_{n-1}}{\sigma_{n-1}}$ and $r_n = r_{n-1} - \alpha_{n-1} t$.
    Set $t = B_1^{-1} t,\ \theta_n = \frac{||t||_2}{\tau_{n-1}},\ c_n = \frac{1}{\sqrt{1+\theta_n^2}},\ \tau_n = \tau_{n-1}\theta_n c_n,$
    and $d_n = c_n^2 \theta_{n-1}^2 d_{n-1} + c_n^2 \alpha_{n-1} q_{n-1},\ x_n = x_{n-1} + d_n.$
    if *convergence* **then**
        stop
    if $\rho_{n-1} = 0$ **then**
        stop
    Set $u_n = B_2^{-1} t,\ \rho_n = r_n^T u_n,\ \beta_n = \frac{\rho_n}{\rho_{n-1}},$ and $q_n = u_n + \beta_n q_{n-1}$.

For general A, V_m and W_m are not unitary matrices, but when A is a symmetric matrix, the algorithm simplifies significantly. In fact, without preconditioning, the symmetric QMR and the MINRES are mathematically equivalent [151]. However, when using a preconditioner, the two methods differ. While the QMR can deal with indefinite preconditioners, the MINRES method is restricted to symmetric positive definite preconditioners. This seems a bit unnatural but is due to the fact that MINRES requires a symmetric matrix in the iteration. Therefore, the preconditioner has to be applied in a symmetric way. As we will see in the next section, this requires the preconditioner to be symmetric, positive definite. The symmetric QMR on the other hand can directly incorporate an in-

definite preconditioner. However, as noted in [152], a sophisticated 'look-ahead' is needed to prevent numerical breakdown. A simple preconditioned algorithm without look-ahead for the symmetric QMR is given in Algorithm 7.5.

7.3.3 Preconditioning

The error bounds e.g. for the MINRES method show that for the problems under investigation here, the convergence will be quite slow. Therefore, these methods are usually used with a preconditioner.

Refering to a linear problem of type

$$Ax = b, \tag{7.23}$$

preconditioning means transforming this linear system into one which is easier to solve for an iterative method, and has the same solution as the original one. Most commonly, the preconditioner is a matrix itself, let us denote it by B. Application of the preconditioner (B^{-1}) should be inexpensive. Then the left preconditioned system is

$$B^{-1}Ax = B^{-1}b. \tag{7.24}$$

Similarly, right preconditioning leads to

$$AB^{-1}u = b, \quad x = B^{-1}u. \tag{7.25}$$

If a factorization $B = B_L B_R$ is available, then it can be split into

$$B_L^{-1}AB_R^{-1}u = B_L^{-1}b, \quad B_R^{-1}u. \tag{7.26}$$

Since A is symmetric, it seems important to preserve symmetry, i.e. the first two approaches are not symmetric in general. However, symmetry can also be preserved by using an B-inner product without factorizing or splitting B (e.g. sym. QMR, or preconditioned CG method [144, Ch. 9.2.1]).

A good preconditioner usually fulfills $B^{-1}A \approx I$, but anything which improves convergence is acceptable. The ideal preconditioner is the matrix A itself, and the iterative method would finish in one step. For obvious reasons, this choice is out of the question. Classical choices for preconditioners (where $A = D - E - F$ with D diagonal, E, F lower and upper triangular matrices with zeros on the diagonal) are:

- Jacobi: $B = D$,
- Gauss-Seidel: $B = (D - E)$,
- Symmetric Gauss-Seidel: $B = (D - E)D^{-1}(D - F)$. Note that this can be written as $B = LU$ with $L = (D - E)D^{-1} = I - ED^{-1}$ and $U = D - F$. Hence, this can be realized by two solves with an upper/lower triangular matrix [144].

However, these preconditioners do not work well for the problem under consideration. A more elaborate approach consists in computing an approximate factorization

$$A \approx LU$$

where L and U have some favorable properties (for example sparsity, or low cost) and hopefully $||A - LU||$ small. Many strategies exist for sparse matrices, e.g. incomplete LU or incomplete Cholesky factorizations. For symmetric indefinite matrices, an incomplete LDL^T factorization with lower triangular L and diagonal D can be computed [144, 147, 153].

Preconditioning methods are available in software packages such as ILUPACK [154]. ILUPACK provides state of the art algorithms for computing approximate matrix decompositions based on the incomplete LU decomposition including a multilevel approach [155]. Additionally, iterative solvers such as GMRES, PCG and symmetric QMR are provided, which can make use of the preconditioners. Matrix reordering schemes are also available. Altogether, the algorithms in ILUPACK allow for the efficient solution of sparse linear systems.

Let us return to the aforementioned limitation, that MINRES requires a symmetric positive definite preconditioner. This is due to the fact that the iteration matrix in MINRES needs to be symmetric. As a consequence, the preconditioning has to be of the form $B^{-1/2}AB^{-1/2}$ for a preconditioner $B = B^{1/2}B^{1/2}$. Thus, B is symmetric positive definite and we transform to a symmetric problem with the matrix square root. Such a restriction is unnatural since for an indefinite matrix A and a symmetric positive definite preconditioner B, we cannot have $B^{-1/2}AB^{-1/2} \approx I$. However, the problem can be partly circumvented [156]. First, an approximate LDL^T factorization with indefinite D is computed. Then $L|D|L^T$ is used as preconditioner. $L|D|L^T$ has the same eigensystem but positive eigenvalues. However, convergence might be slower compared to a preconditioned symmetric QMR [151].

On the other hand, the symmetric QMR only requires a relationship of the form $A^TP = PA$ for the Lanczos process to simplify. This can be fulfilled by an arbitrary nonsingular symmetric conditioner, see [151] and Algorithm 7.5. More precisely, if $B = B_1B_2 = B_2^TB_1^T = B^T$ is a split preconditioner, then the preconditioned system $A' = B_1^{-1}AB_2^{-1}$ can be used, since with $P = B_1^TB_2^{-1}$ we have

$$(A')^TP = B_2^{-T}A^TB_2^{-1} = B_1^TB_2^{-1}B_1^{-1}AB_2^{-1} = PA'.$$

So far, we have derived preconditioners which have a direct matrix representation. However, this is not a requirement. In fact, as in many iterative methods, only the action of the preconditioner to a vector is required. In the next section, we explore an elaborate approach based on a multilevel structure. This method can be used as a solver for the linear system itself, or as a preconditioner built into an outer iterative method.

7.4 A Multilevel Approach for Solving the Linear System

The idea of multilevel methods for PDEs is based on a particular convergence behavior of certain splitting methods. For most classical iteration methods such as the Richardson iteration (with ω suitably chosen)

$$x_{k+1} = x_k + \omega(b - Ax_k) \tag{7.27}$$

the convergence is quite slow. However, oscillatory components of the error are damped significantly after only a few steps.

The large linear systems can be solved efficiently by a multigrid approach by exploiting information gained by discretizing the PDE on a sequence of meshes. On a fine mesh, iteration methods - called *smoothers* - will allow us to reduce oscillatory components and thus smooth the error. The idea of multigrid now relies on the fact that a smooth function can be well approximated on a coarser mesh. Hence, on the coarse mesh a (linear) error equation for the error is solved, which is then used to correct the approximation on the fine mesh. The key point is that the resulting linear system on the coarse mesh is of much smaller dimension. Obviously, this idea can be applied recursively, and the linear system on the coarse mesh can itself be solved by first iterating with a cheap method, and then solving the error equation on an even coarser grid.

All in all, a hierarchy of meshes offers to move between different resolutions. On each level, a few steps of the smoother reduce the corresponding fastest error components which allows the representation of the error on a coarser mesh. The obtained approximation is then improved by a correction from a coarser grid. Combining these ideas results in an efficient solver.

This section investigates whether a similar approach can be employed in solving linear systems for normal mode analysis. At first, it is not at all obvious how this could work. NMA is neither based on meshes nor does the concept of a smoother directly translate. However, a molecule possesses a strong, fixed structure itself. The success of coarse graining methods seems to indicate that it is possible to reduce the 'resolution' within a molecular model while still resolving its essential dynamics. It seems worth trying a series of such models for use in a multilevel approach. The role of the smoother is not obvious, but we can employ an iteration method with similar properties. Its task is to reduce the error such that the result can be well represented on the next coarser level. Since coarsening usually reduces local and internal high frequency motions, we suspect that a simple method such as the Richardson iteration might be a good candidate.

7.4.1 Basic Algorithm

We directly use ideas motivated by classical multigrid schemes. We closely follow the notation and ideas in [144, Ch. 13]. In the context of molecular dynamics, however, no grids are present. Instead, we will use the term 'level'. For a successful multilevel approach,

two ingredients are essential: first, one needs a hierarchy of levels with matching operators for projection and prolongation; this makes it possible to switch between different levels. Second, a numerical method which reduces the fast error components on each particular level has to be selected. Those two ingredients have to match, i.e. the smoother needs to produce an approximation which can be well represented on the next coarser level.

Algorithm 7.6: Basic two-level algorithm with two levels (A_f, A_c) and starting value x_0

Function $x = \texttt{twolevel}(b_f, x_0)$
- $x = \texttt{smooth}(A_f, b_f, x_0, \mu_1)$ // μ_1 *smoothing steps, starting value* x_0
- $r = b_f - A_f x$ // *compute residual*
- $A_c e_c = P^T r$ // *solve for* e_c
- $x = x + P e_c$ // *correct*
- $x = \texttt{smooth}(A_f, b_f, x, \mu_2)$ // μ_2 *smoothing steps, starting value* x

In Algorithm 7.6, the two-level procedure is shown for solving $A_f x = b_f$. There, we have a fine and a coarse level with symmetric matrices $A_f \in \mathbb{R}^{n\times n}$ and $A_c \in \mathbb{R}^{m\times m}$ with $m < n$, respectively. The relationship between these levels is expressed by a mapping $P \in \mathbb{R}^{n\times m}$ which allows the restriction and prolongation between levels. At the beginning and end of the algorithm, a certain number of smoothing steps are computed. Here, the function $\texttt{smooth}(A_f, b_f, x_0, \mu)$ denotes the application of μ steps of the smoother applied to the problem $A_f x = b_f$ with starting value x_0. In between the residual vector is projected to the coarse level. There, the corresponding linear system is solved exactly. The coarse level solution is then used as a correction for the fine level.

We impose a few assumptions to simplify the analysis. First, we assume that $P \in \mathbb{R}^{n\times m}$ is an orthogonal matrix with $P^T P = I_m$. Second, assume that the matrix A_f is nonsingular and the same holds for A_c. Third, we restrict ourselves to employing a Galerkin projection

$$A_c = P^T A_f P \tag{7.28}$$

to obtain the coarse matrix. Note that in general, for a multilevel algorithm the interpolation and prolongation matrices do not need to be each other's transpose.

For a further analysis, we write the smoothing iterations in the form

$$\tilde{x}_{k+1} = S\tilde{x}_k + g, \qquad k = 0, 1, ... \tag{7.29}$$

where S is the iteration matrix for one smoothing step. Then the application of μ smoothing steps can be written as

$$\tilde{x}_\mu = S^\mu \tilde{x}_0 + g_\mu, \quad g_\mu = \sum_{i=0}^{\mu-1} S^i g.$$

The coarse level correction (lines 3-5 in Algorithm 7.6) reads

$$\begin{aligned} \hat{x}_{k+1} &= \hat{x}_k + PA_c^{-1}P^T(b_f - A_f\hat{x}_k) \\ &= (I_n - PA_c^{-1}P^TA_f)\hat{x}_k + PA_c^{-1}P^Tb_f \\ &= T\hat{x}_k + b_c \end{aligned} \tag{7.30}$$

where

$$T = I_n - PA_c^{-1}P^TA_f \qquad \text{and} \qquad b_c = PA_c^{-1}P^Tb_f. \tag{7.31}$$

Thus, one iteration of the two-level algorithm (Algorithm 7.6) can be expressed as an iteration scheme with

$$\begin{aligned} x_{k+1} &= S^{\mu_2}(T(S^{\mu_1}x_k + g_{\mu_1}) + b_c) + g_{\mu_2} \\ &= S^{\mu_2}TS^{\mu_1}x_k + \tilde{g}_{\mu_1,\mu_2} \\ &= Gx_k + \tilde{g}_{\mu_1,\mu_2}. \end{aligned} \tag{7.32}$$

Here,

$$G = S^{\mu_2}TS^{\mu_1} \qquad \text{and} \qquad \tilde{g}_{\mu_1,\mu_2} = S^{\mu_2}(Tg_{\mu_1} + b_c) + g_{\mu_2}. \tag{7.33}$$

G is the iteration matrix of the two-level cycle, and the matrix T as defined in (7.31) acts as the coarse level correction. μ_1 and μ_2 denote the number of smoothing iterations before and after the coarse grid correction.

The extension to a hierarchy of levels $l = 1, 2, ..., L$ is straight forward. In the fourth line of Algorithm 7.6, the method requires us to solve the coarse level equation exactly. However, $A_ce_c = P^Tr$ might still be too large to be solved directly. Instead, the two-level idea can be applied recursively, and the coarse level correction can itself be approximated by yet another two-level method on an even coarser level. The obtained recursive algorithm is given Algorithm 7.7. There, we assume that we have a sequence of suitable projectors $P_l, l = 2, ..., L$ and set

$$A_1 = A_f, \qquad A_l = P_l^TA_{l-1}P_l, \quad l = 2, ..., L$$

to create a hierarchy of L levels. The function $\texttt{smooth}(A_l, b_l, x_0^l, \mu_1^l)$ denotes the application of μ_1^l steps of the smoother applied to the problem $A_lx = b_l$ with starting value x_0^l. The initial call to this algorithm is simply

$$x = \texttt{multilevel}(1, b_f, x_0^1)$$

where x_0^1 is a starting vector for level one, e.g. the zero vector, and b_f is the right hand side on the finest level. The method then recursively calls itself iterating through the coarser

and coarser levels. On each level l, we can choose the number of pre- and postsmoothing steps μ_1^l, μ_2^l as well as the starting vector x_0^l.

Algorithm 7.7: Basic multilevel algorithm with L levels

```
1 Function x = multilevel(l, b_l, x_0^l)
2     if l < L then
3         x = smooth(A_l, b_l, x_0^l, μ_1^l)
4         r = b_l − A_l x
5         e = multilevel(l + 1, P_{l+1}^T r, x_0^{l+1})
6         x = x + P_{l+1} e
7         x = smooth(A_l, b_l, x, μ_2^l)
8     else
9         x = A_L^{-1} b_L
```

The way this algorithm walks through the levels, has similarities to the letter V, therefore Algorithm 7.7 is known as a V-cycle. If line 5 is replaced by a double call of the multilevel function, that is

$$e = \texttt{multilevel}(l+1, P_{l+1}^T r, \texttt{multilevel}(l+1, P_{l+1}^T r, x_0^{l+1})),$$

this method corresponds to a W-cycle [144, Ch. 13]. See Figure 7.3 for a schematic representation of a V- and W- cycle with four levels.

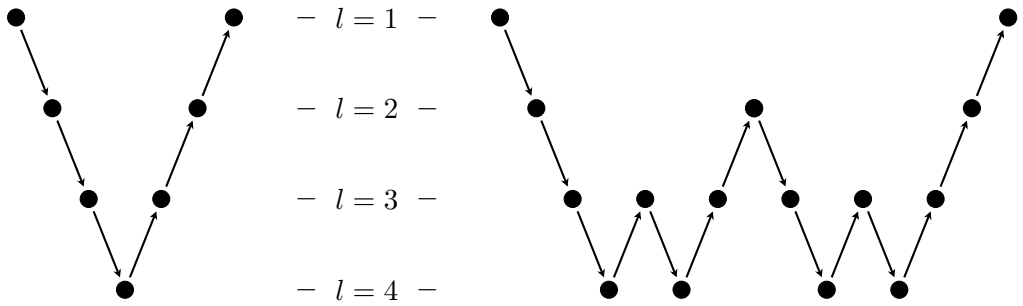


Figure 7.3: *Schematic representation of a V- and a W-cycle for a four-level algorithm. A downwards arrow from level i to $i+1$ corresponds to computing μ_1^i pre-smoothing steps on level i, and then computing the projected residual for the next level. An upwards arrow from $i+1$ to i corresponds to correcting the approximation on level i by the coarse level correction from level $i+1$, and then computing μ_2^i post-smoothing steps. On a node at level 4, the exact solution $A_L^{-1} b_L$ is computed.*

7.4.2 Properties and General Convergence of the Two-Level Algorithm

We now return to the two-level algorithm, and highlight a few properties of the coarse level correction.

Lemma 24. *The coarse level correction T defined in (7.31) is a projector.*

Proof. By definition we have

$$\begin{aligned} T^2 &= (I_n - PA_c^{-1}P^TA_f)(I_n - PA_c^{-1}P^TA_f) \\ &= I_n - 2PA_c^{-1}P^TA_f + PA_c^{-1}\underbrace{P^TA_fP}_{=A_c}A_c^{-1}P^TA_f \\ &= I_n - PA_c^{-1}P^TA_f = T. \end{aligned}$$

Hence, T is a projector. □

With the projector T, we can write $\mathbb{R}^n$ as a direct sum of two subspaces

$$\mathbb{R}^n = \mathcal{P} \oplus \mathcal{T} = \text{Null}(T) \oplus \text{Range}(T).$$

This implies

$$Tx = 0, \quad \forall x \in \mathcal{P} \quad \text{and} \quad Tx = x, \quad \forall x \in \mathcal{T}.$$

It is a small further observation that

$$\mathcal{P} = \text{Null}(T) = \text{Range}(P),$$

and

$$\begin{aligned} \mathcal{T} &= \text{Range}(T) = \text{Null}(I - T) \\ &= \text{Null}(PA_c^{-1}P^TA_f) \\ &= \text{Null}(P^TA_f). \end{aligned}$$

Additionally, $\mathcal{P}$ and $\mathcal{T}$ are orthogonal with respect to the A_f-weighted inner product (if A_f is s.p.d) or, in general, for all $x \in \mathcal{T} : P^TA_fx = 0$. By construction, application of T annihilates components in the range of P, but leaves components in the kernel of P^TA_f untouched.

We now turn to the convergence of the two-level iteration (7.32). In the literature, the convergence theory has been established for many cases, and precise, technical conditions are available. We refer to [157] and references therein. Here, however, we want to give a very simple, and straight forward sufficient condition for convergence.

Lemma 25. *Assume that the exact solution $x_\star$ of $A_fx = b_f$ is a fixpoint of the smoother (7.29), e.g. $x_\star = Sx_\star + g$, then $x_\star$ is also a fixpoint of the two-level algorithm (7.32).*

Proof. Inserting the exact solution $x_\star$ into (7.32) we obtain

$$\begin{aligned} Gx_\star + \tilde{f}_{\mu_1,\mu_2} &= S^{\mu_2}(T(S^{\mu_1}x_\star + g_{\mu_1}) + b_c) + g_{\mu_2} \\ &= S^{\mu_2}((I_n - PA_c^{-1}P^TA_f)x_\star + PA_c^{-1}P^Tb_f) + g_{\mu_2} \\ &= S^{\mu_2}((x_\star - PA_c^{-1}P^Tb_f + PA_c^{-1}P^Tb_f) + g_{\mu_2} \\ &= S^{\mu_2}x_\star + g_{\mu_2} \\ &= x_\star. \end{aligned}$$

Thus, $x_\star$ is also a fixpoint of the two-level algorithm. □

Convergence of the two-level iteration (7.32) is then proven by showing that $||G|| < 1$ under certain assumptions on the smoother.

Lemma 26. *Let the coarse level correction T be defined as in (7.31), and $\mathcal{T} = \text{Range}(T)$. If the smoother defined in (7.29) is a convergent method with $||S|| < 1$, and fulfills*

$$||S^{\mu_2}t|| \leq \frac{||t||}{||T||}, \quad \forall t \in \mathcal{T}, \tag{7.34}$$

where μ_2 denotes the number of post-smoothing steps, then (7.32) describes a convergent method.

Proof. We show that $||G|| < 1$ with G as in (7.32). Note that

$$\begin{aligned} ||G|| = ||S^{\mu_2}TS^{\mu_1}|| &\leq ||S^{\mu_2}T|| \cdot ||S^{\mu_1}|| \\ &< ||S^{\mu_2}T|| \end{aligned}$$

since $||S|| < 1$. With $x = s + t,\ s \in \mathcal{P}, t \in \mathcal{T}$ we have

$$\begin{aligned} ||S^{\mu_2}Tx|| = ||S^{\mu_2}T(s+t)|| &= ||S^{\mu_2}t|| \\ \leq \frac{||t||}{||T||} &\leq ||x|| \qquad \forall x \in \mathbb{R}^n\,. \end{aligned}$$

The last inequality holds since $t = Tx$ and therefore $||t|| \leq ||T|| \cdot ||x||$. Thus, $||G|| < 1$ and the method (7.32) is convergent. □

The lemma is based on a very simple estimate. Therefore, it requires a very strict condition on the smoother. This can be improved in many ways. First, it basically ignores the pre-smoothing iterations. Second, the inequality (7.34) can be weakened significantly depending on the specific angles between $\mathcal{P}$ and $\mathcal{T}$.

Nevertheless, the lemma is useful in establishing a baseline. The assumption in (7.34) requires the smoother to converge quickly on the subspace $\mathcal{T}$. Elsewhere, the smoother can have a very limited convergence rate. This is the case for many simple iteration methods such as the Richardson iteration.

Now, the general heuristic is that we find a subspace with basis P such that the associated subspace $\mathcal{T}$ is the space where a selected smoother's convergence is very fast. We can approach this from both ends: either we select a smoother, check where it converges fast, and then try to construct a suitable subspace $\mathcal{P}$, or we select the space first, and then try to combine it with a suitable smoother.

7.4.3 Simple Analysis for Eigenspace Decomposition

There is one special case that we want to investigate a little bit further. The ideal choice of P, though not practical, would be a basis of the subspace containing the eigenvectors of A_f corresponding to the lowest eigenvalues. We want to show that this choice combined with the simple Richardson iteration as a smoother leads to fast convergence.

Since A_f is symmetric, it is diagonizable, i.e.

$$A_f = VDV^T, \qquad D = \operatorname{diag}(\lambda_1, ..., \lambda_n)$$

with eigenvalues $\lambda_1 \leq \cdots \leq \lambda_n$ and an orthogonal matrix V. Note that in the linear systems required for normal mode analysis, the eigenvalues are contained in an interval (a, b) where a is a small negative number and b is a large positive number. In the examples of Chapter 8, $a \in [-100, -10]$ and $b \in [10^5, 10^6]$.

Now, we fix $1 < p \ll n$ and let

$$V = [V_c, V_f], \qquad D = \begin{bmatrix} D_c & \\ & D_f \end{bmatrix}, \qquad \begin{aligned} D_c &= \operatorname{diag}(\lambda_1, ..., \lambda_{p-1}) \\ D_f &= \operatorname{diag}(\lambda_p, ..., \lambda_n) \end{aligned}$$

be a splitting of V into A_f invariant subspaces where V_c contains all eigenvectors for eigenvalues $\lambda_i < \lambda_p$, and V_f contains all eigenvectors to eigenvalues $\lambda_i \geq \lambda_p$. We choose $P = V_c$, and obtain

$$A_c = V_c^T A_f V_c = D_c.$$

As we will see later, the factor

$$K = 1 - \frac{\lambda_p}{\lambda_n}$$

plays a significant role in the convergence of the smoother. The smaller this factor, the faster the convergence in the smoothing step. However, decreasing this factor corresponds to choosing a larger p. On the other hand, the smaller we choose the space V_c or p, the easier it will be to solve the coarse level equation.

The coarse level corrector T with the choice of $P = V_c$ is

$$\begin{aligned} T &= I - PA_c^{-1}P^T A_f = I - V_c A_c^{-1} V_c^T A_f \\ &= I - V_c V_c^T \\ &= V_f V_f^T. \end{aligned}$$

Since T is actually symmetric, T is an orthogonal projector and $||T|| = 1$.

The smoother, a Richardson iteration, has the iteration matrix

$$S = I - \omega A_f.$$

Here, we choose $\omega = \frac{1}{\lambda_n}$. Hence,

$$\begin{aligned} & S = I - \frac{1}{\lambda_n} A_f \\ \Rightarrow \quad & S^\mu V = V \left(I - \frac{1}{\lambda_n} D \right)^\mu . \end{aligned}$$

Note, that under the assumption that $\frac{\lambda_p}{\lambda_n}$ is sufficiently small, the smoother converges quickly on V_f:

$$||SV_f|| = ||I - \frac{1}{\lambda_n} D_f|| = 1 - \frac{\lambda_p}{\lambda_n} \ll 1.$$

However, on V_c we have

$$||SV_c|| = ||I - \frac{1}{\lambda_n} D_c|| = 1 - \frac{\lambda_1}{\lambda_n} \approx 1,$$

and it might even be slightly larger than 1 in the presence of negative eigenvalues of small magnitude. Note, that therefore $||S|| > 1$ and we do not fulfill the assumptions in Lemma 26. As already mentioned, though, the assumptions in Lemma 26 are very strict. And in fact, since Range (V_c) is an S-invariant subspace with Range $(V_c) \subset \mathcal{P}$, the two-level algorithm will still converge [157, cf. Fact 1.1].

For the two-level algorithm with G as in (7.33) we have

$$\begin{aligned} GV = S^{\mu_2} T S^{\mu_1} V &= S^{\mu_2} V_f V_f^T V \left(I - \frac{1}{\lambda_n} D \right)^{\mu_1} \\ &= S^{\mu_2} V_f \left(I - \frac{1}{\lambda_n} D_f \right)^{\mu_1} \\ &= V_f \left(I - \frac{1}{\lambda_n} D_f \right)^{\mu_1 + \mu_2} \end{aligned}$$

and thus

$$||G|| = ||GV|| = \left(1 - \frac{\lambda_p}{\lambda_n} \right)^{\mu_1 + \mu_2} = K^{\mu_1 + \mu_2}.$$

Hence, in this simple example the two-level algorithm has a convergence rate of $K^{\mu_1+\mu_2}$. Furthermore, it does not make any difference whether we iterate k times with G, or simply once but with $k\mu_1$ and $k\mu_2$ smoothing steps. This behavior is specific to the very simple special case using exact invariant subspaces of A_f. The coarse grid correction solves its part on V_c exactly in a single iteration. There is no benefit in multiple iterations.

However, in practice, a few obstacles remain. Most importantly, a decomposition in V_c and V_f is obviously not available, but instead target of the eigensolver which calls the multilevel algorithm as a linear solver. Instead, we will have to work with approximations of invariant subspaces. Also, the coarse level equation will not be solved exactly but will be replaced by an approximation itself, introducing further challenges. This will be discussed in the following sections.

7.4.4 A Subspace Based on Translation and Rotation

A key part of the multilevel approach is the generation of a hierarchy of levels which provide descriptions of different resolutions of the underlying molecular model. A straight forward idea from coarse graining (see e.g. ideas used for the proposed corotational filter in Chapter 5) decomposes the molecule into disjunct clusters. Internal motion is removed for each cluster, and the movement is restricted to translation and rotation. If the allowed moves are collected in an orthonormal matrix $P \in \mathbb{R}^{n\times m}$, thus leads to the coarse Hessian $A_c = P^T A_f P$. Multiple levels can easily be formed; a coarser level is obtained by simply connecting two or more neighboring clusters to a new, larger cluster. Finer levels are obtained by dividing clusters into multiple, smaller ones.

