New path integral simulation algorithms and their application to creep in the quantum sine-Gordon chain

Dissertation

zur Erlangung des Grades
“Doktor der Naturwissenschaften”
am Fachbereich Physik
der Johannes Gutenberg-Universität
in Mainz

vorgelegt von

Florian Krajewski
geboren in Köln

4. August 2003

Datum der mündlichen Prüfung: 24. Oktober 2003

Abstract

New path integral simulation algorithms and their application to creep in the quantum sine-Gordon chain

A path integral simulation algorithm which includes a higher-order Trotter approximation (HOA) is analyzed and compared to an approach which includes the correct quantum mechanical pair interaction (effective Propagator (EPr)). It is found that the HOA algorithm converges to the quantum limit with increasing Trotter number P as P^{-4}, while the EPr algorithm converges as P^{-2}. The convergence rate of the HOA algorithm is analyzed for various physical systems such as a harmonic chain, a particle in a double-well potential, gaseous argon, gaseous helium and crystalline argon. New simulation techniques for the HOA algorithm are developed: The correct estimator for the pair correlation function is presented and it is demonstrated that the estimators for the simulation box volume or the lattice constant do not receive higher-order corrections in the HOA algorithm.

A new path integral algorithm, the hybrid algorithm, is developed. It combines an exact treatment of the quadratic part of the Hamiltonian and the higher-order Trotter expansion techniques. For the discrete quantum sine-Gordon chain (DQSGC), it is shown that this algorithm works more efficiently than all other improved path integral algorithms discussed in this work.

The new simulation techniques developed in this work allow the analysis of the DQSGC and disordered model systems in the highly quantum mechanical regime using path integral molecular dynamics (PIMD) and adiabatic centroid path integral molecular dynamics (ACPIMD). The ground state phonon dispersion relation is calculated for the DQSGC by the ACPIMD method. It is found that the excitation gap at zero wave vector is reduced by quantum fluctuations. Two different phases exist: One phase with a finite excitation gap at zero wave vector, and a gapless phase where the excitation gap vanishes. The mean square displacement of the center of mass mode and the diffusion constant are calculated for both phases. In the gapless phase, the diffusion constant shows an Einstein diffusion like behavior, while it vanishes exponentially with decreasing temperature in the phase with a gap. The reaction of the DQSGC to an external driving force is analyzed at $T = 0$. In the gapless phase the system creeps if a small force is applied, and in the phase with a gap the system is pinned. At a critical force, the systems undergo a depinning transition in both phases and flow is induced.

The analysis of the DQSGC is extended to models with disordered substrate potentials. Three different cases are analyzed: Disordered substrate potentials with roughness exponent $H = 0$, $H = 1/2$, and a model with disordered bond length. For all models, the ground state phonon dispersion relation is calculated. The system with $H = 0$ and the system with disordered bond length behave qualitatively similar to the DQSGC: A phase with a gap and a gapless phase are found. In the phase with a gap, the phonon dispersion relation has one broad branch, while in the gapless phase the dispersion relation is similar to that of the DQSGC. Disordered systems with roughness $H = 1/2$ show a more complicated behavior. For these systems, the chain is always pinned and no mobility due to quantum fluctuations can be observed. Here, the effect of quantum and classical fluctuations on the mobility is different. Classical systems at finite temperature show finite mobility, while the quantum system at $T = 0$ is always pinned. For finite wave vectors, two different phonon branches are found for the quantum system.

Contents

Introduction

The numerical treatment of many particle quantum systems is an important issue of theoretical condensed matter physics. Quantum effects become important when the typical thermal energy of the physical system $k_{\mathrm{B}}T$ is smaller or of the same order as the energy quanta of the microscopic system or when exchange processes due to the indistinguishability become important. For atomistic simulations, this is the case in low temperature physics, e.g. liquid helium or crystalline argon, but also for simulations in organic chemistry where hydrogen atoms have to be treated by quantum mechanics. Recently it was found that the quantum nature of heavy molecular frame atoms substantially enhances proton tunneling and can not be neglected even at room temperature [TM01]. This has far-reaching implications for common modeling strategies of proton transfers in complex systems such as biomolecules and shows that the correct quantum description of atoms is an necessary element of computer aided molecular design and materials modeling. For this reason it is very desirable to have efficient algorithms which allow the numerical treatment of these systems.

Here Feynman's formulation of statistical mechanics in terms of path integrals has turned out to be a very powerful tool for the development of numerical methods treating quantum mechanical many-body problems at finite temperature. Not only do path integrals possess mathematical elegance, but they can be rendered into a computationally traceable form with an inherent structure perfectly suited for implementation on modern day parallel computing architectures. While in the calculation of the canonical partition function a classical particle appears as the integral over one coordinate, for a quantum particle a functional integral over closed paths in imaginary-time has to be performed. In a picture with discretized imaginary-time, one can treat a quantum system in analogy to a classical system: Each particle has to be replaced by a ring polymer representing an imaginary-time path (see Fig. 1). In this sense the path integral expression of the partition function for an N particle system can be mapped to a classical system of N ring polymers. This mapping is called quantum classical isomorphism and allows to perform the integration of the discretized path integrals with path integral Monte Carlo (PIMC) methods and with path integral molecular dynamics (PIMD) methods. Thus, it is possible to calculate numerous properties of quantum systems including thermodynamic quantities, structural properties, and quantum effective potentials with numerical methods similar to those developed for classical statistical mechanics.

With standard path integral simulations, it is not possible to calculate real-time dynamics on long timescales because the inverse Laplace transformation which maps imaginary-time to real-time dynamics is numerically unstable. Recently it was found that the center of mass dynamics

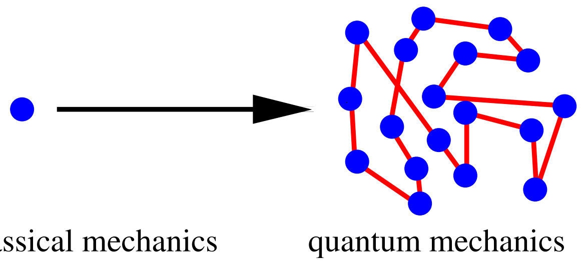

Figure 1: Illustration of the transition from a classical to a quantum mechanical description of a particle in statistical mechanics.

of the closed imaginary-time paths moving on the potential surface given by the average over all closed paths with fixed center of mass is related to Kubo transformed time correlation functions [CV93, JV99]. This leads to the centroid molecular dynamics method which is explained in detail in chapter 4. The described development leads to PIMD and PIMC algorithms which scale algebraically with the system size. The systematic error of the results is controlled by a small parameter, the imaginary-time step, which discretize the imaginary-time path. Statistical errors can be estimated from the standard derivation of the averages and can be reduced systematically with longer simulation runs. In this sense, path integral algorithms allow the efficient treatment of quantum many body problems at finite temperature, what means that if the error has to be reduced by a factor f, the necessary computation time scales algebraically with f.

In spite of the power of path integrals as computational approach to quantum statistical mechanics, several important problems remain unsolved. The required antisymmetry of the wave function describing fermions leads to the so called “Fermi sign problem”. A further disadvantage of PIMC and PIMD is the scaling of the required computing time t_{CPU} with decreasing temperature T at a given required accuracy: Using the so-called primitive approximation for the discretization of imaginary-time together with the most efficient sampling algorithms that completely eliminate critical slowing down (see Ref. [TBMK93] and Ref. [CGC98] for a thorough discussion), it is not possible to overcome $t_{\text{CPU}} \propto 1/T$. The reason for that is that the length of the imaginary-time on which a closed path can evolve is proportional to the inverse temperature and if one wants to keep the discretization step of the imaginary-time constant, the number of time slices which is given by the so called Trotter number P has to be increased as $P \propto 1/T$.

Because of the wide range of possible applications for path integral simulations, it is desirable to have efficient path integral algorithms. These algorithms should allow the treatment of physical systems at low temperatures in the regime where quantum effects are very strong at a reasonable computational cost. Different improvements of the primitive algorithm have been suggested to render path-integral simulations more efficient. The bulk of such attempts may be subdivided into three categories:

(i) Methods that are based on higher-order approximants (HOA) of the high-temperature density matrix [TI84, LB87, KM02].

(ii) Methods that use effective propagators (EPr) that automatically yield the proper two-particle behavior [Cep95, Pol88].

(iii) Methods that decompose the Hamiltonian into a quadratic part and a rest (QR) contribution before applying the Trotter formula [GT86, CMTV93].

One intention of this work is the further development of more efficient path integral algorithms. The three different approaches (i, ii, iii) are analyzed [KM02] (chapter 1, 2) with various test models, e.g. the HOA (i) and EPr (ii) methods are applied to a harmonic chain, because the analysis can be done analytically in this case. The convergence rate of the different algorithms towards the quantum limit with increasing Trotter number is analyzed. New results on the efficient calculation of the pair correlation function or the particle probability distribution within the HOA (i) approach are found. Simulation techniques for the calculation of the volume respectively lattice constant in the constant pressure ensemble (NPT) are developed for the HOA (i) algorithm. The new techniques are applied to test models, e.g. a particle in a double well potential and crystalline Argon.

In chapter 3, a new path integral algorithm, the so called hybrid algorithm, is developed. The hybrid algorithm combines higher order approximants (i) and an exact treatment of the harmonic part of the Hamiltonian (iii). It is applied to the discrete sine-Gordon chain for which it is shown that it converges as $1/P^4$ and furthermore shows a much smaller systematic error than the PA, HOA (i), EPr (ii) or QR (iii) algorithms.

With the new developed simulation techniques, the discrete quantum sine-Gordon chain (DQSGC), also known as Frenkel-Kontorova (FK) model [FK38, BK98] is analyzed in the highly quantum mechanical regime. The classical version of the FK model was originally suggested by Frenkel and Kontorova in 1938 for the description of dislocations in metals. Since then it has been successfully applied to various different physical problems. The discrete sine-Gordon (SG) chain or Frenkel-Kontorova model can be considered as a generic model for the motion of an elastic object composed of discrete degrees of freedom through an external potential (see Fig. 2). To

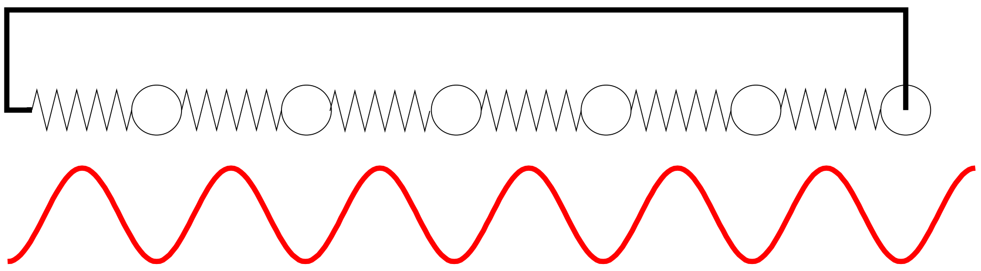

Figure 2: Illustration of the discrete sine-Gordon chain with periodic boundary conditions.

name a few of its applications, the SG model is used for the description of driven charge-density waves in solids, coupled Josephson junctions [FM96], the sliding motion of an adsorbed layer of atoms over a substrate [MUR03], and most recently electronic conductance in nanotubes [LT03], see also the reviews [AA98, Zs99] on electronic transport in one-dimensional structures. A lot of attention has been devoted to the (discrete) classical FK model both at zero and finite temperatures [BK98, FM96] and the (continuum) quantum-mechanical SG model [Zs99, STF79, STF].

However, less is known about the quantum-mechanical properties of the discrete quantum FK (QFK) model, in particular about its dynamical properties.

In the analysis of the quantum SG model or models with disordered substrate potentials, it is necessary to go beyond standard renormalization group (RG) theories [CGD98, GFB02, Keh99] because otherwise the effect of creep or, alternatively, the effect of a vanishing excitation gap at zero wave vectors might be artificially removed. It is thus desirable to have numerical techniques which allow to (a) verify the results of RG studies, (b) obtain results beyond continuum approximations allowing an accurate determination of critical values, and (c) obtain dynamical responses of the system for arbitrary external forces. The author intends to demonstrate that adiabatic centroid path integral molecular dynamics (ACPIMD) [TBMK93, CM96] is a well-suited technique to tackle the many-body dynamics of elastic manifolds moving through embedding systems. A path integral analysis of the commensurate discrete quantum sine-Gordon chain is given in chapter 5. Special emphasis is given to the ground state structure of the DQSGM. The phonon dispersion relation and various correlation functions are calculated. The study is extended to the DQSGM in the presence of an external driving force. The dynamics in the presence of an external driving force is analyzed in both, the linear response region and the region far from equilibrium. At a critical value for the external force, a pinning-depinning transition is found.

An advantage of the numerical techniques developed in this work is that the effort for the calculation does not increase if the underlying potential becomes more complex. This allows to extend the analysis of the quantum sine-Gordon chain to related disordered models, where the sinusoidal substrate potential is replaced by a disordered potential. Different models of this type are analyzed in chapter 6 and the dispersion relations are calculated.

Chapter 1

Discretized path integrals: The so called primitive and the higher-order approximation

In the first part of this chapter a short introduction of the primitive algorithm is given, starting with the path integral expression for the partition function, and it is explained that observables are calculated as averages of so called estimators within this framework

Because of the increasing CPU time needed as the temperature is decreased ($t_{\mathrm{CPU}} \propto 1/T$), it is desirable to use more efficient algorithms. The so called primitive algorithm converges to the quantum limit as $1/P^2$ if the potential is sufficiently well behaved. One very effective method to save CPU time and improve the convergence rate to the quantum limit is the so called higher-order approximation (HOA). This method uses a higher-order Trotter decomposition instead of the leading order expression used in the primitive approximation.

The HOA algorithm is derived and compared to the primitive algorithm in the second part of this chapter. In a HOA path integral simulation, generalized estimators have to be defined even for those observables that are diagonal in real space. In particular, an expression for the HOA estimator of the radial distribution function is derived. This has not been given hitherto which might explain the sparse use of the method in the literature. The new estimator is used to calculate the pair correlation function of an argon crystal and the probability distribution function of a particle in a double-well potential. Another crucial point which has not yet been discussed in the literature concerns simulations in the constant pressure (NPT) ensemble. It is important to note that here the volume is a classical observable and that the estimator for the volume has no corrections in the HOA algorithm. Besides the fact that the correct higher-order corrections for the estimators have to be used, another reason why the HOA algorithm is not as established as the standard technique for PIMD and PIMC simulations is that the HOA approach may become problematic if the interaction potentials are not well-behaved. It has been argued that the HOA algorithm has problems if the interaction diverges ([Cep95] page 315) as, for example, the Lennard Jones potential which has a $1/r^{12}$ singularity at small interatomic distances r. In such a situation, the prefactor of the $1/P^4$ correction term may be ill-defined, resulting in a convergence that is less favorable than that of well-behaved potentials. Therefore the study is extended to the

HOA convergence of expectation values for observables in realistic systems. It is shown numerically for two examples, gaseous helium and crystalline argon, that the convergence rate is $1/P^4$, even though the Lennard Jones potential diverges at small distances. It is known that for other divergent interactions as the attractive Coulomb potential the PA and HOA the convergence rate is different from the standard behavior because the wave function does not vanish exponentially at the divergence. It was also argued that the commutators appearing in the correction terms can be slow to evaluate ([Cep95] page 315). It is shown in paragraph 1.5.2 that the scaling behavior of the HOA algorithm is the same as for the primitive algorithm in the case of pair potentials.

1.1 The primitive algorithm (PA)

Path integral simulations are based on an expression for the partition function for the quantum system which has the form similar to that of a classical system. This reduces the problem of calculating statistic properties of the system to the calculation of a high-dimensional integral. In the case of Boltzmann or Bose statistics this integral can be calculated with Monte Carlo or molecular dynamics methods. For systems including fermions the situation is more difficult because of the alternating sign stemming from the anti-symmetrization of the wave function. In this work the expression for the partition function is integrated with molecular dynamics techniques. The classical equations of motion for the ring polymers representing quantum particles are solved numerical with a 5^{th} order Gear predictor-corrector algorithm or a 2^{nd} order velocity Verlet algorithm. To fix the kinetic temperature a heat bath is coupled to the system which is modeled with a Langevin thermostat. In the following, the path integral expression for the partition function (Eq. 1.4) and for the potential used in the molecular dynamics simulation (Eq. 1.8) will be derived. For an N particle system with the Hamilton operator

$$\hat{H} = \sum_{j=0}^{N-1} \frac{\hat{p}_j^2}{2m} + V\left(\{\hat{\mathbf{r}}_j\}\right) \ , \tag{1.1}$$

the partition function is given by

$$Z\left(\beta\right) = \mathrm{Tr}\ e^{-\beta \hat{H}} \ . \tag{1.2}$$

The expression for the partition function can be rewritten using the Trotter expansion formula

$$\mathrm{tr} \exp\left[-\beta\left(\hat{T}+\hat{V}\right)\right] = \mathrm{tr}\left[\exp\left(-\frac{\beta}{P}\hat{T}\right)\exp\left(-\frac{\beta}{P}\hat{V}\right)\right]^P + \mathcal{O}\left(1/P^2\right) \ . \tag{1.3}$$

The application of the Trotter expansion formula (1.3) to the expression for the partition function (Eq. 1.2) and $(P-1)$ insertions of the unity representation in real space $\mathbb{1} = \int \prod_{j=0}^{N-1} d\mathbf{r}_j^t |\mathbf{r}_j^t\rangle\langle\mathbf{r}_j^t|$ lead to Feynman's path integral expression for the partition function

$$\begin{aligned} Z = \int \prod_{t=0}^{P-1}\prod_{j=0}^{N-1} \left(\sqrt{\frac{m}{2\pi\hbar^2\left(\beta/P\right)}}\right)^d d^d\mathbf{r}_j^t \\ \times \exp\left\{-\frac{\beta}{P}\sum_{t=0}^{P-1}\left[\sum_{j=0}^{N-1}\frac{m}{2\hbar^2(\beta/P)^2}\left(\mathbf{r}_j^{t+1}-\mathbf{r}_j^t\right)^2 + V\left(\{\mathbf{r}_j\}^t\right)\right]\right\} + \mathcal{O}\left(1/P^2\right) \ . \end{aligned} \tag{1.4}$$

The integral measure appearing in Eq. (1.4) can be identified as the discretized version of the functional integration measure over N closed paths. The continuum limit is given by

$$\prod_{t=0}^{P-1}\left(\sqrt{\frac{m}{2\pi\hbar^2\left(\beta/P\right)}}\right)^d d^d\mathbf{r}_j^t \overset{P\to\infty}{\longrightarrow} \mathcal{D}\left[\mathbf{r}_j\left(\tau\right)\right] \ . \tag{1.5}$$

Here $\mathcal{D}[r_j(\tau)]$ stands for the functional integration over all paths $r_j : \tau \longrightarrow \mathbb{R}^d$ with $r_j(0) = r_j(\tau)$. τ is the imaginary-time and the upper index t labels the discretized imaginary-time. They are connected according to

$$\tau = \frac{\hbar\beta}{P}t \ . \tag{1.6}$$

The number of discrete imaginary-time steps P is commonly called the Trotter number, and the index t which labels the imaginary-time is called the Trotter index. In the limit $P \to \infty$, the partition function (Eq. (1.4)) can be written as the functional integral

$$Z = \int\prod_{j=0}^{N-1}\mathcal{D}\left[\mathbf{r}_j\left(\tau\right)\right]\exp\left\{-\frac{1}{\hbar}\int_0^{\beta\hbar}d\tau\left[\frac{1}{2}m\dot{\mathbf{r}}_j^2\left(\tau\right)+V\left(\{\mathbf{r}_j(\tau)\}\right)\right]\right\} \ . \tag{1.7}$$

An alternative interpretation of the discretized path integral expression for the partition function given in Eq. (1.4) is to consider Z to be the partition function of N classical ring polymers where the neighboring monomers interact with a harmonic potential. This allows one to map a quantum-mechanical N particle system onto a classical system of N ring polymers. This mapping becomes exact as the number of monomers P of each ring polymer goes to infinity. The potential $U_{\mathrm{rp}}(\{\mathbf{r}_j^t\})$ that corresponds to the classical picture on N ring polymers is given by

$$U_{\mathrm{rp}}^{\mathrm{PA}}(\{\mathbf{r}_j^t\}) = \sum_{t=0}^{P-1}\sum_{j=0}^{N-1}\left[\frac{1}{2}\frac{m}{\hbar^2(\beta/P)^2}\left(\mathbf{r}_j^t-\mathbf{r}_j^{t+1}\right)^2+V(\{\mathbf{r}_j\}^t)\right] , \tag{1.8}$$

with $\beta = 1/(k_BT)$. $\mathbf{r}_j^t$ represents the position of monomer t in the ring polymer j, and V is the real (physical) potential.

1.1.1 Estimators for the primitive algorithm

In a path integral simulation, many physical quantities can be calculated from the sampled imaginary-time paths $\{\mathbf{r}_j^t\}$ as averages of so called estimators for the corresponding quantity. The statistical average of an observable L is given by

$$\langle L\rangle = -\frac{d}{d\lambda}\ln\left(\ \mathrm{Tr}\ e^{-\{\beta H+\lambda L\}}\right)\Big|_{\lambda=0} \ . \tag{1.9}$$

If the observable L is diagonal in real space $[L = L(\{\hat{\mathbf{r}}_j\})]$, the expectation value of L can be calculated as an average of the estimator for L. The estimator is a function of the coordinates $\{\mathbf{r}_j^t\}$.

