Appendix A

Numerical Methods for Quantum Mechanics

A.1 The Fourier Transform

In quantum mechanics it is possible to represent the wavefunctions describing a system in different representations, i.e. one can represent a state $|\Phi\rangle$ in position space ($\Phi(x) = \langle x|\ \Phi\rangle$) or in momentum space ($\Phi(k) = \langle k|\ \Phi\rangle$) [92]. These two representations are connected through a Fourier transform (FT) defined (for one dimensional wavefunctions) as [152]

$$\tilde{\Psi}(k) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} dx\ e^{-ikx} \Psi(x) \tag{A.1}$$

and the inverse relation as

$$\Psi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} dk\ e^{ikx} \tilde{\Psi}(k). \tag{A.2}$$

Other physical representations connected in such a way by a FT are, e.g. the time and frequency domain. The possibility to change between representations is often employed in quantum mechanical applications. The most prominent is the split operator propagator (presented in Section 2.4.3), which needs a FT performed on a grid, and, because one has to do two transformations per time step, one which scales efficiently.

In the following description, the notation for a FT between a function of time $f(t)$ and its counterpart in the frequency domain $F(\nu)$

$$F(\nu) = \int_{-\infty}^{+\infty} dt\ e^{-i2\pi\nu t} f(t) \tag{A.3}$$

will be used.

A.1.1 The Discrete Fourier Transform

For the discrete FT usually needed for numerical applications, one starts with a set of N discretely sampled function values of $f(t)$:

$$f_n = f(t_n) \quad t_n = n\ \Delta t \quad n = 0, 1, 2, ...N-1, \tag{A.4}$$

with the (convenient, but not necessary) condition of even N. The introduction of a finite sampling time Δt will introduce the so called Nyquist frequency $\nu_{\max}$, which defines the maximum frequency which can be resolved with the existing data points. The sampling theorem gives

$$\nu_{\max} = \frac{1}{2\Delta_t}. \tag{A.5}$$

As the discrete FT [152] starts from a set of N sampled points, the transformation will only be able to generate N discrete values for the frequency domain. These, according to Eq. (A.5), have to be in the range $-\frac{1}{2\Delta_t} \leq \nu_n \leq \frac{1}{2\Delta_t}$, which leads to a choice of the grid in the frequency domain in the form

$$F_j = F(\nu_j) \quad ;\ \nu_j = \frac{j}{N\Delta t} \quad ;\ j = -\frac{N}{2}, ..., \frac{N}{2}. \tag{A.6}$$

Now Eq. (A.3) can be approximated by a discrete sum

$$\begin{aligned} F_j = F(\nu_j) &= \int_{-\infty}^{+\infty} dt\ e^{-i2\pi\nu_j t} f(t) \\ &\approx \sum_{n=0}^{N-1} \Delta t\ e^{-j2\pi\nu_j t_n} f(t) \\ F_j &= \Delta t \sum_{n=0}^{N-1} e^{-i2\pi j\ n/N} f_n. \end{aligned} \tag{A.7}$$

This equation is the discrete FT. It has to be noted, that Eq. (A.7) is periodic in n with a period of N, so that from the $N+1$ values F_i defined in Eq. (A.6) only N are independent, i.e., $F_{-\frac{N}{2}}$ is equal to $F_{\frac{N}{2}}$.

A.1.2 The Fast Fourier Transform

The discrete FT given by Eq. (A.7) can be rewritten as a multiplication of a matrix with a vector:

$$\begin{pmatrix} F_{j_0} \\ \vdots \\ F_{j_{N-1}} \end{pmatrix} = \begin{pmatrix} e^{aj_0\ n_0} & \dots & e^{aj_0\ n_{N-1}} \\ \vdots & \ddots & \vdots \\ e^{aj_{N-1}\ n_0} & \dots & e^{aj_{N-1}\ n_{N-1}} \end{pmatrix} \begin{pmatrix} f_{n_0} \\ \vdots \\ f_{n_{N-1}} \end{pmatrix} \tag{A.8}$$

with $a = -i2\pi/N$. This implies that each discrete FT would need $N \times N$ multiplications of complex numbers, making it an algorithm scaling on the order of $\mathcal{O}(N^2)$ with the number

of grid points. This effort can be reduced to $\mathcal{O}(N \log_2 N)$ by the Fast Fourier transform (FFT) [152] algorithm. This makes use of the Danielson-Lanczos Lemma, which states that a discrete Fourier transform of length N can be rewritten as the sum of two discrete Fourier transforms, each of length $N/2$. One of them containing the even-numbered points e and the other the odd-numbered o. This can be shown as follows:

$$\begin{aligned} F_j &= \sum_{n=0}^{N-1} e^{-i2\pi j\ n/N} f_n \\ &= \sum_{n=0}^{N/2-1} e^{-i2\pi j\ 2n/N} f_{2n} + \sum_{n=0}^{N/2-1} e^{-i2\pi j\ (2n+1)/N} f_{2n+1} \\ &= \sum_{n=0}^{N/2-1} e^{-i2\pi j\ n/(N/2)} f_{2n} + e^{-i2\pi j/N} \sum_{n=0}^{N/2-1} e^{-i2\pi j\ n/(N/2)} f_{2n+1} \\ &= F_j^e + e^{-i2\pi j/N} F_j^o. \end{aligned} \tag{A.9}$$

This can be applied recursively, to halve the length the two resulting discrete FT again, until the point is reached, where one has N discrete FT of length one. A Discrete FT for $N = 1$ is just the copying of the input to the output:

$$F_j^{eeoeeoo...eeooe} = f_n. \tag{A.10}$$

Each F_j is then calculated by first reordering the input, and then calculating the transforms of length 2,4,8,...N by multiplying with the factors $e^{-i2\pi j/N}$ from Eq. (A.9). As this has to be done $\log_2 N$ times, the total numerical effort of the FFT is $\mathcal{O}(N \log_2 N)$.