Each cluster yields three translation vectors and three rotation vectors. In Cartesian coordinates, these vectors can be quickly computed. The translation vectors for a single cluster with k atoms are a simple Kronecker product

$$N^{\text{trans}} = \frac{1}{\sqrt{k}} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \otimes I_3 \in \mathbb{R}^{3k\times 3}.$$

The corresponding rotational vectors for that cluster at position $q = [q_1, \cdots, q_k]^T \in \mathbb{R}^{3k}$ are

$$N^{\text{rot}} = [(I_k \otimes \Omega_x)\hat{q}, \quad (I_k \otimes \Omega_y)\hat{q}, \quad (I_k \otimes \Omega_z)\hat{q}] \in \mathbb{R}^{3k\times 3}$$

with internal coordinates

$$\hat{q} = q - \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \otimes \frac{1}{k} \sum_{i=1}^{k} q_i \in \mathbb{R}^{3k}$$

and generators of rotations

$$\Omega_x = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}, \quad \Omega_y = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{pmatrix}, \quad \Omega_z = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

Normal mode analysis is usually conducted in mass-weighted coordinates, and the

translation and rotation vectors need to be transformed into that mass-weighted space. The resulting vectors $M^{1/2}N^{\text{trans}}, M^{1/2}N^{\text{rot}}$ for all m clusters are collected in a matrix $N \in \mathbb{R}^{n\times 6m}$ with $N^T N = I_{6m}$.

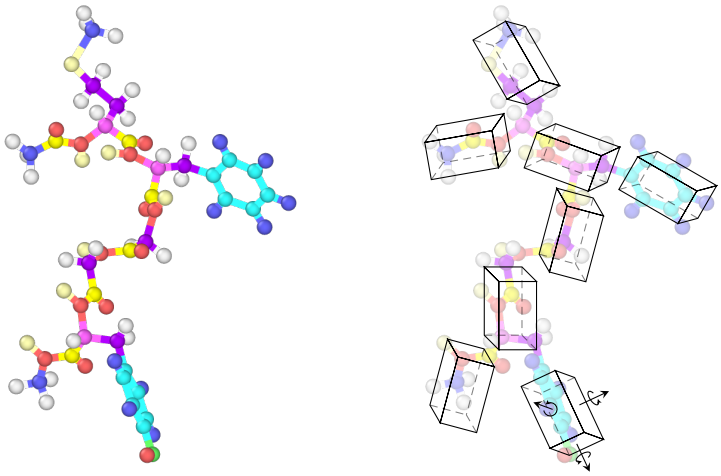


Figure 7.4: *Translation-and-rotation approach: decomposition of the peptide from Problem (B) in clusters depicted by cuboids. Each cuboid is then restricted to translation and rotation.*

Note that this approach can be modified in many ways. First, the restriction to local translation and rotation is not a necessary requirement and other motions could also be considered. Second, the clusters cut the molecule at selected bonds. Bonds, however, are strong connections within a molecule. Decompositions which allow an overlap, that is, which allow an atom to be part of two or maybe multiple clusters, might be better suited. Basically, it is about whether we can approximately guess the slow subspace correctly, any physically motivated decomposition can be incorporated.

7.4.5 A Subspace Based on Torsion Angles

A more elaborate and physically motivated subspace is used in quite a few coarse graining models where a reduction to internal coordinates is performed. A very common choice is the reduction to dihedral coordinates. Here, only a few selected (usually important backbone) dihedral angles are kept as variables. The remaining components, i.e. all bonds and angles, are kept fixed. This results in a rigid model which has some rotational degrees of freedom along some significant chains.

In a very similar manner such 'significant' dihedrals can be identified and used to construct a relevant subspace. However, the resulting N will be, in contrast to the shift-and-rotation approach, a dense matrix. This is not a problem if a small enough number of dihedrals is selected.

Similar to the translation-and rotation approach, the construction of the subspace is straight forward. For each dihedral (with four assigned atoms), we compute the rotation

around the axis $\omega = \frac{x_j - x_i}{||x_j - x_i||}$ indicated by the two interior, middle atoms i, j. For a single torsion axis the idea is shown in Figure 7.5.

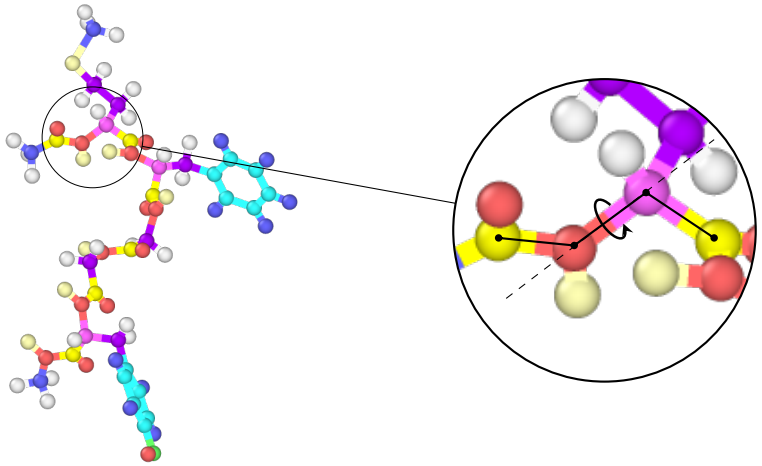


Figure 7.5: *Torsion approach: a single torsional degree of freedom from the peptide of Example (B) is shown. The black points indicate the atoms forming the dihedral, the dashed line is the rotation axis.*

However, there is one situation in which such an approach fails: if there are loops in the main chain of a molecule, it is not clear how to construct the corresponding torsion vectors. In a loop of a molecule, a dihedral cannot be freely moved since both ends of the dihedral are connected through the loop. The problem is resolved by calculating the response of the model when twisting the dihedral. This can be fairly expensive, though, and instead a cheap approximation e.g. by a coarse grained model should be used.

The idea of using rotational degrees of freedom can also be used to refine the translation-and-rotation approach. Recall, that each cluster of atoms was restricted to overall translation and rotation. In such a cluster, additional internal degrees of freedom can be incorporated by allowing the movement along certain dihedral angles.

7.4.6 A Decomposition Approach

The methods presented in [39, 40, 42] start by decomposing a large molecule into smaller, connected domains. Assume we already have a sensible decomposition of a molecule into some larger clusters. Instead of relying purely on translation and rotation, or torsional degrees of freedom, one can simply compute the exact eigenpairs of A restricted onto each cluster. Instead of one big eigenvalue problem, many much smaller ones have to be solved. Since we are only interested in the slow directions, we simply discard the fastest eigenpairs and take the remaining ones as a basis.

The decomposition approach has a clear relation to the translation-and-rotation approach. Both will yield similar results for small clusters since the slowest six eigenvectors will most likely correspond to translation and rotation. However, larger clusters can be

chosen here, and then other slow eigenvectors contribute.

7.4.7 Smoother

Usually, for the smoother (7.29), a cheap iterative method which rapidly converges on the fast modes is chosen.

For solving $Ax = b$, classical iteration methods are of the form

$$x_{k+1} = Sx_k + f,$$

and some examples are (where $A = D - E - F$, D diagonal, E, F strict lower/upper part)

- Richardson iteration: $S_R = I - \omega A$, $f_R = \omega b$ with suitably chosen ω,
- Jacobi iteration: $S_{Ja} = I - D^{-1}A$, $f_{Ja} = D^{-1}b$,
- Gauss-Seidel iteration: $S_{GS} = I - (D - E)^{-1}A$, $f_{GS} = (D - E)^{-1}b$.

More details on each of these methods as well as a general convergence analysis can be found in [144, Ch. 4]. Note, that we can write a Gauss-Seidel iteration in the form

$$\begin{aligned} x_{k+1} &= S_{GS}x_k + f_{GS} \\ &= (I - (D - E)^{-1}A)x_k + (D - E)^{-1}b \\ &= (D - E)^{-1}(Fx_k + b) \end{aligned}$$

and thus, one iteration step has a similar cost compared to one iteration step with the Richardson iteration.

8 Numerical Results for Normal Mode Analysis

This section gives numerical results for the algorithms discussed in the previous chapter. First, after presenting a few suitably chosen test examples, different numerical methods for solving the eigenvalue problem are examined in Section 8.3. Afterwards, numerical results for solving the linear system $(A-\sigma I)x = b$ are evaluated in Section 8.4. We present results for the multilevel approach in Section 8.5.

8.1 Workflow

Fully computing a Hessian takes a lot of resources. First, the molecular problem needs to be built and equilibrated. Then, the structure needs to be minimized until a local minimum of the potential function has been found. Finally, the Hessian can be extracted.

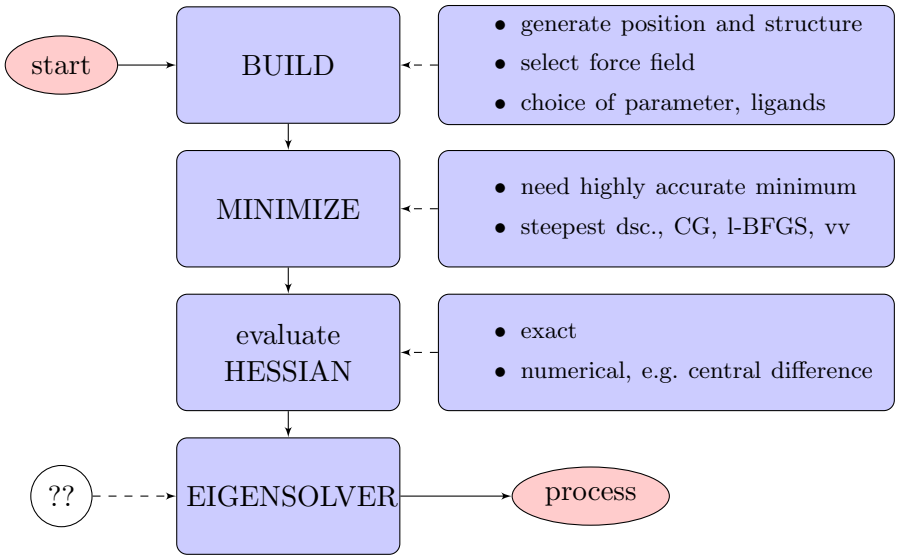


Figure 8.1: *Workflow for normal mode analysis*

When building the problem, the given structure and position data must be thoroughly checked. Since the data is usually obtained by some experimental methods, there may be a large error in the position data due to the resolution limit of the employed method. Frequently, some data, such as hydrogen bonds, have been left out and need to be added. Then a force field, that is, a set of parameters and a functional form of the potential landscape, has to be selected. This step is a choice, and many force fields for different classes of molecules are available. Usually, the force field that most closely matches the requirements is chosen. However, many force fields do not offer a complete parametrization for every possible molecular situation. This is often the case if there are ligands (ions or molecules that bind central metal atoms) present. Those interactions then need to be

parametrized by some force field generation tool (such as SwissParam [60]). Finally, we have a full atomistic model (position data and structure) combined with a complete description of the force field which is ready for simulation.

Now, this model needs to be minimized with respect to its potential function. The attained minimum should be highly accurate, since even minor deviations can lead to quite large negative eigenvalues in the Hessian. Usually, multiple algorithms are employed. It is observed that, for example, the steepest descent method is good for locating a minimum, but the convergence is slow. Hence, after some initial minimization, a more elaborate algorithm such as a CG or l-BFGS method is used for convergence until machine precision. Sometimes, if the initial position data is very inaccurate it is common to first equilibrate the molecular system with a regular molecular dynamics simulation e.g. with the Verlet method.

When the minimum is attained, the Hessian can be calculated. If all second derivatives of the (very complex) force field are available in an analytical form, the Hessian is directly evaluated. Otherwise, a numerical approach e.g. with central differences can be taken.

The software package GROMACS provides solutions for all the required steps. It can read a structure file and automatically assign the correct parameters according to a user-selected force field. Multiple minimizers are implemented. It has the capability to numerically compute the Hessian, and finally can compute a few of the smallest eigenvalues by calling the implicitly restarted Arnoldi method provided by ARPACK.

8.2 Selection of Examples

An excellent database for molecular structures is the RCSB database [158]. It primarily contains structure information of large biological molecules, such as proteins and nucleic acids. Scientists can upload their results in a publicly available format. For each entry in the archive, the method and resolution of the data is listed.

We chose six examples with varying degrees of freedom. Note, though, that in the community those examples are considered rather small structures. However, for larger structures, the necessary effort required to obtain a valid minimized structure grows rapidly.

Each structure in the RCSB database is identified by a unique four character combination. We chose *1PGB*, an immunoglobin binding protein with 855 atoms, *4PTI*, the classical BPTI example with 892 atoms, *1TNG*, a hydrolase inhibitor with 3,247 atoms, *1AI0*, a human insulin hexamer with 4,740 atoms, *5C7X*, a neutralizing antibody with 16,165 atoms, and *2BG9*, a nicotinic acetylcholine receptor with 29,879 atoms. More background information about each example can be found in the RCSB database. We plot each molecule in its cartoon style in Figure 8.2.

For each of the examples, we went through the process described in Figure 8.1. We modeled the interactions with the CHARMM force field, and set a cutoff radius for the interactions to 10Å. After the initial setup of the problem and parametrization, we chose to 'relax' each model with a short MD simulation (500,000 steps with the Verlet method

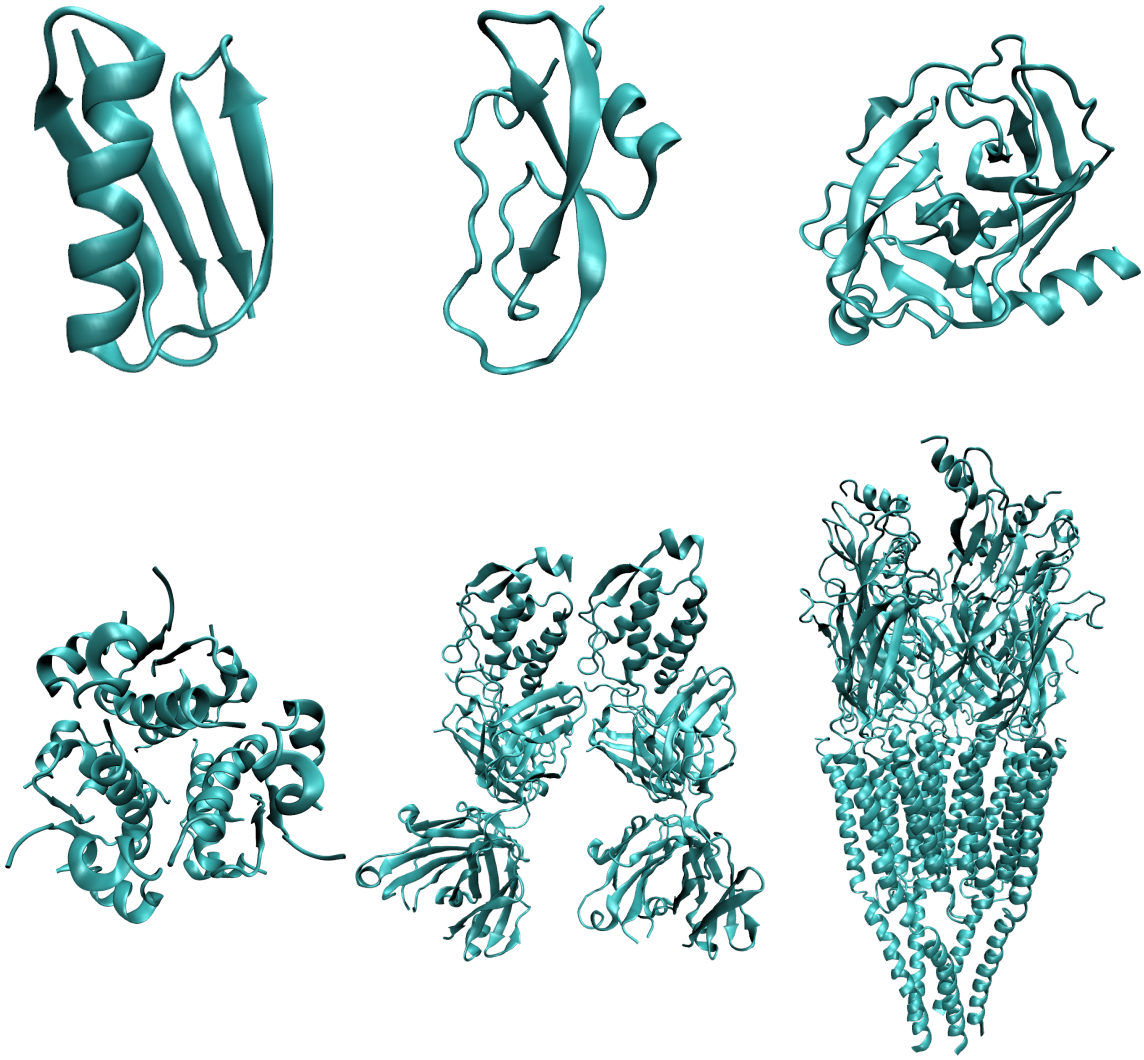


Figure 8.2: *Schematic representation of the selected examples 1pgb, 4pti, 1tng (top row) and 1ai0, 5c7x, 2bg9 (bottom row)*

and 0.5fs step size) starting at 50°K and then slowly cooling it to a near zero temperature. From there on, the l-bfgs minimizer iterated until convergence to machine precision was obtained (between 600 and 3072 iterations depending on the model). At the obtained position, the Hessian was evaluated. The MD simulation took approximately 5 hours on 4 cores for the largest model, and the minimizer afterwards required approximately half an hour. The bottleneck was evaluating the Hessian, which used more than 14 hours. However, the code in GROMACS for evaluating the Hessian is still at the prototype stage.

We list some important facts in Table 8.3. From now on, we use the abbreviations H1, H2, ..., H6 to refer to each of the problems. Across the models, the maximum number of entries per row seems to be between 1,100 and 1,400. It is no surprise that this number is similar for all molecules since it is directly related to the cutoff radius in the potential which we chose to be the same for all models. As a consequence of an almost constant number of entries per row, the density, that is, the relative amount of nonzero entries,

PDB ID	1pgb	4pti	1tng	1ai0	5c7x	2bg9
#dof	2,565	2,676	9,741	14,220	48,495	89,637
max. entries/row	1,269	1,194	1,380	1,245	1,419	1,389
density	0.267	0.245	0.090	0.053	0.018	0.010
#MB	50.2	54.6	48.7	61.5	245.1	436.1
	H1	H2	H3	H4	H5	H6

Table 8.3: *Number of degrees of freedom, maximal nonzero entries per row in Hessian, density, and storage required by Gromacs .mtx format (in MB) for each Hessian*

quickly drops with increasing dimension. Also, there is relatively small variation of the number of entries per row within each model. This is simply attributed to the fact that each atom has a similar number of neighbors within the cutoff radius.

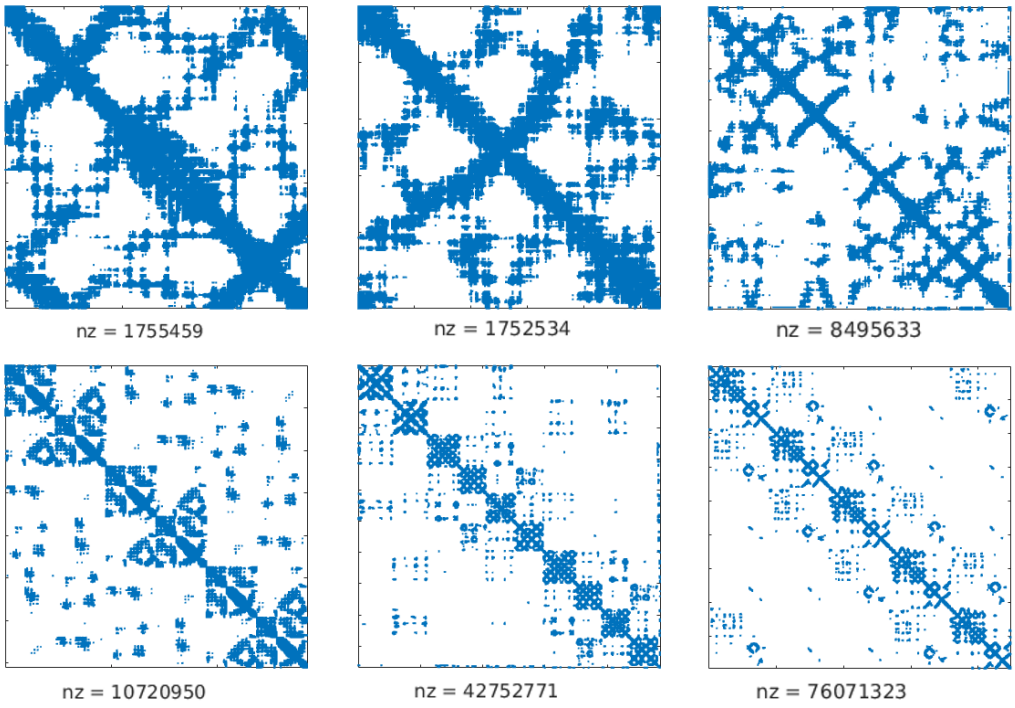


Figure 8.4: *Sparsity pattern of the Hessian matrices, 'nz' denotes the number of nonzero entries. From top left to bottom right: 1pgb, 4pti, 1tng, 1ai0, 5c7x and 2bg9*

Furthermore, Table 8.3 lists the storage required to store the Hessian (actually, due to the symmetry of the matrix only the lower or upper triangular is stored) in GROMACS internal .mtx format (a version of compressed row storage). The anomaly in the storage size for the two small problems is due to the fact that, in GROMACS, a full Hessian and not a sparse Hessian is used for systems with less than 1,000 atoms. In Figure 8.4, the structure of the obtained Hessians is shown.

Table 8.5 lists the smallest and largest eigenvalue of each Hessian. It is noteworthy

	H1	H2	H3	H4	H5	H6
#dof	2,565	2,676	9,741	14,220	48,495	89,637
$\lambda_{\min}$	-2e-07	-2e-07	-0.005	-6e-08	-0.006	-583
$\lambda_{\max}$	5.3e05	5.0e05	5.3e05	5.3e05	5.4e05	5.5e05

Table 8.5: *List of smallest and largest exact eigenvalue ($\lambda_{min}, \lambda_{max}$) of each Hessian*

that the largest eigenvalue is independent of the dimension. Also note that the largest problem, despite having converged to a local minimum (up to machine precision), has one significant negative eigenvalue.

8.3 Solving the Eigenvalue Problem

In this section we test the methods introduced in Section 7.2 and compare their performance on the six examples. A common goal in normal mode analysis is computing a subspace corresponding to slow modes. Therefore, with each algorithm we try to compute the 50 smallest eigenpairs up to residual accuracy of 1e-6.

8.3.1 Lanczos Method and Shift-and-Invert Iteration

In Figure 8.6 we plot the number of converged 'target' eigenvalues with magnitude smaller than 1,000 versus the number of iterations when using the Lanczos method. As predicted,

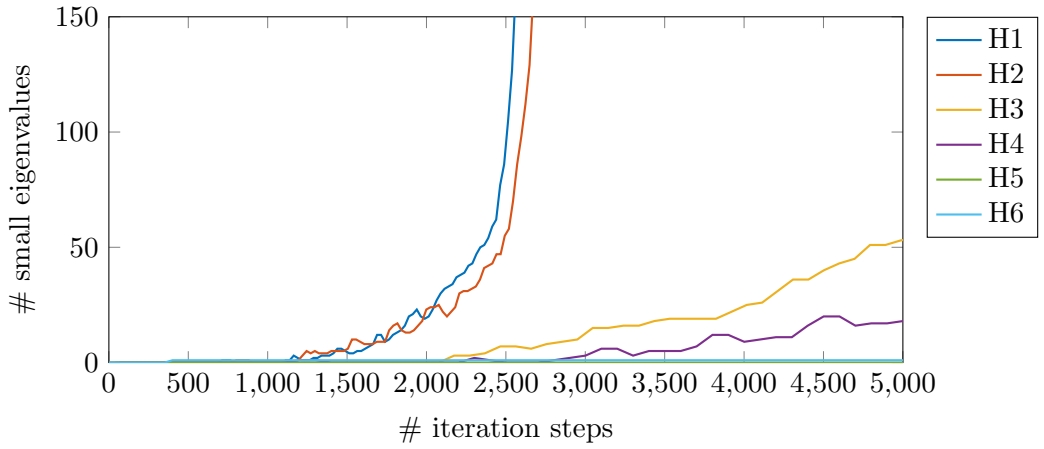


Figure 8.6: *Lanczos method: number of converged eigenvalues with magnitude smaller than 1000 vs number of iteration steps*

it takes many iterations for the eigenpairs to converge. Note that for the two small problems the convergence of small eigenpairs happens close to the full dimension of the matrix. A similar behavior is expected for the larger problems. However, we simply cannot iterate for that long because of two major issues: loss of orthogonality and memory requirements.

A major advantage of the Arnoldi process for symmetric matrices is the reduction to the Lanczos process, that is, H_m reduces to a tridiagonal matrix, and in theory, a three term recurrence allows for cheap iteration steps. However, due to roundoff, the orthogonality between the vectors in V_m is quickly lost. In Figure 8.7 we see that this is a serious issue even for quite small iteration numbers.

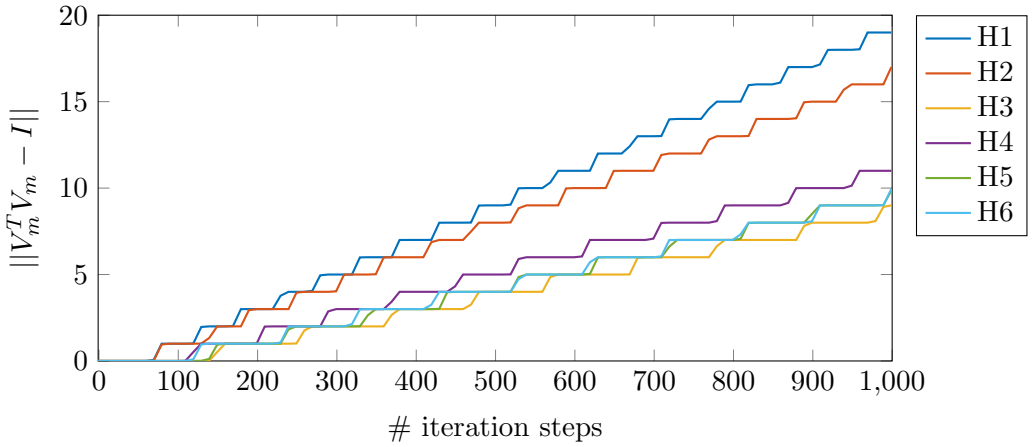


Figure 8.7: *Lanczos method: loss of orthogonality in V_m vs number of iterations m*

In fact, the plot looks quite peculiar and exhibits a funny step behavior. While the algorithm seems to maintain orthogonality for some steps, it then quickly drifts off a little bit. It maintains the new level for a few steps, before it again quickly drifts to a new level. There is an explanation to this behavior. Orthogonality is well maintained until a Ritz vector converges. In this case, the orthogonality can be lost. For more details, we refer to Chapter 13 in [159].

Therefore, the Lanczos process needs to be upgraded with some strategy of reothogonaliation. Hence, a major advantage of the Lanczos process is lost. Full reorthogonalization is not necessary and many more elaborate strategies exist, however, they still add significant cost.