Inserting the path integral expression of $\mathrm{Tr}\left(\exp\left[-(\beta\hat{H}+\lambda L)\right]\right)$ in Eq. (1.9) and calculating the derivative leads to the statistical average

$$\begin{aligned}\langle L\rangle &= \int\prod_{t=0}^{P-1}\prod_{j=0}^{N-1}\left(\sqrt{\frac{m}{2\pi\hbar^2\,(\beta/P)}}\right)^d d^d\mathbf{r}_j^t\,\times \\ &\qquad\times L_{\mathrm{est}}^{\mathrm{PA}}\left(\{\mathbf{r}_j^t\}\right)\,\exp\left\{-\frac{\beta}{P}U_{\mathrm{rp}}^{\mathrm{PA}}(\{\mathbf{r}_j^t\})\right\}\Big/Z+\mathcal{O}\left(1/P^2\right) \\ &=: \left\langle L_{\mathrm{est}}^{\mathrm{PA}}\left(\{\mathbf{r}_j^t\}\right)\right\rangle_{U_{\mathrm{rp}}^{PA}}+\mathcal{O}\left(1/P^2\right)\end{aligned} \tag{1.10}$$

with the estimator

$$L_{\mathrm{est}}^{\mathrm{PA}}\left(\{\mathrm{r}_j^t\}\right)=\frac{1}{P}\sum_{t=0}^{P-1}L\left(\{\mathbf{r}_j\}^t\right)\ . \tag{1.11}$$

The PIMD or PIMC program is then assumed to generate distributions in a way that the probability of configuration $\{\mathbf{r}_j^t\}$ to occur is proportional to $\exp[-\,(\beta/P)\,U_{\mathrm{rp}}(\{\mathbf{r}_j^t\})]$. All statistical averages of observables diagonal in real space can be determined directly from the configurations

$$\langle L\rangle=\lim_{P\to\infty}\lim_{M\to\infty}\frac{1}{M}\sum_{i=1}^{M}L_{\mathrm{est}}(\{\mathbf{r}_j^t\}_i), \tag{1.12}$$

where $\mathbf{r}_{j,i}^t$ is the position of the t'th monomer of the polymer representing particle j in the i'th Monte Carlo or molecular dynamics step and M is the number of observations in the Monte Carlo or molecular dynamics simulation [Bar79, TBMK93, Cep95]. For example, the primitive estimator for the potential energy is given by

$$V_{\mathrm{est}}^{\mathrm{PA}}\left(\{\mathbf{r}_j^t\}\right)=\frac{1}{P}\sum_{t=0}^{P-1}V\left(\{\mathbf{r}_j\}^t\right)\ . \tag{1.13}$$

The expectation value of the kinetic energy can be calculated easily as well. The thermodynamic relation

$$\langle T\rangle=\frac{m}{\beta}\frac{d}{dm}\ln\,Z\left(\beta\right) \tag{1.14}$$

leads to the so called primitive estimator for the kinetic energy

$$T_{\mathrm{est}}^{\mathrm{PA}}\left(\{\mathbf{r}_j^t\}\right)=\frac{dNP}{2\beta}-\frac{1}{P}\sum_{t=0}^{P-1}\sum_{j=0}^{N-1}\frac{m}{2\hbar^2\left(\beta/P\right)^2}\left(\mathbf{r}_j^t-\mathbf{r}_j^{t+1}\right)^2\ . \tag{1.15}$$

The estimator given in Eq. (1.15) is problematic if large Trotter numbers P are used because the statistical fluctuations diverge if P is increased, as $\langle\left(T_{\mathrm{est}}^{\mathrm{PA}}\right)^2\rangle-\langle T_{\mathrm{est}}^{\mathrm{PA}}\rangle^2=\mathcal{O}\left(P\right)$ [HBB82]. In order to rectify this, one can derive an equivalent form of the estimator using the virial theorem that involves only the potential and its first derivative. This alternative estimator, known as the virial estimator is given by

$$T_{\text{est}}^{\text{PA(vir)}}\left(\{\mathbf{r}_j^t\}\right) = \frac{1}{2}\frac{dN}{\beta} + \frac{1}{2P}\sum_{t=0}^{P-1}\sum_{j=0}^{N-1}\left(\mathbf{r}_j^t - \mathbf{r}_{\text{cms}\,j}\right)\cdot\nabla_{\mathbf{r}_j^t}V\left(\{\mathbf{r}_j\}^t\right) \ , \tag{1.16}$$

where $\mathbf{r}_{\text{cms}\,j} = 1/P\sum_{s=0}^{P-1}\mathbf{r}_j^s$ is known as the path's centroid and is simply the center of the path. A short and new derivation of the virial estimator which takes advantage of the normal mode representation of the path integral is given in appendix B. Alternatively, Müser showed that if the term $dNP/2\beta$ in Eq. (1.15) is replaced with the actual dynamic kinetic energy of the PIMD simulation, the time dependent variance [1] can be reduced [Müs02].

It is important to note that two arbitrary observables which are a function of the momentum or space operator respectively, cannot be calculated in the same simulation. This is a reflection of the Heisenberg uncertainty relation. For the calculation of the momentum distribution of a particle given by $P(p) = \langle\delta\left(p - \hat{p}\right)\rangle$, for example, open paths have to be sampled. The momentum distribution is then related to the Fourier transformed distribution of the polymer's end point distances [Cep95].

1.2 The higher-order Trotter approximation (HOA)

The HOA method is based on a fourth-order Trotter decomposition of the high-temperature density matrix [RR83]. The decomposition was first applied to continuous degrees of freedom by Takahashi and Imada [TI84] as well as by Li and Broughton [LB87]. The basic idea is to use a higher-order Trotter expansion formula instead of Eq. (1.3). The fourth-order Trotter expansion formula

$$\text{tr}\ \exp\left[-\beta\left(\hat{T}+\hat{V}\right)\right] = \text{tr}\left[\exp\left(-\frac{\beta}{P}\hat{T}\right)\exp\left(-\frac{\beta}{P}\hat{\hat{V}}\right)\right]^P + \mathcal{O}\left(1/P^4\right) \tag{1.19}$$

with

$$\hat{\hat{V}} = \hat{V} + \frac{1}{24}\left(\frac{\beta}{P}\right)^2\left[\hat{V},\left[\hat{T},\hat{V}\right]\right] \tag{1.20}$$

[1] The time dependent variance of the kinetic energy in a MD simulation is given by

$$\left(\Delta T(t_{\text{MD}}^{\max})\right)^2 = \frac{1}{t_{\text{MD}}^{\max}}\int_0^{t_{\text{MD}}^{\max}} dt_{\text{MD}}\left(T_{\text{est}}^{\text{PA}}\left(\mathbf{r}_j^t(t_{\text{MD}})\right) - \langle T\rangle\right)^2 \tag{1.17}$$

with

$$\langle T\rangle = \lim_{t_{\text{MD}}^{\max}\to\infty}\frac{1}{t_{\text{MD}}^{\max}}\int_0^{t_{\text{MD}}^{\max}} dt_{\text{MD}}T_{\text{est}}^{\text{PA}}\left(\mathbf{r}_j^t(t_{\text{MD}})\right) \ . \tag{1.18}$$

turned out to be very useful for numerical simulations. [2] For the physical case where $\hat{T} = \frac{\hat{p}^2}{2m}$ and $\hat{V}$ is only a function of the space operator $\hat{r}$, the double commutator in Eq. (1.20) is diagonal in real space making the effective potential $\tilde{V}$ a function of only the space operator

$$\tilde{V} = V + \frac{1}{24}\frac{\hbar^2}{m}\left(\frac{\beta}{P}\right)^2 \sum_{j=0}^{N-1} \left(\nabla_{\mathbf{r}_j} V\right)^2 =: V + V_c \, . \tag{1.21}$$

The configurations $\{\mathbf{r}_j^t\}$ that are generated in a PIMD or PIMC simulation have to be distributed according to the weight function $\exp[-(\beta/P)U_{\text{rp}}^{\text{HOA}}]$ with the classical ring polymer potential

$$U_{\text{rp}}^{\text{HOA}} = \sum_{t=0}^{P-1}\left[\sum_{j=0}^{N-1} \frac{m}{2\hbar^2(\beta/P)^2}\left(\mathbf{r}_j^{t+1} - \mathbf{r}_j^t\right)^2 + \tilde{V}\left(\{\mathbf{r}_j\}^t\right)\right] \, . \tag{1.22}$$

1.2.1 Estimators for the higher-order approximation algorithm (HOA)

The estimators for the higher-order approximation algorithm can be derived in analogy to the estimators in the primitive algorithm starting from Eq. (1.9). If one uses the higher-order Trotter expansion formula from Eq. (1.19), one finds for the thermodynamic average of an operator $L\left(\{\mathbf{r}_j\}\right)$

$$\begin{aligned}\langle L \rangle = &\int \prod_{t=0}^{P-1}\prod_{j=0}^{N-1}\left(\sqrt{\frac{m}{2\pi\hbar^2\left(\beta/P\right)}}\right)^d d^d\mathbf{r}_j^t \\ &\times L_{\text{est}}^{\text{HOA}}\left(\{\mathbf{r}_j^t\}\right)\,\exp\left\{-\frac{\beta}{P}U_{\text{rp}}^{\text{HOA}}(\{\mathbf{r}_j^t\})\right\}\Big/ Z + \mathcal{O}\left(1/P^4\right) \\ =: &\left\langle L_{\text{est}}^{\text{HOA}}\left(\{\mathbf{r}_j^t\}\right)\right\rangle_{U_{\text{rp}}^{\text{HOA}}} + \mathcal{O}\left(1/P^4\right)\end{aligned} \tag{1.23}$$

with the estimator

$$L_{\text{est}}^{\text{HOA}} = \frac{1}{P}\sum_{j=0}^{P-1}\left(L\left(\{\mathbf{r}_j\}^t\right) + \frac{1}{12}\frac{\hbar^2}{m}\left(\frac{\beta}{P}\right)^2\sum_{j=0}^{N-1}\left[\nabla_{\mathbf{r}_j^t} V\left(\{\mathbf{r}_j\}^t\right)\right]\left[\nabla_{\mathbf{r}_j^t} L\left(\{\mathbf{r}_j\}^t\right)\right]\right) \tag{1.24}$$

and the ring polymer potential for the HOA algorithm (Eq. (1.22)). Eq. (1.24) allows one to find the estimator for the potential energy to be

$$V_{\text{est}}^{\text{HOA}}\left(\{\mathbf{r}_j^t\}\right) = \frac{1}{P}\sum_{t=0}^{P-1}\left[V\left(\{\mathbf{r}_j\}^t\right) + 2V_{\text{c}}\left(\{\mathbf{r}_j\}^t\right)\right] \, . \tag{1.25}$$

According to Eq. (1.15). The HOA estimator for the kinetic energy is given by

$$T_{\text{est}}^{\text{HOA}}\left(\{\mathbf{r}_j^t\}\right) = T_{\text{est}}^{\text{PR}}\left(\{\mathbf{r}_j^t\}\right) + \frac{1}{P}\sum_{t=0}^{P-1} V_{\text{c}}\left(\{\mathbf{r}_j\}^t\right) \tag{1.26}$$

[2] In principal it is possible to formulate Trotter decompositions like Eq. (1.19) which include higher-orders of (β/P) than Eq. (1.19). But in these expressions the operator $\tilde{V}$ is not diagonal in real space. For this reason, expansions applicable for path integral simulations which go further than $1/P^4$ are not known.

and can be rewritten with the virial theorem as

$$T_{\mathrm{est}}^{\mathrm{HOA(vir)}}\left(\{\mathbf{r}_j^t\}\right) = \frac{1}{2}\frac{dN}{\beta} + \frac{1}{2P}\sum_{t=0}^{P-1}\left[\sum_{j=0}^{N-1}\left(\mathbf{r}_j^t - \mathbf{r}_{\mathrm{cms}\,j}\right)\cdot\nabla_{\mathbf{r}_j^t}\tilde{V}\left(\{\mathbf{r}_j\}^t\right) + V_{\mathrm{c}}\left(\{\mathbf{r}_j\}^t\right)\right] \; . \tag{1.27}$$

For further details on the calculation of thermal expectation values see Refs. [TI84, LB87].

The radial distribution function and the distance estimator

In this work and in [KM02], it is discussed for the first time that also the estimator for the radial distribution function $g(r)$ needs to be altered with respect to the primitive approach, for which the estimator can be written as

$$g_{\mathrm{est}}^{\mathrm{PA}}\left(\{r_j^t\}\right)(r) \propto \frac{1}{P}\sum_{t=0}^{P-1}\delta\left(r- \mid \mathbf{r}_j^t - \mathbf{r}_{j'}^t \mid\right)\,/r^2 \, , \tag{1.28}$$

$\mathbf{r}_j^t$ denoting the position of the t's monomer in ring polymer (particle) j. Applying the expression for the estimator in the HOA algorithm (Eq. (1.24)) to $g(r)$ leads to a shift of the estimator in the squared distance between particle j and j'. Simply applying Eq. (1.24) to the operator for the squared distance between particle j and j' yields the following distance estimator $r_{j,j'\,\mathrm{est}}^{\mathrm{HOA}}$:

$$r_{j,j'\,\mathrm{est}}^{\mathrm{HOA}} = \left\{\left(\mathbf{r}_j^t - \mathbf{r}_{j'}^t\right)^2 + \Delta_{j,j'}\right\}^{1/2} \tag{1.29}$$

with

$$\Delta_{j,j'} = \frac{\beta^2\hbar^2}{6P^2}\left(\frac{\nabla_{\mathbf{r}_j^t}V}{m_j} - \frac{\nabla_{\mathbf{r}_{j'}^t}V}{m_{j'}}\right)\cdot\left(\mathbf{r}_j^t - \mathbf{r}_{j'}^t\right) \; . \tag{1.30}$$

Thus $r_{j,j'\mathrm{est}}^{\mathrm{HOA}}$ should replace $\mid \mathbf{r}_j^t - \mathbf{r}_{j'}^t \mid$ in the argument of the δ-function on the right hand side of Eq. (1.28) in order to calculate $g(r)$. Note that the choice of the distance estimator in Eq. (1.29) is by no means unique, because one can add correction terms in higher-orders of $(\beta/P)^2\hbar^2/m$ without affecting the convergence rate for well-behaved potentials. An alternative expression for Eq. (1.29) which is always real is

$$r_{j,j'\,\mathrm{est}}^{\mathrm{HOA}} = \left|\mathbf{r}_j^t - \mathbf{r}_{j'}^t\right| \exp\left(\frac{\Delta_{j,j'}}{2\left(\mathbf{r}_j^t - \mathbf{r}_{j'}^t\right)^2}\right) \; . \tag{1.31}$$

Eq. (1.31) may prove to be important if the potential energy V has strongly repulsive parts at small distances. Eq. (1.29) can then result in imaginary values of $r_{j,j'\mathrm{est}}$ if the distance between two interacting particles is sufficiently small, even though the probability of the configuration to occur is non-zero. This situation was never observed in any of the simulations, because such small separations are extremely unlikely.

1.3 The double-well potential

The HOA method works without difficulty if the potentials are well behaved and the second derivative in Eq. (1.21) is always well defined. To analyze the convergence, one-particle in a double-well potential is considered as a test model. The numerical matrix multiplication (NMM) method is applied to solve this one particle problem.

1.3.1 The numerical matrix multiplication method (NMM)

NMM [Sto68, TBB83, PC84, ML02] exploits the semi-group property of the density matrix. Squaring of the thermal density matrix $\rho(\beta/2)$ at temperature $T = k_\mathrm{B}/(2\beta)$ results in the density matrix $\rho(\beta)$ at $T = k_\mathrm{B}/\beta$.

$$\rho(\beta) = \langle x|e^{-\beta\hat{H}}|x''\rangle = \int dx' \langle x|e^{-\beta\hat{H}/2}|x'\rangle\langle x'|e^{-\beta\hat{H}/2}|x'\rangle \tag{1.32}$$

This procedure is repeated n times so that the low-temperature density matrix $\rho(\beta)$ is caluculated from a high-temperature density matrix $\rho(\beta/P)$ with $P = 2^n$. For the high-temperature density matrix, the action is approximated either by the decomposition underlying the primitive approach or by the highr-order decomposition. The high-temperature approximations of $\rho(\beta/P)$ for the primitive (PA) and higher-order (HOA) algorithm are given by

$$\begin{aligned}
&\mathrm{PA:} && (1.33)\\
&\rho_P^{\mathrm{PA}}(\beta/P) = \left\langle x \middle| \exp\left\{-\left(\frac{\beta}{2P}\right)\hat{V}(x)\right\}\exp\left\{-\left(\frac{\beta}{P}\right)\hat{T}\right\}\exp\left\{-\left(\frac{\beta}{2P}\right)\hat{V}(x)\right\} \middle| x'\right\rangle \\
&\mathrm{HOA:} && (1.34)\\
&\rho_P^{\mathrm{HOA}}(\beta/P) = \left\langle x \middle| \exp\left\{-\left(\frac{\beta}{2P}\right)\hat{\tilde{V}}(x)\right\}\exp\left\{-\left(\frac{\beta}{P}\right)\hat{T}\right\}\exp\left\{-\left(\frac{\beta}{2P}\right)\hat{\tilde{V}}(x)\right\} \middle| x'\right\rangle \\
&\text{with}\quad \tilde{V}(x) = V(x) + \frac{1}{24}\frac{\hbar^2}{m}\left(\frac{\beta}{P}\right)^2\left(\frac{dV(x)}{dx}\right)^2
\end{aligned}$$

Iterative squaring of the high-temperature approximation for the density matrix results in expressions for the low temperature matrix

$$\begin{aligned}
&\mathrm{PA:} && (1.35)\\
&\rho_\mathrm{P}^{\mathrm{PA}}(\beta) = \left\langle x \middle| \left(\exp\left\{-\left(\frac{\beta}{2P}\right)\hat{V}(x)\right\}\exp\left\{-\left(\frac{\beta}{P}\right)\hat{T}\right\}\exp\left\{-\left(\frac{\beta}{2P}\right)\hat{V}(x)\right\}\right)^P \middle| x'\right\rangle \\
&\mathrm{HOA:} && (1.36)\\
&\rho_\mathrm{P}^{\mathrm{HOA}}(\beta) = \left\langle x \middle| \left(\exp\left\{-\left(\frac{\beta}{2P}\right)\hat{\tilde{V}}(x)\right\}\exp\left\{-\left(\frac{\beta}{P}\right)\hat{T}\right\}\exp\left\{-\left(\frac{\beta}{2P}\right)\hat{\tilde{V}}(x)\right\}\right)^P \middle| x'\right\rangle .
\end{aligned}$$

For the primitive algorithm, the radial distribution function and the free and potential energy are simply given by

$$\begin{aligned} g_P(r) &= \frac{\int dx\, \rho_{\mathrm{P}}^{\mathrm{PA}}(\beta,x,x)\,\delta(r-x)}{\int dx\, \rho_{\mathrm{P}}^{\mathrm{PA}}(\beta,x,x)} && (1.37)\\ \langle F_P\rangle &= -\frac{1}{\beta}\ln\int dx\,\rho_{\mathrm{P}}^{\mathrm{PA}}(\beta,x,x) && (1.38)\\ \langle V_P\rangle &= \frac{\int dx\, \rho_{\mathrm{P}}^{\mathrm{PA}}(\beta,x,x)\,V(x)}{\int dx\, \rho_{\mathrm{P}}^{\mathrm{PA}}(\beta,x,x)}\,. && (1.39)\end{aligned}$$

If higher-order corrections are used to approximate the high-temperature matrix, higher-order corrections have to be considered in the expressions for the estimators as well. This results in

$$\begin{aligned} g_P(r) &= \frac{\int dx\, \rho_{\mathrm{P}}^{\mathrm{HOA}}(\beta,x,x)\left(\delta(r-x)+\frac{1}{12}\frac{\hbar^2}{m}\left(\frac{\beta}{P}\right)^2\delta'(r-x)\frac{dV(x)}{dx}\right)}{\int dx\, \rho_{\mathrm{P}}^{\mathrm{HOA}}(\beta,x,x)}\,, && (1.40)\\ \langle F_P\rangle &= -\frac{1}{\beta}\ln\int dx\,\rho_{\mathrm{P}}^{\mathrm{HOA}}(\beta,x,x)\ , && (1.41)\\ \langle V_P\rangle &= \frac{\int dx\, \rho_{\mathrm{P}}^{\mathrm{HOA}}(\beta,x,x)\left(V(x)+\frac{1}{12}\frac{\hbar^2}{m}\left(\frac{\beta}{P}\right)^2\left(\frac{dV(x)}{dx}\right)^2\right)}{\int dx\, \rho_{\mathrm{P}}^{\mathrm{HOA}}(\beta,x,x)}\,. && (1.42)\end{aligned}$$

Here $\delta'(r-x)$ stands for the derivative of the delta distribution. For the NMM method, it turned out to be practical not to use the expression for the radial distribution function given in Eq. (1.29) directly, but to expand Eq. (1.29) in $(\beta/P)^2\,\hbar^2/m$. This leads to the estimator for the radial distribution function used in Eq. (1.40).

NMM becomes increasingly more complex with increasing dimensionality of the Hilbert space, i.e., exponentially more complex with the number of particles. As described previously in Refs. [Sto68, TBB83, PC84], the calculations can be simplified for systems whose description can be reduced to the case of one particle moving in a central potential. To analyze the convergence with P in dimensions larger than one, it is necessary to expand the density matrix in partial waves and use the proper modified Bessel functions for the free-particle kernel at given angular momentum l, see Eq. (4.43) in Ref. [Cep95]. Otherwise the leading corrections cannot vanish faster than $1/P^2$. Moreover, it is necessary to choose the discretization of the real space sufficiently small as one uses larger Trotter numbers.