A.2 The Fourier–Grid Method

The goal of this method is to give a closed expression for the matrix elements of the total Hamiltonian on a discrete grid in position space [99]. Writing the Hamiltonian as a function of the position and the momentum operator one has

$$\hat{\mathbf{H}} = \frac{\hat{\mathbf{p}}^2}{2m} + V(\hat{\mathbf{x}}). \tag{A.11}$$

As the next step the two bases for the Hilbert space are defined. In position space one has

$$\hat{\mathbf{x}} \left| x \right\rangle = x \left| x \right\rangle \tag{A.12}$$

with the orthonormality and completeness relation

$$\left\langle x' \right| \left. x \right\rangle = \delta(x - x') \tag{A.13}$$

$$\mathbf{1} = \int_{-\infty}^{\infty} dx\ \left| x \right\rangle \left\langle x \right|. \tag{A.14}$$

In this basis the potential is diagonal:

$$\langle x'|\, V(\hat{\mathbf{x}})\, |x\rangle = V(x)\delta(x - x'). \tag{A.15}$$

The equivalent relations for the momentum space are:

$$\hat{\mathbf{p}}\, |k\rangle = \hbar k\, |k\rangle \tag{A.16}$$

$$\langle k'|\, k\rangle = \delta(k - k') \tag{A.17}$$

and

$$\mathbf{1} = \int_{-\infty}^{\infty} dk\ |k\rangle\, \langle k|\,. \tag{A.18}$$

This basis results in a diagonal kinetic energy part:

$$\langle k'|\, \hat{\mathbf{T}}\, |k\rangle = \frac{\hbar^2 k^2}{2m}\delta(k - k'). \tag{A.19}$$

In addition the Fourier transformation gives a direct relation between the two basis sets:

$$\langle k|\, x\rangle = \frac{1}{\sqrt{2\pi}} e^{-ikx}. \tag{A.20}$$

Now it is possible to write the total Hamiltonian $\hat{\mathbf{H}}$ in position representation:

$$\langle x'|\, \hat{\mathbf{H}}\, |x\rangle = \langle x'|\, \hat{\mathbf{T}}\, |x\rangle + V(x)\delta(x - x'). \tag{A.21}$$

The first term of this result can be transformed by inserting the completeness relation or the momentum (Eq. (A.18)) twice and applying Eq. (A.20) and Eq. (A.19):

$$\langle x'|\, \hat{\mathbf{H}}\, |x\rangle = \frac{1}{2\pi} \int_{-\infty}^{\infty} dk\, e^{ik(x-x')} \hat{\mathbf{T}}(k) + V(x)\delta(x - x'), \tag{A.22}$$

where the exponential term can be interpreted as arising from taking a forward and backward Fourier transformation.

For numerical treatment now the continuous range of coordinate values has to be replaced by a discrete grid of N values

$$x_n = n\Delta x \qquad n = 1, \ldots, N \tag{A.23}$$

where Δx is the grid spacing. This discrete position grid defines the grid spacing in momentum space via the relation

$$\Delta k = \frac{2\pi}{N\Delta x}. \tag{A.24}$$

With this one can write a discretized version of the Hamilton matrix element in position space as

$$H_{mn} = \frac{1}{\Delta x} \left[\sum_{l=-K}^{K} \frac{e^{il2\pi(m-n)/N}}{N} \cdot T_l + V(x_m)\delta_{mn} \right] \tag{A.25}$$

with $K = (N-1)/2$ and

$$T_l = \frac{\hbar^2}{2m}(l\Delta k)^2. \tag{A.26}$$

Diagonalizing this Hamiltonian matrix will result in the eigenenergies and eigenfunctions for the discretized system.

A.3 Integrators

In the most simple approximation, an initial value problem for a first order differential equation like

$$\frac{dy}{dx} = z(x, y), \tag{A.27}$$

as they are encountered in the numerics of the time dependent Schrödinger equation, can be solved with the so called Euler method. (As any ordinary differential equation can be rewritten as a set of coupled first order differential equations, this is true for all initial value problems [152].) To start with, one gets the initial values x_i and y_i for Eq. (A.27) and then has to calculate the value y_f for the location x_f. In the Euler method one now rewrites dx as a finite step Δx and multiplies Eq. (A.27) by Δx. This gives an approximation for the change in y_n when the independent variable x is increased by one step Δx, i.e., from x_n to $x_{n+1} = x_n + \Delta x$. Now it is possible to calculate the values of y_{n+1} starting from y_i until one reaches y_f as follows:

$$y_{n+1} = y_n + z(x_n, y_n)\Delta x. \tag{A.28}$$

In this method the error made in each iteration is only one order smaller than the length of the step taken [$\mathcal{O}(\Delta x^2)$]. Due to this the Euler method is not suitable for long propagations, as even for very small steps the inaccuracies will start to add up quickly.