The second big drawback of the Lanczos process is its memory requirement. While the iterations are very cheap, until convergence is reached many iterations have to be done. Yet, we need to store the Lanczos vectors in V_m (the alternative, computing a second, run is too expensive). For example for problem H6, 1000 steps require storing 1000 vectors of size 89,637. In regular doubles (64bits, 8bytes each) this amounts to roughly 700MB. However, since many more iterations than 1000 are required, this quickly becomes unwieldy.

With shift-and-invert iteration the convergence can be accelerated significantly, however, at the cost of repeatedly solving large linear systems. In Figure 8.8 the number of converged target eigenpairs versus the number of required iterations is shown. Two things stick out. First, compared to the Lanczos method very few iterations are required. Second, the convergence of eigenpairs starts almost immediately, and the convergence speed does not majorly differ between the different problems. I.e. the smallest and the largest problem require a similar number of iterations.

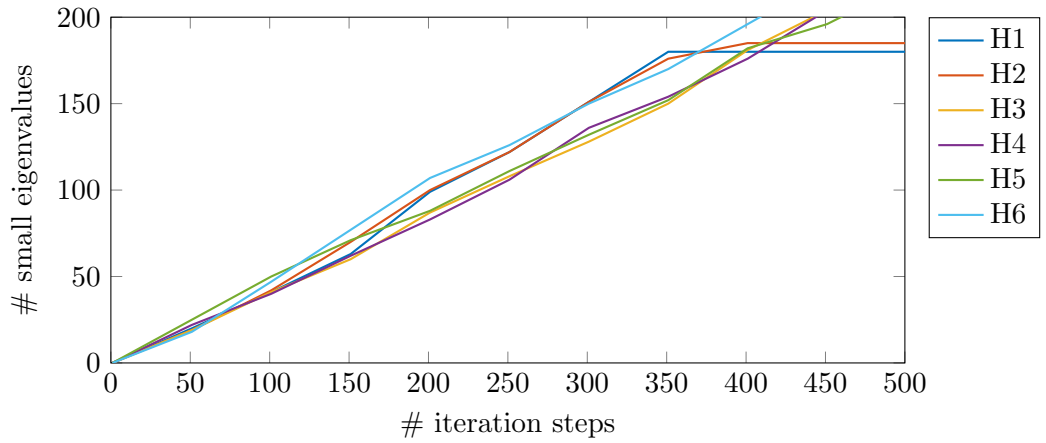


Figure 8.8: *Shift-and-invert iteration: number of converged eigenvalues with magnitude smaller than 1000 vs number of iteration steps*

We also computed harmonic Ritz pairs but could not find a major advantage. Polynomial filtering can speed up convergence. However, one has to take into account that each iteration step has the cost of multiple multiplications with A which (in our tests) nullified the accelerated convergence.

8.3.2 The Implicitly Restarted Arnoldi Method (IRAM)

When using the IRAM, due to the restarts, the memory requirements do not increase the more iterations we compute. We compare the number of required iterations until convergence of the smallest 50 eigenvectors for standard and shift-and-invert mode in the IRAM.

PDB ID	1pgb	4pti	1tng	1ai0	5c7x	2bg9
#dof	2,565	2,676	9,741	14,220	48,495	89,637
n_{it} (standard)	6,440	6,717	12,659	17,851	28,850	35,117
n_{it} (shift-and-invert)	154	154	154	154	154	154
	H1	H2	H3	H4	H5	H6

Table 8.9: *Number of outer iterations n_{it} in IRAM when applied to the six test problems in standard and shift-and-invert mode*

Recall that in the standard mode each iteration consists of a matrix vector product like in the Lanczos method. In the shift-and-invert mode, in each step a large linear system with the shifted matrix $A - \sigma I$ has to be solved. We choose $\sigma = 10$ and give the results in Table 8.9. In contrast to the Lanczos method without restarts, the IRAM in standard mode can run many more iterations without running into memory restrictions. Still, the standard mode requires a very high number of iterations. On the other hand, when using the shift-and-invert mode only few iterations are required. This is no surprise

since there is not much difference between the shift-and-invert iteration and the IRAM in that mode. The obtained number of iterations n_{it} with the IRAM in shift-and-invert mode is 154 for all problems. This comes from the fact that in the IRAM there is no testing for convergence during extending the search space (see 4. in Algorithm 7.2). The default choice for the search space is twice the number of wanted eigenvectors. Thus, in the first iteration a Lanczos basis of size 100 is computed. The restarting procedure then reduces the factorization by half. In the second iteration, the basis is extended back to the size 100 by adding around 50 more Lanczos vectors. Note that the number of iterations could be easily tuned by manually adjusting the size of search space.

Some troubles arise from having a symmetric indefinite matrix A. Since the negative eigenvalues are of much smaller magnitude compared to the largest positive ones, it might seem like a good idea to choose the shift σ in the IRAM such that the resulting matrix $A-\sigma I$ is symmetric positive definite. This can provide some advantage since solving linear systems with a positive definite matrix is more efficient and reliable than for indefinite problems. However, there is a major drawback to this approach which jeopardizes all that is gained.

	H1	H2	H3	H4	H5	H6
n_{it} for $\sigma = 10$	154	154	154	154	154	154
n_{it} for $\sigma = -100$	183	184	216	228	261	362
#dof	2,565	2,676	9,741	14,220	48,495	89,637

Table 8.10: *Influence of different choices for σ on the number of required iterations n_{it} in shift-and-invert mode*

When the shift-and-invert iteration is performed and the shift is not chosen closest to the targeted eigenvalues, the convergence speed significantly decreases. This phenomena is shown in Table 8.10. In this table we list the number of required iterations until the smallest 50 eigenpairs have converged. The shift $\sigma = 10$ is closer to the eigenvalues of interest but the iteration matrix is then indefinite. The number of iterations stays constant in this case. If $\sigma = -100$ is chosen the shift is further away from the eigenvalues but the iteration matrix is now positive definite (except for problem H6 which still has one negative eigenvalue). The number of required iterations, though, is much higher, and seems to increase with the dimension of the problem.

8.3.3 Jacobi-Davidson Method

For performing computations with the Jacobi-Davidson method we use the JADAMILU software package [160]. JADAMILU stands for JAcobi-DAvidson method with Multilevel ILU preconditioning. It is coded in Fortran 77 and the name hints that its linear solver is tuned by a multilevel ILU preconditioner.

The code runs an outer loop which repeatedly solves the Jacobi-Davidson correction

equation to expand the search space. An inner loop iteratively solves the linear systems by using a preconditioned CG or a symmetric QMR method. When using standard settings for computing the smallest 50 eigenpairs the following results are obtained (see Table 8.11).

PDB ID	1pgb	4pti	1tng	1ai0	5c7x	2bg9
#dof	2,565	2,676	9,741	14,220	48,495	89,637
outer steps	195	186	197	194	183	379
inner steps	1,852	1,833	1,984	1,880	1,686	7,824
shift	-5.75	-5.37	-3.02	0.00	0.00	-594
	H1	H2	H3	H4	H5	H6

Table 8.11: *Outer iterations in Jacobi-Davidson method*

The number of inner and outer steps seems to be mostly independent of the problem size. Problem H6 seems to be a special case due to its single negative eigenvalue. In this case, JADAMILU chose a shift slightly below the negative eigenvalue. The number of inner and outer iteration steps is drastically larger compared to the other problems. We suspect that this behavior is similar to the behavior the IRAM exhibits when using a shift further away from the eigenvalues.

By its performance, this method behaves somewhere in between the standard and the shift-and-invert mode in the IRAM. Compared to the shift-and-invert mode, it uses more outer steps, however, each step is only solved approximately. The numbers of inner iteration steps combined, though, are significantly less than required by the standard IRAM.

The JADAMILU has one very interesting possibility: in contrast to the Krylov methods, the user can supply a start space. If a good guess of the 'target' eigenspace is available, the method is expected to converge much faster. Such an initial guess could be cheaply obtained, e.g. by using an elastic network model.

8.3.4 Contour Integration: the FEAST Eigensolver

The FEAST eigensolver is distributed with the Intel MKL libraries under the name Intel MKL Extended Eigensolver. As such, it is expected to be implemented very well. Especially on the smaller four problems it is very quick in converging. It only required 2 or 3 outer iterations for convergence to machine precision. However, one has to keep in mind that each iteration consists of solving N_e (number of quadrature points, default: 8) linear systems with m_0 (here, 100) right hand sides each!

On the larger two problems, namely 5C7X and 2BG9, the eigensolver faces more troubles: for 5C7X, it needs 7 iterations until machine precision, or 4 iterations until a residual error of 1e-6. For problem 2BG9, the solver requires around 12GB in memory, and also requires 3 iterations until 1e-6, and then runs until the maximum number of

refinement loops allowed (by default: 20). However, it converges to a residual of around 1e-12 in a mere 5 steps but then cannot reach its own (very strict) stopping criteria.

Compared to the other eigensolvers, the FEAST eigensolver requires many more linear systems to be solved, however, this method still has some benefits. Due to the algorithm, parallelization is straight forward. Availability and level of implementation are also in favor of this eigensolver.

8.3.5 Conclusion on Eigensolvers

Before we turn to investigate efficient ways to solve the arising linear systems, we want to give a short summary of the discussed methods.

In general, there are two extremes. Methods, which are purely based on polynomial matrix-vector multiplications with A, allow for cheap iteration steps, but require a very large number of iterations. For methods entirely based on inverse iteration, quick convergence can be expected, yet each step is costly.

The performance of methods based on inverse iteration depends strongly on the efficiency when solving the resulting linear systems. If they can be solved at fair cost, all inverse based methods (IRAM in shift-and-invert mode, JADAMILU, FEAST) are quite efficient. Actually, there is not much of a difference, if each of the method's parameters is chosen suitably. In fact, the literature suggests that a simple shift-and-invert Lanczos version is just as good, or maybe even better than contour integral based spectral projection methods [161].

Some of the methods presented (JADAMILU, FEAST) can benefit from a properly chosen starting space. Sometimes, a good starting space is available at low cost. In those cases, it seems advisable to use such a method. Later, when presenting the multilevel approach, we also try to guess certain subspaces corresponding to slow motion. The ideas discussed there could also be used to construct a proper starting space. However, we did not investigate this possibility further.

8.4 Solving the Shifted Linear System

The linear systems are of the type

$$(M^{-1/2}\,\mathcal{H}\,M^{-1/2} - \sigma I)x = b\ ,$$

which we abbreviate by $Ax = b$. For testing, we set $\sigma = 10$ unless otherwise stated.

First, we show how direct methods perform for solving the linear system. Afterwards, we investigate iterative methods. The multilevel approach is evaluated in Section 8.5.

8.4.1 Direct Methods: Factorization of Matrices

When using a direct method to solve the linear system $Ax = b$, with symmetric, and indefinite A, the main step is computing the factorization $A = LDL^T$. This task is not

trivial for large sparse matrices. By reordering the matrix A, the fill-in in the L factor is reduced. See [162] for a comparison and details of some implementations of available direct solvers.

Here, we compare the sparsity and the fill-in obtained by reordering the matrix A with the reverse Cuthill-McKee algorithm (RCM) and the symmetric approximate minimum degree permutation (AMD). For problems H3 and H4, we plot the reordered matrix and the obtained L factors (see Figure 8.12 and Figure 8.13). It is nice to observe how the RCM attempts to minimize the bandwidth, and AMD tries to obtain an arrowhead structure.

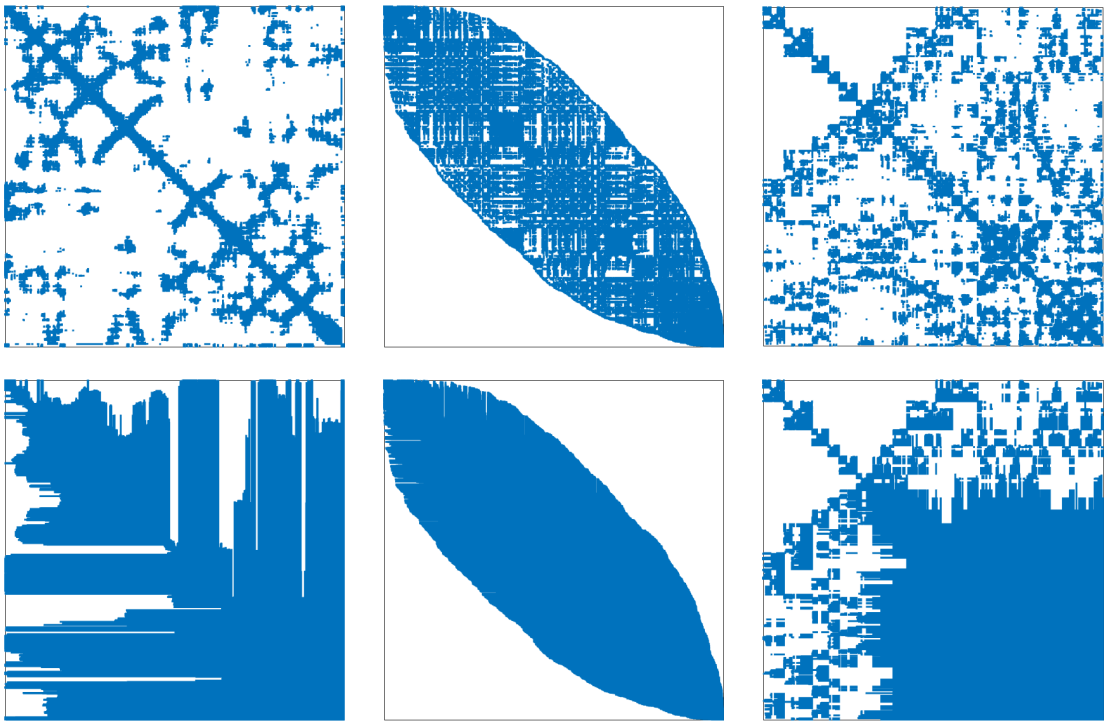


Figure 8.12: *H3: sparsity pattern of the reordered matrices. In top row, we plot the reordered Hessian matrices, in the bottom row, we plot the corresponding $L + L^T$ factor. Left: unsorted, middle: RCM, right: AMD, nz denotes the number of nonzero entries.*

In Table 8.14, we list the number of nonzero entries (nnz) in the matrix A and the L factors. By $\text{nnz}(L_{\text{rcm}})$ we denote the number of nonzero entries in the matrix L as obtained from the LDL^T decomposition applied to the matrix A reordered with the RCM. A similar notation is used for the AMD. We also list the memory requirement in MB for storing each matrix. In contrast to the values listed earlier, here, we use the storage required by MATLAB without taking advantage of the symmetry. Also, MATLAB uses 8-byte integers for storing the row index of non-zero elements. Thus, the values for $\#\text{MB}(A)$ are more than twice as large compared to the .mtx format reported earlier.

The number of non-zero entries in the matrix L is significant lower when using a reordering strategy. The RCM performs well on the three small Hessians, however, it is less efficient than the AMD on the larger problems. On problem H4, the RCM fails to decrease the fill-in at all. For a sparse matrix, the number of non-zero entries is directly

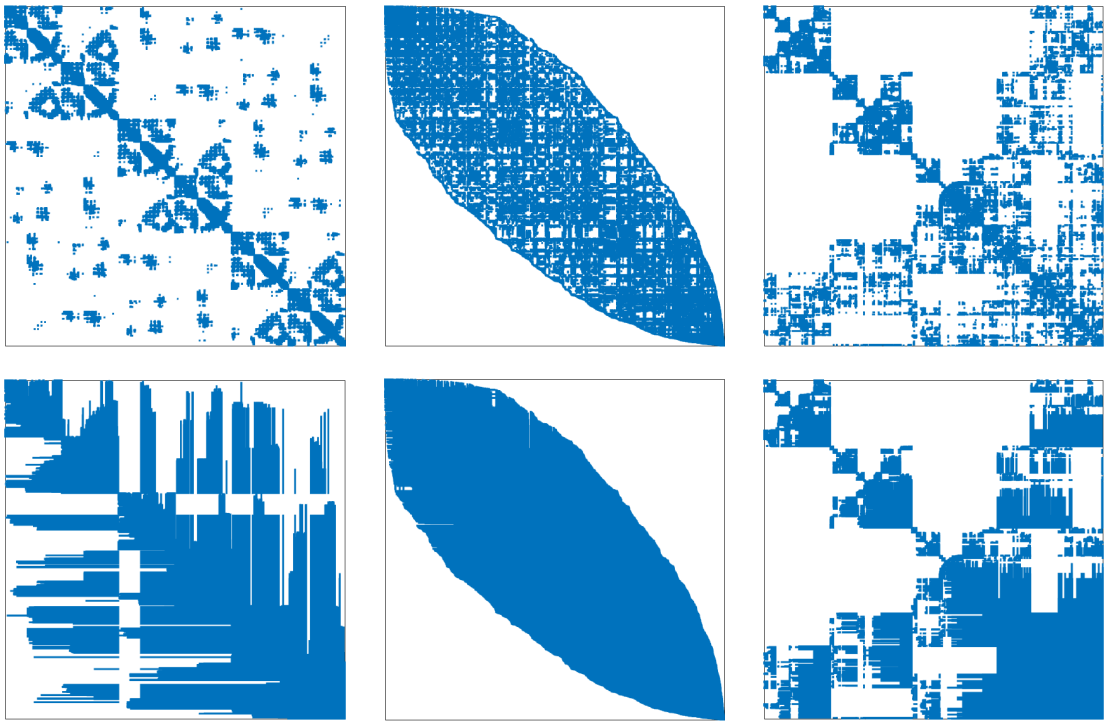


Figure 8.13: *H4: sparsity pattern of the reordered matrices. In top row, we plot the reordered Hessian matrices, in the bottom row, we plot the corresponding $L + L^T$ factor. Left: unsorted, middle: RCM, right: AMD, nz denotes the number of nonzero entries.*

PDB ID	1pgb	4pti	1tng	1ai0	5c7x	2bg9
#dof	2,565	2,676	9,741	14,220	48,495	89,637
nnz(A)	1.76m	1.75m	8.50m	10.7m	42.8m	76.1m
nnz(L)	2.82m	2.80m	33.8m	45.5m	307m	1,077m
nnz(L_{RCM})	1.88m	1.92m	22.0m	46.5m	234m	634m
nnz(L_{AMD})	2.60m	2.58m	24.7m	31.5m	136m	496m
#MB(A)	26.8	26.8	130	164	653	1,161
#MB(L)	43.1	42.8	516	694	4,677	16,437
#MB(L_{RCM})	28.5	29.4	336	710	3,571	9,682
#MB(L_{AMD})	39.7	39.3	377	480	2,078	7,569
	H1	H2	H3	H4	H5	H6

Table 8.14: *Fill-in and memory usage of the LDL^T decomposition applied to the matrix A reordered with RCM and AMD*

related to the amount of memory required to store the matrix. Hence, the reordered L factors are less demanding in memory. Though, they quickly fill a couple of GB even for the problems under consideration here.

The amount of fill-in is large. To estimate the amount of fill-in ϵ_{fill}, we divide the number of non-zero entries in L_{AMD} or L_{RCM} (whichever is lower) by the number of non-zero entries in the lower triangular part of A, that is approximately $\text{nnz}(A)/2$:

$$\epsilon_{\text{fill}} = \frac{2\min\{\text{nnz}(L_{\text{AMD}}), \text{nnz}(L_{\text{RCM}})\}}{\text{nnz}(A)}.$$

For problems H1 to H6 we obtain the fill-in factors 2.14, 2.19, 5.18, 5.89, 6.36, and 13.0.

Let us make a crude estimate of the computational cost of a regular Lanczos iteration compared to a shift-and-invert iteration. The main computational task of the Lanczos iteration is a matrix-vector multiplication with A. This costs approximately $2{\cdot}\text{nnz}(A)$ operations each. For the shift-and-invert method, first we need to compute the factorization $A = LDL^T$ once. Each step then requires the triangular and diagonal solves $Lz = b$, $Dy = z$ and $L^T x = y$ for some right-hand side b.

Let us ignore the factorization cost for the moment, and compare the cost for each iteration. The two triangular solves require approximately $2{\cdot}\text{nnz}(L)$ operations each, and the diagonal solve is of negligible computational cost. Each step in the shift-and-invert iteration thus costs $4{\cdot}\text{nnz}(L) = 2 \cdot \epsilon_{\text{fill}}{\cdot}\text{nnz}(A)$ operations. Therefore, the fill-in factor ϵ_{fill} provides us with an estimate of how much more expensive solving $Ax = b$ is compared to a matrix-vector multiplication with A.

Obviously, the factorization cost is the main computational task for the shift-and-invert iteration. In the end, it comes down to the question whether the LDL^T factorization can be computed cheaper than the cost of the many matrix-vector multiplications in the Lanczos iteration. For example, in problem H5, the IRAM requires 28,850 matrix-vector multiplications until convergence. If we subtract the cost for the iterations in the shift-and-invert iteration ($\approx 150 \cdot \epsilon_{\text{fill}}$), we see that shift-and-invert iteration breaks even when the factorization is cheaper than approximately 27,896 matrix-vector multiplications. We want to emphasize again that this a very crude estimate since we simply neglected the computational cost of anything other than the matrix-vector multiplications for both algorithms.

8.4.2 Iterative Methods

Iterative methods are usually employed when solving large, sparse linear systems of equations. However, the MINRES method without preconditioning requires many iterations until convergence. In Table 8.15, we list the number of iterations until a relative residual of approximately 1.49e-08 ($= \sqrt{\text{mach. prec.}}$) has been obtained. The number of iterations for solving $Ax = b$ is averaged over ten random vectors b.

Clearly, the number of iterations is unacceptably high, and one should not use an iterative method without a preconditioner for these applications.

PDB ID	1pgb	4pti	1tng	1ai0	5c7x	2bg9
n_{it}	916.2	1,082	1,558	1,925	6,778	13,751
	H1	H2	H3	H4	H5	H6

Table 8.15: *Number of iterations n_{it} for the MINRES Method without using a preconditioner*

8.4.3 Preconditioned Iterative Methods

ILUPACK [154] is a very versatile software package for solving large and sparse linear systems. At its core, it computes an incomplete LU or LDL^T decomposition which then is used as a preconditioner in a linear solver. Trying to keep as much sparsity as possible, it offers many different reordering strategies. By adjusting drop tolerances, the fill-in can be controlled. Furthermore, ILUPACK provides the ability to use a multilevel framework for the preconditioner. Finally, several built-in iterative solvers such as the symmetric QMR complete the package.

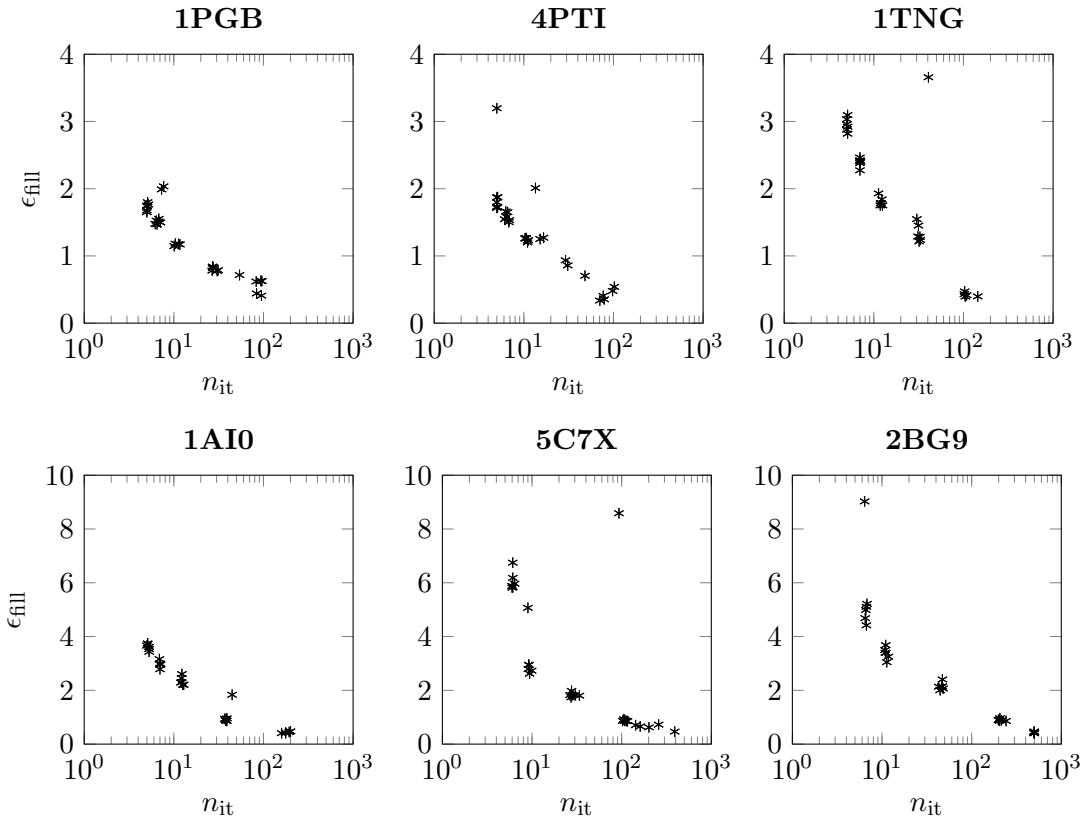


Figure 8.16: *Fill-in ϵ_{fill} vs number of iterations n_{it} with ILUPACK on Hessians H1-H3 (top row) and H4-H6 (bottom row)*

Thus, ILUPACK allows for balancing between the effort for precomputing a factorization and the number of iterations required in the iterative solver. In general, the more expensive the factorization (both in computing time and storage), the less iterations are

required later (and vice versa). Since the eigensolvers required around 150 solves, and the preconditioner only needs to be computed once, it is worth investing some effort into the decomposition.

We present the results as obtained by ILUPACK in the following way. For each model, we compute the factorization for all available reordering methods and a number of different drop tolerances. ILUPACK has six different reordering strategies, and we chose drop tolerances from 1e-2 to 1e-6. Then, for each factorization, we compare the fill-in to the number of iterations the preconditioned symmetric QMR requires on average to solve a linear system. In Figure 8.16 we plot the results for the Hessian matrices.

We do not specify to which reordering strategy each of the points belong. In fact, there is no clear superior or inferior reordering method. As we can see, the overall differences are fairly low. The results clearly show the interplay between the fill-in ϵ_{fill} and the number of iterations n_{it}. A low number of iterations in the solver can be obtained with a high fill-in and vice-versa.

E.g. for 2BG9, if a fill-in of 4 can be handled, the method allows for convergence in around 10 steps. So, ignoring all other costs, the eigensolver requires 150 solves times 10 iterations. Each iteration requires a preconditioned application of A, so with a fill-in of 4, this costs roughly 5 MV. In total this is of the order 7,500 MV ($\ll 35,117$ MV what the standard IRAM requires).