1.3.2 Numerical results for the quartic oscillator and the double-well potential

The $1/P^4$ convergence of the HOA method is confirmed for the one-dimensional quartic oscillator and the double-well potential in this work. The double-well potential is given by

$$V(x) = \frac{m\omega^2}{8a^2}\left(x^2-a^2\right)^2\ . \qquad (1.43)$$

Fig. 1.1 shows the convergence of the probability distribution function of a particle in a double-well potential with increasing Trotter number. For $P = 4$, the contribution stemming from the the correction of the estimator for the probability distribution function is not small compared to the leading order term. This leads to the extra oscillations of the probability distribution function calculated with the HOA algorithm. For $P = 8$, the HOA result for the probability distribution function is almost exact while the primitive algorithm still shows an error of up to 10%. At $P = 16$, the error of the HOA approach is already smaller than the line thickness.

The convergence for the HOA algorithm is shown in Fig. 1.2 for the free and potential energy. One may conjecture that similar behavior is found for more general Φ^4 potentials, i.e., multi-well potentials. Indeed, the HOA treatment of a linear rotor impurity in a three-dimensional multi-well potential showed similar behavior as that found for our simple one-dimensional quartic oscillator [Müs96]. In that HOA study, the Debye anomaly in the specific heat (due to tunneling between equivalent minima in the potential energy) could be observed at much smaller Trotter numbers than if the primitive approach had been used.

1.4 Gaseous helium

The proof for the $1/P^4$ convergence of the HOA approach relies on the assumption that the thermal expectation value of the commutator $[\hat{T}, \hat{V}]$ as well as higher-order commutators, such as $[\hat{V}, [\hat{T}, \hat{V}]]$ are well-defined [TI84, Suz87, RR83]. More realistic potentials like the Lennard Jones potential $V = 4\epsilon[(\sigma/r)^{12} - (\sigma/r)^6]$ may not satisfy this assumption. Hence it is important to test how higher-order approximants perform for this class of potentials in particular in a situation where the system is far from being harmonic. The study of gaseous helium can therefore elucidate the convergence of HOA methods because quantum effects are strong even in the dilute gas phase as discussed recently, for example by Müser and Luijten [ML02]. Moreover, gaseous helium satisfies the condition of being strongly anharmonic.

In the low-density limit, noble gases are approximated rather well in terms of the second-order virial coefficient B_2 which can be expressed in terms of $g(r)$

$$B_2(\beta) = -2\pi \int_0^\infty drr^2 \left[g_{12}(r) - g_0(r)\right] \ . \tag{1.44}$$

Here $g_{12}(r)$ and $g_0(r)$ denote the radial distribution function in the interacting and non-interacting case, respectively. The calculation of B_2 can be done in terms of numerical matrix multiplication (NMM) method.

The analysis of the convergence rate of B_2 for gaseous argon (Lennard Jones potential) and gaseous ^{4}He (Aziz potential [ACW87]) showed that the convergence of the higher-order approximants remains $1/P^4$ like. However, the crossover to the regime in which the leading correction is in the order of $1/P^4$ happens at values of P that are larger than those values of P where the crossover takes place for the primitive approach. The reason why the singularity does not affect the convergence rate at large P lies in the fact that there is no wave function in the singularity and that its statistical weight near the singularity vanishes sufficiently fast.

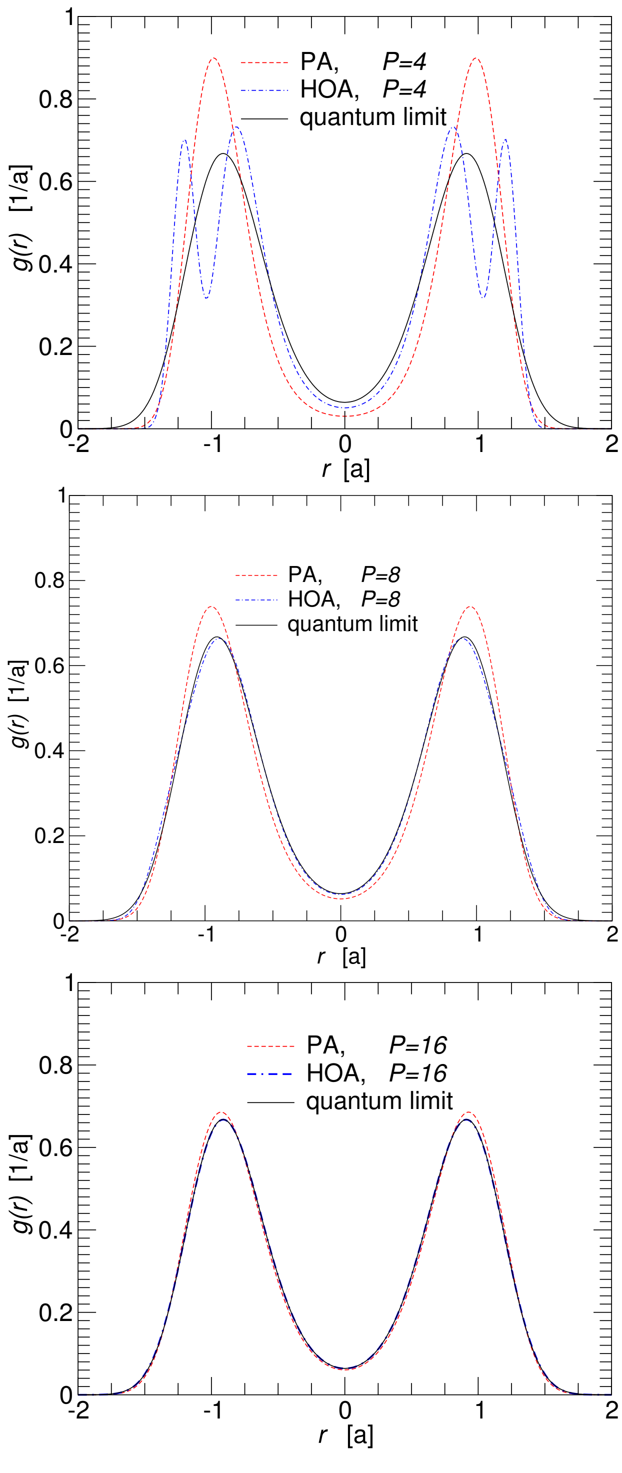

Figure 1.1: Convergence of the probability distribution function of a particle in a double-well potential with $\hbar\omega\beta = 8\sqrt{2}$ and $m\omega^2 a^2 \beta = 80$. The calculation was done with the Fortran 77 numerical matrix multiplication code given in appendix E.

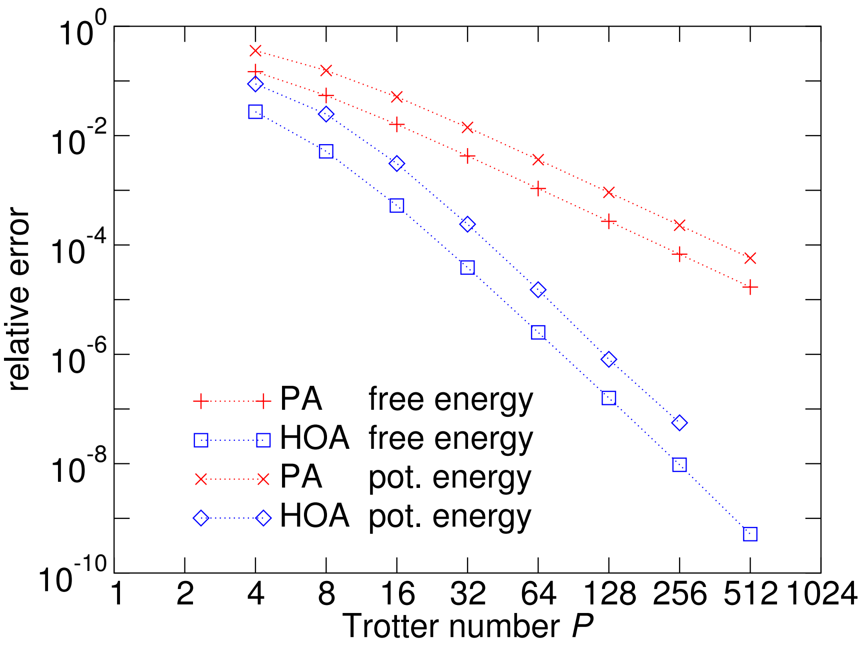

Figure 1.2: Relative error of the free and potential energy for the double-well potential with $\hbar\omega\beta = 8\sqrt{2}$ and $m\omega^2 a^2 \beta = 80$. The calculation was done with the Fortran 77 numerical matrix multiplication code given in appendix E.

Fig. 1.3 shows the relative error in the second virial coefficient of one dimensional Lennard Jonesium at two different temperatures. The choice of the space dimension $d = 1$ is motivated in part by the argument that the effect of a singularity is particularly large for small-dimensional systems. Moreover, the numerical stability and the range of Trotter numbers that can be investigated are larger in $d = 1$ than in $d = 3$. In Fig. 1.3, results are shown for Lennard Jonesium characterized by a de Boer parameter $\lambda = h/\sqrt{M\sigma^2\epsilon} = 2.7$, which reflects approximately ^{4}He. The two thermal energies investigated in Fig. 1.3 correspond to $T = 2$ K and $T = 16$ K if we interpret $\lambda = 2.7$ Lennard Jonesium as ^{4}He. Fig. 1.3 shows that the convergence of the virial coefficient in $d = 1$ is similar to that described above in the previous paragraph for $d = 3$. Despite the singularity, the HOA approach converges with $1/P^4$ to the proper quantum limit. The convergence starts at values of P that are slightly larger than those of the primitive approach. The author has also analyzed the kinetic and potential energy in the dilute gas phase as a function of Trotter number P. Both observables converge in a similar way as B_2. The leading corrections being in the order of $1/P^4$ for HOA and of order $1/P^2$ for PA.

1.5 Crystalline argon

It has been pointed out in section (1.2) that in order to calculate radial correlation functions and energies, it is necessary to alter the estimators with respect to the the primitive algorithm. An important question to address is, how well the HOA approach allows $g(r)$ to be calculated. In

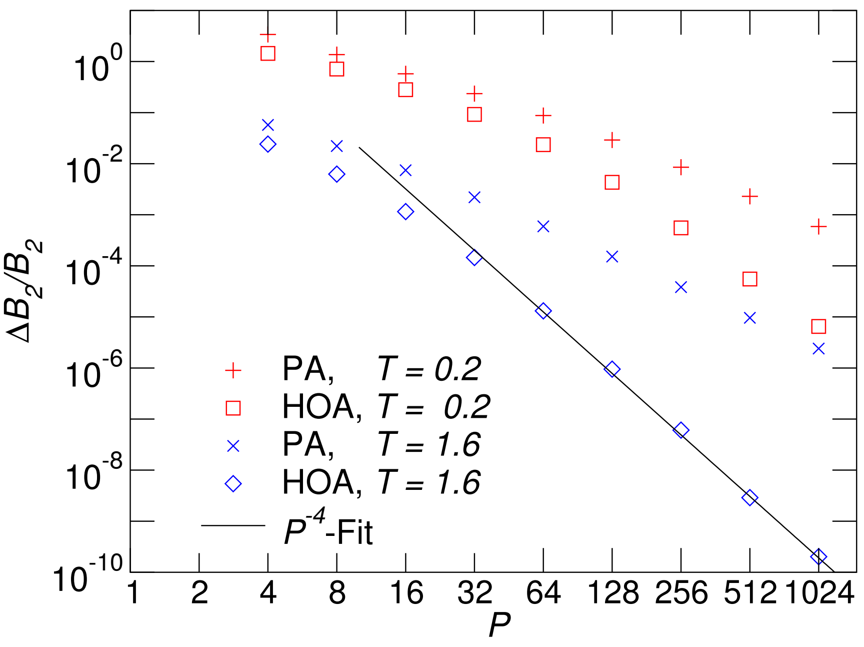

Figure 1.3: Relative error of the second virial coefficient B_2 of Lennard Jonesium with de Boer parameter $\lambda = 2.7$, an approximate description of ^{4}He. PA method and HOA method are compared at two different temperatures, where the dimensionless temperatures $T = 0.2$ and $T = 1.6$ correspond to 2 K and 16 K, respectively in the case of ^{4}He. The calculation was done with the Fortran 77 numerical matrix multiplication code given in appendix E.

order to examine this issue the HOA method is applied to crystalline argon.

1.5.1 Constant pressure path integral simulations

The simulation of crystalline argon is performed in the constant pressure (NPT) ensemble. To the knowledge of the author it has not yet been explained how calculations in the NPT ensemble are performed within the HOA algorithm. In the following, it is shown why the estimator for the volume does not receive higher-order corrections. Anderson [AAB$^+$84] originally proposed a method for constant pressure molecular dynamics, which involves coupling the system to an external variable V, the volume of the simulation box. This mimics the action of a piston on a real system. The method was extended by Parrinello and Rahman [PR80] to allow the simulation box to change shape as well, which is done through the introduction of scaled coordinates $\hat{\mathbf{s}}_i$ and momenta $\hat{\mathbf{q}}_i$ through

$$\begin{aligned} \hat{\mathbf{r}}_i &= h\hat{\mathbf{s}}_i \\ \hat{\mathbf{p}}_i &= h\hat{\mathbf{q}}_i \; . \end{aligned} \tag{1.45}$$

The columns of the transformation matrix h are the three vectors representing the sides of the simulation box. A classical Hamilton function H_V for the box geometry is added to the Hamilton operator:

$$\hat{H}_{\mathrm{NPT}} = \hat{H} + H_V \; . \tag{1.46}$$

The Hamilton function for the transformation matrix h is given by

$$H_V = T_{\mathrm{V}} + V_{\mathrm{v}} \tag{1.47}$$

with the kinetic part

$$T_{\mathrm{V}} = \frac{1}{2} Q \sum_{i,j} \dot{h}_{ij}^2 \tag{1.48}$$

and the potential energy for the box geometry

$$V_{\mathrm{V}} = pV = p \det(h) \; . \tag{1.49}$$

Here p stands for the pressure and $V = \det(h)$ is the volume of the simulation box. Rescaling the coordinates and the canonical conjugated momenta with a non-orthogonal matrix is not a symplectic transformation and the canonical commutation relations (CCR) do not remain form invariant under such a transformation. In Cartesian coordinates, the CCR read

$$\left[\hat{p}_i^m, V\left(\{\hat{r}_j^n\}\right)\right] = \frac{\hbar}{i} \frac{\partial V}{\partial r_i^m} \; . \tag{1.50}$$

For the scaled coordinates, they transform to

$$\left[\hat{q}_i^m, V\left(\{\hat{s}_j^n\}\right)\right] = \frac{\hbar}{i} G_{mr}^{-1} \frac{\partial V}{\partial s_i^r} \; . \tag{1.51}$$

with the metric tensor $G_{mn} = (h^T h)_{mn}$. If the geometry of the box is not changed and the box is only rotated, h is an orthogonal matrix and the metric tensor G is trivial $(G = \mathbb{1})$. It is important to note that the matrix h is a classical variable. Therefore the commutators of H_V with $\hat{s}_i^n$ and $\hat{q}_i^n$ vanish:

$$[H_V, \hat{s}_i^m] = [H_V, \hat{q}_i^m] = 0 \; . \tag{1.52}$$

The partition function is then given by

$$\begin{aligned} Z &= \left(\frac{Q}{2\pi\hbar^2\beta}\right)^{6/2} \int d^6h \prod_{j=0}^{N-1} d^3s_j \left\langle \{\mathbf{s}_j\} \left| e^{-\beta \hat{H}_{\mathrm{NPT}}} \right| \{\mathbf{s}_j\} \right\rangle \\ &= \left(\frac{Q}{2\pi\hbar^2\beta}\right)^{6/2} \int d^6h \prod_{j=0}^{N-1} d^3s_j \, e^{-\beta V_V} \left\langle \{\mathbf{s}_j\} \left| e^{-\beta \hat{H}} \right| \{\mathbf{s}_j\} \right\rangle \; . \end{aligned} \tag{1.53}$$

In the last step, the commutation relation (Eq. (1.52)) was used to factorize the exponential function. The statistical average is only performed over 6 of the $3 \times 3 = 9$ degrees of freedom of the box geometry h. The three degrees of freedom corresponding to simple rotations would contribute a factor $[m/(2\pi\hbar^2\beta)]^{(3/2)}$ to the partition function which does not appear because the orientation of the box is assumed to be fixed. The thermodynamic average of the volume is given by the logarithmic derivative of the partition function.

$$\begin{aligned} \langle V \rangle &= -\frac{1}{\beta}\frac{d}{dp}\ln Z \qquad (1.54) \\ &= \left(\frac{m}{2\pi\hbar^2\beta}\right)^{6/2} \int d^6h \prod_{j=0}^{N-1} d^3s_j \, \det(h)\, e^{-\beta V_V} \left\langle \{\mathbf{s}_j\} \left| e^{-\beta\hat{H}} \right| \{\mathbf{s}_j\} \right\rangle \Big/ Z \end{aligned}$$

Eq. (1.54) shows that the estimator of the volume is always given by $\det(h)$. This statement is independent of the Trotter equation that is used to calculate $\left\langle \{\mathbf{s}_j\} \left| e^{-\beta\hat{H}} \right| \{\mathbf{s}_j\} \right\rangle$.

1.5.2 Implementation of the effective force for a pair potential

In a PIMD simulation, an artificial dynamic has to be created in a way that in the coordinates are distributed according to the weight function $\exp\{-\beta U_{\mathrm{rp}}^{\mathrm{HOA}}\}$ in the simulation time average, with $U_{\mathrm{rp}}^{\mathrm{HOA}}$ given in Eq. (1.22). The effective force corresponding to the ring polymer potential U_{rp} is needed as an input parameter for the integration routine. The force belonging to the harmonic part of $U_{\mathrm{rp}}^{\mathrm{HOA}}$ which connects the monomers together to form a ring polymer is simply given by $\mathbf{F}_{\mathrm{i}}^{\mathrm{Harm},t} = \frac{m}{\hbar^2(\beta/P)^2}(2\mathbf{r}_i^t - \mathbf{r}_i^{t+1} - \mathbf{r}_i^{t-1})$. Here, the index t is the Trotter index and i labels the particles. The interaction of different ring polymers with each other is determined by the effective potential $\tilde{V}$ given in Eq. (1.21). In the following, the Trotter index will be suppressed because all expressions are local in imaginary-time. For many applications such as the simulation of noble gas or $\mathrm{Si0}_2$, the potential can be approximated reasonably well as a sum of pair potentials

$$V = \frac{1}{2}\sum_{i,j;i\neq j}^{N} V_{\mathrm{pair}}\left(|\mathbf{r}_{ij}|\right) \qquad (1.55)$$

with V_{pair} being the interaction of two particles and $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$. In the simulation of crystalline argon, the pair potential is of the Lenard Jones type:

$$V_{\mathrm{pair}}\left(|\mathbf{r}_{ij}|\right) = 4\epsilon\left(\left(\frac{\sigma}{|\mathbf{r}_{ij}|}\right)^{12} - \left(\frac{\sigma}{|\mathbf{r}_{ij}|}\right)^{6}\right) . \qquad (1.56)$$

The specific form of the pair interaction will not be important in the following. The classical forces $\mathbf{F}_i^{\mathrm{cl}} = -\nabla_i V$ can be calculated in the simulation as

$$\mathbf{F}_i^{\mathrm{cl}} = -\sum_{j=0;j\neq i}^{N-1} \left(DV_{\mathrm{pair}}\right)\left(|\mathbf{r}_{ij}|\right)\mathbf{r}_{ij} \qquad (1.57)$$

with the operator $D = (1/r)(d/dr)$. The effective potential which has to be used for force calculation reads according to Eq. (1.20):

$$\tilde{V} = V + \frac{1}{24}\hbar^2\left(\frac{\beta}{P}\right)^2 \sum_{j=0}^{N-1} \frac{1}{m_j}\mathbf{F}_j^{\mathrm{cl}} \cdot \mathbf{F}_j^{\mathrm{cl}} . \qquad (1.58)$$

The effective force on particle i is given by

$$\begin{aligned}\mathbf{F}_i^{\mathrm{eff}} &= \mathbf{F}_i^{\mathrm{cl}} + \frac{1}{12}\hbar^2\left(\frac{\beta}{P}\right)^2 \sum_{j=0;j\neq i}^{N-1}\left(\left[\frac{\mathbf{F}_j^{\mathrm{cl}}}{m_j} - \frac{\mathbf{F}_i^{\mathrm{cl}}}{m_i}\right]\cdot\mathbf{r}_{ij}\ \left(D^2V_{\mathrm{pair}}\right)(|\mathbf{r}_{ij}|)\,\mathbf{r}_{ij}\right. \\ &\quad \left. + \left(DV_{\mathrm{pair}}\right)(|\mathbf{r}_{ij}|)\left[\frac{\mathbf{F}_j^{\mathrm{cl}}}{m_j} - \frac{\mathbf{F}_i^{\mathrm{cl}}}{m_i}\right]\right)\ . \qquad (1.59)\end{aligned}$$

In order to save CPU time, the functions DV_{pair} and D^2V_{pair} are tabulated up to the cut-off radius. The calculation of the classical force scales linearly with the number of particles in the simulation and the average number of neighbors of a particle with a distance to the particle smaller than the cut-off radius. Please note that the classical forces have to be calculated first and stored, otherwise the calculation of the effective force would become of quadratic order in the number of particles and linear in the number of neighbors. They are also needed in the observation routine because they enter the higher-order corrections of the estimators (Eq. (1.24)). The implementation of Eq. (1.59) in a PIMD program which allows the box geometry to vary is given in appendix D.