A.3.1 The Runge–Kutta Method

To improve the Euler method, one approach is to use information about the derivative of y not only from the beginning of the interval, but also from trial points within it. The Runge–Kutta [152] method does this by using one or several trial steps within the interval to determine $z(x, y)$ for different points in the interval Δx. Adding up the right combinations of these trial steps can eliminate the higher order error terms. In the commonly used forth-order Runge–Kutta formula one can reduce the order of the error to $\mathcal{O}(\Delta x^5)$ by applying the following algorithm:

$$\begin{aligned} y_{s1} &= z\left(x_n, y_n\right)\Delta x \\ y_{s2} &= z\left(x_n + \frac{\Delta x}{2}, y_n + \frac{y_{s1}}{2}\right)\Delta x \\ y_{s3} &= z\left(x_n + \frac{\Delta x}{2}, y_n + \frac{y_{s2}}{2}\right)\Delta x \end{aligned}$$

$$\begin{aligned} y_{s4} &= z\left(x_n + \Delta x, y_n + y_{s3}\right)\Delta x \\ y_{n+1} &= y_n + \frac{y_{s1}}{6} + \frac{y_{s2}}{3} + \frac{y_{s3}}{3} + \frac{y_{s4}}{6}. \end{aligned}$$

While this requires several evaluations of the derivative in the interval, it usually allows a much larger step size, without accumulating significant errors.

Appendix B

Evolutionary Algorithms

Evolutionary algorithm (EA) is an umbrella term used to describe computer- based problem solving systems which use computational models of evolutionary processes as key elements in their design and implementation. More precisely, EAs maintain a population of structures, that evolve according to rules of selection and other operators, that are referred to as "search operators", (or genetic operators), such as recombination and mutation. Each individual in the population receives a measure of it's fitness in the environment, which has to be defined as a function of all the characteristics this individual has. Reproduction focuses attention on high fitness population members, introducing the the process of natural selection. Recombination and mutation perturb those individuals, providing a random way to explore the fitness landscape. This landscape is described by the defined quality of the individual, in dependence on all the N parameters, which are allowed to vary, thereby generating a N-dimensional "quality hypersurface". Although simplistic from a biologist's viewpoint, these algorithms are sufficiently complex to provide robust and powerful adaptive search mechanisms, which are especially useful for problems where no analytical searching method exists. As the methods are very general, it is also possible, to search, e.g., for an optimal fitting formula of a potential, as shown in [153]

There are two different approaches to implement the search for the optimal individual. In genetic algorithms, the parameters are encoded into discrete sequences or strings, similar to the human genes, which are then allowed to recombine and mutate using certain rules. In contrast to this, the evolutionary approach normally codes the parameters as their real values and varies the population by small random fluctuations of these numbers. The algorithms used in this work are based on the genetic approach as presented in the book of, for example, Goldberg [114].

The advantage of these methods over the usual, gradient based approaches is, that they stay robust in the presence of several local extrema. While a gradient search will always converge to the nearest local extremum, a EA will normally not get trapped there. The reasons for this will be shortly highlighted in the following sections.

B.1 Genetic Algorithms (GA)

GAs differ from the more traditional gradient based search algorithms in four important points:

1. GAs work with a coding of the parameter set, not the parameters themselves.
2. GAs search from a population of points, not from a single point.
3. GAs use quality information (fitness) of the points, not gradients or other derivative information.
4. GAs propagate by random transitions, not via deterministic rules.

The second and the last point are mainly responsible for the ability of the GAs to find several local extrema – the fact that the scattered population can move to several, different minima, and the random mutations, which prevent trapping in locally optimal niches.

In practice, this genetic model of computation is implemented by having arrays of bits or characters to represent the parameters in finite–length strings. An individual is then described by a set of these strings, its "chromosomes". Within a population of these individuals, new members are then created by "mating" two parents, giving parents with higher fitness a better chance to reproduce. The chromosomes of the child are created from the ones of the two parents by crossover and mutation. For a crossover, the corresponding chromosomes of the parents are cut at a random position, and then the parts are swapped between them, to produce two new genes. After two new individuals are created in this way, each of the encoded genes has a chance of mutation. To give an example, lets consider a chromosome representing a parameter allowed to vary from $[0, 1]$. For the computer, a binary encoding is most natural, so this parameter is encoded in a binary string of arbitrary, fixed length. If one chooses ten bits for this chromosome, one gets an encoding in the form

$$\begin{array}{rcl} \texttt{0000000000} & \equiv & 0 \\ \texttt{0000000001} & \equiv & 1/1022 \\ \vdots & & \vdots \\ \texttt{1111111111} & \equiv & 1 \end{array}$$

The chromosome allows to encode 1023 discrete values of the parameter within its domain. This already shows the one limitation of this approach – to encode a large parameter domain, one often needs quite a lot of genes to allow for a fine enough grid. The crossover between two of these chromosomes is the performed by choosing a random position to cut these strings and recombine the exchanged parts:

$$\begin{array}{c} \texttt{0110}\,|\,\texttt{001101} \\ \texttt{0001}\,|\,\texttt{111010} \end{array} \Longrightarrow \begin{array}{c} \texttt{0001}\,|\,\texttt{001101} \\ \texttt{0110}\,|\,\texttt{111010} \end{array}$$

After this crossover, each of the the bits in the new individuals has a small random chance of mutating, i.e. flipping from zero to one or vice versa. Due to a single mutation in a high significant bit the new individual could end up in a completely different part of the parameter space, than the parents. This shows, that this arbitrary mutation rate should be small enough to prevent completely random fluctuation of the parameter, but large enough to allow some individuals to explore new parts of parameter space.