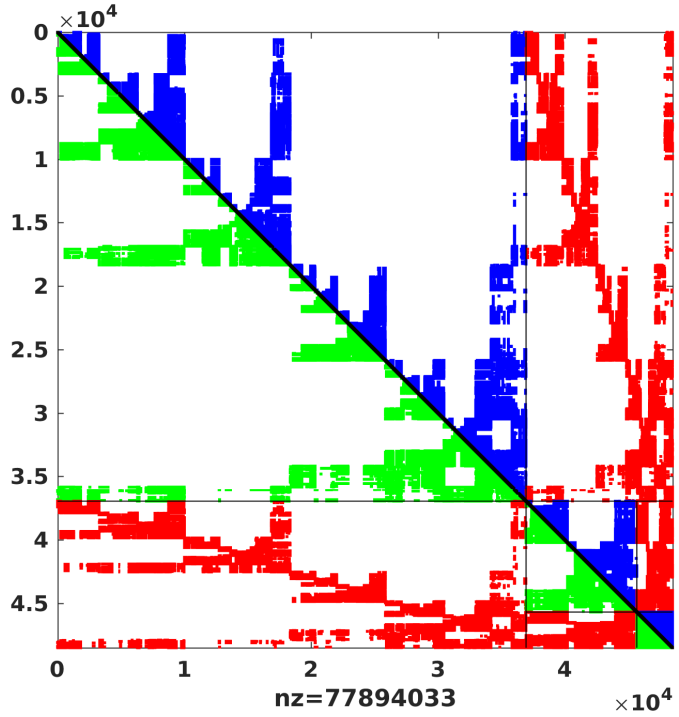


Figure 8.17: *ILUPACK preconditioner for Hessian H5, three levels, reordering 'metise', drop tolerance 1e-4*

In Figure 8.17 we plot the multilevel decomposition for the model 5C7X. We chose the

reordering strategy 'metise', and used a drop tolerance of 1e-4. The green and blue parts are just the transpose of each other (since it is a symmetric problem). In fact, only one set is stored. The red parts are the connection between levels, of which there are three in this example. For more information about the multilevel strategy we refer to [154].

8.5 Multilevel Algorithm

In this section, we present the numerical results when using a multilevel approach such as proposed in Section 7.4. We compare the numerical convergence factors obtained when using different subspaces. In detail, we show the results for BPTI, that is, the model problem H2. These results are presented throughout Section 8.5.1 to Section 8.5.4. Afterwards, in Section 8.5.5, we give results for H4. A short summary of the results for H6 can be found in Section 8.5.6.

Recall, that BPTI has 892 atoms, and hence the Hessian is a 2676-by-2676 matrix. BPTI has 58 residues. We construct the subspaces based on the ideas presented earlier, and test them together with the Richardson and Gauss-Seidel iteration in the two-level algorithm (7.32). We did not use the Jacobi iteration, since it never converges for the problems under consideration here. Also, we start by considering the two-level algorithm (Algorithm 7.6) which has one coarse and one fine level. In particular, that means that the coarse grid correction equation is solved exactly. Later, we will also consider a hierarchy of levels with more than two levels (see Section 8.5.4).

8.5.1 Subspaces Based on Translation and Rotation

Subspaces following the idea proposed in Section 7.4.4 are very easy to construct. They do not even require any knowledge of the Hessian. Instead, they are purely based on the underlying structure.

Using simple heuristics, we construct a sequence of seven subspaces. A very coarse subspace of dimension 348 is obtained by simply using one translation-and-rotation cluster for each residue. For constructing a finer subspace, we use the strategy employed to construct the corotational filter: we force all hydrogen bonds (which are expected to be the fastest bonds) into clusters. Usually, a cluster decomposition with clusters of sizes two to four is obtained. Atoms which do not have a hydrogen bond form clusters of size one. The resulting subspace is of dimension 1987.

Starting at these two 'basic' subspaces, we construct a total of seven subspaces by dividing or combining clusters. The full list with a short description is given in Table 8.18. We abbreviate the subspaces by RT1 to RT7.

In Figure 8.19, we plot the original Hessian aside two coarse Hessians which correspond to the Galerkin projection with RT5 and RT2. We now evaluate the performance as follows: for each combination of smoother and subspace, we compute 20 steps of the two-level cycle for ten random vectors b with unit length. We then plot the norm of the residual versus

ID	dimension	description
RT1	174	two residues per cluster
RT2	348	one cluster per residue
RT3	510	split clusters larger than 14 atoms in subspace 2
RT4	738	split clusters larger than 10 atoms in subspace 2
RT5	1499	subspace 7 but try to merge clusters of size one and two
RT6	1937	subspace 7 but prohibit clusters with one atom
RT7	1987	cluster decomposition based on hydrogen bonds

Table 8.18: *List of the constructed translation-and-rotation subspaces for H2*

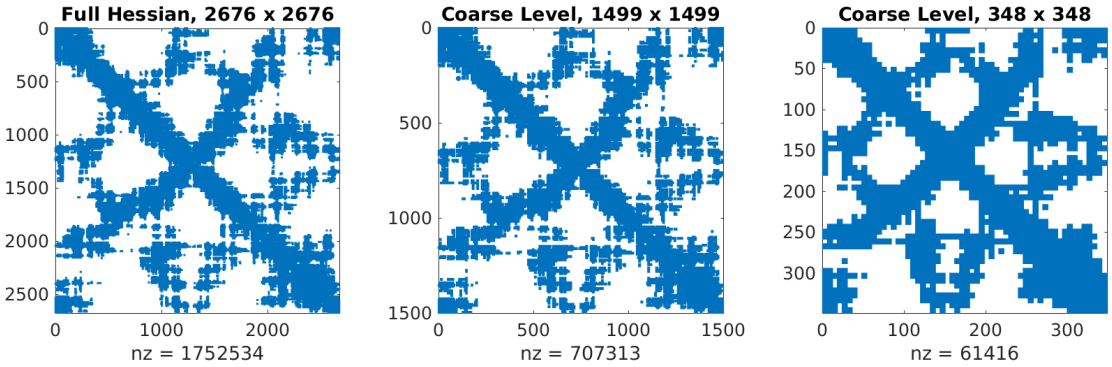


Figure 8.19: *Sparsity pattern of full and coarse Hessians for subspaces RT5 and RT2*

the iteration number. Also, we compute the numerical convergence factor as the quotient between two consecutive residual norms.

In our experiments, the Gauss-Seidel method converges much faster than the Richardson iteration. Therefore, in Algorithm 7.6 we set $\mu_1 = \mu_2 = 1$ when using the Gauss-Seidel method as smoother, and $\mu_1 = \mu_2 = 2$ when using the Richardson iteration. In Figure 8.20 we plot the results for all subspaces listed in Table 8.18. On the left, we plot the relative residual norm

$$||r_i|| = ||b - Ax_i||/||b||$$

where x_i is the approximation obtained after i steps with the two-level algorithm. On the right, we plot a numerical convergence factor

$$c_i = ||r_i||/||r_{i-1}||.$$

All methods converge with some factor smaller than 1. However, the convergence speed is extremely slow for the subspaces RT1 to RT4. This is not really surprising, since these subspaces are quite small.

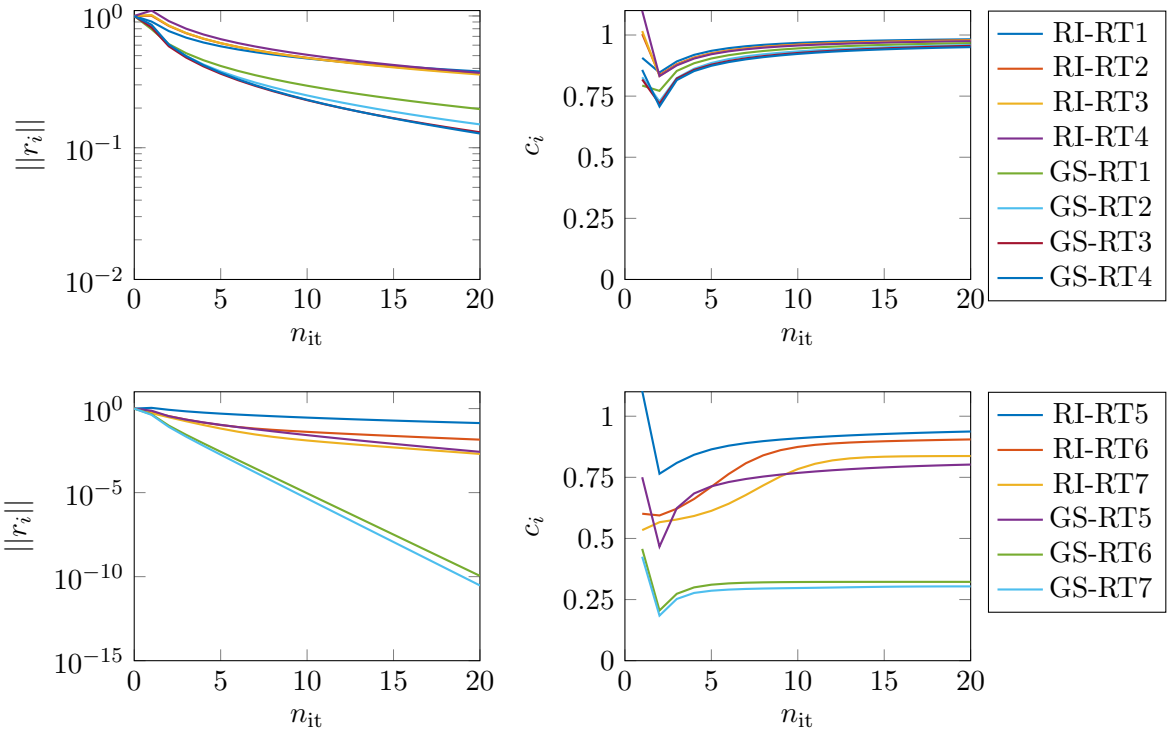


Figure 8.20: *Relative residual norm $||r_i||$ and convergence factor c_i for subspaces RT1-RT4 (top row) and RT5-RT7 (bottom row) in combination with Richardson iteration (RI) and Gauss-Seidel iteration (GS)*

Much more promising are the results for the larger subspaces RT5-RT7. We can observe a very good convergence speed for the two large subspaces RT6 and RT7 combined with the Gauss-Seidel method. However, the dimension of these subspaces is fairly large. The Gauss-Seidel method with the subspace RT5 performs well, considering that subspace RT5 is of roughly half the dimension of the original Hessian.

We give the convergence factor and residual at step 20 for other numbers of smoothing steps in a table. In Table 8.21, this is done for the Richardson iteration. We use two, four and ten smoothing steps each for pre- and for post-smoothing. Good convergence is obtained for subspaces RT6 and RT7 when using many smoothing steps.

In Table 8.22, the results for the Gauss-Seidel iteration are plotted. We use one, two and five smoothing steps each for pre- and post-smoothing. The results are much better compared to the Richardson iteration, and a convergence factor of $c_{20} = 0.268$ can be obtained. As before, though, this corresponds to choosing a fairly large subspace and using many pre- and post-smoothing iterations. Considering the dimension of the subspace, RT5 again performs very well. The convergence factor is between $c_{20} = 0.8023$ and $c_{20} = 0.6719$.

Let us investigate how well the exact eigenvectors $v_i \in V$ of the Hessian can be represented in each of the subspaces P. As a measure, we compute the norm of the projection of v_i onto P, that is $||PP^T v_i||$. Here, P is an orthonormal basis of the subspace, and

	$\mu_1 = \mu_2 = 2$		$\mu_1 = \mu_2 = 4$		$\mu_1 = \mu_2 = 10$	
ID	$\|\|r_{20}\|\|$	c_{20}	$\|\|r_{20}\|\|$	c_{20}	$\|\|r_{20}\|\|$	c_{20}
RT1	3.792e-01	0.9825	2.852e-01	0.9791	1.956e-01	0.9742
RT2	3.656e-01	0.9789	2.730e-01	0.9759	1.662e-01	0.9685
RT3	3.591e-01	0.9774	2.525e-01	0.9723	1.495e-01	0.9659
RT4	3.702e-01	0.9748	2.497e-01	0.9702	1.206e-01	0.959
RT5	1.372e-01	0.9375	4.222e-02	0.9127	8.088e-03	0.8648
RT6	1.417e-02	0.9051	2.849e-03	0.8433	3.453e-05	0.6844
RT7	2.005e-03	0.8372	8.071e-05	0.7042	7.258e-09	0.4368

Table 8.21: *Richardson iteration and subspaces RT1-RT7: influence of number of smoothing steps on numerical convergence factor and relative residual norm as obtained after 20 steps.*

	$\mu_1 = \mu_2 = 1$		$\mu_1 = \mu_2 = 2$		$\mu_1 = \mu_2 = 5$	
ID	$\|\|r_{20}\|\|$	c_{20}	$\|\|r_{20}\|\|$	c_{20}	$\|\|r_{20}\|\|$	c_{20}
RT1	1.964e-01	0.9669	1.082e-01	0.9569	4.871e-02	0.9425
RT2	1.503e-01	0.959	8.200e-02	0.9506	3.243e-02	0.9332
RT3	1.310e-01	0.9556	8.305e-02	0.9496	2.871e-02	0.9276
RT4	1.282e-01	0.9504	5.414e-02	0.9325	1.946e-02	0.9014
RT5	2.671e-03	0.8023	3.004e-04	0.742	1.985e-05	0.6719
RT6	1.111e-10	0.3224	2.979e-11	0.2945	7.198e-13	0.3025
RT7	2.996e-11	0.3039	5.768e-11	0.2777	3.576e-12	0.268

Table 8.22: *Gauss-Seidel iteration and subspaces RT1-RT7: influence of number of smoothing steps on numerical convergence factor and relative residual norm as obtained after 20 steps.*

$v_i \in V_{\text{exact}}$ is an exact, orthonormal eigenvector. E.g. $||PP^T v_i|| = 1$ if the subspace entirely contains the eigenvector v_i, and $||PP^T v_i|| = 0$ if the subspace is orthogonal to that eigenvector.

The results are plotted in Figure 8.23 for the subspaces RT1-RT7. For visual purposes, we sorted the eigenvectors in V by their eigenvalue in ascending order, that is, the eigenvectors corresponding to the lowest eigenvalues are left.

Clearly, Figure 8.23 indicates why some of the subspaces are much better than others. Also, when dividing the decomposition from subspace RT2 into the decomposition for subspace RT4, we did not respect the underlying molecular structure but simply cut the clusters in half. Apparently, this is not a good strategy, since subspace RT4 has the highest

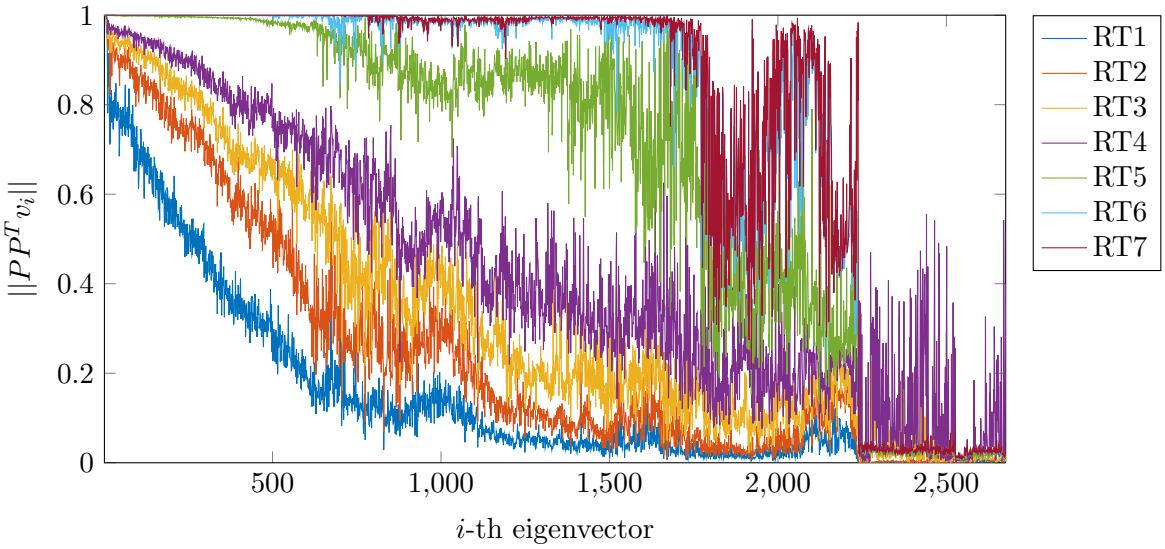


Figure 8.23: *Representation of an eigenvector in the subspaces RT1-RT7. Eigenvectors are sorted in ascending order.*

overlap with fast eigenvectors.

8.5.2 Improving the Subspace with Dihedrals

The molecular structure of BPTI has some salt bridges, and thus long loops inside the main chain. Hence, a direct approach with dihedrals does not work. Instead, we again decompose the molecule into clusters corresponding to the residues. For each residue, we compute the six translation and rotation vectors (compare to subspace RT2 in the previous section) and then inject further vectors based on the torsional directions indicated by some of the dihedrals inside the cluster.

Our chosen selection strategy was to exclude all dihedrals which connect residues. Also, we exclude all dihedrals which are part of a loop within a residue (e.g. aromatic carbon ring). Then, the obtained subspace has dimension 611. We abbreviate this subspace with TO (for torsion).

	$\mu_1 = \mu_2 = 2$		$\mu_1 = \mu_2 = 4$		$\mu_1 = \mu_2 = 10$	
ID	$\|\|r_{20}\|\|$	c_{20}	$\|\|r_{20}\|\|$	c_{20}	$\|\|r_{20}\|\|$	c_{20}
RT2	3.656e-01	0.9789	2.730e-01	0.9759	1.662e-01	0.9685
TO	3.540e-01	0.9749	2.246e-01	0.9681	1.226e-01	0.9615

Table 8.24: *Richardson iteration and subspaces RT2 and TO: influence of number of smoothing steps on numerical convergence factor and relative residual norm as obtained after 20 steps*

The results of that strategy are given in Table 8.24 and Table 8.25. They do not seem to provide a specific benefit. The new subspace TO gives results which are in between the

results for subspaces RT3 and RT4 of the precious section.

	$\mu_1 = \mu_2 = 1$		$\mu_1 = \mu_2 = 2$		$\mu_1 = \mu_2 = 5$	
ID	$\|\|r_{20}\|\|$	c_{20}	$\|\|r_{20}\|\|$	c_{20}	$\|\|r_{20}\|\|$	c_{20}
RT2	1.503e-01	0.959	8.200e-02	0.9506	3.243e-02	0.9332
TO	1.054e-01	0.9519	5.722e-02	0.9445	2.567e-02	0.9315

Table 8.25: *Gauss-Seidel iteration and subspaces RT2 and TO: influence of number of smoothing steps on numerical convergence factor and relative residual norm as obtained after 20 steps*

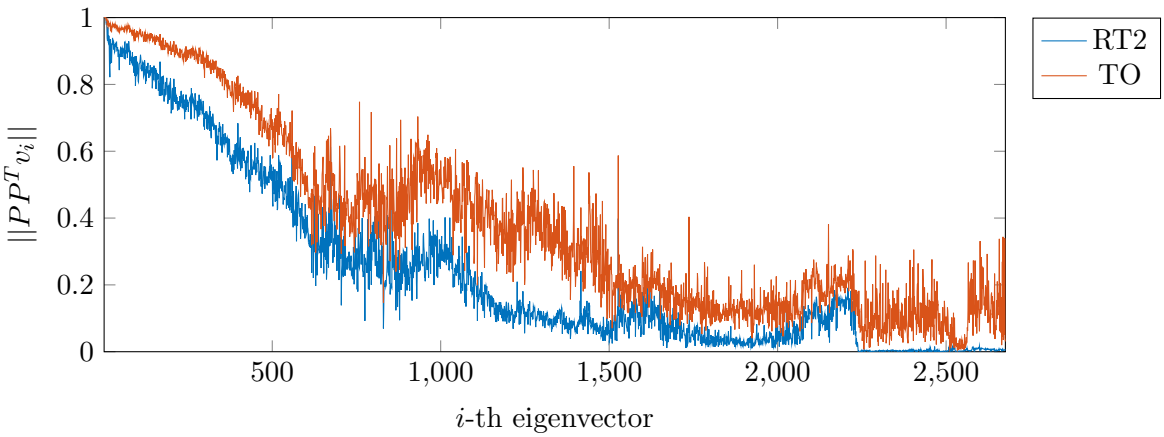


Figure 8.26: *Representation of an eigenvector in the subspaces RT2 and TO. Eigenvectors are sorted in ascending order.*

Similar to Figure 8.23, we compare the eigenspace overlap for the dihedral space. The results are given in Figure 8.26. There we see, that the dihedral space is a bit better than the RT2, however, not much better.

8.5.3 Domain Decomposition and Subspaces

Domain decomposition as a basis for constructing a subspace was discussed in Section 7.4.6. We divide BPTI in its 58 residues. Then, for each residue, we first compute the six translation and rotation vectors, and then elevate that subspace by a number of slow eigenvectors of the Hessian restricted to that residue. Basically, the large eigenvalue problem is divided into many smaller eigenvalue problems. In the case of BPTI, 58 small eigenvalue problems with sizes in between 21-by-21 and 78-by-78 have to be solved.

For each residue we take $\lceil \eta \cdot N_{res} \rceil$ additional slow eigenvectors, where η is a scaling factor and N_{res} denotes the number of degrees of freedom of that residue. We construct a series of subspaces with $\eta = 0.1, 0.2, ..., 0.6$. We list their dimension in Table 8.27.

We plot the sparsity pattern of three matrices in Figure 8.28. Left is the original Hessian, the middle one is the coarse Hessian corresponding to subspace DD4, and the right

ID	DD1	DD2	DD3	DD4	DD5	DD6
dimension	644	909	1170	1437	1696	1976
η	0.1	0.2	0.3	0.4	0.5	0.6

Table 8.27: *List of the constructed domain decomposition subspaces*

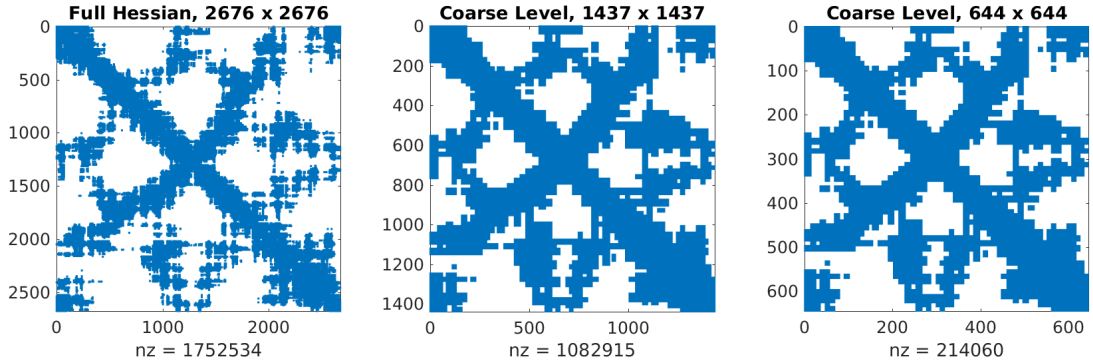


Figure 8.28: *Sparsity pattern of full and coarse Hessians for subspaces DD4 and DD1*

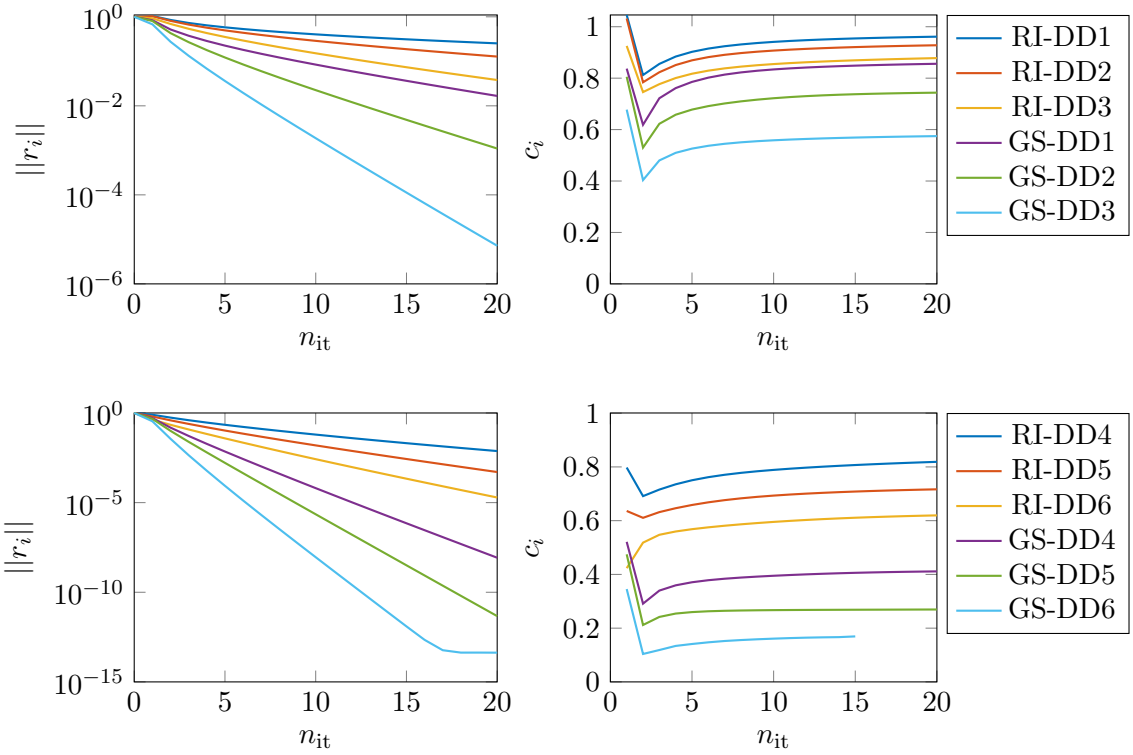


Figure 8.29: *Relative residual norm $||r_i||$ and convergence factor c_i for subspaces DD1-DD3 (top row) and DD4-DD6 (bottom row) in combination with Richardson iteration (RI) and Gauss-Seidel iteration (GS)*

matrix is the coarse Hessian for DD1. In contrast to the subspaces based on translation and rotation, visually, there is not much of a resolution difference between the two coarse

Hessians. The reason is that the subspaces here are always based on an entire residue, and not on smaller entities.

As before, we plot results for $\mu_1 = \mu_2 = 1$ (Richardson iteration: $\mu_1 = \mu_2 = 2$) smoothing steps in Figure 8.29. Again, the methods converge with some factor smaller than 1. The convergence speed is much better compared to the subspaces RT1-RT7.

More results for different numbers of smoothing steps can be found in Table 8.30 and Table 8.31. Note that for the methods converging to machine precision in less than 20 steps, we give the convergence factor just before convergence. For all others we use the numerical convergence factor after 20 steps.