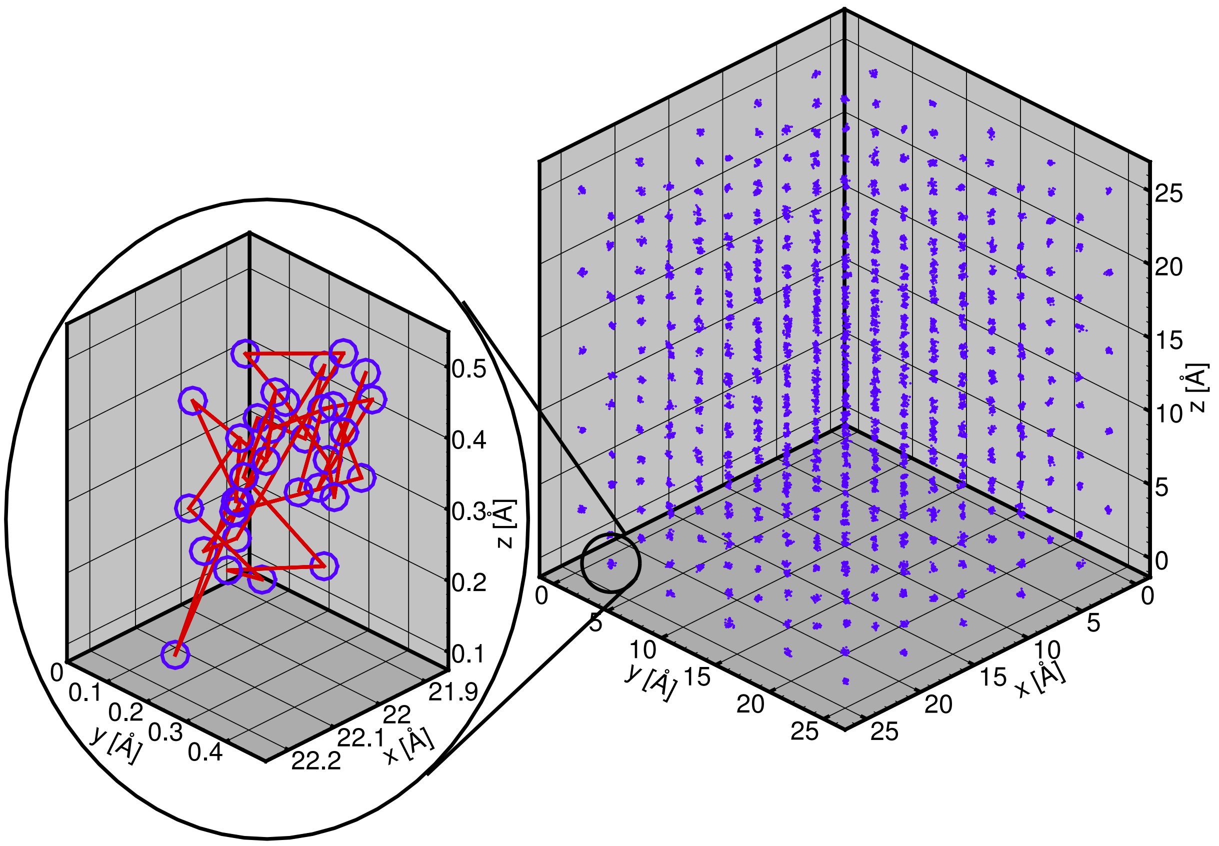

Figure 1.4: Snapshot of an argon crystal with 500 atoms at $T = 2\mathrm{K}$. The HOA algorithm is used in the simulation which reduces the systematic error to less than one percent. Each atom is represented as a ring polymer with Trotter number $P = 32$. In every space direction there are five fcc cells. For the Fortran 77 source code of the HOA force routine applied, see appendix D.

1.5.3 Simulation details

The system is modeled with the same Lennard Jones potential as in Ref. [MNB95], namely $\epsilon = 120$ k$_B$K and $\sigma = 3.405$ Å. A cut-off-radius for the interaction $r_{\text{cut-off}} = 5.79$ Å is used. This means that the first two neighbor shells of a particle in the argon crystal are able to interact with the particle. The number of atoms used in the simulation cell was $N = 500$. The simulation is performed in the NPT ensemble, the geometry and the size of the simulation cell are allowed to fluctuate, and the initial state is chosen such that the average geometry is cubic. Fig. 1.4 shows a snapshot of an argon crystal with 500 atoms at $T = 2$K. Each atom is represented as a ring polymer with $P = 32$ monomers. In every space direction there are five fcc cells.

1.5.4 Simulation results

One intention of this work is to study the convergence with respect to P. Therefore, finite size effects and the corrections due to the finite cut-off radius for the potential are not discussed.

The pair correlation function

The results for the pair correlation function $g(r)$ are presented in Fig. 1.5. It is interesting to

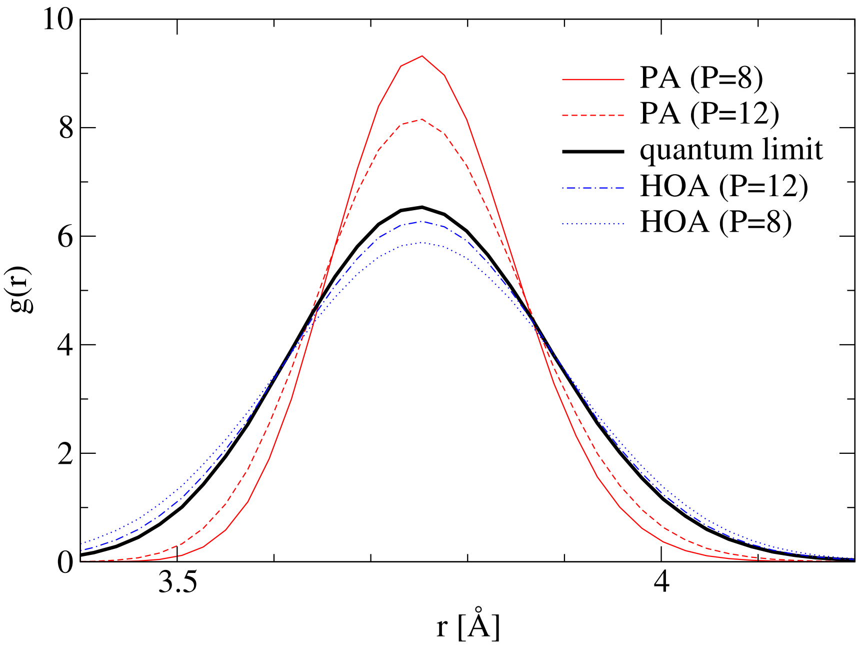

Figure 1.5: Next neighbor peak of the pair correlation function $g(r)$ of crystalline argon at $T = 2$K calculated with the PA and HOA algorithm for Trotter numbers $P = 8$ and $P = 12$. As a reference a quasi-exact correlation function which corresponds to the quantum limit (HOA, $P = 256$) is included.

note that $g(r)$ is too broad for the HOA approach, while it is too narrow using the PA algorithm.

Obviously, the agreement of the $P = 12$ HOA calculation is already very close to the quantum limit. This is a little surprising as the product of $k_B T P$ is still far below the thermal energy of the Debye temperature which is about $T_D \approx 85$ K. If the pair correlation function $g(r)$ is obtained without corrections, the agreement is distinctly reduced. This might explain the poor convergence of $g(r)$ reported by Li and Broughton for the attractive Coulomb potential [LB87]. They omitted the corrections to the $g(r)$-estimator leading to peaks in $g(r)$ that were even sharper than those obtained with PA using identical Trotter numbers.

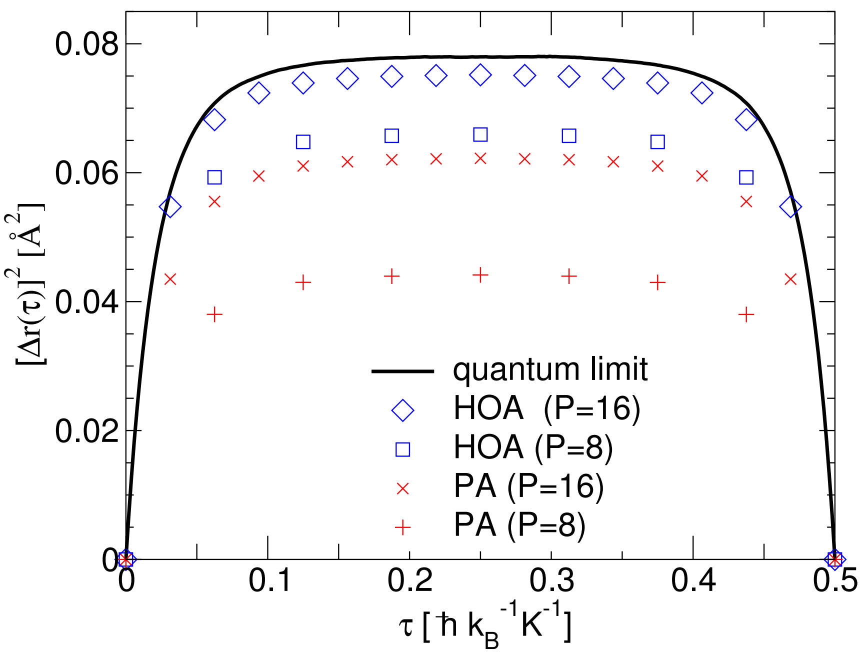

Figure 1.6: Imaginary-time mean square displacement $[\Delta r(\tau)]^2$ of crystalline argon at $T = 2$K, calculated with the PA and HOA algorithms for Trotter numbers $P = 8$ and $P = 16$. As a reference, a quasi-exact correlation function (HOA, $P = 256$) is included.

Imaginary-time mean square displacement

The delocalization of the argon atoms can be estimated with the imaginary-time mean square displacement $[\Delta r(\tau)]^2$. This function also contains useful information on the short-time dynamics of the system. It is defined as

$$[\Delta r(\tau)]^2 = \left\langle \frac{1}{N} \sum_{j=0}^{N-1} \left(\hat{\mathbf{r}}_\mathbf{j} - e^{\tau \hat{H}/\hbar} \hat{\mathbf{r}}_j e^{-\tau \hat{H}/\hbar} \right)^2 \right\rangle . \tag{1.60}$$

The estimators for $[\Delta x(\tau)]^2$ for the PA and HOA algorithm respectively are given by:

$$\text{PA :} \qquad [\Delta r]^2\left(\{\mathbf{r}_j^t\}\right)(\tau) = \frac{1}{NP}\sum_{j=0}^{N-1}\sum_{t=0}^{P-1}\left(\mathbf{r}_j^t - \mathbf{r}_j^{t+t'}\right)^2 \tag{1.61}$$

$$\text{HOA :} \qquad [\Delta r]^2\left(\{\mathbf{r}_j^t\}\right)(\tau) = \frac{1}{NP}\sum_{j=0}^{N-1}\sum_{t=0}^{P-1}\left[\left(\mathbf{r}_j^t - \mathbf{r}_j^{t+t'}\right)^2 + \Delta_{j,j}^{t,t+t'}\right] \tag{1.62}$$

$$\text{with} \quad \Delta_{j,j}^{t,t+t'} \;=\; \frac{\beta^2\hbar^2}{6P^2}\left(\frac{\nabla_{\mathbf{r}_j^t}V}{m_j} - \frac{\nabla_{\mathbf{r}_j^{t+t'}}V}{m_{j'}}\right)\cdot\left(\mathbf{r}_j^t - \mathbf{r}_j^{t+t'}\right) \quad \text{and} \quad \tau = t'\frac{\hbar\beta}{P}\,.$$

The results for $[\Delta r(\tau)]^2$ are shown in Fig. 1.6. The $P = 16$ HOA result approximates the exact result with an error of about 5% while the corresponding PA calculation shows a rather large error of about 20%. The $P = 8$ HOA calculation approximates the maximum of the imaginary correlation function already better than the $P = 16$ PA result.

Energy

The convergence of the kinetic and potential energy is shown in Fig. 1.7. The kinetic and potential energy both cross over (individually) to the $1/P^4$ convergence if the HOA is employed. The cross-over to the values of P, where the leading corrections $1/P^n$ dominate the finite P error, is larger for HOA than for PA. Hence this behavior resembles that of the B_2 expansion (discussed in paragraph 1.4), despite the fact that the argon crystal is relatively harmonic, that is to say, the harmonic approximation is stable and accounts for most of the ground state energy.

Volme

It is well-known that deviations of the (average) atomic volume $\langle V/N\rangle = \langle v\rangle$ from its value in the harmonic approximation are due to anharmonicity. Quantum fluctuations often enhance anharmonicity. For solid argon at $T = 2$ K, the anharmonicity is dominated by quantum fluctuations. Hence, the convergence of $\langle v\rangle$ with P to the quantum mechanical reference value is an important test for the HOA method. Results are shown in Fig. 1.8 for crystalline argon at $T = 2$ K. As in all other cases discussed above, HOA leads to distinctly reduced systematic errors with respect to PA. It is possible to confirm the $1/P^4$ convergence for the HOA algorithm within the statistical error bars.

Concluding all numerical results of this chapter, one finds that the HOA approach allows to reduce the Trotter number, which leads to shorter computation times. The price for this improvement is that higher effort has to be put into the development of the code. This effort pays off especially when one needs high accuracy, since one can take advantage of the better scaling behavior of the HOA approach.

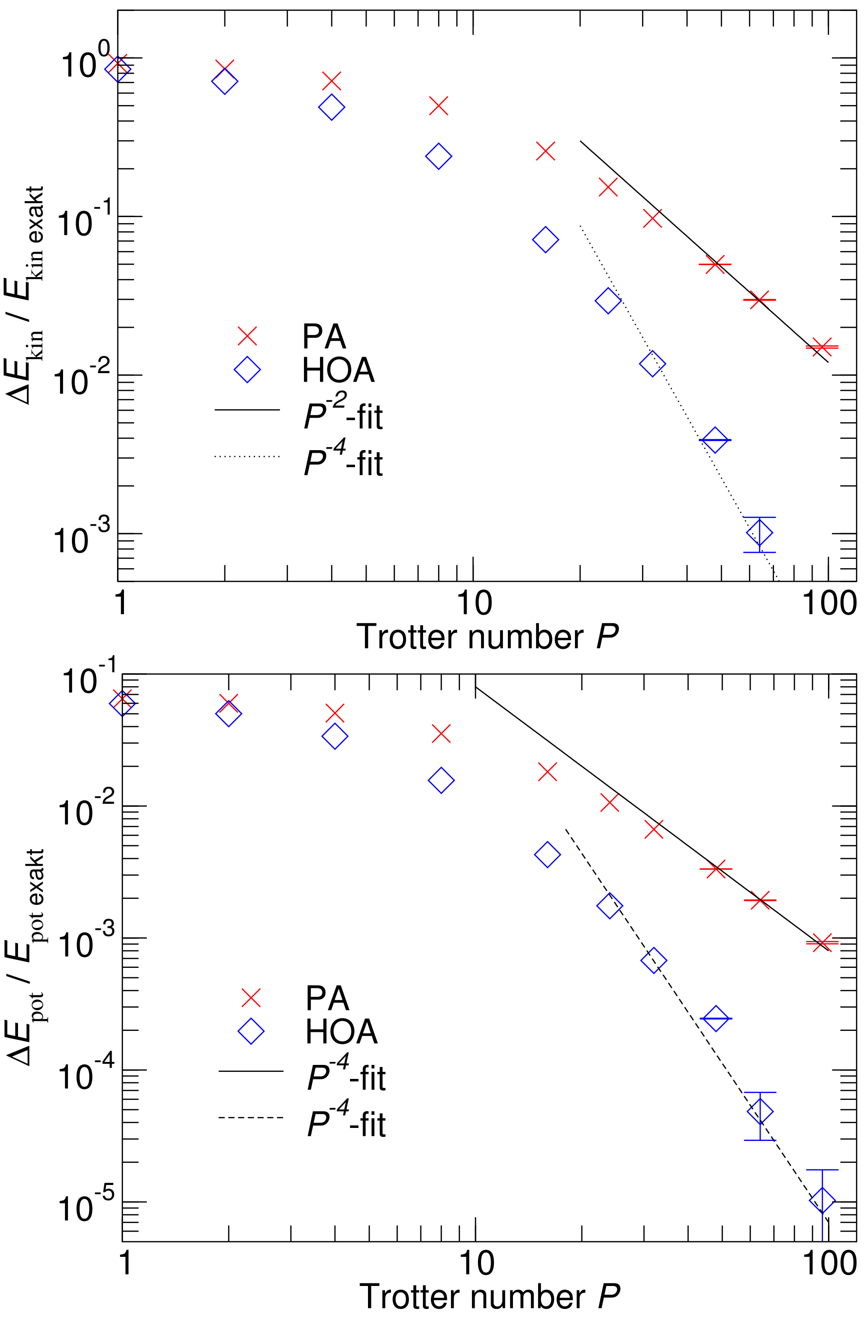

Figure 1.7: Relative error of the kinetic and potential energy for crystalline argon at $T = 2$K as a function of Trotter number P.

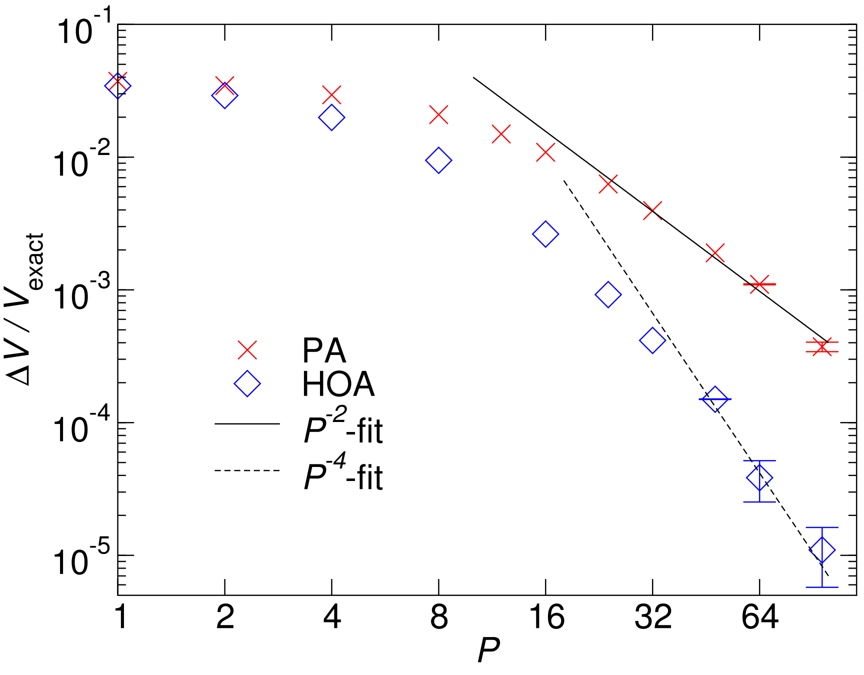

Figure 1.8: Relative error of the atomic volume for crystalline argon at $T = 2$K as a function of Trotter number P calculated with the PA (crosses) and HOA (diamonds) algorithm.

Chapter 2

Higher order Trotter approximation vs. an effective propagator (EPr) approach

2.0.5 The EPr method

The effective propagator approach is a non-primitive path integral method which is an alternative to the higher-order algorithm. It is based on the idea that the two-particle propagators are calculated prior to the simulation and are then reflected accurately in the simulation, e.g., are correct in all orders of $\hbar$. This is done by introducing an effective pair potential $V_{\text{pair}}^{\text{EPr}}$ which is defined as

$$V_{\text{pair}}^{\text{EPr}}\left(\mathbf{r}_1\mathbf{r}_2;\mathbf{r}_1'\mathbf{r}_2';\beta/P\right)=-\frac{P}{\beta}\ln\frac{\left\langle \mathbf{r}_1\mathbf{r}_2\left|\exp\left\{-\frac{\beta}{P}\left(\hat{T}+\hat{V}\right)\right\}\right|\mathbf{r}_1'\mathbf{r}_2'\right\rangle}{\left\langle \mathbf{r}_1\mathbf{r}_2\left|\exp\left\{-\frac{\beta}{P}\left(\hat{T}\right)\right\}\right|\mathbf{r}_1'\mathbf{r}_2'\right\rangle}\ . \tag{2.1}$$

$V_{\text{pair}}^{\text{EPr}}$ is a function that depends on $\mathbf{r}_1, \mathbf{r}_1', \mathbf{r}_2$, and $\mathbf{r}_2'$. Therefore, the interaction is said to be non-local in imaginary-time, e.g. $\mathbf{r}_1$ does not couple directly to $\mathbf{r}_2'$ in the primitive decomposition. Configurations $\{\mathbf{r}_j^t\}$ have to be sampled in the simulation such that they are distributed according to the statistical weight $\exp\left\{-\frac{\beta}{P}U_{\text{rp}}^{\text{EPr}}\right\}$ with the potential

$$U_{\text{rp}}^{\text{EPr}}\left(\{\mathbf{r}_j^t\}\right)=\sum_{t=0}^{P-1}\sum_{j=0}^{N-1}\left(\frac{m}{2\hbar^2\left(\beta/P\right)^2}\left(\mathbf{r}_j^{t+1}-\mathbf{r}_j^t\right)^2+\frac{1}{2}\sum_{j'=0;j'\neq j}^{N-1}V_{\text{pair}}^{\text{EPr}}\left(\mathbf{r}_j\mathbf{r}_{j'};\mathbf{r}_j'\mathbf{r}_{j'}';\beta/P\right)\right)\ . \tag{2.2}$$

The statistical averages of an observable diagonal in real space $L\left(\{\mathbf{r}_j\}\right)$ are calculated with the same estimators as in the primitive algorithm.

$$\left\langle L\left(\{\mathbf{r}_j\}\right)\right\rangle=\int\prod_{t=0}^{P-1}\prod_{j=0}^{N-1}\left(\sqrt{\frac{m}{2\pi\hbar^2\left(\beta/P\right)}}\right)^d d^d\mathbf{r}_j^t\ L_{\text{est}}^{\text{PA}}\left(\{\mathbf{r}_j^t\}\right)\ e^{-\frac{\beta}{P}U_{\text{rp}}^{\text{EPr}}\left(\{\mathbf{r}_j^t\}\right)}\Big/Z(\beta)+\mathcal{O}\left(1/P^2\right)\ . \tag{2.3}$$

Compared to the primitive algorithm, only the interaction potential which determines the dynamics of a PIMD program or the probability for the acceptance/rejection of a Monte Carlo move in a PIMC simulation has to be changed. Still, due to the non-locality of the effective pair potential, the implementation is more complicated than it is for the primitive algorithm. It is necessary to calculate $V_{\rm pair}^{\rm EPr}$ prior to the simulation and store the result such that it can be evaluated quickly during the simulation. While the classical pair potential used in the primitive approach can be tabulated as a one dimensional table, a three-dimensional table is needed to store the effective pair potential if the Hamiltonian is Galilei invariant. [1] Alternately, fit functions for $U_{\rm rp}\left(\left\{\mathbf{r}_j^t\right\}\right)$ can be defined which reduce the three dimensional table to several one dimensional tables for the fit coefficients [Cep95]. The calculation of both, the diagonal and the non-diagonal elements, is not trivial for non-harmonic potentials. It can be done with the NMM method: First, the pair density is factorized into a center of mass term that is free-particle like and a term that is a function of the relative coordinates. The factor of the density matrix which depends on the relative coordinate can be treated like a single particle in a spherical potential. In one dimension, it can be calculated as described in paragraph 1.3.1. For two- or three- dimensional systems, one expands the relative pair density into partial waves. Each partial wave component is the density matrix of a particle in one dimension with an additional centrifugal term and can be treated as described in paragraph 1.3.1. In Ref. [Cep95] an in-depth discussion of that problem is given. For the one-dimensional model system discussed in this work, however, the approach can be simplified significantly, i.e. the pair density matrix can be calculated analytically. It should be noted that the effective propagator (EPr) method is particularly useful for ill-behaved potentials such as the attractive Coulomb potential [Pol88] because the singularity of the interaction is regularized due to quantum fluctuations.