The implementation of GAs on the computer can be written in the following pseudo code segment:

```
PSEUDO CODE
   Algorithm GA is

        // start with an initial time
        t := 0;

        // initialize a usually random population of individuals
        initpopulation P (t);

        // evaluate fitness of all initial individuals of population
        evaluate P (t);

        // test for termination criterion (time, fitness, etc.)
        while not done do

             // increase the time counter
             t := t + 1;

             // select a sub-population for offspring production
             P' := selectparents P (t);

             // recombine the "genes" of selected parents
             recombine P' (t);

             // perturb the mated population stochastically
             mutate P' (t);

             // evaluate it's new fitness
             evaluate P' (t);

             // select the survivors from actual fitness
             P := survive P,P' (t);
```

```
    od
end GA.
```

B.2 Evolutionary Programming (EP)

A slightly different approach than GAs is used within EP. There are two main differences.

First, there is no constraint on the representation. The typical GA approach involves encoding the parameters of the problem into strings of representative tokens, the chromosomes. In EP, the parameters are used directly, i.e. a parameter varying in the domain $[0, 1]$ would simply be represented as a real number in this range.

Second, the new generation normally is only generated by mutation of the parents, there will be no crossover between individuals. A mutation in this approach changes the parameters of the offspring according to a statistical distribution which weights minor variations in the parameter value as highly probable and substantial variations as increasingly unlikely. This mutation method differs drastically from the one used for GAs, where it is equally likely for all bits to mutate, no matter if this results in a large or a small shift of the parameter.

Therefore, the EP strategy places ore emphasis on the connection between parents and their offspring. It does not employ the techniques, which are usually termed as genetic operations (e.g. the crossover between two individuals), but more resembles a random search with a multitude of starting points.

An pseudo code implementation of this approach is given below:

```
PSEUDO CODE
   Algorithm EP is

        // start with an initial time
        t := 0;

        // initialize a usually random population of individuals
        initpopulation P (t);

        // evaluate fitness of all initial individuals of population
        evaluate P (t);

        // test for termination criterion (time, fitness, etc.)
        while not done do

            // perturb the whole population stochastically
            P'(t) := mutate P (t);
```

```
            // evaluate it's new fitness
            evaluate P' (t);

            // stochastically select the survivors from actual fitness
            P(t+1) := survive P(t),P'(t);

            // increase the time counter
            t := t + 1;
      od
end EP.
```

Appendix C

Molecular Dynamics (MD) Algorithms

When a molecular system gets larger, or an ensemble of several molecules has to be examined (e.g. a microscopic description for the bath of a dissipative quantum system, cf. Section 2.3.3ff.), the methods of quantum mechanics cannot be used any more. Even with severe approximations, it is not possible to solve the time dependent Schrödinger equation for a system of several hundred interacting particles (which would only represent a relatively small bio–molecule, to represent a bath, several thousand atoms normally have to be included). To simulate the behavior of such "macroscopic" (with respect to the quantum world) systems, the methods of molecular dynamics can be used. These methods use Newtons classical equations of motion for each atom, thereby loosing the quantization of the energy, but simplifying the problems greatly. These methods are discussed in several textbooks for further reference [141, 142].

C.1 Equations of Motion

For an ensemble of N interacting particles, located at the positions $\mathbf{r}_i$ $(i = 1, \ldots, N)$, the motion of each particle is given by the solution of the equation of motion derived from Newtons second law ($m_i \ddot{\mathbf{r}}_i = \mathbf{F}_i$), where $\mathbf{F}_i$ is the total force acting on the i^{th} particle. The total force is calculated from the pair interaction potential between the atoms, which are modeled differently for inter- and intramolecular interactions. Two particles, separated by $r_{ij} = |\mathbf{r}_i - \mathbf{r}_j|$, belonging to different molecules will normally use a Lennard–Jones (LJ) interaction potential and, if they are charged, a Coulomb potential, in the form

$$u(r_{ij}) = 4\epsilon_{ij} \left\{ \left(\frac{\sigma_{ij}}{r_{ij}} \right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}} \right)^{6} \right\} + \frac{q_i q_j e^2}{r_{ij}}. \tag{C.1}$$

Here the first (LJ) part of the sum describes the short range van der Waals type interactions. The r^{-6} term describes the potential between two induced dipoles, resulting in

a weak attractive force. The r^{-12} term models the repulsion at extremely short range, when the electron clouds of the atoms begin to touch. This modeling is chosen for numerical convenience, as this term can be calculated by simply squaring the r^{-6} term, saving computational effort. σ and ϵ are the parameters of the LJ model potential, and have to be fitted to experimental results. They are different for each pair of different atoms. The second part of the sum gives the Coulomb interaction of the particles, which exists when they carry any partial charge q_i. The potential between particles belonging to the same molecule are normally modeled using bond-, angle- and torsion forces, which can be derived from quantum chemical calculations or from spectroscopic data. A more detailed treatment of those intramolecular potentials can be found in the literature [141, 142, 154].