	$\mu_1 = \mu_2 = 2$		$\mu_1 = \mu_2 = 4$		$\mu_1 = \mu_2 = 10$	
ID	$\|\|r_{20}\|\|$	c_{20}	$\|\|r_{20}\|\|$	c_{20}	$\|\|r_{20}\|\|$	c_{20}
DD1	2.532e-01	0.9613	1.442e-01	0.9477	4.379e-02	0.9149
DD2	1.279e-01	0.928	3.741e-02	0.884	2.579e-03	0.7861
DD3	3.802e-02	0.878	5.696e-03	0.8103	6.779e-05	0.6642
DD4	7.526e-03	0.8189	4.227e-04	0.7207	7.211e-07	0.5461
DD5	5.086e-04	0.7167	3.067e-06	0.5608	4.879e-11	0.3345
DD6	1.930e-05	0.6197	2.648e-09	0.402	4.252e-14	0.1448

Table 8.30: *Richardson iteration and subspaces DD1-DD6: influence of number of smoothing steps on numerical convergence factor and relative residual norm as obtained after 20 steps.*

	$\mu_1 = \mu_2 = 1$		$\mu_1 = \mu_2 = 2$		$\mu_1 = \mu_2 = 5$	
ID	$\|\|r_{20}\|\|$	c_{20}	$\|\|r_{20}\|\|$	c_{20}	$\|\|r_{20}\|\|$	c_{20}
DD1	1.668e-02	0.8559	2.157e-03	0.7931	1.653e-04	0.7092
DD2	1.098e-03	0.7436	3.519e-05	0.6433	2.082e-06	0.5809
DD3	7.181e-06	0.5746	2.555e-07	0.5008	1.854e-08	0.4516
DD4	8.447e-09	0.4113	2.040e-10	0.3508	3.073e-12	0.2979
DD5	4.667e-12	0.2694	7.614e-13	0.2314	3.358e-11	0.1917
DD6	7.326e-12	0.1666	3.959e-12	0.09701	2.320e-12	0.08191

Table 8.31: *Gauss-Seidel iteration and subspaces DD1-DD6: influence of number of smoothing steps on numerical convergence factor and relative residual norm as obtained after 20 steps.*

In general, we can expect faster convergence using the Gauss-Seidel method instead of the Richardson iteration. Even when using a single pre- and post-smoothing step with the Gauss-Seidel method, the two-level algorithm combined with DD4 converges with a

factor of 0.4113. With more smoothing steps, faster convergence can be achieved.

Finally, we plot the eigenspace-subspace overlap in Figure 8.32. Compared to the subspaces RT1-RT7, the subspaces DD1-DD6 are much better in reproducing the slow eigenvectors.

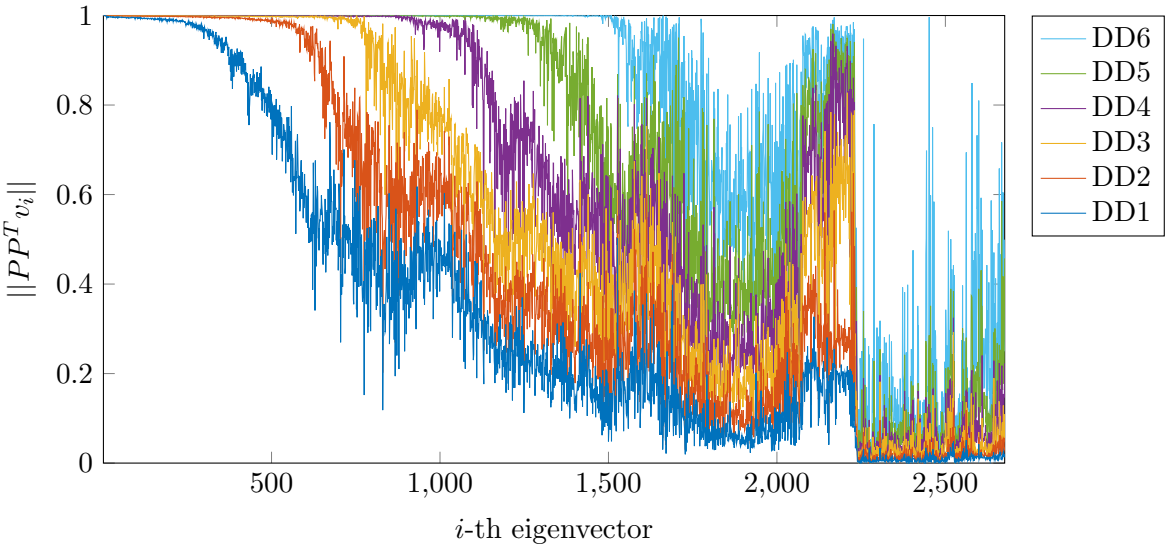


Figure 8.32: *Representation of an eigenvector in the subspaces DD1-DD6. Eigenvectors are sorted in ascending order.*

8.5.4 Multilevel Algorithms with Three and Four Levels

So far, we only considered the two-level cycle. Here, we show numerical convergence for one three-level algorithm and one four-level algorithm (cf. Algorithm 7.7). In general, it is advantageous to choose subspaces from the same category (e.g. either from the RT* series or the DD* series) because we can then be assured that prolongation and restriction operators between levels are guaranteed to have beneficial properties such as sparsity and simplicity. We only give results for subspaces in the DD* series, since they yield better results. Algorithms using translation and rotation subspaces also work, yet at slightly worse factors.

We start with three levels, and construct a three-level algorithm with subspaces from domain decomposition, specifically, subspaces DD1 and DD4. Note that subspace DD4 is a true extension to subspace DD1, so there is a simple prolongation and interpolation between them. The corresponding matrices are the ones shown in Figure 8.28. We investigate different numbers of smoothing steps, and also show results for V- and W-cycles (see Table 8.33 and Table 8.34). We choose the same number of pre- and post-smoothing steps for all levels, so setting $\mu_1 = \mu_2 = 2$ is short for setting $\mu_1^l = \mu_2^l = 2,\ l = 1, 2$ in Algorithm 7.7.

The convergence speed is somewhere in between the expected factors for the smaller and larger subspaces. Clearly, convergence can be accelerated by choosing more smoothing

	$\mu_1 = \mu_2 = 2$		$\mu_1 = \mu_2 = 4$		$\mu_1 = \mu_2 = 10$	
	$\|\|r_{20}\|\|$	c_{20}	$\|\|r_{20}\|\|$	c_{20}	$\|\|r_{20}\|\|$	c_{20}
V-cycle	1.371e-01	0.9461	5.087e-02	0.9197	1.077e-02	0.8676
W-cycle	9.677e-02	0.9336	3.068e-02	0.8999	2.561e-03	0.8127

Table 8.33: *Richardson iteration for a three-level algorithm with DD1 and DD4: influence of number of smoothing steps on numerical convergence factor and relative residual norm as obtained after 20 steps.*

	$\mu_1 = \mu_2 = 1$		$\mu_1 = \mu_2 = 2$		$\mu_1 = \mu_2 = 5$	
	$\|\|r_{20}\|\|$	c_{20}	$\|\|r_{20}\|\|$	c_{20}	$\|\|r_{20}\|\|$	c_{20}
V-cycle	5.246e-04	0.7326	4.999e-05	0.6703	1.747e-06	0.5861
W-cycle	9.089e-06	0.6025	1.594e-07	0.5179	2.621e-09	0.4208

Table 8.34: *Gauss-Seidel iteration for a three-level algorithm with DD1 and DD4: influence of number of smoothing steps on numerical convergence factor and relative residual norm as obtained after 20 steps.*

steps in the inner levels. Then, the limit convergence factor is expected to be close to the factor of the two-level algorithm with exact inner solves.

It makes much more sense to choose different smoothing steps for the different levels. In a four-level approach, we use subspaces DD1, DD2 and DD4 and choose as smoothing steps $\mu_1^1 = \mu_2^1 = 1$, $\mu_1^2 = \mu_2^2 = 3$ and $\mu_1^3 = \mu_2^3 = 5$ (outer to inner levels, cf. Algorithm 7.7) for the Gauss-Seidel iteration. As before, we choose twice the number of smoothing steps for Richardson iteration. The following results are obtained (see Table 8.35).

	V-cycle		W-cycle	
	$\|\|r_{20}\|\|$	c_{20}	$\|\|r_{20}\|\|$	c_{20}
RI	3.063e-02	0.8854	1.661e-02	0.8448
GS	1.443e-05	0.6128	3.689e-08	0.4376

Table 8.35: *Richardson (RI) and Gauss-Seidel (GS) iteration for a four-level algorithm with DD1, DD2 and DD4: numerical convergence factor and relative residual norm as obtained after 20 steps for V- and W-cycle.*

Here, for the W-cycle, the results are very close to the corresponding results reported in Table 8.30 and Table 8.31. Especially, using only one (Richardson: two) outer smoothing steps make this choice much cheaper than the only slightly better results in Table 8.33 and Table 8.34.

8.5.5 Results for H4

After giving the results in detail for H2, we now quickly give some of the results for H4. H4 has around five times the number of atoms compared to H2, and its Hessian is a 14,220-by-14,220 matrix.

ID	H4-RT1	H4-RT2	H4-RT3	H4-RT4	H4-RT5	H4-RT6	H4-RT7
dim.	918	1,836	2,988	3,780	7,572	10,351	10,525

Table 8.36: *List of the constructed rotation-and-translation subspaces for H4*

In Table 8.36, we list the dimension of the rotation and translation subspaces. To avoid confusion, we abbreviate those subspaces with H4-RT*. We use exactly the same heuristics as detailed in Table 8.18.

	$\mu_1 = \mu_2 = 1$		$\mu_1 = \mu_2 = 2$		$\mu_1 = \mu_2 = 5$	
ID	$\|\|r_{20}\|\|$	c_{20}	$\|\|r_{20}\|\|$	c_{20}	$\|\|r_{20}\|\|$	c_{20}
H4-RT1	1.832e-01	0.9658	1.086e-01	0.9584	4.870e-02	0.9498
H4-RT2	1.615e-01	0.9632	8.158e-02	0.9519	3.632e-02	0.9415
H4-RT3	1.436e-01	0.9598	7.590e-02	0.9493	2.902e-02	0.9419
H4-RT4	1.200e-01	0.9546	6.763e-02	0.9496	3.033e-02	0.9419
H4-RT5	1.719e-02	0.8965	6.701e-03	0.8654	9.728e-04	0.8154
H4-RT6	2.515e-09	0.3952	6.547e-11	0.3397	7.406e-12	0.3226
H4-RT7	3.295e-10	0.3539	1.339e-11	0.3163	1.670e-12	0.297

Table 8.37: *Gauss-Seidel iteration and subspaces H4-RT1 to H4-RT7: influence of number of smoothing steps on numerical convergence factor and relative residual norm as obtained after 20 steps.*

For brevity, we only give results for the Gauss-Seidel iteration (Table 8.37). Similar to the behavior on H2, the convergence factor is good for the large subspaces H4-RT6 and H4-RT7. For the smaller subspaces, the convergence is significantly slower.

When we follow the approach of domain decomposition for H4, we obtain the subspaces listed in Table 8.38. Again, we only give results with the Gauss-Seidel iteration (see Table 8.39).

ID	H4-DD1	H4-DD2	H4-DD3	H4-DD4	H4-DD5	H4-DD6
dimension	3,420	4,818	6,240	7,650	9,048	10,482
η	0.1	0.2	0.3	0.4	0.5	0.6

Table 8.38: *List of the constructed domain decomposition subspaces for H4*

	$\mu_1 = \mu_2 = 1$		$\mu_1 = \mu_2 = 2$		$\mu_1 = \mu_2 = 5$	
ID	$\|\|r_{20}\|\|$	c_{20}	$\|\|r_{20}\|\|$	c_{20}	$\|\|r_{20}\|\|$	c_{20}
H4-DD1	3.850e-02	0.9106	1.137e-02	0.8754	1.510e-03	0.8231
H4-DD2	3.078e-03	0.7978	3.579e-04	0.7344	4.334e-05	0.6901
H4-DD3	4.087e-05	0.6483	6.018e-06	0.6027	5.668e-07	0.5636
H4-DD4	7.381e-08	0.4561	1.664e-09	0.4018	2.440e-10	0.3902
H4-DD5	7.558e-12	0.2894	5.446e-12	0.2547	2.216e-12	0.2492
H4-DD6	8.233e-13	0.1796	2.895e-12	0.1663	2.339e-12	0.1598

Table 8.39: *Gauss-Seidel iteration and subspaces H4-DD1 to H4-DD6: influence of number of smoothing steps on numerical convergence factor and relative residual norm as obtained after 20 steps.*

The convergence is a lot faster on all domain decomposition subspaces compared to the H4-RT* subspaces of similar dimension. A good convergence factor of $c_{20} \approx 0.4$ is already obtained with H4-DD4. Note that this subspaces has around half the dimension of the original Hessian.

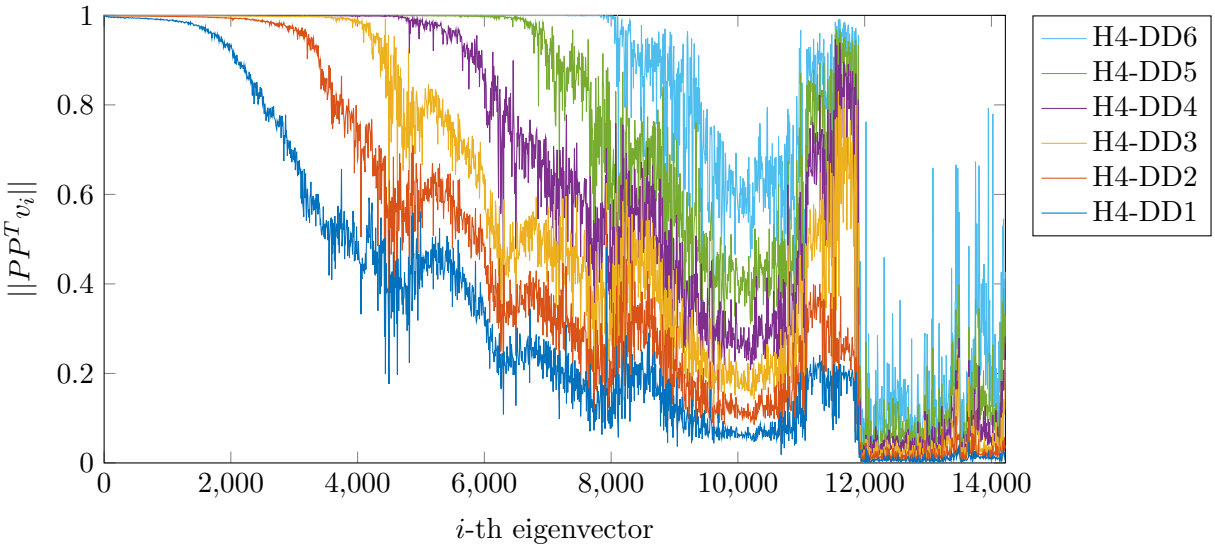


Figure 8.40: *Representation of an eigenvector in the subspaces H4-DD1 to H4-DD6. Eigenvectors are sorted in ascending order.*

We plot the projection of the eigenvectors into the subspaces H4-DD1 to H4-DD6 in Figure 8.40. For H4, the combined results indicate that an efficient multilevel algorithm can likely be constructed with a proper choice of subspaces.

8.5.6 Results for H6

It is nice to see that the multilevel approach works well on the smaller problems. However, on current hardware, these problems can be solved directly since the memory requirements of a factorization can easily be handled. For the larger problems, though, the direct approach becomes quickly unfeasible. Therefore, it is of great interest whether the multilevel approach actually works for large problems. The problem H6 is on the verge of being too large to be handled by direct methods on common hardware. Recall that H6 has Hessian matrix of dimension 89,637-by-89,637 containing approximately 76,1 million non-zero entries.

ID	H6-DD1	H6-DD2	H6-DD3	H6-DD4	H6-DD5	H6-DD6
dimension	20,992	29,953	38,846	47,648	56,367	65,738
η	0.1	0.2	0.3	0.4	0.5	0.6

Table 8.41: *List of the constructed Domain Decomposition Subspaces for H6*

We only give results for the Gauss-Seidel method combined with domain decomposition subspaces (see Table 8.41). Furthermore, the results are averaged over 5 runs and not 10 runs as previously. Also, we have to choose a slightly different shift, since for the shift $\sigma = 10$, the matrix is very close to singular. We set $\sigma = -10$ and obtain the following results, which are summarized in Table 8.42.

	$\mu_1 = \mu_2 = 1$	
ID	$\|\|r_{20}\|\|$	c_{20}
H6-DD1	2.410e-02	0.9064
H6-DD2	2.209e-03	0.8195
H6-DD3	1.686e-04	0.7205
H6-DD4	1.457e-06	0.5764
H6-DD5	8.511e-10	0.3942
H6-DD6	1.965e-12	0.286

Table 8.42: *Gauss-Seidel iteration and subspaces H4-DD1 to H4-DD6: influence of number of smoothing steps on numerical convergence factor and relative residual norm as obtained after 20 steps.*

In Table 8.43, we show convergence results for a three-level algorithm using subspaces H6-DD2 and H6-DD4. We set $\mu_1^1 = \mu_2^1 = 1$ and $\mu_1^2 = \mu_2^2 = 3$. The results are given for the V- and W- cycle.

	V-cycle		W-cycle	
	$\|\|r_{20}\|\|$	c_{20}	$\|\|r_{20}\|\|$	c_{20}
GS	9.409e-05	0.7335	3.514e-06	0.6144

Table 8.43: *Gauss-Seidel (GS) iteration for a three-level algorithm with H6-DD2 and H6-DD4: numerical convergence factor and relative residual norm as obtained after 20 steps for V- and W-cycle.*

8.5.7 Conclusion on Linear Solvers

Direct methods are very efficient for small problems. For larger problems, the fill-in in the L-factor can grow rapidly. Also, it is difficult to decide a priori which reordering strategy will be effective.

Therefore, for larger problems, one is advised to use a preconditioned iterative solver. ILUPACK provides elaborate software for this task. The different parameters allow the balance between competing factors: low memory on one side, and a low number of iterations on the other. However, if the focus is on a low number of iterations, there is a large overhead in precomputing time for the incomplete multilevel LDL^T decomposition. Also, the large fill-in will make each application of the preconditioner fairly expensive. The reward is an extremely low number of iterations required to solve a linear system. Even for the large problem H6, it is possible to achieve convergence in less than 7 iterations on average.

The multilevel algorithm has little memory overhead, and requires almost no precomputing time. For the large problems H4 and H6, we showed that it is possible to achieve good convergence for a W-cycle in a three or four level environment. These algorithms only used one pre- and post-smoothing step on the outer level, and the next level is around half the dimension. With a convergence factor of 0.4, a relative residual of around 1e-4 can be reached in ten steps. With twenty steps, a relative accuracy of 1e-8 can be expected. Also, in most settings, a faster convergence can be observed for the first few steps.

It is difficult to compare this method to ILUPACK, though. Since we only implemented a "proof of concept" version of this multilevel algorithm, it does not make sense to directly compare computing times.

Specifically, for H6, ILUPACK can achieve convergence in 11 steps with a fill-in of around 3.5, or in 45 steps with a fill-in of around 2.1. This is equivalent to the cost of roughly 50, or 140 respectively, matrix-vector (MV) multiplications. The three-level algorithm for H6 (see Table 8.43) requires at least 20 steps for the W-cycle, and each iteration step computes 2 MV on the full problem. The inner levels of the W-cycle require $2(\mu_1^2+\mu_2^2) = 12$ iterations with the coarse matrix on level 2. Then, per iteration step, four linear systems with the coarse matrix of level three need to be solved. In total, this will be more expensive than the solver implemented in ILUPACK, though, the difference will not be large.

8.6 The Slowest Eigenmodes of H6

We want to finish this chapter by presenting some of the eigenmodes for the problem H6. Recall, that the lowest eigenvalue of H6 is negative. The next six eigenvectors corresponding to the lowest frequencies are related to overall translation and rotation. Therefore, we are interested in eigenmodes starting at the 8th lowest frequency. However, for visual purposes, we also included the 7th eigenmode, which corresponds to an overall rotation of the molecule.

plotted in	number of eigenmode	wave number [1/cm]
Figure 8.45	7th	2.16399
Figure 8.46	8th	6.58859
Figure 8.47	9th	8.05627
Figure 8.48	10th	8.73612
Figure 8.49	11th	10.544
Figure 8.50	12th	11.3367
Figure 8.51	13th	12.6236
Figure 8.52	14th	12.8965
Figure 8.53	15th	13.2836

Table 8.44: *List of figures corresponding to the eigenmodes and wave numbers*

The eigenmodes are plotted as displacement vectors (cf. Figure 7.1). Here, however, we do not plot the individual atoms anymore. Depending on the eigenmode, we decide to either plot a side or a top view. All the eigenmodes have the common feature that they are vast collective motions with contributions from the entire molecule.