2.1 The reduced effective propagator (r-EPr) method

As mentioned above, the implementation of the full two-particle pair propagator with correct diagonal and non-diagonal elements is rather expensive. On the other hand, if one restricts oneself to the diagonal elements of the density matrix, it is possible to construct an algorithm which can be implemented similarly to the primitive one: Only the specific shape of the classical pair potential has to be changed to an effective potential depending on the temperature and the particle mass. The main idea of the r-EPr is to incorporate only those corrections to the primitive decomposition that are local or in other words the corrections on the diagonal elements of the density matrix only. The reduced effective pair potential is given by

$$V_{\rm pair}^{\rm r-EPr}\left(\mathbf{r}_1\mathbf{r}_2;\mathbf{r}_1'\mathbf{r}_2';\beta/P\right)=\frac{1}{2}\left(V_{\rm pair}^{\rm EPr}\left(\mathbf{r}_1\mathbf{r}_2;\mathbf{r}_1\mathbf{r}_2;\beta/P\right)+V_{\rm pair}^{\rm EPr}\left(\mathbf{r}_1'\mathbf{r}_2';\mathbf{r}_1'\mathbf{r}_2';\beta/P\right)\right)\ . \qquad (2.4)$$

The terms in the high-temperature density matrix that involve coordinates at different Trotter indices are neglected.

The EPr and r-EPr method are the same if $P=1$. In the gas phase, the r-EPr approach allows an almost exact treatment of the quantum effects already for $P=1$. That fact was conjectured

[1] Due to the translational invariance of the Hamiltonian the density matrix can be factored into two scalar factors $\rho_{\rm CMS}(\mathbf{X},\mathbf{X}')$ and $\rho_{\rm rel}(\mathbf{r},\mathbf{r}')$. The center of mass part $\rho_{\rm CMS}(\mathbf{X},\mathbf{X}')$ has the form of the density matrix for a free particle. The interaction part $\rho_{\rm rel}(\mathbf{r};\mathbf{r}')$ is a scalar and therefore invariant under rotations. For this reason it can only depend on the three scalars $\mathbf{r}^2$, $\mathbf{r}'^2$ and $\mathbf{r}\cdot\mathbf{r}'$.

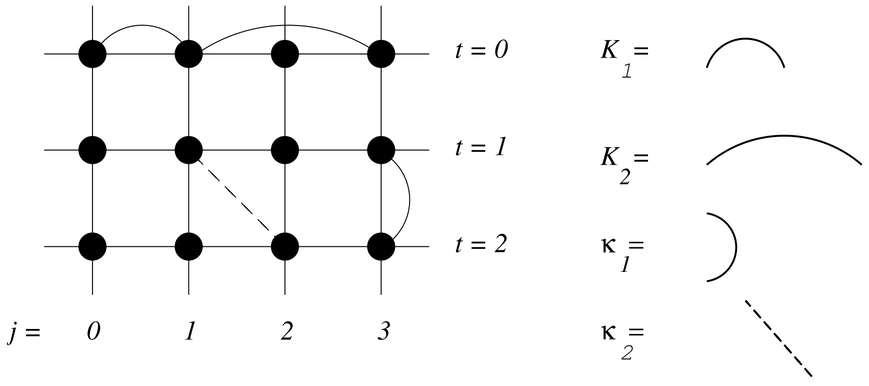

Figure 2.1: Illustration of coupling between atoms of the harmonic chain at discretized imaginary-time τ. The straight horizontal lines represent springs between nearest neighbors of stiffness K, the solid vertical lines represent springs of stiffness $m/(\hbar^2(\beta/P)^2)$.

in this work for gaseous Argon. For well-behaved potentials, the r-EPr method must converge to the proper quantum limit because the leading differences between the r-EPr effective potential and the true potential vanish with $1/P^2$. The calculation of observables in r-EPr approach is similar to the EPr approach.

The r-EPr approach is similar to an idea suggested by Thirumalai et al. [THB84], who constructed effective interaction potentials from the diagonal elements of the high-temperature density matrix, see also a related paper by Pollock and Ceperley [PC84].

2.2 A test model for the convergence: The linear harmonic chain

In order to analyze the convergence of PA, HOA and EPr and r-EPr method respectively, a one-dimensional linear chain with harmonic next neighbor coupling

$$V = \frac{1}{2} \sum_{j=0}^{N-1} K \left(x_j - x_{j+1}\right)^2 \tag{2.5}$$

is chosen. Periodic boundary conditions, $x_N \equiv x_0$ are applied and the masses m are identical for all atoms.

The HOA, EPr and r-EPr method invoke correction terms in the potential energy of the ring polymers with respect to the original expression of the primitive algorithm in Eq. (1.8). All four approaches can be represented as the limiting cases of a classical scalar field theory in a discrete (2+1) dimensional space-time. A graphical illustration is given in Fig. 2.1. The new effective energy U_{rp} that enters the Boltzmann factor reads

$$\begin{aligned} U_{\mathrm{rp}} &= \frac{1}{2}\sum_{t=0}^{P-1}\sum_{j=0}^{N-1}\sum_{\pm}\Big[\ (\kappa+\kappa_1)\left(x_j^t - x_j^{t+1}\right)^2 \\ &\quad + (K+\ K_1)\left(x_j^t - x_{j+1}^t\right)^2 + \kappa_2(x_j^t - x_{j+1}^{t\pm1})^2 \\ &\quad + K_2(x_j^t - x_{j+2}^t)^2\ \Big]. \end{aligned} \tag{2.6}$$

The expression for U_{rp} is diagonal in the normal mode representation with the normal coordinates u_k^s given by

$$u_k^s = \sum_{t=0}^{P-1}\sum_{j=0}^{N-1} U_{st}U_{kj}x_j^t\ . \tag{2.7}$$

The orthogonal normal mode transformation U is defined in appendix A. In terms of the normal mode coordinates, U_{rp} reads

$$U_{\mathrm{rp}} = \frac{1}{2}\sum_{s=0}^{P-1}\sum_{k=0}^{N-1} K\left(k,s\right)\,(u_k^s)^2\ , \tag{2.8}$$

with

$$\begin{aligned} K\left(k,s\right) &= (\kappa+\kappa_1)4\sin^2\left(\frac{\pi}{P}s\right) + (K+K_1)4\sin^2\left(\frac{\pi}{N}k\right) \\ &\quad + \kappa_2 4\left[\sin^2\left(\pi(k/N+s/P)\right) + \sin^2\left(\pi(k/N-s/P)\right)\right] \\ &\quad + K_2 4\sin^2\left(\frac{\pi}{N}2k\right) \end{aligned} \tag{2.9}$$

and $\kappa = mP^2/(\beta^2\hbar^2)$. The partition function Z_c for the classical system illustrated in Fig. 2.1 can be reduced to NP Gaussian integrals. Z is given by

$$Z = \prod_{s=1}^{P-1}\prod_{k=0}^{N-1}\sqrt{\frac{2\hbar^2\beta^2}{mP^2}K\left(k,s\right)}\ . \tag{2.10}$$

For the four different algorithms (PA, HOA, EPr and r-EPr method), there are different functions for K_1, K_2, κ_1, and κ_2. The expressions for these effective coupling coefficients are summarized in Tab. (2.1).

In the following, a derivation for the coupling coefficients and the thermal expectation value of the potential energy $\langle V_P\rangle$ for a given Trotter number P will be given. One can expect that the errors in different observables vanish with the same power of P, which is the reason why it is sufficient to investigate $\langle V_P\rangle$ only. The prefactor of the corrections can certainly depend on the observable. The calculations for the four approaches will be separated into four paragraphs.

	PA	HOA	EPr	r-EPr
K_1	0	$(\hbar\beta K)^2/(3mP^2)$	$K(A-C-1)$	$K(A-1)$
K_2	0	$-(\hbar\beta K)^2/(12mP^2)$	0	0
κ_1	0	0	$-KC$	0
κ_2	0	0	$\frac{1}{2}KC$	0

Table 2.1: Expressions for the effective coupling coefficients that are represented in Fig. 2.1

2.2.1 Solution for the PA method

For the primitive method U_{rp} is simply given by

$$U_{\text{rp}} = \sum_{t=0}^{P-1}\sum_{j=0}^{N-1}\left[\frac{m}{\hbar^2(\beta/P)^2}\left(x_j^{t+1}-x_j^t\right)^2+\frac{1}{2}K\left(x_{j+1}^t-x_j^t\right)^2\right] . \tag{2.11}$$

Comparison of Eq. (2.11) with Eq. (2.8) leads to $K_1 = K_2 = \kappa_1 = \kappa_2 = 0$. The usual thermodynamic relationships can be applied in order to calculate the thermal expectation value of $\langle V_P\rangle$:

$$\langle V\rangle = -\frac{K}{\beta}\frac{d}{dK}\ln(Z) . \tag{2.12}$$

This relationship simply follows the formal expression for the quantum mechanical partition function of a linear, monoatomic harmonic chain with coupling constant K. The result for $\langle V_P\rangle$ calculated with the primitive approximation is then given by

$$\langle V_P\rangle = \frac{K}{2\beta}\sum_{s=0}^{P-1}\sum_{k=0}^{N-1}\frac{4\sin^2\left(\pi q/N\right)}{K^{\text{PA}}\left(k,s\right)} . \tag{2.13}$$

2.2.2 Solution for the HOA method

As explained in chapter 1, the application of the higher-order Trotter decomposition (Eq. (1.19)) leads to a modification of the potential. The gradient of the potential energy V according the particle coordinate x_j is given by

$$\frac{d}{dx_j}V = \sum_{j'=0}^{N-1}K\left(x_j'-x_{j'+1}\right)\left(\delta_{j,j'}-\delta_{j'+1,j}\right) = K\left(2x_j-x_{j+1}-x_{j-1}\right) . \tag{2.14}$$

For the HOA effective potential $\tilde{V}$ one finds

$$\tilde{V} = \sum_{j=0}^{N-1}\left\{\frac{1}{2}K\left(x_j-x_{j+1}\right)+\frac{1}{24}\frac{\hbar^2}{m}\left(\frac{\beta}{P}\right)^2K^2\left(2x_j-x_{j+1}-x_{j-1}\right)^2\right\}\Bigg|_{x_0=x_N} . \tag{2.15}$$

The dependence of $\tilde{V}$ according to the particle coordinates can be expressed as squares of next neighbor differences and next to next neighbors differences because

$$(2x_j - x_{j+1} - x_{j-1})^2 = 2\,(x_j - x_{j+1})^2 + 2\,(x_j - x_{j-1})^2 - (x_{j-1} - x_{j+1})^2 \ . \tag{2.16}$$

This results in the effective potential in the HOA algorithm:

$$\begin{aligned} \tilde{V} &= \frac{1}{2}\sum_{j=0}^{N-1}\left\{ \left(K + \frac{1}{3}\frac{\hbar^2}{m}\left(\frac{\beta}{P}\right)^2 K^2\right)(x_j - x_{j+1})^2 \right. \\ &\qquad \left. \left. - \frac{1}{12}\frac{\hbar^2}{m}\left(\frac{\beta}{P}\right)^2 K^2\,(x_{j-1} - x_{j+1})^2 \right\}\right|_{x_0 = x_N} . \end{aligned} \tag{2.17}$$

The coefficients $K_1,\ K_2,\ \kappa_1$ and κ_2 for the HOA method, given in Tab. (2.1) result from the expression for $\tilde{V}$. With the coupling coefficients, the thermal expectation value at finite Trotter number P for the potential energy can be calculated according to Eq. (2.12). The thermodynamic relation given in Eq. (2.12) is also applicable for the HOA method because the only modification with respect to the primitive approximation (PA) is that a better approximant for the high-temperature density matrix is employed. This leads to the expression

$$\begin{aligned} \langle V_P \rangle &= \frac{K}{2\beta}\sum_{s=0}^{P-1}\sum_{k=0}^{N-1}\left\{ \left(1 + \frac{2}{3}\frac{\hbar^2}{m}\left(\frac{\beta}{P}\right)^2 K\right) 4\sin^2\left(\frac{\pi}{N}q\right) \right. \\ &\qquad \left. - \frac{2}{3}\frac{\hbar^2}{m}\left(\frac{\beta}{P}\right)^2 K \sin^2\left(\frac{\pi}{N}2q\right) \right\} \bigg/ K^{\mathrm{HOA}}(k,s) \end{aligned} \tag{2.18}$$

for the thermal expectation value of the potential energy at the finite Trotter number P where $K^{\mathrm{HOA}}(k,s)$ refers to that expression for $K(k,s)$ in Eq. (2.10) which is obtained by inserting the HOA values for K_1 and K_2.

The same result for $\langle V_P \rangle$ could have been obtained by calculating the second moments of the eigenmodes $\langle (u_j^s)^2 \rangle$ from equipartition. The resulting averages $\langle (u_j^s)^2 \rangle$ could then have been used to calculate the proper HOA potential energy estimator $V + 2V_{\mathrm{cor}}$.

2.2.3 Solution for the EPr method

The idea of the effective propagator algorithm (EPr) is to introduce an effective potential $V_{\mathrm{pair}}^{\mathrm{EPr}}$ which produces the correct pair correlation function for an arbitrary Trotter number in the low density limit. In this paragraph, $V_{\mathrm{pair}}^{\mathrm{EPr}}$ is derived for the harmonic chain. To do this, one has to consider a dimer of two atoms coupled by a harmonic spring of stiffness K. The density matrix of the dimer can be written as the product of a free particle density matrix for the center-of-mass mode X and the density matrix for a harmonic oscillator for the reduced distance x.

$$\rho\,(x_1, x_2; x_1', x_2'; \beta/P) = \rho(x, X; x', X'; \beta/P) = \rho_{\mathrm{CMS}}(X; X'; \beta/P)\ \rho_{\mathrm{rel}}(x, ; x'; \beta/P) \tag{2.19}$$

Both, the free particle density matrix and the density matrix of an oscillator with spring constant K and mass μ are known exactly. The free particle density matrix is simply

$$\rho\,(X, X', \beta/P) = \sqrt{\frac{M}{2\pi\hbar\,(\beta/P)}}\exp\left\{-\frac{M}{2\hbar^2(\beta/P)}(X-X')^2\right\} \tag{2.20}$$

(see Eq. (1.8)), while the oscillator's density matrix is given by [Fey72]:

$$\begin{aligned}\rho(x, x', \beta/P) = \sqrt{\frac{\sqrt{\mu k}}{2\pi\hbar\sinh(2f)}} \times \qquad (2.21)\\ \exp\left\{-\frac{\beta}{P}\frac{K}{f\sinh(f)}\left[(x^2+x'^2)\cosh(f) - 2x\,x'\right]\right\},\end{aligned}$$

with the definition

$$f = \frac{\beta}{P}\hbar\sqrt{\frac{K}{\mu}}\,. \tag{2.22}$$

The Gallilei transformation from the center of mass (CMS) system to the laboratory system reads:

$$\begin{aligned} x &= x_1 - x_2 & X &= (x_1 + x_2)/2 \\ x' &= x_1' - x_2' & X' &= (x_1' + x_2')/2 \\ \mu &= m/2 & M &= 2m\,. \end{aligned} \tag{2.23}$$

The density matrix for the dimer in the CMS system is:

$$\begin{aligned} &\rho_{\text{dimer}}(x, X; x', X'; \beta/P) \qquad (2.24)\\ = \;& \rho_{\text{CMS}}(X; X'; \beta/P)\;\rho_{\text{rel}}(x, ; x'; \beta/P) \\ = \;& \sqrt{\frac{M}{2\pi\hbar^2(\beta/P)}}\exp\left\{-\frac{M}{2\hbar^2(\beta/P)}\left(X-X'\right)^2\right\} \\ & \times\sqrt{\frac{\mu}{2\pi\hbar^2(\beta/P)}\frac{f}{\sinh[f]}} \\ & \times\exp\left\{-\frac{\mu}{2\hbar^2\left(\beta/P\right)}\frac{f}{\sinh[f]}\left(\left(x^2+x'^2\right)\cosh[f] - 2xx'\right)\right\}\,. \qquad (2.25)\end{aligned}$$

Next, the product of the free and the oscillator density matrix needs to be transformed back into the laboratory frame and the effective potential needs to be expressed in a form corresponding to Eq. (2.6). It is useful to introduce the following variables for the squared particle distances:

$$\begin{aligned} \Delta^2 &= (x_1 - x_2)^2 + (x_1' - x_2')^2 \\ q^2 &= (x_1 - x_1')^2 + (x_2 - x_2')^2 \\ d^2 &= (x_1 - x_2')^2 + (x_1' - x_2)^2\,, \end{aligned} \tag{2.26}$$

which satisfy the relations

$$\begin{aligned} -2xx' &= q^2 - d^2 \\ 4\left(X - X'\right)^2 &= q^2 + d^2 - \Delta^2 \\ x^2 + x'^2 &= \Delta^2 \, . \end{aligned} \tag{2.27}$$

The density matrix in the laboratory frame follows from Eq. (2.24) with Eq. (2.27):

$$\begin{aligned} \rho_{\text{dimer}}(x_1, x_2; x_1', x_2'; \beta/P) &= \frac{m}{2\pi\hbar^2(\beta/P)} \sqrt{\frac{f}{\sinh[f]}} \\ &\quad \times \exp\left\{-\frac{\beta}{P}\frac{1}{2}\left(2\frac{m}{\hbar^2(\beta/P)^2} - KC\right)\frac{q^2}{2}\right\} \\ &\quad \times \exp\left\{-\frac{\beta}{P}\frac{1}{2}K\left(A - C\right)\frac{\Delta^2}{2}\right\} \\ &\quad \times \exp\left\{-\frac{\beta}{P}\frac{1}{2}KC\frac{d^2}{2}\right\} \, . \end{aligned} \tag{2.28}$$

According to the definition of the pair potential given in Eq. (2.1), the effective potential $V_{\text{pair}}^{\text{ERp}}$ for a harmonic chain can be written in the form

$$\begin{aligned} V_{\text{pair}}^{\text{EPr}}(x_1, x_2; x_1', x_2'; \beta/P) &= -\frac{1}{\beta}\ln\left(\frac{\rho_{\text{dimer}}(x_1, x_2; x_1', x_2'; \beta/P)}{\rho_{\text{free}}(x_1, x_2; x_1', x_2'; \beta/P)}\right) + \text{const} \\ &= -\frac{1}{2}KC\frac{q^2}{2} + \frac{1}{2}K\left(A - C\right)\frac{\Delta^2}{2} + \frac{1}{2}K\frac{C}{2}d^2 \\ &= \frac{1}{2}\left(\kappa_1\frac{q^2}{2} + \left(K + K_1\right)\frac{\Delta^2}{2} + \kappa_2 d^2\right) \, , \end{aligned} \tag{2.29}$$

with the parameters

$$A = \frac{2}{f}\tanh\left(\frac{f}{2}\right) \quad \text{and} \quad C = \frac{2}{f^2}\left\{1 - \frac{f}{\sinh(f)}\right\} \, . \tag{2.30}$$

The parameters K_1, K_2, κ_1 and κ_2 are introduced in Tab. (2.1). They can be calculated by inserting the result for $V_{\text{pair}}^{\text{EPr}}$ into the expression for $U_{\text{rp}}^{\text{EPr}}$ in Eq. (2.2) and by compaing the result with Eq. (2.8). As explained above the estimators do not receive corrections in the EPr algorithm. According to Eq. (2.12), the potential energy $\langle V_p \rangle$ is given by:

$$\langle V_p \rangle = \frac{1}{2}K\frac{1}{P}\sum_{t=0}^{P-1}\sum_{j=0}^{N-1}\left\langle \left(x_j^t - x_j^{t+1}\right)^2 \right\rangle = \frac{1}{2}K\frac{1}{P}\sum_{s=0}^{P-1}\sum_{j=1}^{N-1} 4\sin^2\left(\frac{\pi}{N}k\right)\left\langle (u_k^s)^2 \right\rangle \, . \tag{2.31}$$

The expectation value of the squared normal modes can be found with the equipartition theorem as

$$\frac{1}{2}K\left(k, s\right)\left\langle (u_k^s)^2 \right\rangle = \frac{1}{2(\beta/P)} \, . \tag{2.32}$$

This leads to the result for the potential energy calculated with Trotter number P by the EPr approach which is simply

$$\langle V_P \rangle = \frac{K}{2\beta} \sum_{s=0}^{P-1} \sum_{j=1}^{N-1} \frac{4\sin^2(\pi q/N)}{K^{\mathrm{EPr}}(k,s)} \ . \tag{2.33}$$

Here K^{EPr} is given by Eq. (2.10) with the effective coupling coefficients for the Epr method given in Tab. 2.1.