From these pair potentials the interaction force between the atoms is then calculated ($\mathbf{f}_{ij} = -\nabla u(r_{ij})$). These pair forces add up independently, resulting in the total force $\mathbf{F}_i$ on one particle. Writing Newtons equation with this force in the Hamilton form (i.e. split into two first order differential equations), one gets the basic equations of motion for a single particle within a larger ensemble:

$$\begin{aligned} \dot{\mathbf{r}}_i &= \mathbf{p}_i/m_i = \mathbf{v}_i \\ \dot{\mathbf{p}}_i &= \mathbf{F}_i = \sum_{i \neq j} \mathbf{f}_{ij} \end{aligned} \tag{C.2}$$

The solution of this problem for a system of N particles then requires the integration of the $6N$ first order differential equations given by Eq. (C.2).

C.2 Numerical Integration of the MD Equations

The numerical solution of the equations of motion given by Eq. (C.2) requires a method which is very fast, as it has to handle a very large number of equations. At the same time it must be accurate enough to correctly approximate the properties of classical trajectories: For time- and velocity–independent pair potentials the total energy of the system has to be conserved, and the trajectories have to be invariant under time reversal. Maybe the most widely used integrator for MD calculations is the Verlet algorithm [154]. It is based on a Taylor expansion around $\mathbf{r}(t)$ (in the following $\mathbf{a} = \dot{\mathbf{v}} = \ddot{\mathbf{r}}$ and a quantity without index represents a vector containing all $3N$ coordinates of the system):

$$\mathbf{r}(t+\delta t) = \mathbf{r}(t) + \delta t \mathbf{v}(t) + \frac{1}{2}\delta t^2 \mathbf{a}(t) + \cdots \tag{C.3}$$

$$\mathbf{r}(t-\delta t) = \mathbf{r}(t) - \delta t \mathbf{v}(t) + \frac{1}{2}\delta t^2 \mathbf{a}(t) - \cdots \tag{C.4}$$

From these the velocities can be eliminated by subtraction, resulting in

$$\mathbf{r}(t+\delta t) = 2\mathbf{r}(t) - \mathbf{r}(t-\delta t) + \delta t^2 \mathbf{a}(t). \tag{C.5}$$

This simple equation is enough to calculate the trajectories, but does not contain the velocities, which are needed to calculate the kinetic energy (and therefore the total energy) of the system. So instead of calculating $\mathbf{v}(t)$ from the finite differences of $\mathbf{r}(t)$, one can use the so called "velocity Verlet" algorithm, which uses the velocities as well:

$$\begin{aligned} \mathbf{r}(t+\delta t) &= \mathbf{r}(t) + \delta t \mathbf{v}(t) + \frac{1}{2}\delta t^2 \mathbf{a}(t) \\ \mathbf{v}(t+\delta t) &= \mathbf{v}(t) + \frac{1}{2}\delta t \left[\mathbf{a}(t) + \mathbf{a}(t+\delta t)\right]. \end{aligned} \tag{C.6}$$

These simple methods are in principle able to model even large systems of complex molecules, as long as accurate pair potentials for all possible inter- and intra–molecular interactions exist. Some of the problems which can develop with more complex or larger systems are given in the next section.

C.3 Complex MD Systems

Periodic Boundaries and Cutoffs

The total number of particles, which are theoretically manageable with the numerical MD methods, is limited by the amount of computer memory available. The velocity Verlet method (Eq. (C.7)) needs to store a total amount of $9N$ words (one word equals 8 bytes, needed to store on double precision number) for the positions, velocities and accelerations of the particles. So even with memory in the order of TBytes ($\approx 10^{12}$) the amount of storeable particles is on the order of 10^{10} – still far away from a macroscopic system. While this is enough for the simulation of clusters, the modeling of a bulk liquid with this small amount of particles would be dominated by surface effects. To avoid this, one usually introduces periodic boundary conditions. This means, that a cubic simulation volume is replicated as an infinite lattice through space. As a particles leaves the box through one side, the periodicity requires the image particle from a neighboring box to enter from the opposite side. This removes the surface effects, and now models the macroscopic system as a small, infinitely periodic system. If the result will reflect this depends on the range of the inter-molecular potentials and on the actual size of the simulation box. For a pure LJ potential a cubic box with edges of length $L \approx 6\sigma$ (σ from Eq. (C.1)) is large enough, so that a particle will not interact significantly with any of its images in neighboring boxes. This allows the use of the minimum image convention: A particle will only interact with partners located within a box with edges of length L, centered on its position. This ensures, that it does not influence its own image. This sort of system therefore do not notice the symmetry of the periodic lattice, as long as the forces neglected are small enough. Problems will arise for smaller simulation boxes and for molecules with longer range potentials, e.g. polar molecules (dipole-dipole interaction $u(r) \sim r^{-3}$ or charged particles with Coulomb interaction ($u(r) \sim r^{-1}$). Despite these limitations, the

equilibrium properties of liquids away from phase transitions are not disturbed by including periodic boundary conditions.