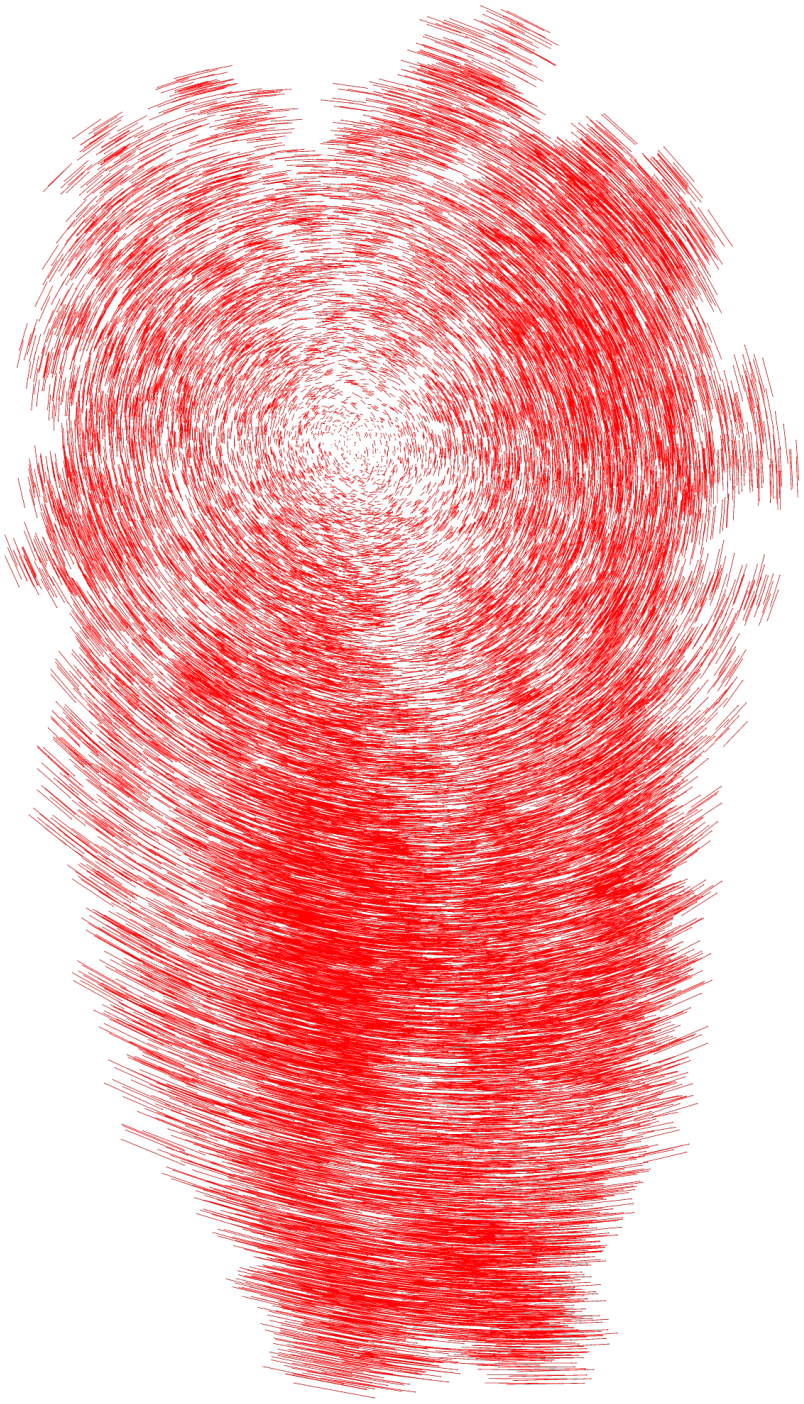


Figure 8.45: *H6 - 7th eigenmode*

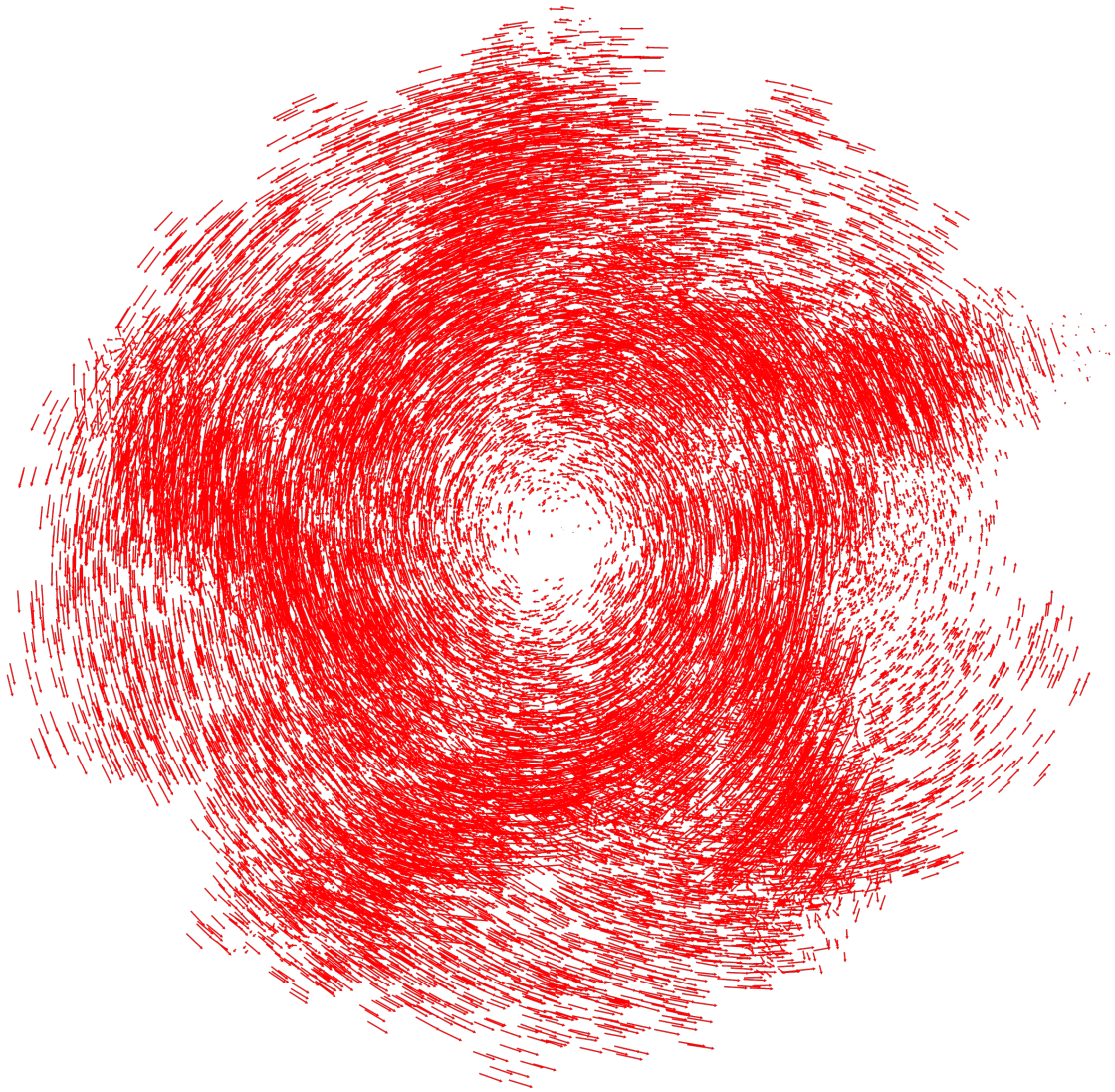


Figure 8.46: *H6 - 8th eigenmode*

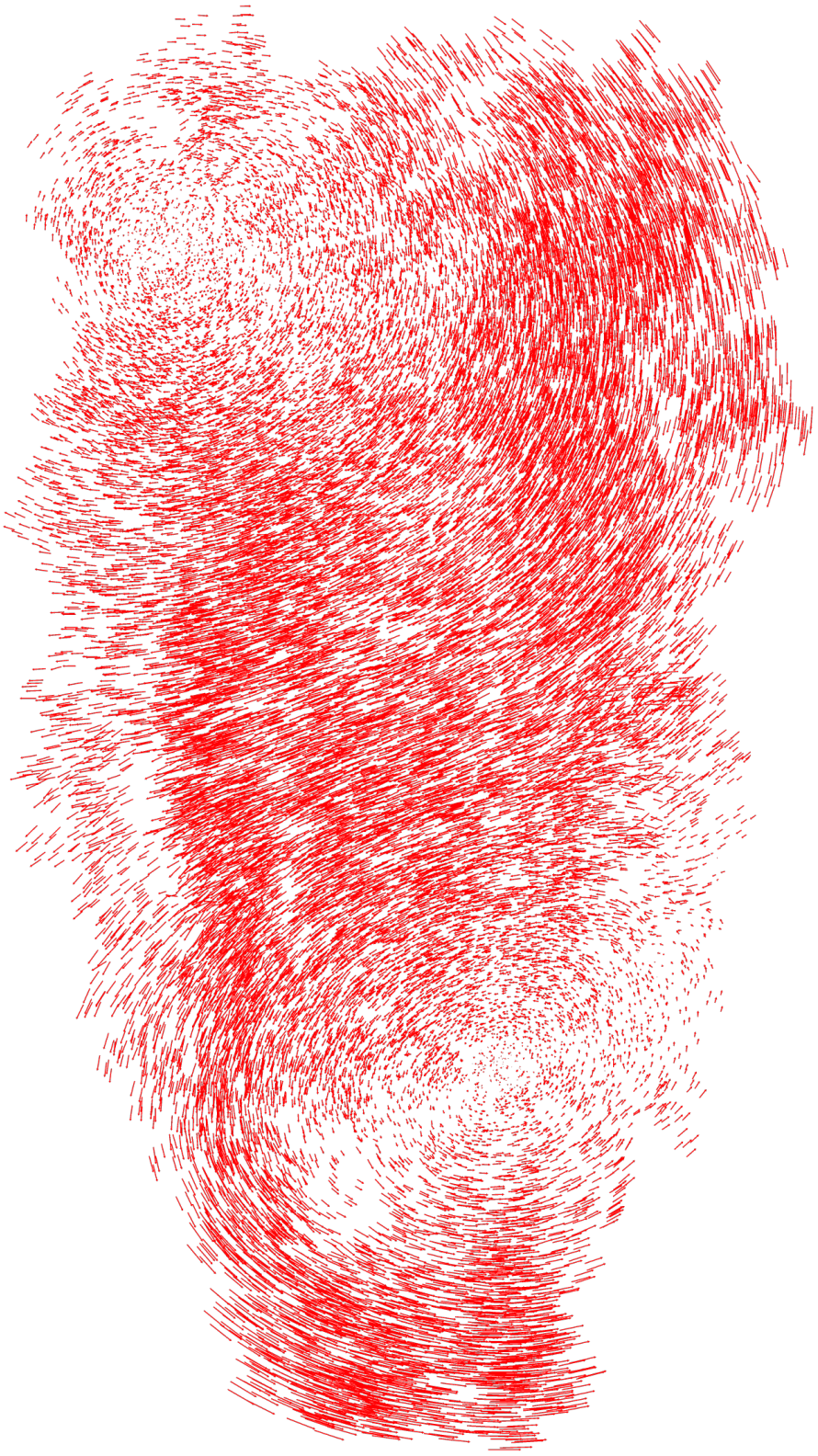


Figure 8.47: *H6 - 9th eigenmode*

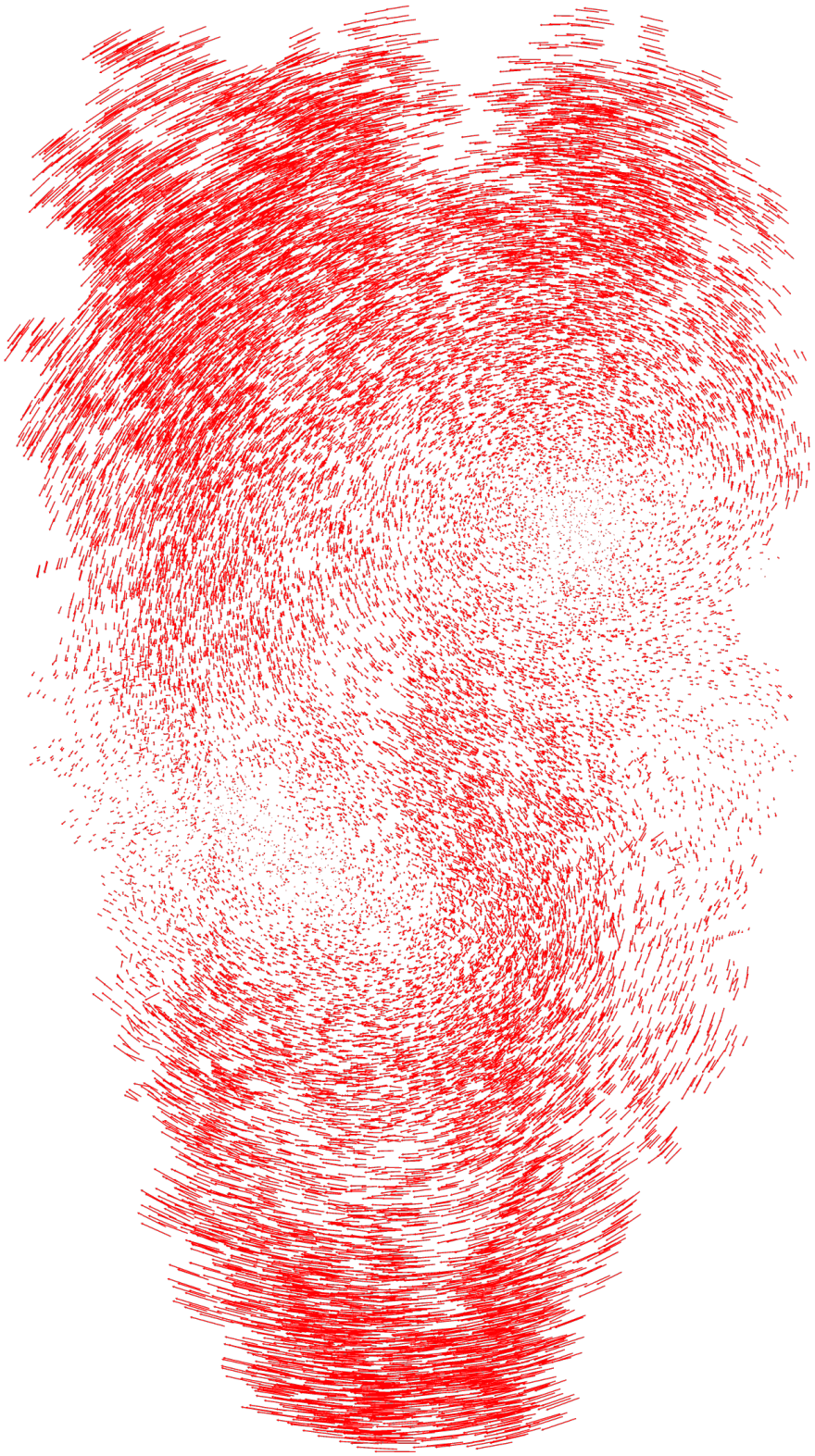


Figure 8.48: *H6 - 10th eigenmode*

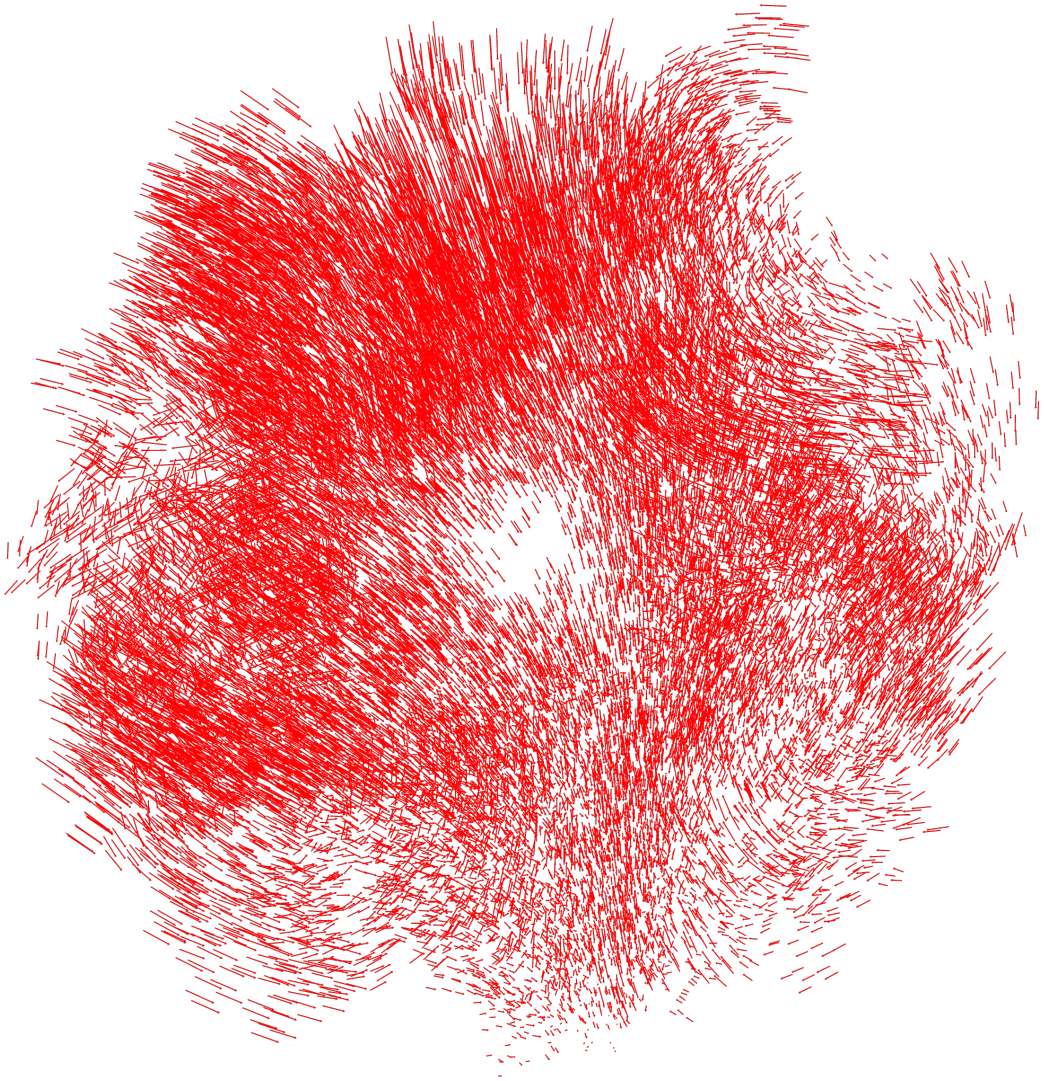


Figure 8.49: *H6 - 11th eigenmode*

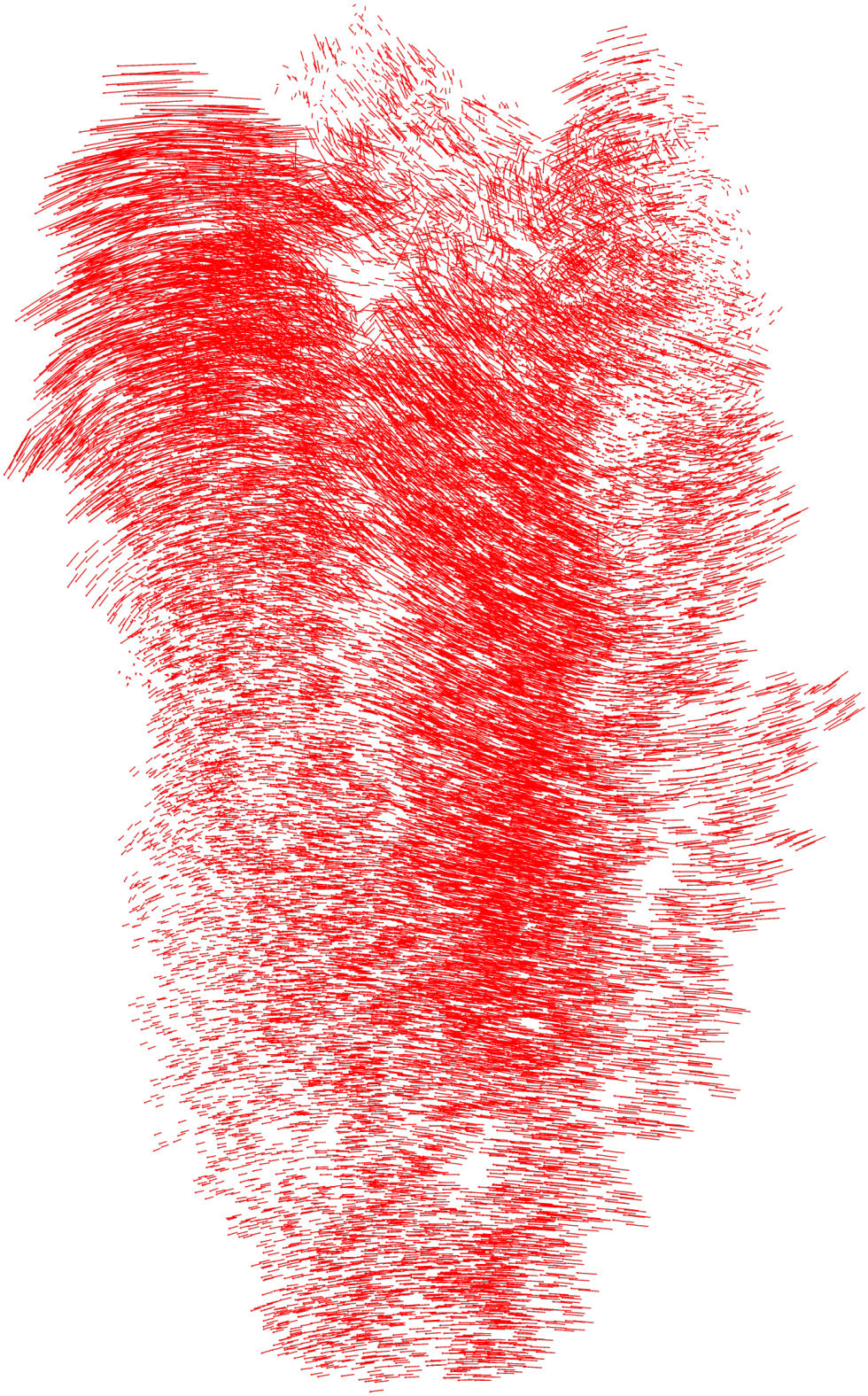


Figure 8.50: *H6 - 12th eigenmode*

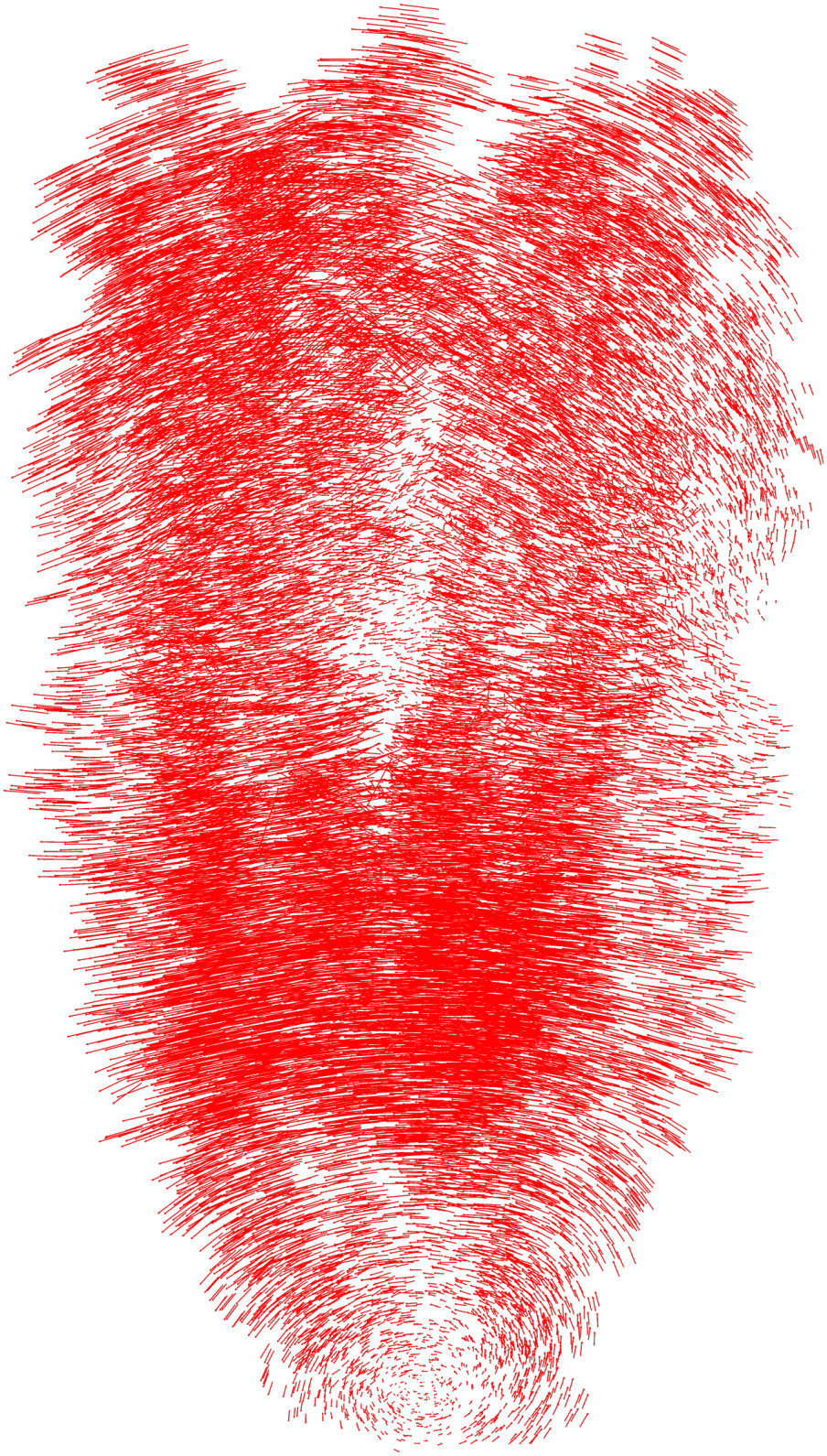


Figure 8.51: *H6 - 13th eigenmode*

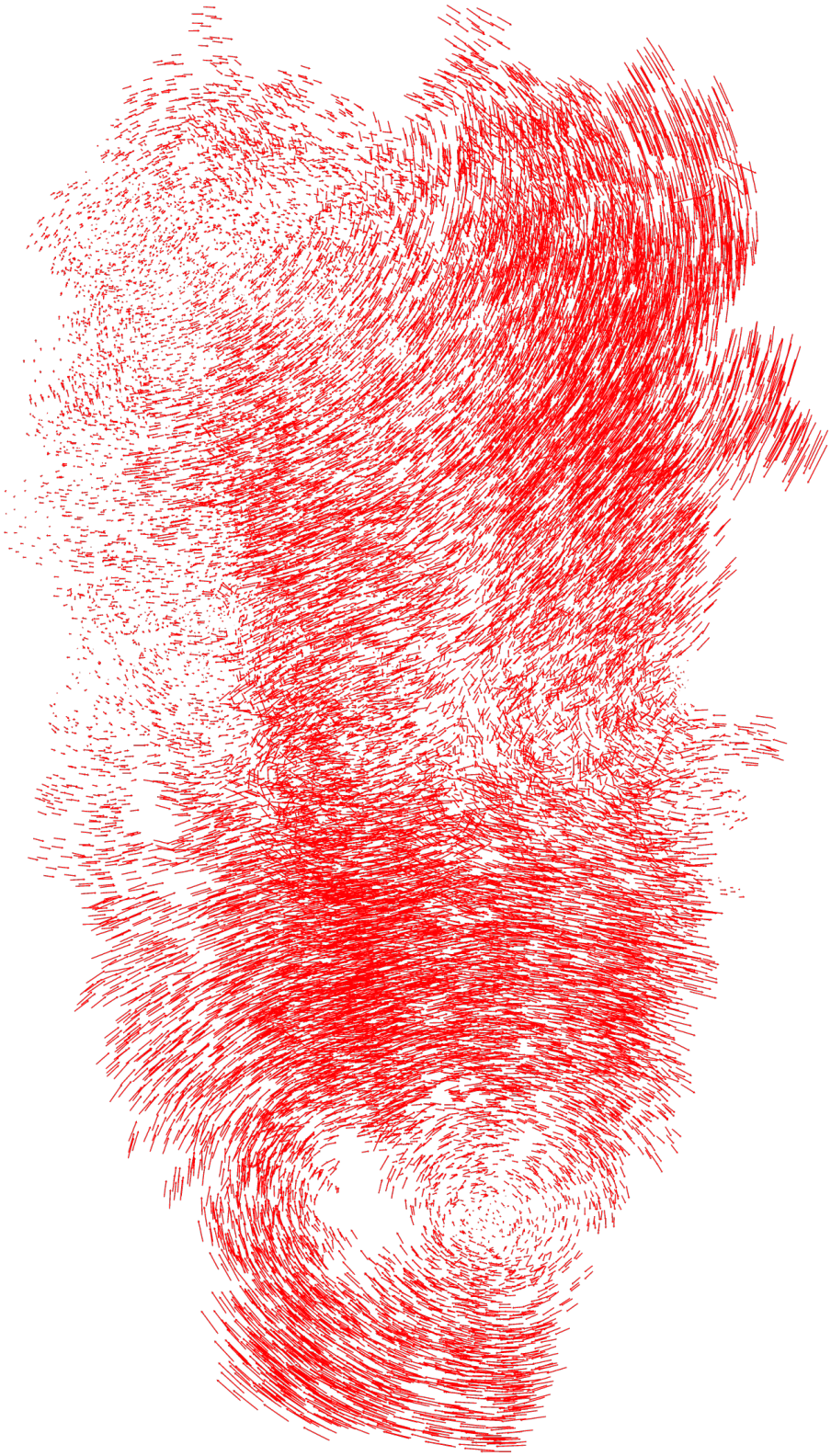


Figure 8.52: *H6 - 14th eigenmode*

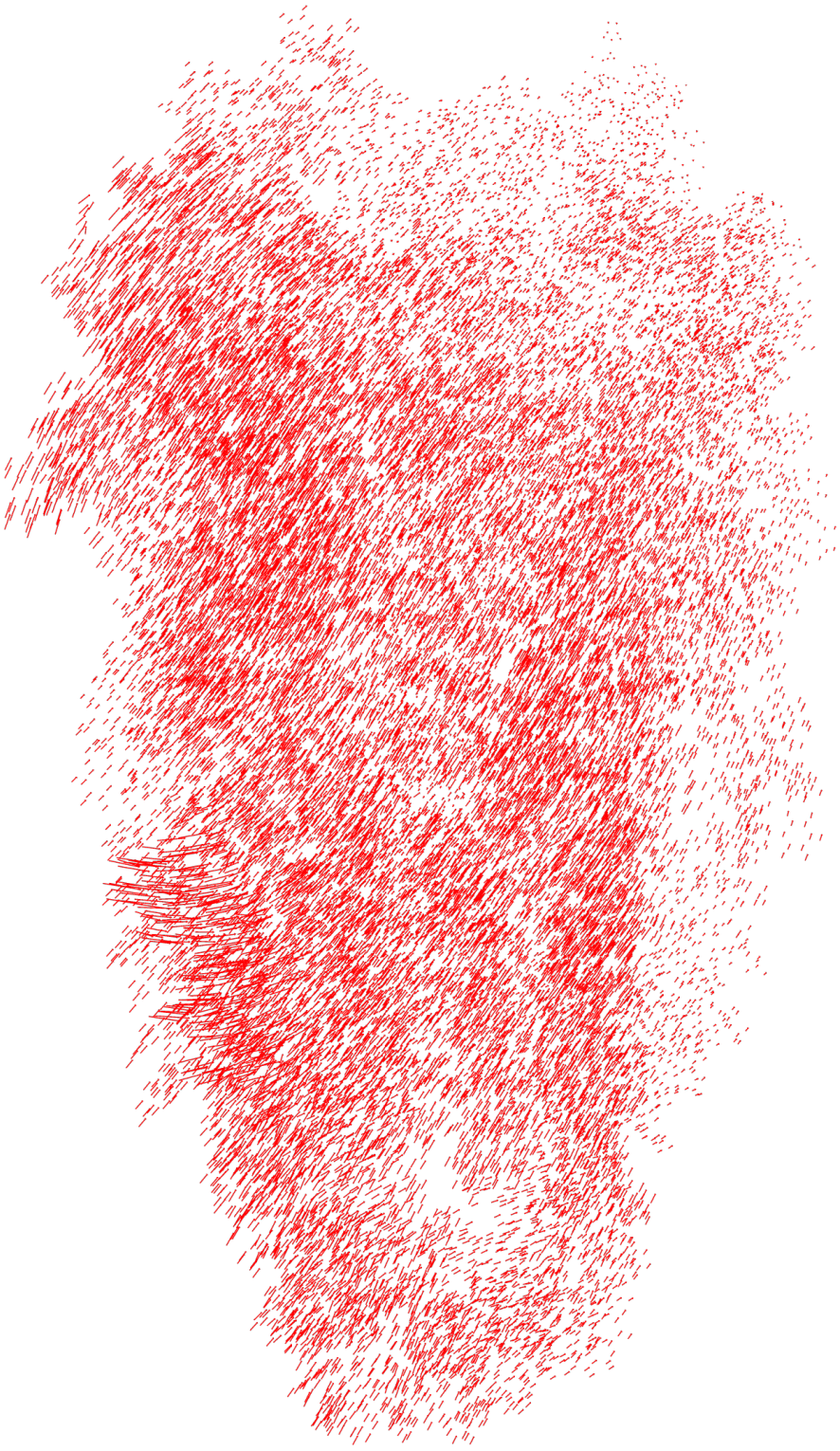


Figure 8.53: *H6 - 15th eigenmode*

9 Conclusions

In the first part of the thesis, we showed the significance of resonance issues in standard integrators for molecular dynamics. Such resonance problems effectively limit the time step of the employed numerical integrator. We proposed a new filtering technique, which can be used to increase the accessible time step in an impulse method. In detail, we compared it to already established methods such as the equilibrium method. On some realistic examples, we showed that the new filtering technique works reliably. Furthermore, it can be implemented in a very efficient way. The results are published in [21].

The second part of the thesis compares different numerical methods for normal mode analysis. In normal mode analysis, the main task is computing eigenpairs corresponding to the smallest eigenvalues of large Hessian matrices. Essentially, the problem requires solving linear systems with the mass-weighted Hessian. We show the limitations of direct methods, and the performance of iterative solvers. Finally, we introduce a new multilevel algorithm which tries to take advantage of a coarse grained molecular description to extract information. A first implementation of this algorithm shows the potential to efficiently tackle the problem at hand. At its core, we need to correctly guess subspaces corresponding to slow motion using structural information. In this thesis, we propose three different strategies, and critically evaluate the success on six molecular system of different sizes. However, while it shows good performance for the examples investigated here, the algorithm reacts very sensitively to small changes. This thesis provides a "proof of concept", but for usage in productive implementations, more research needs to be conducted.