2.2.4 Solution for the r-EPr method

The r-EPr potential $V_{\mathrm{pair}}^{\mathrm{r-EPr}}$ is related to the diagonal elements of the pair density matrix. Inserting the result for $V_{\mathrm{pair}}^{\mathrm{EPr}}$ (Eq. (2.29)) into Eq. (2.4) leads to the reduced pair potential for the harmonic chain

$$V_{\mathrm{pair}}^{\mathrm{r-EPr}}(x_1, x_2; x_1', x_2', \beta/P) = \frac{1}{2} K A \frac{\Delta^2}{2} \ . \tag{2.34}$$

From the expression for $V_{\mathrm{pair}}^{\mathrm{r-Epr}}$, the effective coupling coefficients for the reduced propagator algorithm given in Tab. 2.1 follow. The coupling of the neighbored particles is altered $[K_1 = (A-1)K]$ while there is no coupling $(K_2 = 0)$ to the second neighbor, and the coupling to modes at different imaginary-time is not changed with respect to the PA approach $(\kappa_1 = \kappa_2 = 0)$. The expression for the potential energy follows in analogy to the EPr result. It is given by

$$\langle V_P \rangle = \frac{K}{2\beta} \sum_{s=0}^{P-1} \sum_{j=1}^{N-1} \frac{4\sin^2(\pi q/N)}{K^{\mathrm{r-EPr}}(k,s)} \ . \tag{2.35}$$

Here $K^{\mathrm{r-EPr}}$ is given by Eq. (2.10) with the effective coupling coefficients for the r-Epr method given in Table 2.1.

2.3 Comparison of the methods

The main issue of the comparative study presented here is the analysis of the convergence of thermal expectation values such as the potential energy $\langle V_P \rangle$ to the proper quantum limit as a function of Trotter number P. As a test model, a linear chain consisting of $N = 5$ atoms and periodic boundary conditions is considered. The sums appearing in the expressions for the thermal average of the potential energy are performed numerically with a Mathematica program. For the source code see appendix F. The convergence does not depend on N in a qualitative way. It is examined at a fixed thermal energy well below the Debye frequency of the chain, namely at inverse temperature $\beta = 64/(\hbar\sqrt{K/m})$.

A linear plot of $\langle V_P \rangle$ is shown in Fig. 2.2. It can be seen that at $P = 1$ the EPr and the r-EPr method start off with estimates that are very close to the quantum limit while PA and HOA start off with an estimate near the classical value. Upon increasing P, the EPr approaches the proper value from below, while for the r-EPr method the deviation between estimate and proper result

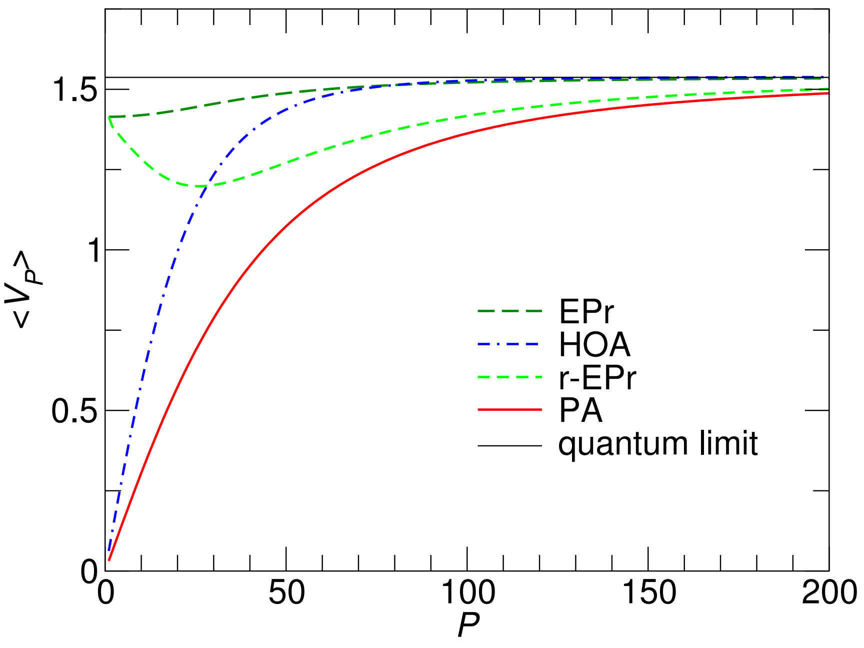

Figure 2.2: Thermal expectation value of the potential energy $\langle V_P \rangle$ for a $N = 5$ chain as a function of Trotter number P; $\beta = 64, \quad \hbar = K = m = 1$.

first increases before it decreases again. At a Trotter number $P \approx 64$, the HOA method becomes similarly good as the EPr approach.

In order to study the convergence in a more quantitative way, it is convenient to consider the relative deviation of $\langle V_P \rangle$ from the exact value $\langle V_\infty \rangle$ as a function of P in a double logarithmic plot (Fig. 2.3). It can be seen that at large Trotter numbers the HOA method converges with $1/P^4$ to the quantum limit, while all other methods converge with $1/P^2$ only. The EPr method starts at small Trotter numbers with a much smaller error than the PA and HOA methods. The value of P at which convergence starts is similar in all approaches. The accuracy where HOA becomes better than EPr is 1.7%. For this particular model system, this value of 1.7% was found to be independent of temperature. One can expect it to be similar for all systems that are dominated by harmonic interactions. An important practical question is how P has to be increased for the various approaches if the temperature T is lowered and the required relative accuracy is constant, e.g. 1%. The results are shown in Fig. 2.4. In order to keep the relative accuracy constant, all methods require that P increases linearly with inverse temperature β. The r-EPr approach, which is a little more difficult to implement than the PA method, requires slightly reduced Trotter numbers with respect to PA. It should be emphasized that the behavior shown in Fig. 2.4 is qualitatively similar if the accuracy criterion for P is changed, however, the stricter the criterion the larger the gap between EPr and HOA. This trend can be seen in Fig. 2.4 right, where 0.1 % accuracy is required instead of 1 % as shown in Fig. 2.4 left. Only if one is confined to the use of very small P, EPr might be the better choice. One may conclude that the optimal method depends on the desired accuracy: Effective propagator methods can be used if one is restricted

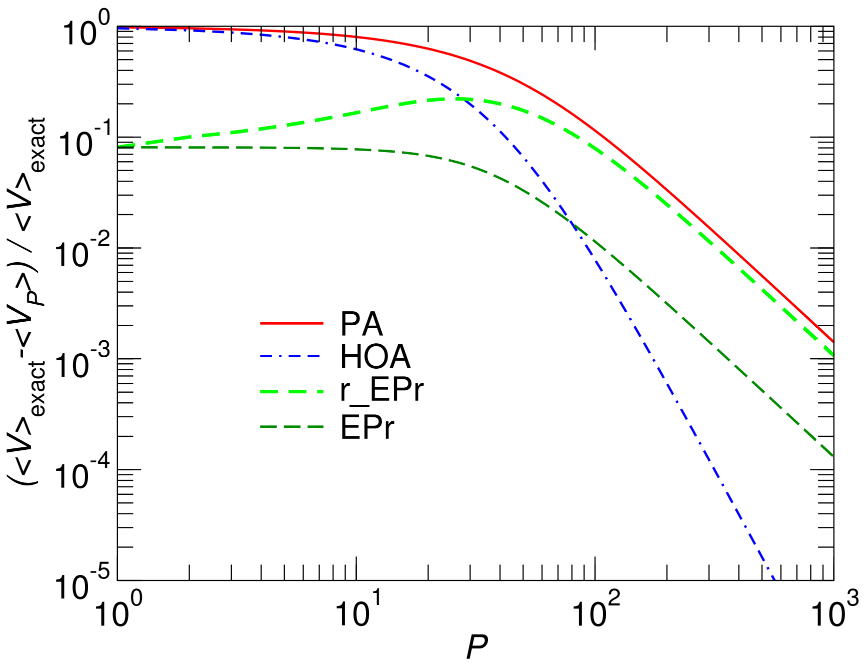

Figure 2.3: Relative error of the potential energy for a $N = 5$ chain as a function of Trotter number P; $\beta = 64, \quad \hbar = K = m = 1$.

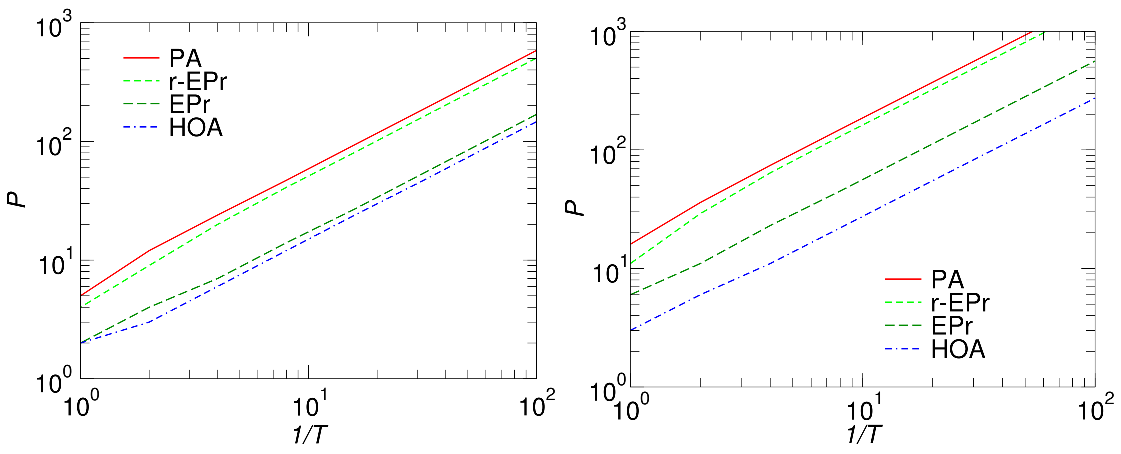

Figure 2.4: Necessary Trotter number P to reach a relative accuracy of 10^{-2}(left) and 10^{-3}(right) in the potential energy $\langle V_P \rangle$ at different inverse temperatures $1/T$ for the linear chain consisting of $N = 5$ atoms. All parameters other than temperature ($\hbar$, k, m) are set to unity.

to small Trotter numbers. In this regime they have advantages over the higher-order method. It has to be taken care that the density is low enough that two particle collisions are the dominating kind of interaction and three and more particles are unlikely to come close together at the same time. The HOA method is optimal if a high accuracy is needed because one can take advantage of the better scaling behavior. In addition to that, the approximation is well controlled by a small parameter, namely $(\beta/P)^2\hbar^2/m$. No such parameter can be identified for the EPr method.

Chapter 3

The hybrid algorithm (QR-HOA)

One of the main tasks of this work is the numerical study of the discrete quantum sine-Gordon model and generalizations of this model with disorder. The results are presented in the last two chapters (5 and 6). For this purpose, a new algorithm for path integral simulations was developed in this work. To achieve better performance than the PA, HOA, ePr and r-EPr methods, three steps are needed:

- The Hamiltonian is split into a harmonic part which is quadratic in the operators $\hat{x}$ and $\hat{p}$ and in a rest part (quadratic-rest decomposition QR).
- The harmonic part is treated exactly by the analytic solution.
- The remaining part is treated with HOA techniques, as explained in chapter 1.

The result is an algorithm which we call hybrid or QR-HOA algorithm. For the discrete sine-Gordon chain, it will be shown that the application of the hybrid algorithm can reduce the computation time dramatically with respect to the PA or HOA algorithm. In addition, the knowledge of the harmonic approximation of a system automatically implies eigen coordinates and dynamical masses for the PIMD simulation. These coordinates and dynamical masses suppress critical slowing down and improve the efficiency of sampling states. In the case of the Frenkel Kontorova model, further optimization of the dynamical masses can scale all time scales of the system to one single time scale.

Possible further applications for the hybrid algorithm are all systems with an interaction which can be separated into a harmonic part and a small anharmonic rest. Examples for such systems are crystals and molecules: For crystals, the harmonic approximation is well known and can be treated analytically. The internal vibrational degrees of freedom of a molecule can be parameterized in terms of normal mode coordinates and treated analytically in a harmonic approximation, while the anharmonic rest is rather small and can be treated with HOA technics.

3.1 Derivation of the hybrid algorithm

In the first part of the derivation, the partition function is rewritten as a discretized path integral including the improvements described above. From this expression, the weight function according to which states have to be sampled and estimators for the calculation of the observables can be found.

3.1.1 The partition function

The main idea of the hybrid algorithm is to express the partition function as a discretized path integral which has already included the exact solution of the harmonic part of the interaction and higher-order corrections for the rest part. The starting point is the N particle Hamiltonian

$$H = \sum_{j=0}^{N-1} \frac{\hat{p}_j^2}{2m} + V\left(\{\hat{x}_j\}\right) \tag{3.1}$$

with a potential V which can be is decomposed into a harmonic part V_{harm} and a rest part V_{I}:

$$V\left(\{\hat{x}_j\}\right) = \sum_{j,j'=0}^{N-1} \frac{1}{2}\hat{x}_j W_{j\,j'}\,\hat{x}_{j'} + V_{\mathrm{I}}\left(\{\hat{x}_j\}\right)\ . \tag{3.2}$$

$W_{j\,j'}$ is a symmetric real matrix which can be diagonalized by an orthogonal transformation A. In terms of the new eigen coordinates $q_k = \sum_j A_{kj}x_j$ of the matrix W, the Hamiltonian reads:

$$\begin{aligned} \hat{H} &= \sum_{k=0}^{N-1} \left(\frac{\hat{p}_k^2}{2m} + \frac{1}{2}K_k\hat{q}_k^2\right) + V_{\mathrm{I}}\left(\{\hat{q}_k\}\right) \\ &=: \sum_{k=0}^{N-1} \frac{\hat{p}_k^2}{2m} + \hat{V}_{\mathrm{harm}} + V_{\mathrm{I}}\left(\{\hat{q}_k\}\right)\ . \end{aligned} \tag{3.3}$$

Here K_k are the eigenvalues of the matrix W. The partition function is given by

$$Z(\beta) = \int \prod_{k=0}^{N-1} dq_k\ \langle\{q_k\}|e^{-\beta\left(\hat{T}+\hat{V}_{\mathrm{harm}}+\hat{V}_{\mathrm{I}}\right)}|\{q_k\}\rangle. \tag{3.4}$$

With the higher-order Trotter formula for the trace given in Eq. (1.19), the partition function can be written as

$$\begin{aligned} Z(\beta) = \int \prod_{t=0}^{P-1}\prod_{k=0}^{N-1} dq_k^t\ &\langle\{q_k\}^0|e^{-\frac{\beta}{2P}\tilde{V}_{\mathrm{I}}}e^{-\frac{\beta}{P}(T+V_{\mathrm{harm}})}e^{-\frac{\beta}{2P}\tilde{V}_{\mathrm{I}}}|\{q_k\}^1\rangle\times \\ &\cdots\times\langle\{q_k\}^{P-1}|e^{-\frac{\beta}{2P}\tilde{V}_{\mathrm{I}}}e^{-\frac{\beta}{P}(T+V_{\mathrm{harm}})}e^{-\frac{\beta}{2P}\tilde{V}_{\mathrm{I}}}|\{q_k\}^0\rangle + \mathcal{O}\left(1/P^4\right)\ , \end{aligned} \tag{3.5}$$

with

$$\begin{aligned} \tilde{V}_{\mathrm{I}} &= V_{\mathrm{I}} + \frac{1}{24}\left(\frac{\beta}{P}\right)^2 [V_{\mathrm{I}},[T+V_{\mathrm{harm}},V_{\mathrm{I}}]] \\ &= V_{\mathrm{I}} + \frac{1}{24}\frac{\hbar^2}{m}\left(\frac{\beta}{P}\right)^2 \sum_{k=0}^{N-1}\left(\frac{d}{dq_k}V_{\mathrm{I}}\right)^2 \\ &= V_{\mathrm{I}} + V_c\ . \end{aligned} \tag{3.6}$$

The high temperature density matrix appearing in Eq. (3.5) can be calculated exactly using the analytical result for the density matrix of a harmonic oscillator [1] The result reads

$$
\begin{aligned}
&\langle \{q_k\}^t | e^{-\frac{\beta}{2P}\tilde{V}_\mathrm{I}} e^{-\frac{\beta}{P}(T+V_\mathrm{harm})} e^{-\frac{\beta}{2P}\tilde{V}_\mathrm{I}} | \{q_k\}^{t+1}\rangle = \\
&= \prod_{k=0}^{N-1} \sqrt{\frac{m}{2\pi\hbar^2\beta/P}\frac{f_k}{\sinh[f_k]}} \\
&\times \exp\left\{-\frac{\beta}{2P}\tilde{V}_\mathrm{I}\left(\{q_k\}^t\right)\right\} \\
&\times \exp\left\{-\frac{\beta}{P}\sum_{k=0}^{N-1}\left[\frac{1}{2}\left(K_k\frac{2\tanh[f_k/2]}{f_k}\right)\frac{(q_k^t)^2+(q_k^{t+1})^2}{2}\right.\right. \\
&\qquad\left.\left. + \frac{m}{2\hbar^2(\beta/P)^2}\frac{f_k}{\sinh[f_k]}\left(q_k^t - q_k^{t+1}\right)^2\right]\right\} \\
&\times \exp\left\{-\frac{\beta}{2P}\tilde{V}_\mathrm{I}\left(\{q_k\}^{t+1}\right)\right\}
\end{aligned}
\tag{3.7}
$$

with $f_k = \hbar(\beta/P)\sqrt{K_k/m}$. Insertion of the high temperature density matrix (Eq. (3.7)) into Eq. (3.5) leads to an expression for the density matrix which includes the exact solution of the harmonic part.

$$
\begin{aligned}
Z(\beta) &= \int \prod_{t=0}^{P-1}\prod_{k=0}^{N-1}\left(\sqrt{\frac{m}{2\pi\hbar^2\beta/P}\frac{f_k}{\sinh[f_k]}}dq_k^t\right) \\
&\exp\left\{-\frac{\beta}{P}\sum_{t=0}^{P-1}\left[\sum_{k=0}^{N-1}\left(\frac{m}{2\hbar^2(\beta/P)^2}\frac{f_k}{\sinh[f_k]}\left(q_k^t - q_k^{t+1}\right)^2\right.\right.\right. \\
&\qquad\left.\left.\left. + \frac{1}{2}K_k\frac{2\tanh[f_k/2]}{f_k}(q_k^t)^2\right) + \tilde{V}_\mathrm{I}\left(\{q_k\}^t\right)\right]\right\}. \\
&+\mathcal{O}(1/P^4)
\end{aligned}
\tag{3.8}
$$

The term $\sum_t (q_k^t - q_k^{t+1})^2$ in Eq. (3.8) can be diagonalized with a normal mode transformation according to the Trotter index t. In terms of the normal coordinates $u_k^s = \sum_t U_{st} q_k^t =$

[1] The density matrix of a harmonic oscillator is given by

$$
\begin{aligned}
\langle q | e^{-\frac{\beta}{P}(\hat{T}+\frac{1}{2}K\hat{q}^2)} | q'\rangle =& \sqrt{\frac{m}{2\pi\hbar^2(\beta/P)}\frac{f}{\sinh(f)}} \\
&\times \exp\left\{-\sqrt{\frac{m}{2\hbar^2(\beta/P)}\frac{f}{\sinh(f)}}\left[\left(q^2+q'^2\right)\left(\cosh(f)-1\right)+\left(q-q'\right)^2\right]\right\}.
\end{aligned}
$$

For a derivation see e.g [Fey72].