If the size of the simulation box is increased, to lessen the effect of the periodicity, one runs into other problems. The number of evaluations of the potential function for N atoms scales with N^2 (each of the N particles interacts with the $N-1$ others). As the time needed for these evaluations normally gets too large long before the memory requirements limit the number of possible particles, an additional approximation is introduced. The main contribution to the forces on an atom normally comes from neighboring short range interactions. Knowing this, one can define a cutoff radius r_C, outside of which again the potential is small enough to be neglected. To ensure consistency with the minimum image convention, this radius must be smaller than $\frac{1}{2}L$. With this inclusion the effort of evaluating the potential scales with $N \cdot N_{r_C}$ for N_{r_C} particles within the cutoff radius. This again does not work well for systems containing long range (e.g. Coulomb type) interactions, and will change the thermodynamic properties of the fluid. Methods to apply long range corrections for these problems to a system are discussed in detail in the literature [155, 156].

Constrained Molecular Systems

The largest time step possible in a MD simulation is limited by the oscillation frequency within the system, to ensure the energy conservation of the integrator. If a molecule in the simulation contains a high frequency mode, e.g. a **C–H** or **O–H** bond, the time resolution necessary will slow down the calculation significantly. There are two possible solutions to this problem, if one wants to increase the time step. The first is to define a so called "united atom", treating the system hydrogen–heavy atom as a single particle with specially tailored LJ–potential parameters. This is useful to simulate regions of molecules of lesser interest, e.g. parts of the scaffold far away from reaction centers. The other method, which allows to model single hydrogens (e.g. the proton involved in a hydrogen bridge), is to introduce constraints to the system. So the bond, which normally would display a high frequency oscillation, will be kept fixed at a set bond length $r_{ij}(t) = d_{ij}$. This condition is introduced into the equations of motion via the Lagrange formalism of constraint forces. Two methods to couple this constraint formalism with the standard integrators of MD are given by the SHAKE algorithm (for the Verlet method), and in a slightly changed form by the RATTLE algorithm (for the velocity Verlet method) [157, 158, 159, 160].

Appendix D

Input Files

To allow for easier reproduction of some of the results, the most important input files for the software packages used are given in this appendix.

D.1 Gaussian98

A good introduction to using Gaussian is given in [134]. Following is the input used to generate the second derivative matrix of PMME along the reaction path. It initiates an optimization, but stops it before starting the second cycle, to get at the force constants, which are really calculated (on Hartree–Fock level) and not only guessed, as would be usual for the first step of an optimization. for MP2 calculations, the keyword `B3LYP` has to exchanged with `MP2` and `calcFC` with `calcHFFC` (to calculate the force constants only on HF level, not the too expensive MP2 one).

```
%KJOB L103 2
# B3LYP/6-31+G(d,p) IOP(1/33=2) NoSymm
 Opt(cartesian,maxcycle=2,calcFC) Test

PMME MOLECULE at position XX of the OD bond

0 1
   C
   C   1 r2
   C   2 r3    1 a3
   C   3 r4    2 a4    1 d4
   C   4 r5    3 a5    2 d5
   C   1 r6    2 a6    3 d6
   C   2 r7    1 a7    3 d7
   O   7 r8    2 a8    1 d8
```

```
   C   3 r9    2 a9    1 d9
   O   9 r10    3 a10    2 d10
   O   7 r11    2 a11    1 d11
   H   4 r12    3 a12    2 d12
   H   5 r13    4 a13    3 d13
   H   6 r14    1 a14    2 d14
   H   1 r15    2 a15    3 d15
   O   9 r16    3 a16    2 d16
   H   8 r17    7 a17    2 d17
   C  16 r18    9 a18    3 d18
   H  18 r19   16 a19    9 d19
   H  18 r20   16 a20    9 d20
   H  18 r21   16 a21    9 d21
Variables:
 r2=1.40432536
 r3=1.42433835
 a3=117.72576173
 r4=1.40882314
 a4=119.20459665
 d4=-0.19311624
 r5=1.39084931
 a5=121.68389305
 d5=1.48565098
 r6=1.39236233
 a6=122.34440553
 d6=-1.36429951
 r7=1.53398645
 a7=112.37631885
 d7=-177.08912612
 r8=1.32976749
 a8=120.21181313
 d8=-158.032569
 r9=1.49375066
 a9=124.85299577
 d9=178.14653929
 r10=1.22916863
 a10=125.93396873
 d10=-23.80676442
 r11=1.21590307
 a11=119.17312995
```

```
d11=19.6129855
r12=1.08272018
a12=118.60759601
d12=-179.07806953
r13=1.08558048
a13=119.88217767
d13=179.00669755
r14=1.08583072
a14=119.72766142
d14=-178.96491785
r15=1.08344076
a15=117.15311728
d15=177.95400614
r16=1.33824679
a16=113.08247147
d16=158.04824707
r18=1.44560255
a18=116.40245396
d18=177.51150072
r19=1.08886698
a19=105.00529711
d19=179.81784864
r20=1.09189824
a20=110.20148122
d20=-60.58234507
r21=1.09165028
a21=110.1115407
d21=60.34718663
r17=XX                # Here goes the position on the 1d reaction path
a17=112.359
d17=-8.334
```