A Potentials

Here, we provide a list of the potentials used during the simulations in this thesis. We want to emphasize that this list does not cover all available potentials, at the contrary, it only contains a few important ones out of the vast number that exist. Afterwards, we quickly explain the force field CHARMM, which provides a full set of parameters for many groups of molecules.

Most potentials directly depend on the interatomic distance. When calculating the force, that is, the derivative of the potential function, the following relations are useful: we denote

$$r_{ij} = q_j - q_i \in \mathbb{R}^3$$

and

$$||r_{ij}|| = ||q_j - q_i|| = \sqrt{(q_{j1} - q_{i1})^2 + (q_{j2} - q_{i2})^2 + (q_{j3} - q_{i3})^2}.$$

Hence,

$$\frac{d||r_{ij}||}{dq_i} = -\frac{1}{||r_{ij}||}(q_j - q_i) = -\frac{r_{ij}}{||r_{ij}||}, \qquad \frac{d||r_{ij}||}{dq_j} = \frac{1}{||r_{ij}||}(q_j - q_i) = \frac{r_{ij}}{||r_{ij}||}. \tag{A.1}$$

If a potential depends only on the interatomic distance, it is clear, that both the potential and the force are invariant with respect to overall translations or rotations.

We group potentials as in (2.10) and

$$U(q) = U_{\text{intra}}(q) + U_{\text{pair}}(q) + U_{\text{long}}(q) \tag{A.2}$$

with

$$U_{\text{intra}} = U_{\text{bond}} + U_{\text{angle}} + U_{\text{dihedral}} + U_{\text{improper}}$$

and

$$U_{\text{pair}} = U_{\text{LJ}} + U_{\text{Coul}}.$$

A.1 Bonded Potentials

Bonded potentials describe interactions within the same molecule between bonded atoms.

A.1.1 Harmonic Bonds

A bond potential models a direct bond between two atoms in the same molecule. Usually, those are very strong bonds which are highly oscillatory and contribute to the fastest vibrations in a molecular model.

A very common and simple harmonic spring model of a two bonded atoms with distance r is given by the potential function

$$U(r) = \frac{k_b}{2}(r - r_0)^2, \tag{A.3}$$

where k_b is a force constant and r_0 is the equilibrium length of that bond.

A.1.2 Morse Bonds

A more elaborate bond potential which is used by the TIP4P/2005f model is the Morse potential

$$U(r) = D[1 - e^{-\alpha(r-r_0)}]^2, \tag{A.4}$$

where D, α and r_0 are suitable constants.

A.1.3 Angle Potential

The harmonic angle potential is similar to the harmonic bond potential. It is harmonic in the angle enclosed by three bonded atoms:

$$U(\theta) = \frac{k_\theta}{2}(\theta - \theta_0)^2, \tag{A.5}$$

where θ is the angle between atom 1 bonded to both atoms 2 and 3:

$$\theta = \theta_{2,1,3} = \arccos\left(\frac{{r_{12}}^T r_{13}}{||r_{12}|| \cdot ||r_{13}||}\right), \qquad r_{ij} = q_j - q_i. \tag{A.6}$$

A.1.4 Dihedrals

Dihedral potentials model a torsion angle ϕ between four consecutive bonded atoms 1-2-3-4. The angle is measured between the two intersecting planes formed by the atoms 1-2-3 and 2-3-4. The harmonic dihedral potential then reads

$$U(\phi) = K(1 + d\cos(n\phi)),$$

where K, d, and n are suitable constants.

In CHARMM, a slightly different version is used with a potential function

$$U(\phi) = K(1 + \cos(n\phi - d)).$$

A.1.5 Impropers

The improper potential is very similar to the dihedral one, however, the four atoms now have a central atom 1 which is bonded to three atoms 2-3-4. The improper angle χ is the angle between the planes formed by 1-2-3 and 2-3-4. As potential form a harmonic

function is chosen, and

$$U(\chi) = K(\chi - \chi_0)^2,$$

with constants K and χ_0.

A.2 Pair Potentials

Pair potentials describe short-range interactions which apply to all pairs of atoms with a distance smaller than a certain cutoff radius r_c. In contrast to the bonded potentials, the 'status' of such interactions will change during a simulation, that is, if two atoms move away from each other their interaction due to the pair potential will eventually vanish. The other way around, if two atoms move towards each other, as soon as they are closer than r_c, a pair interaction needs to be computed.

A.2.1 Lennard-Jones Potential

The Lennard-Jones potential models intermolecular forces due to Van-der-Waals repulsion and attraction. It has the potential function

$$U(r) = 4\ \epsilon \left(\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right) \tag{A.7}$$

with parameters ϵ and σ. The choice of the powers is somewhat arbitrary. This version is sometimes referred to as a 12-6-LJ (because of the powers). A 9-6-LJ version is also common.

Since the Lennard-Jones force decays rapidly with increasing distance it is usually used with a cutoff radius. To avoid non continuous jumps at the cutoff a switching function ensures smooth transition to zero.

Therefore, between an inner and an outer cutoff radius the Lennard-Jones potential is multiplied by a switching function $S(r)$ which goes smoothly to zero:

$$S(r) = \frac{(r_{\text{outer}}^2 - r^2)^2 (r_{\text{outer}}^2 + 2r^2 - 3r_{\text{inner}}^2)}{(r_{\text{outer}}^2 - r_{\text{inner}}^2)^3}, \qquad r_{\text{inner}} < r < r_{\text{outer}}. \tag{A.8}$$

A.2.2 Coulomb Potential

The electrostatic potential models interactions due to the different electric charges in atoms. Denoting by e_i the charge of particle i, it is given by

$$U(r_{ij}) = \frac{1}{4\pi\epsilon_0} \frac{e_i e_j}{r_{ij}}, \qquad r_{ij} = ||q_j - q_i||. \tag{A.9}$$

Since the Coulomb potential decays very slowly with increasing radius, simply using a cut off with a switching function (as for the Lennard-Jones potential) can lead to severe artifacts. Depending on the simulation requirements, neglecting long-range effects can

have an adversary effect. This is an issue for all pair potentials which behave like r^{-3} or slower: using a cutoff can create a significant artifact.

A.3 Long-Range Electrostatic Methods

With periodic boundaries the total electrostatic potential Φ for a cubic box of length L reads

$$\Phi = \frac{1}{2}\frac{1}{4\pi\epsilon_0}\sum_{i,j=1}^{N}\sum_{n\in\mathbb{Z}^3}{}' \frac{e_i e_j}{||q_i - q_j + nL||},$$

where the prime in the second summation means that for $n = 0$ the term $i = j$ is omitted. Unfortunately, this series is only conditionally convergent, so its value depends on the summation order (see [163], good explanation also in [164, pg. 162ff] or [5, Appendix A]).

Directly evaluating this sum is expensive (it would scale like N^2 where N is the number of atoms) and other alternatives have to be considered. Here, we will focus on lattice methods, however, other successful approaches such as multipole methods or reaction field methods exist.

A.3.1 Ewald Summation

With the traditional Ewald summation, the sum can be evaluated with computational complexity in $O(N^{\frac{3}{2}})$. The idea is to decompose the conditionally convergent sum into two rapidly converging sums $\Phi \approx \Phi_{\text{short}} + \Phi_{\text{long}}$. This can be done by adding so-called screening charges to 'shield' the point charges.

With the error function

$$\text{erf}(x) = \frac{2}{\pi}\int_0^x e^{-y^2}dy, \quad \text{erfc}(x) = 1 - \text{erf}(x),$$

we arrive at the following decomposition ($r_{ij}^n = ||q_i - q_j + nL||$):

$$\Phi_{\text{short}} \approx \frac{1}{2}\frac{1}{4\pi\epsilon_0}\sum_{n\in\mathbb{Z}^3}{}' \sum_{\substack{i,j=1 \\ r_{ij}^n < r_{cut}}}^{N} e_i e_j \frac{\text{erfc}(G r_{ij}^n)}{r_{ij}^n},$$

$$\Phi_{\text{long}} \approx \frac{1}{2}\frac{1}{4\pi\epsilon_0}\sum_{n\in\mathbb{Z}^3}{}' \sum_{i,j=1}^{N} e_i e_j \frac{\text{erf}(G r_{ij}^n)}{r_{ij}^n}.$$

The erfc in Φ_{short} ensures that this part quickly goes to zero for increasing radius. Thus, Φ_{short} can be approximated like a regular pair potential with a cutoff radius. Φ_{long} represents the remaining long-range terms, and it can be shown that this sum quickly converges in Fourier - or reciprocal - space.

However, there are even faster methods which use the FFT to speed up computations

for the reciprocal sum. In the best case, the computational cost scales like $O(N \log N)$. One of these methods is the particle-particle particle-mesh (PPPM) method.

A.3.2 PPPM Method

There is some variety for particle-mesh algorithms which only differ in small points. In general, they first proceed similar to the Ewald summation and decompose the sum in a direct and reciprocal sum. The direct sum is handled like a pair potential. With elaborate strategies such as neighbor lists or a linked cell method, the force is evaluated for each pair within a certain cutoff radius. The reciprocal sum is handled by assigning the charges to a mesh, transforming the grid and evaluating the sum in reciprocal space. Then, the force on each atom is obtained by transforming back into real space.

However, these methods differ in their exact choice of screening functions, switching strategies and interpolation method. The PPPM method uses spherical charge clouds to shield the point charges and uses trilinear interpolation to mesh points when evaluating the long-range forces in Fourier space.

A.4 CHARMM Force Fields

CHARMM is an acronym for Chemistry HARvard Molecular Mechanics [46–49]. It is a name for a set of force fields in MD. It also refers to the corresponding MD simulation and analysis software package. We refer to the force fields here. There are different versions for united-atom (CHARMM19) and all-atom fields (CHARMM22). Specific parameter sets for biomolecules such as DNA and lipids exist. These parameter sets try to provide full descriptions for each chemical group.

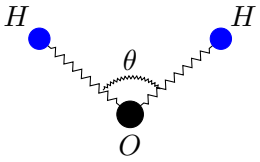
In its core, CHARMM provides an additive force field with a decomposition similar to (2.10) with a separation in bond, angle, dihedral, improper, Van-der-Waals and electrostatic potential. It uses the CHARMM dihedral function, presented second in the dihedral section. Furthermore, the angle potential uses an additional Urey-Bradley term, that is a covalent bond between the terminal atoms of an angle. Nonbonded terms are only computed for atoms separated by three or more covalent bonds.

The force constants and geometric constants (such as bond distance and equilibrium angle) are provided in long parameter tables and are matched based on atom types and structure of the molecular model. The values were obtained from a multitude of sources. In order to provide as accurate simulations as possible, parameters were fitted with vibrational data, geometric matching of crystallographic data, and optimized by empirical energy calculations.

B Models

B.1 SPC/F

The SPC/F model [80] is based on the very common rigid three site SPC model and uses harmonic potentials in bonds and angles to model intra-molecular forces.



model	k_b	r_0	k_a	θ_0	σ_{O-O}	ϵ_{O-O}	σ_{H-H}	ϵ_{H-H}	e_O	e_H
SPC/F	1108.27	1.0	91.5392	109.47	3.166	0.1553943595	0.0	0.0	-0.82	0.41

Table B.1: *model parameters for SPC/F, units as in LAMMPS 'real' settings*

In the flexible SPC/F [80] the bonded contributions are modeled as harmonic springs

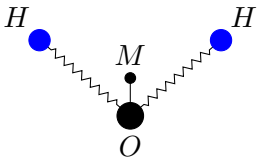
$$U_{\text{intra}} = \sum_i \frac{k_b}{2}(r_i - r_0)^2 + \sum_j \frac{k_a}{2}(\theta_j - \theta_0)^2$$

where the sum runs over all bonds or angles respectively. r_i is the bond length of the i-th bond, and θ_j the j-th angle.

The pair potential is a classical 12-6 Lennard-Jones potential interacting only between oxygen atoms with parameters as in Table B.1. The cut off for the Lennard-Jones potential is set to 9Å using a CHARMM switching function starting at 8Å. The short-range Coulomb interactions also use a cut off of 9Å and the long-range electrostatic contributions are handled by a PPPM-method with accuracy set to 1e-05.

B.2 TIP4P/2005f

Introduced 2011 in [82], the TIP4P/2005f is a flexible water model based on the TIP4P/2005 model.



Each water molecule has two flexible bonds, one flexible angle, and additional charge site M. The exact position of M depends on the bonds.

The TIP4P/2005f uses a Morse potential for the bonds and a harmonic approximation for the angle:

$$U_{\text{intra}} = \sum_i D[1 - e^{-\alpha(r-r_0)}]^2 + \sum_j \frac{k_a}{2}(\theta_j - \theta_0)^2$$

where the sum runs over all bonds or angles respectively. The constants are given in Table B.2.

model	D	α	r_0	k_a	θ_0
TIP4P/2005f	432.581 [kJ/mol]	2.287 [1/Å]	0.9419 [Å]	107.4°	367.81 [kJ/(mol rad^2)]

Table B.2: *Model parameters for TIP4P/2005f*

The position of M-site is calculated relative to the oxygen atom with

$$d_{OM} = d_{OM}^{rel}(z_{OH_1} + z_{OH_2}) \tag{B.1}$$

where z_{OH_i} are the distances to the oxygen of the hydrogen's projections along the HOH bisector, and $d_{OM}^{rel} = 0.13194$.

The oxygen sites do not carry a charge but contribute to the Lennard-Jones term. H and M sites are charged but do not contribute to the Lennard-Jones term. The parameters are $\epsilon = 0.18520627$[kcal/mol], $\sigma = 3.1644$[Å] and $q_H = 0.5564$.

The cutoff for the pair interactions is set to 8.5Å, with a smooth switching function starting at 8Å. The long-range electrostatic contributions beyond 8.5Å are handled by a suitable long-range method such as Ewald summation or PPPM.

B.3 Peptide

The small peptide has only 84 atoms, and it is surrounded by 640 water molecules. In total, the simulation has 14 different types of atoms, 18 different types of bonds, 31 types of angles, 21 dihedral types and 2 improper types. All bonded interactions use the CHARMM versions of the harmonic potentials. The pair style uses a switching function to smoothly cut off the Lennard-Jones potential with $r_{\text{inner}} = 8.0$ Å and $r_{\text{outer}} = 10.0$ Å. The long-range solver is the PPPM method. For a full set of parameters, see the data files from the folder 'peptide' in lammps, see http://lammps.sandia.gov/doc/Section_example.html.

C Integrators

C.1 Velocity-Verlet Method

The Verlet method is the standard integrator as described in Section 3.1. It is a second order, symmetric and symplectic method. Each time step, only one force evaluation is required. For step size h, the numerical flow Φ_h of the Verlet method is given by

$$\Phi_h = \varphi^U_{h/2} \circ \varphi^T_h \circ \varphi^U_{h/2}.$$

Here, φ^T, φ^U describe the exact flow of the kinetic and potential part of the ODE (see (3.2)).

C.2 Impulse Method

The impulse method is described in Section 3.2. In molecular dynamics, we use an impulse method with three stages and the splitting

$$U(q) = U_{\text{intra}}(q) + U_{\text{pair}}(q) + U_{\text{long}}(q) \tag{C.1}$$

with a fast intra-molecular part U_{intra}, short-range pair interactions U_{pair} and remaining long-range electrostatic contributions U_{long}.

The numerical flow of the impulse method with this three stage splitting is given by

$$\Phi_h = \varphi^{U_{\text{long}}}_{h/2} \circ \left(\varphi^{U_{\text{pair}}}_{h/2K} \circ \left(\varphi^{U_{\text{intra}}}_{h/2KL} \circ \varphi^{T}_{h/KL} \circ \varphi^{U_{\text{intra}}}_{h/2KL} \right)^L \circ \varphi^{U_{\text{pair}}}_{h/2K} \right)^K \circ \varphi^{U_{\text{long}}}_{h/2}.$$

Integer factors K, L between the step sizes of the stages are required. If not specified otherwise, we always choose K and L such that

$$\frac{h}{KL} \leq 0.5\text{fs}, \qquad \text{and} \qquad \frac{h}{K} \leq 1\text{fs}.$$

C.3 Implicit Midpoint Method

The implicit midpoint method as introduced in Section 3.4 needs to solve a nonlinear system of equations in every time step. To conduct the simulations in Chapter 4, the implicit midpoint method was implemented with the following simplifications. First, it only runs a simplified Newton iteration, so the derivative is fixed at the initial guess. Second, the derivative is not a full derivative, it uses a smooth CHARMM cutoff for pair forces between 3 and 4 Angstrom. Accuracy of the Newton iteration is set by the user,

we used 1e-4. For solving the linear systems, we used the BiCGstab method from the software package Eigen.

All those simplification make it feasible to run simulations with the implicit midpoint method, but it will have a major impact on the step sizes available. For larger step sizes, due to the nonlinearity of the problem, the simplified equations might not properly converge anymore, and no safeguard, such as re-evaluating the Hessian is implemented.

C.4 Mollified Impulse Method

The mollified impulse method is described in detail in Chapter 5. See also Algorithm 5.4. In principle, it is exactly the same as the impulse method, except that the outer stage is filtered. Specifically, a filter Ψ is inserted in the slow potential. The filtered slow force then reads $\Psi_q^T(q)U_{\text{long}}(\Psi(q))$. If the filter Ψ is chosen as the identity map, this method reduces to the regular impulse method.

If the corotational filter introduced in Section 5.2 is chosen, we refer to this method as the corotational impulse method (CIM). When the outer stage is filtered with the equilibrium filter in (5.5) we call it equilibrium impulse method (EIM). The stage factors K, L are chosen in the same way as for the impulse method.

C.4.1 Corotational Impulse Method

We implemented the corotational impulse method in a high-end fashion in LAMMPS. It automatically decomposes the molecular structure into suitable clusters. The user can influence this decomposition by providing information what kind of interactions have to be contained inside a cluster. Such information can be a specific bond or angle type, atom type or atom mass. Currently, the implementation supports clusters up to 5 atoms.

The decomposition in clusters only depends on local information. So each CPU computes its own decomposition and updates it from time to time due to migrating atoms from neighboring CPUs. A precompiled list of clusters with a reference position each is stored locally. When evaluating the slow force, first the mollified position is computed by iterating through the cluster list and computing rotation and translation of each cluster, and then the corotated reference position is used to obtain the mollified position. To avoid recomputing certain values, at the same time, the derivative is already computed. Then, the mollified position is used to evaluate the slow force. Afterwards, the derivative is applied to the slow force vector and the original position is restored.

It is a straight forward implementation, however, with attention to the specific details of LAMMPS. The filter is able to efficiently run on large processor counts. As such, the filter behaves correctly in the event of migrating atoms between processor borders. All necessary values are communicated to the correct cores. This includes a complicated behavior in some cases of periodic boundaries. The local list of clusters needs updating in the event of resorting of atoms (for performance), and neighbor list changes. All-in-all,

the implementation performs reliable at extremely low cost.

C.4.2 Equilibrium Impulse Method

We implemented the EIM for comparison purposes in LAMMPS. It is mainly based on the SHAKE functions. That is, this method is restricted to small clusters of up to three atoms as obtained from LAMMPS SHAKE implementation. By modifying the position restraint function, the filtered position can be extracted. Additionally, the routines functionality was extended to provide the derivative necessary for filtering.

Notation

(q, p)	atomic positions and momenta in Hamiltonian coordinates, column vectors
N	Number of atoms
$\dot{q}, \ddot{q}$	first, second time derivative of q
$\nabla_q H(q, p)$	gradient of function $H(q, p)$ w.r.t to q, column vector
$\{A, B\}$	Poisson bracket (8)
$0_{3N}, I_{3N}$	zero/identity matrix 3N-by-3N
$\otimes$	Kronecker product
$\|\|q_i - q_j\|\|$	distance/radius between atom i and j with the regular Euclidean norm on $\mathbb{R}^3$
$H(q, p)$	Hamiltonian function
$\mathcal{H}$	Hessian matrix
n	dimension of matrix
$\mathcal{S}, \mathcal{V}, \mathcal{W}$	calligraphic letter: space/subspace - except $\mathcal{H}$
$[q_1, ..., q_n]$	vector, matrix notation
$\mathcal{K}_m$	Krylov space of dimension m
$\mathbb{P}_m$	Set of polynomials with degree smaller or equal than m
$\mathcal{O}(\cdot)$	algorithmic complexity, Landau notation for asymptotic behavior
$\mathrm{diag}(\cdots)$	diagonal matrix
$\mathrm{Range}(A)$	image of matrix A
$\mathrm{Null}(A)$	null space of matrix A
$\mathcal{K}_m(A, v)$	polynomial Krylov subspace

Abbreviations

ODE	ordinary differential equation
MD	molecular dynamics
NMA	normal mode analysis
NVE, NVT, NPT	ensembles
IM	impulse method
CIM	corotational impulse method
EIM	equilibrium impulse method
SPC/F	a flexible simple point charge model of a water molecule, see Appendix B
TIP4P/2005f	a flexible model of a water molecule, see Appendix B
GMRES,MINRES	generalized minimal residual method
QMR	quasi-minimal resiudal method
IRAM	implicitly restarted Arnoldi method
JADAMILU	Jacobi-Davidson method with Multilevel ILU preconditioning
H1-H6	refering to the model problems in Section 8.2
RI,GS	Richardson iteration, Gauss-Seidel iteration
RT	as prefix: for subspaces of type rotation and translation
DD	as prefix: for subspaces of type domain decomposition
CG	conjugate gradients
l-BFGS	low-memory Broyden–Fletcher–Goldfarb–Shanno algorithm
RCM, AMD	matrix reordering algorithms, see Section 7.3.1

List of Figures and Tables

List of Algorithms

Bibliography

[1] T. Schlick, *Molecular Modeling and Simulation: An Interdisciplinary Guide*, 2010.

[2] B. Leimkuhler and S. Reich, *Simulating Hamiltonian Dynamics*, vol. 14, Cambridge University Press, 2004.

[3] M. Griebel, S. Knapek, and G. Zumbusch, *Numerical Simulation in Molecular Dynamics: Numerics, Algorithms, Parallelization, Applications*, Springer Publishing Company, 1st ed., 2007.

[4] M. E. Tuckerman, *Statistical Mechanics: Theory and Molecular Simulation*, Oxford graduate texts, Oxford Univ. Press, 1. publ., reprint. with corr. ed., 2013.

[5] B. Leimkuhler and C. Matthews, *Molecular Dynamics: With Deterministic and Stochastic Numerical Methods*, Interdisciplinary Applied Mathematics, Springer, 2015.

[6] E. Hairer, C. Lubich, and G. Wanner, *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*, vol. 31, 2006.

[7] B. Leimkuhler, S. Reich, and R. Skeel, Integration methods for molecular dynamics, *Mathematical Approaches to Biomolecular Structure and Dynamics*, (1996), pp. 161–185.

[8] T. Schlick, R. D. Skeel, a. T. Brunger, L. V. Kale, J. a. Board, J. Hermans, and K. Schulten, Algorithmic challenges in computational molecular biophysics, *Journal of Computational Physics*, 151 (1999), pp. 9–48.

[9] E. Hairer, *Challenges in Geometric Numerical Integration*, Springer International Publishing, 2014, pp. 125–135.

[10] B. J. Berne, Molecular dynamics in systems with multiple time scales: reference system propagator algorithms, *Computational Molecular Dynamics: Challenges, Methods, Ideas*, (1999), pp. 297–318.

[11] D. D. Humphreys, R. a. Friesner, and B. J. Berne, A multiple-time-step molecular dynamics algorithm for macromolecules, *The Journal of Physical Chemistry*, 98 (1994), pp. 6885–6892.

[12] R. Zhou and B. J. Berne, A new molecular dynamics method combining the reference system propagator algorithm with a fast multipole method for simulating proteins and other complex systems, *Journal of Chemical Physics*, 103 (1995), pp. 9444–9459.

[13] T. Schlick, Some failures and successes of long-timestep approaches for biomolecular simulations, *Computational molecular dynamics: challenges, methods, ideas – Proceedings of the 2nd International Symposium on Algorithms for Macromolecular Modelling, Berlin, May 21-24, 1997*, 4 (1998), pp. 227–262.

[14] R. D. Skeel, Integration schemes for molecular dynamics and related applications, *The Graduate Student's Guide to Numerical Analysis*, (1998), pp. 119–176.

[15] T. C. Bishop, R. D. Skeel, and K. Schulten, Difficulties with multiple time stepping and fast multipole algorithm in molecular dynamics, *Journal of Computational Chemistry*, 18 (1997), pp. 1785–1791.

[16] J. J. Biesiadecki and R. D. Skeel, Dangers of multiple time step methods, *Journal of Computational Physics*, 109 (1993), pp. 318–328.

[17] T. Schlick, M. Mandziuk, R. D. Skeel, and K. Srinivas, Nonlinear resonance artifacts in molecular dynamics simulations, *Journal of Computational Physics*, 140 (1998), pp. 1–29.

[18] Q. Ma, J. A. Izaguirre, and R. D. Skeel, Verlet-I/R-RESPA/Impulse is limited by nonlinear instabilities, *SIAM Journal on Scientific Computing*, 24 (2003), pp. 1951–1973.

[19] R. D. Skeel and J. A. Izaguirre, *The five femtosecond time step barrier*, in Computational Molecular Dynamics, Challenges, Methods, Ideas, 1998, pp. 303–318.

[20] J. a. Morrone, R. Zhou, and B. J. Berne, Molecular dynamics with multiple time scales: how to avoid pitfalls, *Journal of Chemical Theory and Computation*, 6 (2010), pp. 1798–1804.

[21] L. Fath, M. Hochbruck, and C. Singh, A fast mollified impulse method for biomolecular atomistic simulations, *Journal of Computational Physics*, 333 (2017), pp. 180 – 198.

[22] E. B. Wilson, J. C. Decius, and P. C. Cross, *Molecular Vibrations: The Theory of Infrared and Raman Vibrational Spectra*, McGraw-Hill, New York [u.a.], 1955.

[23] Q. Cui and I. Bahar, *Normal Mode Analysis: Theory and Applications to Biological and Chemical Systems*, Chapman and Hall/CRC, 1 ed., Dec. 2005.

[24] K. Itoh and T. Shimanouchi, Vibrational frequencies and modes of α-helix, *Biopolymers*, 9 (1970), pp. 383–399.

[25] S. Lifson and A. Warshel, Consistent force field for calculations of conformations, vibrational spectra, and enthalpies of cycloalkane and n-alkane molecules, *The Journal of Chemical Physics*, 49 (1968), pp. 5116–5129.

[26] M. Tasumi, H. Takeuchi, S. Ataka, A. M. Dwivedi, and S. Krimm, Normal vibrations of proteins: glucagon, *Biopolymers*, 21 (1982), pp. 711–714.

[27] B. Brooks and M. Karplus, Harmonic dynamics of proteins: normal modes and fluctuations in bovine pancreatic trypsin inhibitor, *Proceedings of the National Academy of Sciences*, 80 (1983), pp. 6571–6575.

[28] M. Levitt, C. Sander, and P. S. Stern, Protein normal-mode dynamics: trypsin inhibitor, crambin, ribonuclease and lysozyme, *Journal of Molecular Biology*, 181 (1985), pp. 423 – 447.