$\sum_{j,t} U_{st} A_{kj} x_j^t$, the final result for the diagonalized partition function [2] is

$$\begin{aligned}
Z(\beta) &= \int \prod_{s=0}^{P-1} \prod_{k=0}^{N-1} \left(\sqrt{\frac{m}{2\pi\hbar^2\beta/P} \frac{f_k}{\sinh[f_k]}} du_k^s \right) \\
&\quad \times \exp\left\{ -\frac{\beta}{P} \left[\sum_{s=0}^{P-1} \sum_{k=0}^{N-1} \frac{1}{2} \left[K_{\text{i-time}}^{\text{eff}}(s) + K_{\text{space}}^{\text{eff}}(k) \right] (u_k^s)^2 + \tilde{V}_{\text{I}}(\{u_k^s\}) \right] \right\} \\
&\quad + \mathcal{O}(1/P^4) && (3.9) \\
&= \int \prod_{s=0}^{P-1} \prod_{k=0}^{N-1} \left(\sqrt{\frac{m}{2\pi\hbar^2\beta/P} \frac{f_k}{\sinh[f_k]}} du_k^s \right) && (3.10) \\
&\quad \times \exp\left\{ -\frac{\beta}{P} \left[\sum_{s=0}^{P-1} \sum_{k=0}^{N-1} \frac{1}{2} K_{\text{tot}}^{\text{eff}}(k,s)\, (u_k^s)^2 + \tilde{V}_{\text{I}}(\{u_k^s\}) \right] \right\} \\
&\quad + \mathcal{O}(1/P^4) && (3.11) \\
&= \int \prod_{s=0}^{P-1} \prod_{k=0}^{N-1} \left(\sqrt{\frac{m}{2\pi\hbar^2\beta/P} \frac{f_k}{\sinh[f_k]}} du_k^s \right) \exp\left\{ -\frac{\beta}{P} U_{\text{rp}}^{\text{QV-HOA}}(\{u_k^s\}) \right\} \\
&\quad + \mathcal{O}(1/P^4) && (3.12)
\end{aligned}$$

with the effective spring constants for the space and imaginary-time dimension

$$\begin{aligned}
K_{\text{space}}^{\text{eff}}(k) &= K_k \frac{2\tanh[f_k/2]}{f_k} \\
K_{\text{i-time}}^{\text{eff}}(s) &= \frac{m}{\hbar^2(\beta/P)^2} \frac{f_k}{\sinh[f_k]} 4\sin^2\left(\frac{\pi}{P}s\right) && (3.13) \\
K_{\text{tot}}^{\text{eff}}(k,s) &= K_{\text{i-time}}^{\text{eff}}(s) + K_{\text{space}}^{\text{eff}}(k)\ .
\end{aligned}$$

Note that all functions with the arguments $\{u_k^s\}$ (e.g. V_{I}, V_c, $\tilde{V}_{\text{I}}$, L and $\widetilde{V_{\text{I}} + \lambda L}$) are defined through the corresponding functions with the arguments $\{x_j\}^t$ as

$$V_{\text{I}}(\{u_k^s\}) = \sum_{t=0}^{P-1} V_{\text{I}}\left(\{x_j\}^t\right) \Big|_{x_j^t = x_j^t(\{u_k^s\})} \ . \qquad (3.14)$$

The effective classical potential introduced in Eq. (3.9) which is used in the path integral molecular dynamics (PIMD) simulation to sample states $\{u_k^s\}$ is given by

$$\begin{aligned}
U_{\text{rp}}^{\text{QV-HOA}}(\{u_k^s\}) &= \sum_{s=0}^{P-1} \sum_{k=0}^{N-1} \frac{1}{2} \left[K_{\text{i-time}}^{\text{eff}}(k) + K_{\text{space}}^{\text{eff}}(s) \right] (u_k^s)^2 + \tilde{V}_{\text{I}}(\{u_k^s\}) \\
&= \sum_{s=0}^{P-1} \sum_{k=0}^{N-1} \frac{1}{2} K_{\text{tot}}^{\text{eff}}(k,s)\, (u_k^s)^2 + \tilde{V}_{\text{I}}(\{u_k^s\})\ . && (3.15)
\end{aligned}$$

[2] The decomposition of $K_{\text{tot}}^{\text{eff}}$ into two summands depending on the imaginary-time and the space only is similar to a simple cubic solid with next neighbor interaction and a vanishing shear module. For more complex crystalline structures, e. g. triangular latticed, this decomposition does not hold any more.

3.1.2 Estimators

The estimators for the physical observables differ from the estimators in the primitive algorithm (PA). They can be determined from the corresponding expressions for thermodynamic expectation values.

Energy estimators

The thermodynamic expectation value for the kinetic energy is

$$\begin{aligned}\langle T\rangle &= \frac{m}{\beta}\frac{d}{dm}\ln\left(Z(\beta)\right)\\ &= \frac{P}{4}\sum_{k=0}^{N-1}\left[\frac{f_k}{\tanh[f_k]}+1\right]\frac{1}{\beta}\\ &\quad+\int\prod_{s=0}^{P-1}\prod_{k=0}^{N-1}\left(\sqrt{\frac{m}{2\pi\hbar^2\beta/P}\frac{f_k}{\sinh[f_k]}}du_k^s\right)\\ &\quad\frac{1}{P}\left[\sum_{s=0}^{P-1}\sum_{k=0}^{N-1}\frac{1}{2}\left(-K^{\mathrm{eff}}_{\mathrm{i-time}}+\left[\frac{d}{df_k}K^{\mathrm{eff}}_{\mathrm{tot}}\left(k,s\right)\right]\frac{f_k}{2}\right)\left(u_s^k\right)^2+V_c\left(\{u_k^s\}\right)\right]\\ &\quad\times\exp\left\{-\frac{\beta}{P}U^{\mathrm{QV-HOA}}_{\mathrm{rp}}\left(\{u_k^s\}\right)\right\}\Big/Z(\beta)+\mathcal{O}\left(1/P^4\right)\\ &= \left\langle T^{\mathrm{QV-HOA}}_{\mathrm{est}}\left(\{u_k^s\}\right)\right\rangle_{U^{\mathrm{QV-HOA}}_{\mathrm{rp}}}+\mathcal{O}\left(1/P^4\right)\ .\end{aligned} \tag{3.16}$$

The estimator for the kinetic energy is given by

$$\begin{aligned}T^{\mathrm{QV-HOA}}_{\mathrm{est}}\left(\{u_k^s\}\right) &= \frac{P}{4}\sum_{k=0}^{N-1}\left[\frac{f_k}{\tanh[f_k]}+1\right]\frac{1}{\beta}\\ &\quad+\frac{1}{P}\left[\sum_{s=0}^{P-1}\sum_{k=0}^{N-1}\frac{1}{2}\left(-K^{\mathrm{eff}}_{\mathrm{i-time}}+\left[\frac{d}{df_k}K^{\mathrm{eff}}_{\mathrm{tot}}\left(k,s\right)\right]\frac{f_k}{2}\right)\left(u_s^k\right)^2+V_c\left(\{u_k^s\}\right)\right]\ .\end{aligned} \tag{3.17}$$

The virial estimator can be derived in analogy to the derivation given in appendix B. The result reads:

$$\begin{aligned}T^{\mathrm{QV-HOA(vir)}}_{\mathrm{est}}\left(\{u_k^s\}\right) &= \frac{1}{2}\frac{N}{\beta}+\frac{P}{4}\sum_{k=0}^{N-1}\left[\frac{f_k}{\tanh[f_k]}-1\right]\frac{1}{\beta}\\ &\quad+\frac{1}{2P}\sum_{s=0}^{P-1}\left[\frac{d}{df_k}K^{\mathrm{eff}}_{\mathrm{tot}}\left(k,s\right)\right]\frac{f_k}{2}\left(u_k^s\right)\\ &\quad+\frac{1}{P}\left[\sum_{s=1}^{P-1}\frac{1}{2}u_k^s\left[K^{\mathrm{eff}}_{\mathrm{i-time}}(s)\ u_k^s+\frac{d}{du^s}\tilde{V}_{\mathrm{I}}\left(u^0,u^1,\ldots,u^{P-1}\right)\right]+V_c\left(\{u_k^s\}\right)\right]\ .\end{aligned} \tag{3.18}$$

The fluctuations of this estimator do not diverge with the Trotter number P.

The thermodynamic average of the potential energy can be calculated as

$$\begin{aligned}
&\langle V_{\text{tot}} \rangle = \langle E - T \rangle \\
=\ & \left(-\frac{d}{d\beta} - \frac{m}{\beta}\frac{d}{dm} \right) Z \\
=\ & +\frac{P}{4}\sum_{k=0}^{N-1}\left[\frac{f_k}{\tanh[f_k]} - 1\right]\frac{1}{\beta} \\
& + \int \prod_{s=0}^{P-1}\prod_{k=0}^{N-1}\left(\sqrt{\frac{m}{2\pi\hbar^2\beta/P}\frac{f_k}{\sinh[f_k]}}du_k^s\right)\exp\left\{-\frac{\beta}{P}U_{\text{rp}}^{\text{QV-HOA}}\left(\{u_k^s\}\right)\right\} \\
& \times\frac{1}{P}\left[\sum_{s=0}^{P-1}\sum_{k=0}^{N-1}\frac{1}{2}\left(K^{\text{eff}} + \left[\frac{d}{df_k}K_{\text{tot}}^{\text{eff}}(k,s)\right]\frac{f_k}{2}\right)\left(u_s^k\right)^2 + \tilde{V}_{\text{I}}\left(\{u_k^s\}\right) + 2V_c\left(\{u_k^s\}\right)\right] \\
& \qquad \times\frac{1}{Z(\beta)} + \mathcal{O}\left(1/P^4\right) \\
=\ & \left\langle T_{\text{est}}^{\text{QV-HOA}}\left(\{u_k^s\}\right)\right\rangle_{U_{\text{rp}}^{\text{QV-HOA}}} + \mathcal{O}\left(1/P^4\right) \ . \qquad (3.19)
\end{aligned}$$

The estimator for the potential energy is given by

$$\begin{aligned}
V_{\text{tot est}}^{\text{QV-HOA}}\left(\{u_k^s\}\right) = & \frac{P}{4}\sum_{k=0}^{N-1}\left[\frac{f_k}{\tanh[f_k]} - 1\right]\frac{1}{\beta} \\
+\frac{1}{P}&\left[\sum_{s=0}^{P-1}\sum_{k=0}^{N-1}\frac{1}{2}\left(K^{\text{eff}} + \left[\frac{d}{df_k}K_{\text{tot}}^{\text{eff}}(k,s)\right]\frac{f_k}{2}\right)\left(u_s^k\right)^2 + \tilde{V}_{\text{I}}\left(\{u_k^s\}\right) + 2V_c\left(\{u_k^s\}\right)\right] \ .
\end{aligned} \qquad (3.20)$$

In the expressions for the estimators, the derivative of the effective spring constant appears. It is given by

$$\begin{aligned}
\frac{d}{df}K_{\text{tot}}^{\text{eff}} = \frac{d}{df}\left(K_{\text{i-time}}^{\text{eff}} + K_{\text{space}}^{\text{eff}}\right) = \frac{m}{\hbar^2\left(\beta/P\right)^2}&\left(\frac{1 - f_k\coth[f_k]}{\sinh[f_k]}\right)4\sin^2\left(\frac{\pi}{P}s\right) \\
& + K_k\left(\frac{f_k - \sinh[f_k]}{f_k^2\cosh^2[f_k/2]}\right) \ .
\end{aligned} \qquad (3.21)$$

Estimators for operators diagonal in real space

The estimator for an observable $L\left(\{\hat{q}_k\}\right)$ which is diagonal in real space is the same as that for the higher-order (HOA) method. This can easily be seen if one considers the QR-HOA path integral expression for a generalized partition function which has an additional source term λL

included. It is given by

$$\begin{aligned}
&\operatorname{tr}\exp\left\{-\beta\left(\hat{H}+\lambda\hat{L}\right)\right\}\\
=\;&\int\prod_{s=0}^{P-1}\prod_{k=0}^{N-1}\left(\sqrt{\frac{m}{2\pi\hbar^2\beta/P}\frac{f_k}{\sinh[f_k]}}du_k^s\right)\\
&\exp\left\{-\frac{\beta}{P}\left[\sum_{s=0}^{P-1}\sum_{k=0}^{N-1}\frac{1}{2}K_{\text{tot}}^{\text{eff}}(k,s)\;(u_k^s)^2+(\widetilde{V_\text{I}+\lambda L})\left(\{u_k^s\}\right)\right]\right\}\\
&\mathcal{O}(1/P^4)
\end{aligned}$$

with

$$\begin{aligned}
(\widetilde{V_\text{I}+\lambda L})\left(\{u_k^s\}\right) \;=\; &\sum_{t=0}^{P-1}\Bigg[V_\text{I}\left(\{q_k\}^t\right)+\lambda L\left(\{q_k\}^t\right) \\
&+\frac{1}{24}\frac{\hbar^2}{m}\left(\frac{\beta}{P}\right)^2\sum_{k=0}^{N-1}\left(\frac{d}{dq_k^t}\left(V_\text{I}\left(\{q_k\}^t\right)+\lambda L\left(\{q_k\}^t\right)\right)\right)^2\Bigg]\Bigg|_{q_k^t=q_k^t\left(\{u_k^s\}\right)}\quad.
\end{aligned}\tag{3.22}$$

The thermodynamic expectation value of the operator $\hat{L}$ is the logarithmic derivative

$$\begin{aligned}
&-\frac{1}{\beta}\frac{d}{d\lambda}\ln\operatorname{tr}\exp\left\{-\beta\left(\hat{H}+\lambda\hat{L}\right)\right\}\Bigg|_{\lambda=0}\\
=\;&\int\prod_{s=0}^{P-1}\prod_{k=0}^{N-1}\left(\sqrt{\frac{m}{2\pi\hbar^2\beta/P}\frac{f_k}{\sinh[f_k]}}du_k^s\right)\exp\left\{-\frac{\beta}{P}U_{\text{rp}}^{\text{QV}-\text{HOA}}\left(\{u_k^s\}\right)\right\}\\
&\times\frac{1}{P}\sum_{s=0}^{P-1}\left[L\left(\{u_k\}^s\right)+\frac{1}{24}\frac{\hbar^2}{m}\left(\frac{\beta}{P}\right)^2\sum_{k=0}^{N-1}\left(\frac{d}{du_k^s}V_\text{I}\left(\{u_k\}^s\right)\right)\left(\frac{d}{du_k^s}L\left(\{u_k\}^s\right)\right)\right]\\
&\times\frac{1}{Z(\beta)}+\mathcal{O}\left(1/P^4\right)\;,
\end{aligned}\tag{3.23}$$

which proves that $L_{\text{est}}^{\text{QR}-\text{HOA}}=L_{\text{est}}^{\text{HOA}}$. Note that this allows an expression for the potential energy estimator which is alternative to Eq. (3.20) to be derived.

3.1.3 Limiting cases of the hybrid algorithm

Expressions for the PA, HOA and QR algorithms can be recovered from the expression for the hybrid (QR-HOA) algorithm. If one sets $f_k\to 0$ and $V_\text{I}\to V$ the HOA algorithm is recovered. Skipping of all higher-order terms ($\tilde{V}_I\to V_\text{I}$, $V_c\to 0$ and $L_{\text{est}}^{\text{HOA}}\to L_{\text{est}}^{\text{PA}}$) leads to the equations for the QR algorithm, which includes an exact treatment of the harmonic interaction but scales like $1/P^2$. The limit $f_k\to 0$, $\tilde{V}_\text{I}\to V_\text{I}$, $V_c\to 0$ and $L_{\text{est}}^{\text{HOA}}\to L_{\text{est}}^{\text{PA}}$ leads back to the primitive algorithm (PA).

3.2 Molecular dynamics (MD) implementation

The main task of a path integral simulation program is to sample configurations $\{u_k^s\}$ in such a way that they are distributed according to the weight function

$$W = \exp\left\{-\left(\frac{\beta}{P}\right) U_{\text{rp}}^{\text{QV-HOA}}(\{u_k^s\})\right\} . \tag{3.24}$$

Then, thermodynamic averages of observables can be approximately calculated as averages of their estimators over the sampled configurations as

$$\langle O\rangle = \lim_{P\to\infty} \langle O_{\text{est}}\rangle_{U_{\text{rp}}^{\text{QV-HOA}}} . \tag{3.25}$$

The path integral simulation exploits the fact that classical particles which interact with the potential $U_{\text{rp}}^{\text{QV-HOA}}$ and are coupled to a heat bath with inverse temperature β/P are distributed with the weight W given in Eq. (3.24). In a PIMD simulation, a fictional dynamics for the coordinates u_k^s is used to sample configurations. This dynamic is defined by the equations of motion

$$\begin{aligned} m_k^s \ddot{u}_k^s &= -\nabla_{u_k^s} U_{\text{rp}}^{\text{QV-HOA}}(\{u_k^s\}) + F_{\text{thermostat}} \\ &= -K_{\text{tot}}^{\text{QV-HOA}}(k,s)\, u_k^s - \nabla_{u_k^s}\tilde{V}(\{u_k^s\}) - m_k^s\gamma\dot{u}_k^s + F_{\text{rand}\,k}^s . \end{aligned} \tag{3.26}$$

The dynamical masses m_k^s that appear in the equations of motion do not enter in the averages. In principle, they can be chosen at will. This allows one to adjust the characteristic time scales of the system. In the PIMD simulation, the equations of motion (Eq. (3.26)) are integrated numerically. The finite time step Δt of the numerical integration routine is limited by the smallest time scale of the system t_{min}. On the other hand, the simulation time t_{sim} has to be large compared to the largest time scale of the system t_{max}, otherwise the averages over the slow modes are not accurate. For this reason, it is desirable to choose a set of dynamical masses m_k^s in a way that the ratio $(t_{\text{max}}/t_{\text{min}})$ is as close to unity as possible. The knowledge of the harmonic approximation for the system suggests the choice for the dynamical masses:

$$m_k^s = \frac{1}{\omega_0^2}\left(K_V(k) + K_{\text{tot}}^{\text{eff}}(k,s)\right) . \tag{3.27}$$

Here, ω_0 is the sampling frequency of the simulation and K_V is a correction to the $K_{\text{tot}}^{\text{eff}}$ which reflects the influence of $\tilde{V}$ on the time scales of the coordinates u_k^s. In the optimal case, it is possible to determine the K_V so that all time scales of the system are collapsed. That means that

$$\frac{2\pi}{\omega_0} \approx t_{\text{min}} \approx t_{\text{max}} . \tag{3.28}$$

It is important to note that if all dynamical masses are set to the same value, the time scales belonging to short-wavelength internal modes of one ring polymer diverge as the Trotter number P is increased. This leads to a molecular dynamics time step Δt which has to be reduced proportional to $\Delta t \propto P^{-1/2}$. Because the time scale of the ring polymer's center of mass mode does not depend on the Trotter number, the sampling efficiency is decreased like $t_{\min}/t_{\max} \propto P^{-1/2}$. This problem, known as critical slowing down, is completely eliminated if the dynamical masses are chosen according to Eq. (3.28).

To get the correct distribution W, the kinetic temperature of the molecular dynamics (MD) simulation has to be fixed to be $T = k_{\mathrm{B}} P/\beta$. There are various ways to model the coupling of a system to a heat bath in a molecular dynamics simulation. In this work the so called Langevin thermostat is used. An additional friction term $-m_k^s \gamma \dot{u}_k^s$ and a fluctuating random force $F^s_{\mathrm{rand}\ k}$ are added to the equation of motion. Their size is related to the fluctuation dissipation theorem for the Langevin equation

$$\langle F_{\mathrm{rand}}(t)\rangle = 0 \quad \text{and} \qquad \langle F_{\mathrm{rand}}(t) F_{\mathrm{rand}}(t')\rangle = 2m\gamma k_B T \delta\left(t - t'\right) \ . \tag{3.29}$$

[3] The coupling of the coordinates to the fluctuating force makes the system ergodic. This is especially important for the discrete sine-Gordon chain and related systems which have a negative Lyapunov exponent. The friction constant γ determines the time scale of the thermostat. The relation

$$t_{\mathrm{sim}} \gg \frac{1}{\gamma} \gg \frac{2\pi}{\omega_0} \gg \Delta t \tag{3.32}$$

must hold for the different time scales of the system. Here, Δt is the time step of the integration routine. $2\pi/\omega_0$ is the sampling frequency, $1/\gamma$ the time scale of the thermostat and t_{sim} the simulation time.

3.3 Parallelization of the code

The hybrid path integral MD code developed in this work is parallelized with OpenMP. OpenMP is an open standard for the parallelization of complex algorithms on shared memory machines. The performance is tested on a 4 processor Compaq Alpha ES40 (4 $\times$ 667 MHz) computer, running Compaq Tru64/SC UNIX V5.1 (Rev. 732) as operating system. The Compaq Fortran 90 compiler is used. Fig. 3.1 shows that the performance on the 4-CPU-machine is 3.34 times larger than for the serial code on running on one-CPU. This allowed the calculation of the force velocity relations presented in chapter 5. For more than 4 CPUs the efficiency of the code goes down. The reason is that the creation and destruction of threads leads to a time overhead that becomes more and more relevant compared to the real computation time as the number of threads is increased.