D.2 MCTDH

For the MCTDH calculation, one needs an operator file and an input file, which are documented in [140]. Given here are the time dependent operator for the 3d system and the input file used for the propagation with a laser field.
The 3d operator:

```
OP_DEFINE-SECTION
title
```

```
1D pmme oszillator + x modes (VBFC, fitted for R/E)
end-title
end-op_define-section

PARAMETER-SECTION
mass_x0 = 1.0, D-mass
feld = 0.0005
pi=3.14159
tau=300, fs
pitau=pi/tau
omega = 2425, cm-1
K23_23 =  0.000020088170
F23 =  0.000007713732
K1_1 =  0.000000086875
F1 =  0.000000881969
K23_23C_0 =  0.000004424463
K23_23C_1 =  0.000023298457
K23_23C_2 = -0.000019441142
K23_23C_3 =  0.000012112697
K23_1C_0 =  0.000000188816
K23_1C_1 = -0.000000491782
K23_1C_2 =  0.000000458777
K23_1C_3 = -0.000000158073
K1_1C_0 =  0.000000290335
K1_1C_1 = -0.000000400008
K1_1C_2 =  0.000000213266
K1_1C_3 = -0.000000018034
VC_1 = -0.001465471345
VC_2 =  0.253868818283
VC_3 = -0.384436279535
VC_4 =  0.244873702526
VC_5 = -0.068412922323
VC_6 =  0.007109705824
F23C_0 =  0.000249872537
F23C_1 =  0.000051380171
F23C_2 =  0.001188250375
F23C_3 = -0.001492481446
F1C_1 =  0.000014676155
F1C_2 =  0.000040114362
F1C_3 = -0.000087538589
```

```
F1C_4 =  0.000037307072
DC_0 = 7.217092
DC_1 = 1.576173
DC_2 = 1.418831
DC_3 = 1.484585
DC_4 = -1.676802
DC_5 = 0.3768853
p1 = 3.2
p2 = -2.0
end-parameter-section

HAMILTONIAN-SECTION
  modes  |  x0  |  x1  |  x2  | Time
1.0      | KE
VC_1    | xq1
VC_2    | xq2
VC_3    | xq3
VC_4    | xq4
VC_5    | xq5
VC_6    | xq6

1.0      |2 KE
0.5*K23_23C_0    |2 xq2
0.5*K23_23C_1    | ex1   |2 xq2
0.5*K23_23C_2    | ex2   |2 xq2
0.5*K23_23C_3    | ex3   |2 xq2
K23_1C_0         |2 xq1          |3 xq1
K23_1C_1         | ex1   |2 xq1          |3 xq1
K23_1C_2         | ex2   |2 xq1          |3 xq1
K23_1C_3         | ex3   |2 xq1          |3 xq1

-1.*F23C_0       |2 xq1
-1.*F23C_1       | ex1   |2 xq1
-1.*F23C_2       | ex2   |2 xq1
-1.*F23C_3       | ex3   |2 xq1
-0.5             | gg    |2 xq1

1.0      |3 KE
0.5*K1_1C_0      |3 xq2
0.5*K1_1C_1      | pex1          |3 xq2
```

```
0.5*K1_1C_2      | pex2          |3 xq2
0.5*K1_1C_3      | pex3          |3 xq2

-1.*F1C_1        | xq1   |3 xq1
-1.*F1C_2        | xq2   |3 xq1
-1.*F1C_3        | xq3   |3 xq1
-1.*F1C_4        | xq4   |3 xq1

-1.*DC_0    | 1     |4 efeld*sn2*cs
-1.*DC_1    | xq1   |4 efeld*sn2*cs
-1.*DC_2    | xq2   |4 efeld*sn2*cs
-1.*DC_3    | xq3   |4 efeld*sn2*cs
-1.*DC_4    | xq4   |4 efeld*sn2*cs
-1.*DC_5    | xq5   |4 efeld*sn2*cs
end-hamiltonian-section

HAMILTONIAN-SECTION_basepot
  modes  |  x0 |  x1  |  x2
1.0      | KE
VC_1    | xq1
VC_2    | xq2
VC_3    | xq3
VC_4    | xq4
VC_5    | xq5
VC_6    | xq6
end-hamiltonian-section

HAMILTONIAN-SECTION_dipole
  modes  |  x0 | x1 | x2
DC_0    | 1
DC_1    | xq1
DC_2    | xq2
DC_3    | xq3
DC_4    | xq4
DC_5    | xq5
end-hamiltonian-section

LABELS-SECTION
sn2 = sin[pitau]^2
cs  = cos[omega]
```

```
xq1 = q
xq2 = q^2
xq3 = q^3
xq4 = q^4
xq5 = q^5
xq6 = q^6
ex1 = exp[-1]
ex2 = exp[-2]
ex3 = exp[-3]
pex1 = exp[1]
pex2 = exp[2]
pex3 = exp[3]
gg = gauss[p1,p2]
end-labels-section

end-operator
```

And the input file for the dynamics:

```
################################
#               PMME-D         #
#          LASER DRIVING       #
################################

RUN-SECTION
name = time3
gendvr genoper geninwf
propagation
tfinal= 300.0
tout=   1.0
overwrite
output psi gridpop
end-run-section

OPERATOR-SECTION
oppath = /user/naundorf/work/MCTDH/VBFC/REfit_ops
opname = pmmeD3dTime
alter-parameters
efeld = 0.0005
omega = 2445.0, cm-1
end-alter-parameters
end-operator-section
```

```
SBASIS-SECTION
    x0           =   6
    x1           =   6
    x2           =   5
end-sbasis-section

PBASIS-SECTION
   x0   ho  40  xi-xf   -0.8  1.1
   x1   ho  40  xi-xf   -100.  100.
   x2   ho  40  xi-xf   -250.  250.
end-pbasis-section

INTEGRATOR-SECTION
 VMF
 ABM = 6 , 1.0d-7 , 0.01
end-integrator-section

INIT_WF-SECTION
file =/public/bee/scratch/naundorf/MCTDH/new3D/relax3
end-init_wf-section

end-input
```