[29] S. Hayward, Normal mode analysis of biological molecules, *Computational biochemistry and biophysics*, (2001), pp. 153–168.

[30] J. R. López-Blanco, O. Miyashita, F. Tama, and P. Chacón, *Normal Mode Analysis Techniques in Structural Biology*, John Wiley & Sons, Ltd, 2001.

[31] J. Ma, New advances in normal mode analysis of supermolecular complexes and applications to structural refinement, *Current Protein & Peptide Science*, 5 (2004), pp. 119–123.

[32] ——, Usefulness and limitations of normal mode analysis in modeling dynamics of biomolecular complexes, *Structure*, 13 (2005), pp. 373 – 380.

[33] I. Bahar and A. Rader, Coarse-grained normal mode analysis in structural biology, *Current Opinion in Structural Biology*, 15 (2005), pp. 586 – 592.

[34] N. Kantarci-Carsibasi, T. Haliloglu, and P. Doruker, Conformational transition pathways explored by Monte Carlo simulation integrated with collective modes, *Biophysical journal*, 95 (2008), pp. 5862–5873.

[35] G. H. Zhang and T. Schlick, LIN - a new algorithm to simulate the dynamics of biomolecules by combining implicit-integration and normal-mode techniques, *Journal of Computational Chemistry*, 14 (1993), pp. 1212–1233.

[36] K. Hinsen, Analysis of domain motions by approximate normal mode calculations, *Proteins: Structure, Function, and Bioinformatics*, 33 (1998), pp. 417–429.

[37] K. Hinsen, A. Thomas, and M. J. Field, Analysis of domain motions in large proteins, *Proteins: Structure, Function, and Bioinformatics*, 34 (1999), pp. 369–382.

[38] P. Durand, G. Trinquier, and Y.-H. Sanejouand, A new approach for determining low-frequency normal modes in macromolecules, *Biopolymers*, 34 (1994), pp. 759–771.

[39] M.-H. Hao and S. C. Harvey, Analyzing the normal mode dynamics of macromolecules by the component synthesis method, *Biopolymers*, 32 (1992), pp. 1393–1405.

[40] M.-H. Hao and H. A. Scheraga, Analyzing the normal mode dynamics of macromolecules by the component synthesis method: residue clustering and multiple-component approach, *Biopolymers*, 34 (1994), pp. 321–335.

[41] D. A. Case, Normal mode analysis of protein dynamics, *Current Opinion in Structural Biology*, 4 (1994), pp. 285 – 290.

[42] L. Mouawad and D. Perahia, Diagonalization in a mixed basis: a method to compute low-frequency normal modes for large macromolecules, *Biopolymers*, 33 (1993), pp. 599–611.

[43] A. Ghysels, V. Van Speybroeck, E. Pauwels, S. Catak, B. R. Brooks, D. Van Neck, and M. Waroquier, Comparative study of various normal mode analysis techniques based on partial Hessians, *Journal of Computational Chemistry*, 31 (2010), pp. 994–1007.

[44] L. Skjaerven, S. M. Hollup, and N. Reuter, Normal mode analysis for proteins, *Journal of Molecular Structure: THEOCHEM*, 898 (2009), pp. 42 – 48.

[45] D. A. Case, T. E. Cheatham, T. Darden, H. Gohlke, R. Luo, K. M. Merz, A. Onufriev, C. Simmerling, B. Wang, and R. J. Woods, The Amber biomolecular simulation programs, *Journal of Computational Chemistry*, 26 (2005), pp. 1668–1688.

[46] A. MacKerell and D. Bashford, All-atom empirical potential for molecular modeling and dynamics studies of proteins, *The Journal of Physical Chemistry B*, 5647 (1998), pp. 3586–3616.

[47] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus, CHARMM: a program for macromolecular energy, minimization, and dynamics calculations, *Journal of Computational Chemistry*, 4 (1983), pp. 187–217.

[48] B. R. Brooks, I. C. L. Brooks, J. A. D. Mackerell, L. Nilsson, R. J. Petrella, B. Roux, Y. Won, G. Archontis, C. Bartels, S. Boresch, A. Caflisch, L. Caves, Q. Cui, A. R. Dinner, M. Feig, S. Fischer, J. Gao, M. Hodoscek, W. Im, K. Kuczera, T. Lazaridis, J. Ma, V. Ovchinnikov, E. Paci, R. W. Pastor, C. B. Post, J. Z. Pu, M. Schaefer, B. Tidor, R. M. Venable, H. L. Woodcock, X. Wu, W. Yang, D. M. York, and M. Karplus, CHARMM: the biomolecular simulation program, *Journal of Computational Chemistry*, 30 (2009), pp. 1545–1614.

[49] A. MacKerel Jr., C. Brooks III, L. Nilsson, B. Roux, Y. Won, and M. Karplus, *CHARMM: the Energy Function and Its Parameterization with an Overview of the Program*, vol. 1 of The Encyclopedia of Computational Chemistry, John Wiley & Sons: Chichester, 1998, pp. 271–277.

[50] C. Oostenbrink, A. Villa, A. E. Mark, and W. F. Van Gunsteren, A biomolecular force field based on the free enthalpy of hydration and solvation: the GROMOS force-field parameter sets 53a5 and 53a6, *Journal of Computational Chemistry*, 25 (2004), pp. 1656–1676.

[51] G. A. Kaminski, R. A. Friesner, J. Tirado-Rives, and W. L. Jorgensen, Evaluation and reparametrization of the OPLS-AA force field for proteins via comparison with accurate quantum chemical calculations on peptides, *Journal of Physical Chemistry B*, 105 (2001), pp. 6474–6487.

[52] D. E. Shaw, R. O. Dror, J. K. Salmon, J. P. Grossman, K. M. Mackenzie, J. A. Bank, C. Young, M. M. Deneroff, B. Batson, K. J. Bowers, E. Chow, M. P. Eastwood, D. J. Ierardi, J. L. Klepeis, J. S. Kuskin, R. H. Larson, K. Lindorff-Larsen, P. Maragakis, M. A. Moraes, S. Piana, Y. Shan, and B. Towles, *Millisecond-scale molecular dynamics simulations on Anton*, in Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, Nov 2009, pp. 1–11.

[53] D. E. D. Shaw, P. Maragakis, K. Lindorff-Larsen, S. Piana, R. O. Dror, M. P. Eastwood, J. a. Bank, J. M. Jumper, J. K. Salmon, Y. Shan, and W. Wriggers, Atomic-level characterization of the structural dynamics of proteins, *Science*, 330 (2010), pp. 341–346.

[54] M. Gur, E. Zomot, and I. Bahar, Global motions exhibited by proteins in micro- to milliseconds simulations concur with anisotropic network model predictions, *Journal of Chemical Physics*, 139 (2013).

[55] P. H. Hünenberger, *Thermostat Algorithms for Molecular Dynamics Simulations*, Springer Berlin Heidelberg, 2005, pp. 105–149.

[56] H. J. C. Berendsen, J. P. M. Postma, W. F. van Gunsteren, A. DiNola, and J. R. Haak, Molecular dynamics with coupling to an external bath, *The Journal of Chemical Physics*, 81 (1984), pp. 3684–3690.

[57] W. G. Hoover, Canonical dynamics: equilibrium phase-space distributions, *Physical Review A*, 31 (1985), pp. 1695–1697.

[58] S. Nosé, A molecular dynamics method for simulations in the canonical ensemble, *Molecular Physics*, 52 (1984), pp. 255–268.

[59] http://www.moltemplate.org/, accessed Jan 07, 2017.

[60] http://www.swissparam.ch, accessed Dec 20, 2016.

[61] S. Plimpton, Fast parallel algorithms for short-range molecular dynamics, *Journal of Computational Physics*, 117 (1995), pp. 1 – 19.

[62] M. J. Abraham, T. Murtola, R. Schulz, S. Páll, J. C. Smith, B. Hess, and E. Lindahl, GROMACS: high performance molecular simulations through multi-level parallelism from laptops to supercomputers, *SoftwareX*, 1–2 (2015), pp. 19 – 25.

[63] W. Humphrey, A. Dalke, and K. Schulten, VMD: visual molecular dynamics, *Journal of Molecular Graphics*, 14 (1996), pp. 33–8, 27–8.

[64] A. Stukowski, Visualization and analysis of atomistic simulation data with OVITO - the open visualization tool, *Modelling and Simulation in Materials Science and Engineering*, 18 (2010), p. 015012.

[65] H. Grubmüller, H. Heller, A. Windemuth, and K. Schulten, Generalized Verlet algorithm for efficient molecular dynamics simulations with long-range interactions, *Molecular Simulation*, 6 (1991), pp. 121–142.

[66] M. Tuckerman, B. J. Berne, and G. J. Martyna, Reversible multiple time scale molecular dynamics, *The Journal of Chemical Physics*, 97 (1992), p. 1990.

[67] E. Hairer, C. Lubich, and G. Wanner, Geometric numerical integration illustrated by the Störmer-Verlet method, *Acta numerica*, 12 (2003), pp. 399–450.

[68] J.-P. Ryckaert, G. Ciccotti, and H. J. Berendsen, Numerical integration of the Cartesian equations of motion of a system with constraints: molecular dynamics of n-alkanes, *Journal of Computational Physics*, 23 (1977), pp. 327–341.

[69] H. C. Andersen, RATTLE: a "velocity" version of the SHAKE algorithm for molecular dynamics calculations, *Journal of Computational Physics*, 52 (1983), pp. 24–34.

[70] H. Rubin and P. Ungar, Motion under a strong constraining force, *Communications on Pure and Applied Mathematics*, 10 (1957), pp. 65–87.

[71] S. Reich, Multiple time scales in classical and quantum–classical molecular dynamics, *Journal of Computational Physics*, 151 (1999), pp. 49–73.

[72] D. Dubbeldam, G. A. E. Oxford, R. Krishna, L. J. Broadbelt, and R. Q. Snurr, Distance and angular holonomic constraints in molecular simulations, *The Journal of Chemical Physics*, 133 (2010), p. 034114.

[73] M. Mandziuk and T. Schlick, Resonance in the dynamics of chemical systems simulated by the implicit midpoint scheme, *Chemical Physics Letters*, 237 (1995), pp. 525 – 535.

[74] U. M. Ascher and S. Reich, The midpoint scheme and variants for Hamiltonian systems: advantages and pitfalls, *SIAM Journal on Scientific Computing*, 21 (1999), pp. 1045–1065.

[75] N. Schafer and D. Negrut, A quantitative assessment of the potential of implicit integration methods for molecular dynamics simulation, *Journal of Computational and Nonlinear Dynamics*, 5 (2010), pp. 031012–031012.

[76] A. Stern and E. Grinspun, Implicit-explicit variational integration of highly oscillatory problems, *Multiscale Modeling & Simulation*, 7 (2009), pp. 1779–1794.

[77] V. F. Petrenko and R. W. Whitworth, *Physics of Ice*, Oxford Univ. Press, reprint. of 1999 ed., 2003.

[78] K. Röttger, A. Endriss, J. Ihringer, S. Doyle, and W. F. Kuhs, Lattice constants and thermal expansion of H_2O and D_2O ice I*h* between 10 and 265K. Addendum, *Acta crystallographica Section B*, 68 (2012), p. 91.

[79] M. Chaplin, *Water Structure and Science*, 2015. [Online; accessed 2015-11-13].

[80] O. Teleman, B. Jönsson, and S. Engström, A molecular dynamics simulation of a water model with intramolecular degrees of freedom, *Molecular Physics*, 60 (1987), pp. 193–203.

[81] H. J. C. Berendsen, J. R. Grigera, and T. P. Straatsma, The missing term in effective pair potentials, *The Journal of Physical Chemistry*, 91 (1987), pp. 6269–6271.

[82] M. a. González and J. L. F. Abascal, A flexible model for water based on TIP4P/2005, *The Journal of Chemical Physics*, 135 (2011), p. 224516.

[83] W. M. Brown, P. Wang, S. J. Plimpton, and A. N. Tharrington, Implementing molecular dynamics on hybrid high performance computers – short range forces, *Computer Physics Communications*, 182 (2011), pp. 898–911.

[84] W. M. Brown, A. Kohlmeyer, S. J. Plimpton, and A. N. Tharrington, Implementing molecular dynamics on hybrid high performance computers – particle–particle particle-mesh, *Computer Physics Communications*, 183 (2012), pp. 449–459.

[85] B. García-Archilla, J. M. Sanz-Serna, and R. D. Skeel, Long-time-step methods for oscillatory differential equations, *SIAM Journal on Scientific Computing*, 20 (1998), pp. 930–963.

[86] J. A. Izaguirre, Q. Ma, T. Matthey, J. Willcock, T. Slabach, B. Moore, and G. Viamontes, Overcoming instabilities in Verlet-I/r-RESPA with the mollified impulse method, *Proceedings of 3rd International Workshop on Methods for Macromolecular Modeling*, 24 (2002), pp. 146–174.

[87] J. M. Sanz-Serna, Mollified Impulse Methods for Highly Oscillatory Differential Equations, *SIAM Journal on Numerical Analysis*, 46 (2008), pp. 1040–1059.

[88] J. A. Izaguirre, S. Reich, and R. D. Skeel, Longer time steps for molecular dynamics, *Journal of Chemical Physics*, 110 (1999), p. 9853.

[89] S. Riniker, J. R. Allison, and W. F. van Gunsteren, On developing coarse-grained models for biomolecular simulation: a review, *Physical Chemistry Chemical Physics*, 14 (2012), p. 12423.

[90] E. Brini, E. a. Algaer, P. Ganguly, C. Li, F. Rodríguez-Ropero, and N. F. a. V. D. Vegt, Systematic coarse-graining methods for soft matter simulations – a review, *Soft Matter*, 9 (2013), pp. 2108–2119.

[91] A. Jain, N. Vaidehi, and G. Rodriguez, A fast recursive algorithm for molecular dynamics simulation, *Journal of Computational Physics*, 106 (1993), pp. 258–268.

[92] N. Vaidehi and A. Jain, Internal coordinate molecular dynamics: a foundation for multiscale dynamics, *The Journal of Physical Chemistry B*, 119 (2015), pp. 1233–1242.

[93] M. Müller, J. Dorsey, and L. McMillan, Stable real-time deformations, *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, (2002), pp. 49 – 54.

[94] E. Hairer and C. Lubich, Long-time energy conservation of numerical methods for oscillatory differential equations, *SIAM Journal on Numerical Analysis*, 38 (2000), pp. 414–441.

[95] M. M. Conde, C. Vega, and A. Patrykiejew, The thickness of a liquid layer on the free surface of ice as obtained from computer simulation, *The Journal of Chemical Physics*, 129 (2008), p. 014702.

[96] P. B. Louden and J. D. Gezelter, Simulations of solid-liquid friction at ice-I(h)/water interfaces., *The Journal of chemical physics*, 139 (2013), p. 194710.

[97] N. Samadashvili, B. Reischl, T. Hynninen, T. Ala-Nissilä, and a. S. Foster, Atomistic simulations of friction at an ice-ice interface, *Friction*, 1 (2013), pp. 242–251.

[98] E. M. Schulson and A. L. Fortt, Friction of ice on ice, *Journal of Geophysical Research: Solid Earth*, 117 (2012), pp. 1–18.

[99] N. Maeno, M. Arakawa, A. Yasutome, N. Mizukami, and S. Kanazawa, Ice-ice friction measurements, and water lubrication and adhesion-shear mechanisms, *Canadian Journal of Physics*, 81 (2003), p. 241.

[100] J. R. Errington and P. G. Debenedetti, Relationship between structural order and the anomalies of liquid water, *Nature*, 409 (2001), pp. 318–21.

[101] M. Marquart, J. Walter, J. Deisenhofer, W. Bode, and R. Huber, The geometry of the reactive site and of the peptide groups in trypsin, trypsinogen and its complexes with inhibitors, *Acta Crystallogr.,Sect.B*, 39 (1983), p. 480.

[102] N. Gō, A theorem on amplitudes of thermal atomic fluctuations in large molecules assuming specific conformations calculated by normal mode analysis, *Biophysical Chemistry*, 35 (1990), pp. 105 – 112.

[103] M. M. Tirion, Large amplitude elastic motions in proteins from a single-parameter, atomic analysis, *Physical Review Letters*, 77 (1996), pp. 1905–1908.

[104] A. Kitao, S. Hayward, and N. Go, Comparison of normal mode analyses on a small globular protein in dihedral angle space and cartesian coordinate space, *Biophysical Chemistry*, 52 (1994), pp. 107 – 114.

[105] E. Frezza and R. Lavery, Internal normal mode analysis (iNMA) applied to protein conformational flexibility, *Journal of Chemical Theory and Computation*, 11 (2015), pp. 5503–5512.

[106] O. Marques and Y.-H. Sanejouand, Hinge-bending motion in citrate synthase arising from normal mode calculations, *Proteins: Structure, Function, and Bioinformatics*, 23 (1995), pp. 557–560.

[107] R. G. Grimes, J. G. Lewis, and H. D. Simon, A shifted block Lanczos algorithm for solving sparse symmetric generalized eigenproblems, *SIAM Journal on Matrix Analysis and Applications*, 15 (1994), pp. 228–272.

[108] B. Philippe and Y. Saad, On correction equations and domain decomposition for computing invariant subspaces, *Computer Methods in Applied Mechanics and Engineering*, 196 (2007), pp. 1471 – 1483. Domain Decomposition Methods: recent advances and new challenges in engineering.

[109] C. Bekas and Y. Saad, Computation of smallest eigenvalues using spectral schur complements, *SIAM Journal on Scientific Computing*, 27 (2005), pp. 458–481.

[110] J. K. Bennighof and R. B. Lehoucq, An automated multilevel substructuring method for eigenspace computation in linear elastodynamics, *SIAM Journal on Scientific Computing*, 25 (2004), pp. 2084–2106.

[111] C. Yang, W. Gao, Z. Bai, X. S. Li, L.-Q. Lee, P. Husbands, and E. Ng, An algebraic substructuring method for large-scale eigenvalue calculation, *SIAM Journal on Scientific Computing*, 27 (2005), pp. 873–892.

[112] J. R. López-Blanco, R. Reyes, J. I. Aliaga, R. M. Badia, P. Chacón, and E. S. Quintana-Ortí, Exploring large macromolecular functional motions on clusters of multicore processors, *Journal of Computational Physics*, 246 (2013), pp. 275 – 288.

[113] M. Hochbruck, *Spezielle Themen der numerischen linearen Algebra*. Skriptum zur Vorlesung im SS2013 am KIT.

[114] Z. H. Bai, ed., *Templates for the Solution of Algebraic Eigenvalue Problems: a Practical Guide*, Software, environments, tools, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2000.

[115] http://www.netlib.org/utk/people/JackDongarra/etemplates/node104.html, accessed Dec 06, 2016.

[116] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins studies in the mathematical sciences, Johns Hopkins University Pr., Baltimore, Md., 4. ed., 2013.

[117] Y. Saad, *Numerical Methods for Large Eigenvalue Problems*, Classics in applied mathematics, Society for Industrial & Applied Mathematics, Philadelphia, rev. ed., 2011.

[118] G. H. Golub, Z. Zhang, and H. Zha, Large sparse symmetric eigenvalue problems with homogeneous linear constraints: the Lanczos process with inner–outer iterations, *Linear Algebra and its Applications*, 309 (2000), pp. 289 – 306.

[119] V. Simoncini and D. B. Szyld, Theory of inexact Krylov subspace methods and applications to scientific computing, *SIAM Journal on Scientific Computing*, 25 (2003), pp. 454 – 477.

[120] J. Van Den Eshof and G. L. G. Sleijpen, Inexact Krylov subspace methods for linear systems, *SIAM Journal on Matrix Analysis & Applications*, 26 (2004), pp. 125 – 153.

[121] M. A. Freitag and A. Spence, Shift-invert Arnoldi's method with preconditioned iterative solves, *SIAM Journal on Matrix Analysis & Applications*, 31 (2010), pp. 942 – 969.

[122] D. C. Sorensen, Implicit application of polynomial filters in a k-step Arnoldi method, *SIAM Journal on Matrix Analysis and Applications*, 13 (1992), pp. 357–385.

[123] ———, *Implicitly Restarted Arnoldi/Lanczos Methods for Large Scale Eigenvalue Calculations*, Springer Netherlands, Dordrecht, 1997, pp. 119–165.

[124] R. B. Lehoucq, *ARPACK users guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods*, Software, environments, tools, SIAM, Philadelphia, 1998.

[125] G. L. G. Sleijpen and H. A. V. der Vorst, A Jacobi-Davidson iteration method for linear eigenvalue problems, *SIAM Journal on Matrix Analysis and Applications*, 17 (1996), pp. 401–425.

[126] http://www.netlib.org/utk/people/JackDongarra/etemplates/node138.html, accessed Dec 06, 2016.

[127] http://www.netlib.org/utk/people/JackDongarra/etemplates/node144.html, accessed Dec 06, 2016.

[128] http://www.netlib.org/utk/people/JackDongarra/etemplates/node145.html, accessed Dec 06, 2016.

[129] G. L. G. Sleijpen and H. A. Van der Vorst, A Jacobi-Davidson iteration method for linear eigenvalue problems, *SIAM Review*, 42 (2000), pp. 267–293.

[130] ———, *The Jacobi-Davidson method for eigenvalue problems and its relation with accelerated inexact Newton scheme*, in Proceedings of the Second IMACS International Symposium on Iterative Methods in Linear Algebra, IMACS, 2006.

[131] J. van den Eshof, The convergence of Jacobi-Davidson iterations for Hermitian eigenproblems, *Numerical Linear Algebra with Applications*, 9 (2002), pp. 163–179.

[132] T. Sakurai and H. Tadano, CIRR: a Rayleigh-Ritz type method with contour integral for generalized eigenvalue problems, *Hokkaido Mathematical Journal*, 36 (2007), pp. 745–757.

[133] A. Imakura, L. Du, and T. Sakurai, A block Arnoldi-type contour integral spectral projection method for solving generalized eigenvalue problems, *Applied Mathematics Letters*, 32 (2014), pp. 22 – 27.

[134] E. Polizzi, Density-matrix-based algorithm for solving eigenvalue problems, *Physical Review B*, 79 (2009), p. 115112.

[135] ———, A high-performance numerical library for solving eigenvalue problems: FEAST solver v2.0 user's guide, *CoRR*, abs/1203.4031 (2012).

[136] A. Kuzmin, M. Luisier, and O. Schenk, *Fast methods for computing selected elements of the Greens function in massively parallel nanoelectronic device simulations*, in Euro-Par 2013 Parallel Processing, F. Wolf, B. Mohr, and D. Mey, eds., vol. 8097 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 533–544.

[137] O. Schenk, M. Bollhöfer, and R. A. Römer, On large-scale diagonalization techniques for the Anderson model of localization, *SIAM Review*, 50 (2008), pp. 91–112.

[138] O. Schenk, A. Wächter, and M. Hagemann, Matching-based preprocessing algorithms to the solution of saddle-point problems in large-scale nonconvex interior-point optimization, *Computational Optimization and Applications*, 36 (2007), pp. 321–341.

[139] E. Cuthill and J. McKee, *Reducing the bandwidth of sparse symmetric matrices*, in Proceedings of the 1969 24th National Conference, ACM '69, New York, NY, USA, 1969, ACM, pp. 157–172.

[140] A. George and J. W. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall series in computational mathematics, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[141] W. F. Tinney and J. W. Walker, Direct solutions of sparse network equations by optimally ordered triangular factorization, *Proceedings of the IEEE*, 55 (1967), pp. 1801–1809.

[142] A. George and J. W. Liu, The evolution of the minimum degree ordering algorithm, *SIAM Review*, 31 (1989), pp. 1–19.

[143] L. O. Mafteiu-Scai, The bandwidths of a matrix. a survey of algorithms, *Annals of West University of Timisoara-Mathematics*, 52 (2014), pp. 183–223.

[144] Y. Saad, *Iterative Methods for Sparse Linear Systems*, SIAM, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2. ed., 2003.

[145] A. Greenbaum, *Iterative Methods for Solving Linear Systems*, Frontiers in applied mathematics, Society for Industrial and Applied Mathematics, Philadelphia, 1997.

[146] B. Fischer, *Polynomial Based Iteration Methods for Symmetric Linear Systems*, Wiley-Teubner series advances in numerical mathematics, Wiley-Teubner, Chichester, 1996.

[147] W. Hackbusch, *Iterative Solution of Large Sparse Systems of Equations*, Springer, 2nd ed., 2016.

[148] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, Society for Industrial and Applied Mathematics, 1994.

[149] C. C. Paige and M. A. Saunders, Solution of sparse indefinite systems of linear equations, *SIAM Journal on Numerical Analysis*, 12 (1975), pp. 617–629.

[150] R. W. Freund and N. M. Nachtigal, QMR: a quasi-minimal residual method for non-Hermitian linear systems, *Numerische Mathematik*, 60 (1991), pp. 315–339.

[151] ———, *A new Krylov-subspace method for symmetric indefinite linear systems*, in Proceedings of the 14th IMACS World Congress on Computational and Applied Mathematics, vol. 1253, 1994.

[152] S.-C. T. Choi, C. C. Paige, and M. A. Saunders, MINRES-QLP: a Krylov subspace method for indefinite or singular symmetric systems, *SIAM Journal on Scientific Computing*, 33 (2011), pp. 1810–1836.

[153] M. Benzi, Preconditioning techniques for large linear systems: a survey, *Journal of Computational Physics*, 182 (2002), pp. 418 – 477.

[154] M. Bollhöfer and Y. Saad. ILUPACK - preconditioning software package. Available online at http://ilupack.tu-bs.de/. Release V2.4, June 2011.

[155] ———, Multilevel preconditioners constructed from inverse-based ILUs, *SIAM Journal on Scientific Computing*, 27 (2006), pp. 1627–1650.

[156] P. E. Gill, W. Murray, D. B. Ponceleón, and M. A. Saunders, Preconditioners for indefinite systems arising in optimization, *SIAM Journal on Matrix Analysis and Applications*, 13 (1992), pp. 292–311.

[157] Y. Notay, Algebraic theory of two-grid methods, *Numerical Mathematics: Theory, Methods and Applications*, 8 (2015), p. 168–198.

[158] http://www.rcsb.org, accessed Dec 20, 2016.

[159] B. N. Parlett, *The Symmetric Eigenvalue Problem*, Classics in applied mathematics, Society for Industrial and Applied Mathematics, Philadelphia, 1998.

[160] M. Bollhöfer and Y. Notay, JADAMILU: a software code for computing selected eigenvalues of large sparse symmetric matrices, *Computer Physics Communications*, 177 (2007), pp. 951 – 964.

[161] H. M. Aktulga, L. Lin, C. Haine, E. G. Ng, and C. Yang, Parallel eigenvalue calculation based on multiple shift–invert Lanczos and contour integral based spectral projection method, *Parallel Computing*, 40 (2014), pp. 195 – 212. 7th Workshop on Parallel Matrix Algorithms and Applications.

[162] N. I. M. Gould, J. A. Scott, and Y. Hu, A numerical evaluation of sparse direct solvers for the solution of large sparse symmetric linear systems of equations, *ACM Trans. Math. Softw.*, 33 (2007).

[163] M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids*, Oxford science publications, Clarendon Pr., Oxford, 1987.

[164] C. Sagui and T. A. Darden, Molecular dynamics simulations of biomolecules: long-range electrostatic effects, *Annual Review of Biophysics and Biomolecular Structure*, 28 (1999), pp. 155–179.