D.3 Gromacs

The Gromacs input is documented in [143]. Given here is the topology file for PMME in a solvent of CCl_4 molecules. It uses the simplified PMME geometry described in Section 3.3.5, with partial charges obtained from GAUSSIAN CHELP/CHELPG [150, 161] calculations.

```
; This is your topology file
; PMME in CCl4

; Include forcefield parameters
#include "ffG43a1.itp"

[ moleculetype ]
; Name            nrexcl
PMME             6
```

```
[ atoms ]
; nr   type resnr residue  atom   cgnr   charge    mass
   1    CR1   1   PMME     C1     1      0.041     13.019 ; qtot 0.041
   2    CR1   1   PMME     C2     1      0.048     13.019 ; qtot 0.089
   3      C   1   PMME     C3     1     -0.079     12.011 ; qtot 0.01
   4      C   1   PMME     C4     1     -0.162     12.011 ; qtot -0.152
   5    CR1   1   PMME     C5     1      0.025     13.019 ; qtot -0.127
   6    CR1   1   PMME     C6     1      0.038     13.019 ; qtot -0.089
   7      C   1   PMME     C7     1      0.804     12.011 ; qtot 0.715
   8      O   1   PMME     O1     1     -0.601    15.9994 ; qtot 0.114
   9      C   1   PMME     C8     1      0.849     12.011 ; qtot 0.963
  10     OM   1   PMME     O2     1     -0.443    15.9994 ; qtot 0.52
  11    CH3   1   PMME     CH     1      0.296     15.035 ; qtot 0.816
  12     OA   1   PMME     O3     1     -0.605    15.9994 ; qtot 0.211
  13      O   1   PMME     O4     1     -0.598    15.9994 ; qtot -0.387
  14      H   1   PMME     HO     1      0.387       1.008 ; qtot -2.98e-08

[ bonds ]
;  ai    aj funct       c0       c1      c2       c3
    1     2     2   0.1392  0.0
    1     6     2   0.13544  0.0
    2     3     2   0.1404  0.0
    3     4     2   0.1424  0.0
    3     7     2   0.1534  0.0
    4     5     2   0.1410  0.0
    4     9     2   0.1494  0.0
    5     6     2   0.1391  0.0
    7     8     2   0.1216  0.0
    7    12     2   0.1330  0.0
    9    10     2   0.1338  0.0
    9    13     2   0.1229  0.0
   10    11     2   0.1446  0.0
   12    14     2   0.0992  0.0

;[ dihedrals ]
;  ai    aj    ak    al funct       c0       c1       c2       c3
;    1     2     3     4     2    gi_1
;    2     1     6     5     2    gi_1
;    2     3     4     5     2    gi_1
;    3     4     5     6     2    gi_1
```

```
;    4     5     6     1     2    gi_1
;    5     6     1     2     2    gi_1
;    6     1     2     3     2    gi_1

; Include CCl4 topology
#include "ccl4.itp"

[ system ]
; Name
PMMEinCCl4

[ molecules ]
; Compound        #mols
PMME              1
CCl4            724
```

The first included file `ffG43a1.itp` is supplied by Gromacs and contains the force field parameters, the included file `ccl4.itp` supplies the solvent geometry:

```
[ moleculetype ]
; Name            nrexcl
CCl4              3

[ atoms ]
;  nr   type   resnr residue  atom   cgnr     charge  mass
    1   CCL4     1   CCL4     CCL4    1       0       12.011   ; qtot 0
    2   CLCL     1   CCL4     CL1     1       0       35.453   ; qtot 0
    3   CLCL     1   CCL4     CL2     1       0       35.453   ; qtot 0
    4   CLCL     1   CCL4     CL3     1       0       35.453   ; qtot 0
    5   CLCL     1   CCL4     CL4     1       0       35.453   ; qtot 0

[ bonds ]
;  ai    aj funct            c0            c1            c2            c3
    1     2     2    gb_40
    1     3     2    gb_40
    1     4     2    gb_40
    1     5     2    gb_40
    2     3     2    gb_47
    2     4     2    gb_47
    2     5     2    gb_47
    3     4     2    gb_47
    3     5     2    gb_47
```

The run parameters are then supplied by the `run.mdp` file:

```
title               =  run md
warnings            =  10
cpp                 =  /lib/cpp
constraints         =  none
freezegrps          =  PMME
freezedim           =  Y Y Y
morse               =  no
integrator          =  md
tinit               =  0.0
dt                  =  0.001    ; ps !
nsteps              =  30000     ; total 30.0 ps.
nstcomm             =  1
nstxout             =  0
nstvout             =  0
nstfout             =  1
nstlog              =  10
nstenergy           =  10
nstlist             =  10
ns_type             =  simple
deltagrid           =  2
rlist               =  1.3
coulombtype         =  shift
rcoulomb            =  1.0
epsilon_r           =  2.238
vdwtype             =  shift
rvdw                =  1.0
box                 =  rectangular
tcoupl              =  no
ref_t               =  300.0
tau_t               =  0.1
tc-grps             =  System
pcoupl              =  no
annealing           =  no
gen_vel             =  no
```