[3] In a simulation the time has to be discretized according to the MD time step in the integration routine. For the discrete dynamics, the delta distribution has to be replaced by a Kronecker delta

$$\delta\left(t - t'\right) = \frac{1}{\Delta t}\delta_{t_i,t_{i'}} \ . \tag{3.30}$$

The fluctuating force F_{rand} can be determined from a random number x which is distributed uniformly in the interval $[0,1]$ as

$$F_{\mathrm{rand}} = \sqrt{6\gamma m_k^s k_B T/\Delta t}\,(2x-1) \ . \tag{3.31}$$

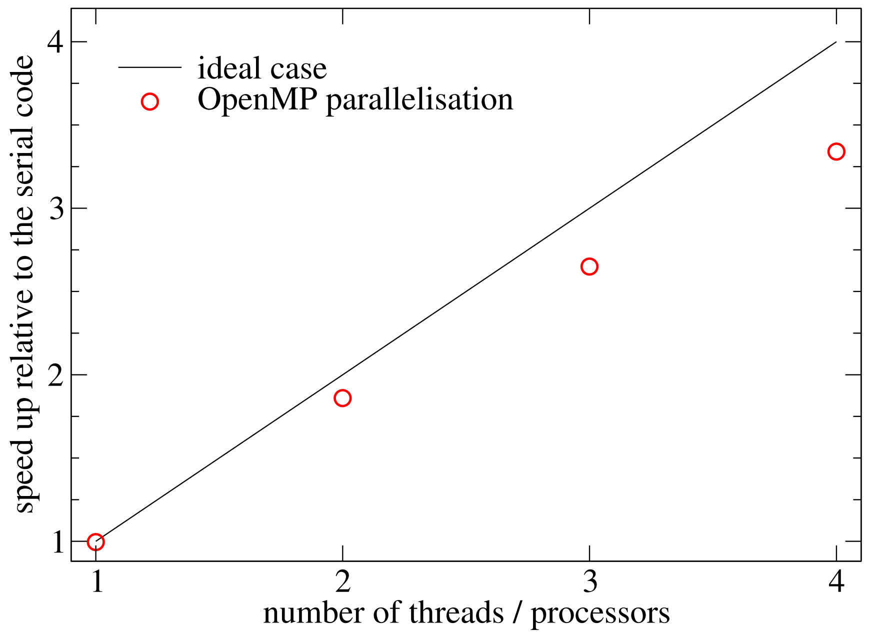

Figure 3.1: Performance scaling with the number of used processors

3.4 Application to the Frenkel Kontorova model

To demonstrate how powerful the hybrid algorithm works, it is applied to the Frenkel Kontorova model, the discrete version of the sine-Gordon equation. The Hamilton operator reads

$$\hat{H} = \sum_{j=0}^{N-1} \left(\underbrace{\frac{\hat{p}_j^2}{2m} + \frac{K}{2}\left(\hat{x}_j - \hat{x}_{j+1}\right)^2}_{Q} - \underbrace{V_0 \cos(\hat{x}_j/b)}_{R} \right) \tag{3.33}$$

with periodic boundary conditions $x_0 \equiv x_N + 2\pi N$. The Hamiltonian can be interpreted as a harmonic chain on a sinusoidal substrate potential. The Trotter breakup is done between the quadratic part of the Hamiltonian Q and the rest R. This leads to the special choice for V, W and K_j, defined in Eq. (3.2)

$$\begin{aligned} V_{\mathrm{I}} &= -V_0 \sum_{j=0}^{N-1} \cos\left(x_j\right) \ , \\ W_{jj'} &= K\left(2\delta_{jj'} - \delta_{j(j'+1)} - \delta_{(j+1)j'}\right) \ , \\ K_k &= 4K \sin^2\left(\frac{\pi}{N}k\right) \ . \end{aligned} \tag{3.34}$$

The harmonic interaction of neighbored particles can be diagonalized with a normal mode transformation (see appendix C). A unit system which is natural for the Frenkel Kontorova model

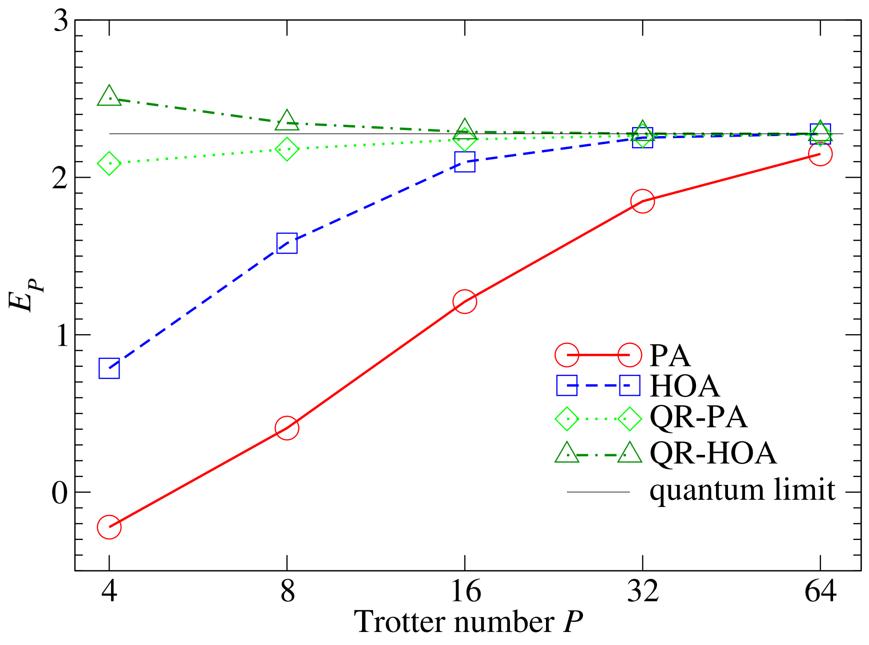

Figure 3.2: Convergence of the total energy of the Frenkel Kontorova model to the quantum limit calculated with different algorithms.

can be defined with V_0 as the energy scale, the length scale b, the Boltzmann constant k_B and the Planck constant $\hbar$. For details see appendix C. In the following, explicit units will be suppressed. In order to analyze the efficiency of the hybrid algorithm, the harmonic coupling is set to $K = 2.0$ and the temperature is $T = 0.2$.

3.4.1 The Trotter convergence

In the following, the Trotter convergence of four different algorithms is analyzed. In the primitive (PA) and the higher-order approximation (HOA) algorithm, the Hamilton operator is split into kinetic and potential energy terms. Higher-order Trotter expansions for the trace are used in the HOA approach. In QR-PA and hybrid (QR-HOA) algorithms, the Hamilton operator is split up into terms quadratic in the momentum and space operators and the rest as explained above. The standard Trotter formula is used in the QR-PA algorithm, while a higher-order Trotter expression for the trace is used for the hybrid algorithm (QR-HOA). The convergence of the total energy to the quantum limit with increasing Trotter number P is shown in Fig. 3.2 for all algorithms. The relative systematic error of the energy due to the finite size of the Trotter number P is shown in Fig. 3.3. The convergence is found to be $1/P^4$ with the HOA and QR-HOA algorithms, while the convergence rate is only $1/P^2$ for the PA and QR algorithms. The exact treatment of the harmonic part of the interaction reduces the systematic error at fixed Trotter number by one order of magnitude for the specific choice of parameters used here. The improvement achieved by the QR technique depends on the accuracy of the harmonic approximation. For large values

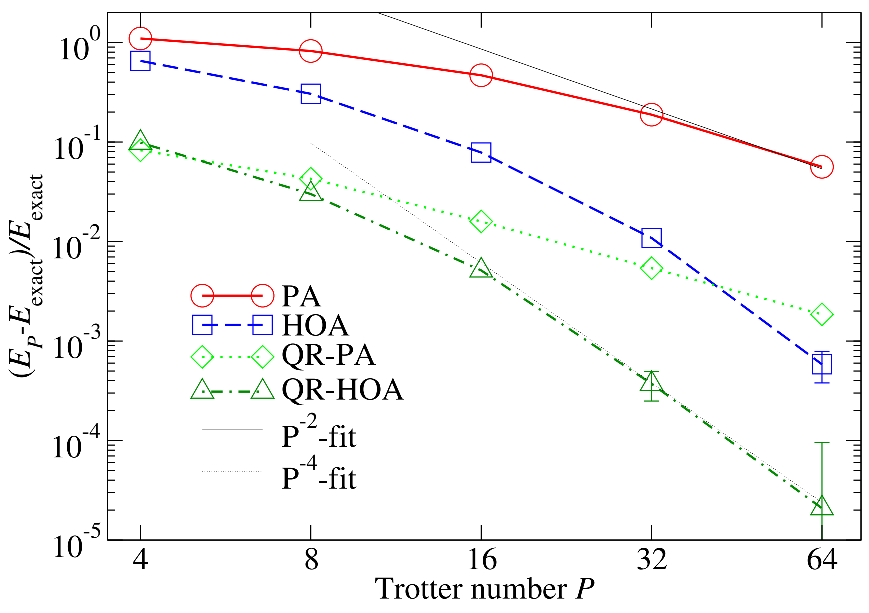

Figure 3.3: Systematic error because of a finite Trotter number

of V_0 it is useful to include the harmonic part of the substrate potential in V_{harm} and choose for the rest $V_{\text{I}} = -V_0 \sum_{j=0}^{N-1} (\cos(x_j/b) - 1/2(x_j/b)^2)$. The result for the energy obtained with the QR and QR-HOA algorithm at $P = 4$ is already better than the PA result with $P = 64$. Using the QR-HOA scheme allows the needed CPU time to be reduced for this specific model system, depending on the required accuracy.

3.4.2 Effective sampling of states with optimized artificial masses

As mentioned above, in order to to sample the phase space of all $N \times P$ coordinates u_k^s, it is important to choose the dynamical masses in a way that the ratio between the smallest and the largest time scale is as close as possible to unity. It is known that the normal mode representation of the closed imaginary-time paths allows to collapse all the internal time scales. This is done by the following choice of dynamical masses

$$m_k^s = m\frac{V_0/b^2 + K_{\text{i-time}}^{\text{eff}}}{V_0/b^2} \tag{3.35}$$

which will be called physical choice for the dynamical masses, because the centroid variables u_k^0 have the physical value m as dynamical mass. V_0^{eff} is an effective strength for the substrate potential that has to be optimized. For the parameters used in this chapter $V_0^{\text{eff}} \equiv V_0$ is a good choice. In the strong quantum regime, V_0^{eff} has to be reduced. The different characteristic time scales of the modes u_k^s are shown in Fig. 3.4. The time scales of the internal modes are all collapsed, and critical slowing down with increasing Trotter number is completely avoided. Still the centroid density of states, shown in Fig. 3.5, is nonzero over a wide frequency spectrum.

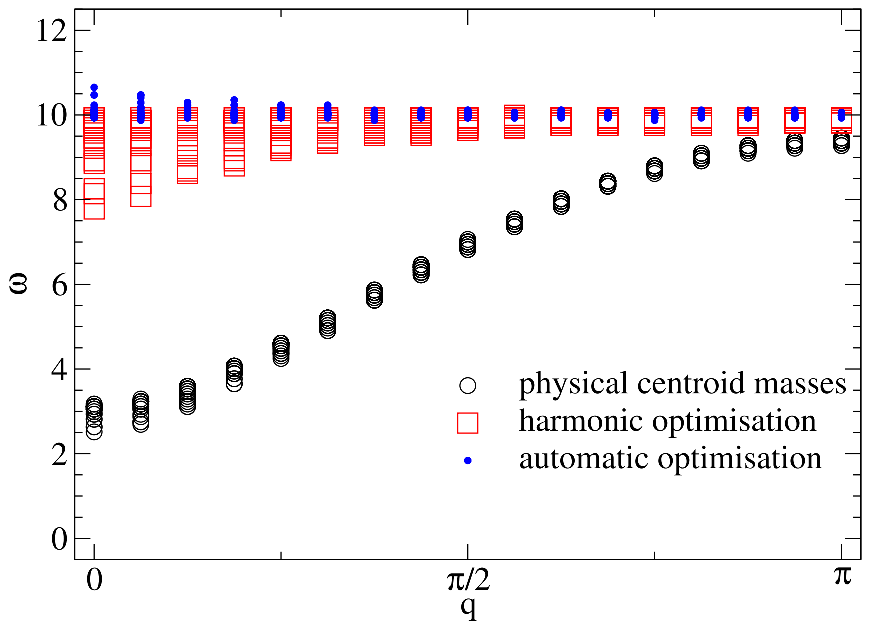

Figure 3.4: Time scales of the coordinates u_k^s for the discrete sine-Gordon chain for three different choices of the dynamical masses. On the x axis the wave vector $q = \frac{k}{N}\frac{\pi}{b}$ is plotted. For each wave vector $q = \frac{k}{N}\frac{\pi}{b}$ the different time scales belonging to the coordinates u_k^s, $s = 0 \ldots (P-1)$ are plotted.

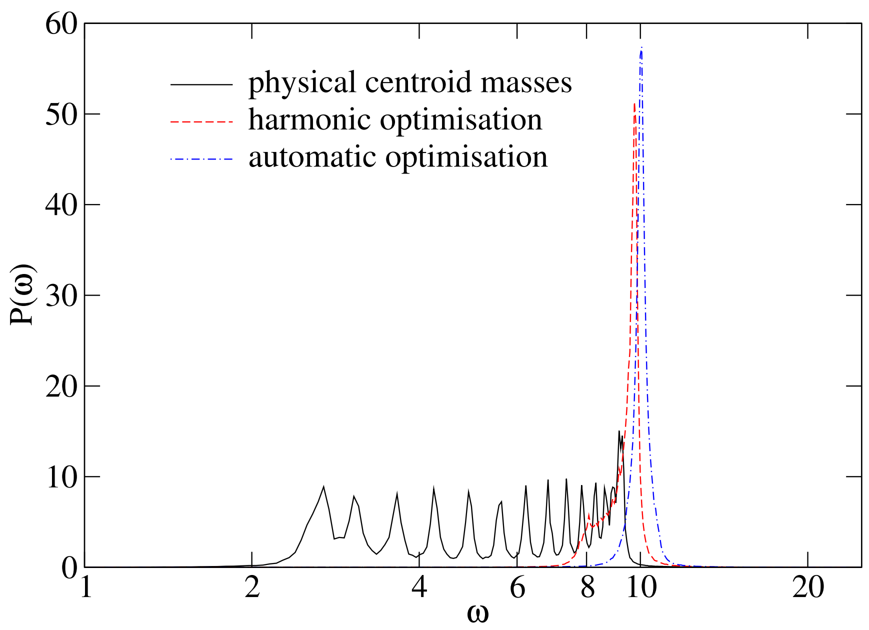

Figure 3.5: Centroid density of states for the discrete sine-Gordon chain. The dotted line shows the physical density of states, calculated with physical values for the dynamical masses belonging to the u_k^0 coordinates (Eq. (3.35)). The dashed line shows the result of the optimization according to Eq. (3.36) and the straight line is the result of an automatic optimization.

Slow centroid modes are still sampled with poor efficiency because the time step is limited by the fastest mode. This problem can be eliminated by a higher sophisticated choice of dynamical masses, which takes advantage of the fact that the simulation is done in coordinates that diagonalize the quadratic part of the Hamilton operator and for which the non-diagonal elements of the complete Hamiltonian are small. The harmonic approximation of the model suggests for the dynamical masses:

$$m_k^s = \frac{1}{\omega_0^2}\left(\frac{V_0^{\text{eff}}}{b^2} + K_{\text{tot}}^{\text{eff}}(k,s)\right) \tag{3.36}$$

as explained in the previous paragraph. The parameter V_0^{eff} is chosen to the classical value V_0 and the typical sampling frequency is set to $\omega_0 = 10$. This very simple optimization reduces the gap between the slowest and fastest time scale by a factor of 4.5 with respect to the physical centroid masses. Fig. 3.5 shows that the typical frequencies of all modes are close to the sampling frequency ω_o. For modes with long wave length (small values of q), the internal modes of the ring polymers differ by a factor of less than 1.25. The centroid spectrum is sharply peaked at the frequency ω_0 but has still a contribution at smaller frequencies.

Further optimization of the masses, which can be done automatically with the data of a test run or during the equilibration, allows to scale all time scales almost perfectly to the value $\frac{2\pi}{\omega_0}$, as it is shown in Fig. 3.4. The centroid density of states given in Fig. 3.5 is sharply peaked around ω_0 if automatic optimization is used. The ratio between the largest and smallest time scale occurring in the PIMD simulation can be reduced by a factor of 8 compared to the standard method which uses the physical masses for the centroid modes. This factor can be much higher if the harmonic coupling K is large and V_o is small.

Chapter 4

The centroid molecular dynamics (CMD) method

Path integral molecular dynamics or path integral Monte Carlo simulations allow one to calculate thermodynamic averages of static quantities such as structural properties or specific heat in the limit $P \to \infty$. Though the imaginary-time correlation functions contain all dynamical information, the inverse Laplace transformation is numerically highly instable, which makes it unfeasible to find the real-time evolution from imaginary-time correlation functions. J. Cao and G.A. Voth introduced a novel and very promising method for the computation of real-time quantum position and velocity correlation functions, which is called centroid molecular dynamics (CMD) [CV93]. This method has been applied to systems such as water [LV97], proton solvated in water [LV96, PCL^{+}97], lithium-para hydrogen clusters [KMK97], liquid para hydrogen [Kin98] and liquid helium [MOK99], to name only a few of its applications.

In the original CMD derivation, it was proven that the centroid trajectories allow the calculation of a well-defined approximation for Kubo transformed time correlation functions correctly up to the the order $\hbar^2$. A more rigorous formulation of centroid molecular dynamics was given by S. Jang and G.A. Voth in 1999 [JV99]. The essential step for this improved derivation was the definition of an operator, called quasi-density operator, whose representation in a position basis corresponds to a fixed centroid path integral. The main result of Jang and Voth was to prove that time correlation functions of the form

$$\Big\langle (\alpha + \beta\hat{x}(t) + \gamma\hat{p}(t))\, \hat{B}(0) \Big\rangle \tag{4.1}$$

can be calculated exactly within CMD. The fundamental quantity in the formulation of centroid molecular dynamics is the geometric center of the closed imaginary-time path

$$x_{\text{cent}}\,[x] = \frac{1}{\hbar\beta} \int_0^{\hbar\beta} d\tau\; x(\tau), \tag{4.2}$$

the so called centroid variable. In the discretized picture, this is simply the center of mass of the classical ring polymer. The centroid density can be defined as the sum over all closed paths with a fixed geometric center, the centroid variable x_c. The centroid density is given by

$$\rho_c(x_c) = \int \mathcal{D}[x]\ \delta\left(x_c - x_{\text{cent}}\left[x\right]\right)\ \exp\left\{\frac{-S_{\text{E}}[x]}{\hbar}\right\}, \tag{4.3}$$

where $S_{\text{E}}[x]$ is the Euclidian-time action functional

$$S_{\text{E}}\left[x\right] = \int_0^{\hbar\beta} d\tau \left(\frac{m}{2}\dot{x}^2(\tau) + V(x(\tau))\right)\ . \tag{4.4}$$

The centroid molecular dynamics method is based on the classical time evolution of the centroid variable on the potential surface defined by the centroid potential

$$V_c(x_c) = -\frac{1}{\beta}\ln\left[\frac{\rho_c(x_c)}{\sqrt{m/(2\pi\hbar^2\beta)}}\right]\ . \tag{4.5}$$

The centroid potential is temperature dependent, for high temperatures $(\beta \longrightarrow 0)$, classical dynamics is recovered because $\lim_{\beta\to 0} V_c(x_c) \sim V(x_c)$. The time evolution of the centroid variable is then defined by the Newton equation of motion

$$m\ddot{x}_c = F_c(x_c) \tag{4.6}$$

with the centroid force [1]

$$\begin{aligned} F_{x_c}(x_c) &= -\frac{d}{dx_c}V_c(x_c) \qquad (4.7)\\ &= \frac{1}{\rho_c(x_c)}\int \mathcal{D}[x]\ \delta\left(x_c - x_{\text{cent}}\left[x\right]\right)\ \frac{1}{\hbar\beta}\int_0^{\hbar\beta} d\tau(-\nabla V)(x(\tau))\exp\left\{\frac{-S_{\text{E}}[x]}{\hbar}\right\}\ . \end{aligned}$$

By integration of the equation of motion (Eq. (4.6)), one can obtain the centroid trajectory $(x_c(t), p_c(t))$, from which correlation functions can be calculated as

$$C_{AB}^{\text{cent}}(t) = \left\langle A\left(x_c(t), p_c(t)\right) B\left(x_c(0), p_c(0)\right)\right\rangle_{\rho_c}\ . \tag{4.8}$$

[1] Because of the special form of the functional $x_{\text{cent}}[x]$, the derivative of the delta distribution according to the number x_c can also be written as a functional derivative according to $x(\tau)$

$$\frac{d}{dx_c}\delta\left(x_c - x_{\text{cent}}\left[x\right]\right) = \frac{\delta}{\delta x(\tau)}\delta\left(x_c - x_{\text{cent}}\left[x\right]\right)\hbar\beta\ .$$

Note that the right hand side of this equation is independent of τ. After a partial integration and the insertion of $1 = \frac{1}{\hbar\beta}\int_0^{\hbar\beta} d\tau$ the, centroid force can be written as

$$-\frac{d}{dx_c}V_c(x_c) \;=\; \frac{1}{\rho_c(x_c)}\int \mathcal{D}[x]\ \delta\left(x_c - x_{\text{cent}}\left[x\right]\right)\frac{1}{\beta}\int_0^{\hbar\beta} d\tau\frac{\delta}{\delta x(\tau)}\exp\left\{\frac{-S[x]}{\hbar}\right\}\ .$$

Computation for the functional derivative yields the result for the centroid force given in Eq. (4.7).

Here $\langle \ldots \rangle_{\rho_c}$ signifies the average over the initial configurations $(x_c(0), p_c(0))$, which are distributed according to the phase space centroid density given in Eq. (4.16). The Kubo transformed time correlation functions of the quantum system

$$C_{AB}^{\mathrm{Kubo}}(t) = \frac{1}{\hbar\beta} \int_0^{\hbar\beta} d\tau \; \langle \hat{A}(t + i\tau)\hat{B}(0) \rangle \tag{4.9}$$

are connected with the centroid correlation functions. The relation

$$C_{AB}^{\mathrm{Kubo}}(\omega) = C_{AB}^{\mathrm{cent}}(\omega) \tag{4.10}$$

between Kubo transformed quantum correlation functions and centroid correlation functions is exact if the operator $\hat{A}$ has the form

$$\hat{A} = \alpha + \beta\hat{x} + \gamma\hat{p} \; . \tag{4.11}$$

For more complicated operators $\hat{A}$, Eq. (4.10) holds up to the order $\hbar^2$. Here $C(\omega)$ is the Fourier transformed real-time correlation function

$$C(\omega) = \frac{1}{2\pi} \int dt \; C(t) e^{-i\omega t} \; . \tag{4.12}$$

For the Fourier transformed time correlation functions of the space and momentum operator, the relations

$$C_{px}(\omega) = i\omega m C_{xx}(\omega) \quad \text{and} \qquad C_{pp}(\omega) = -\omega^2 m^2 C_{xx}(\omega) \tag{4.13}$$

hold. The Fourier transformed time correlation function and the Kubo and Fourier transformed time correlation function are connected by the simple relation

$$C_{AB}^{\mathrm{Kubo}}(\omega) = \hbar\omega\beta/2 \left(\coth\left[\hbar\omega\beta/2\right] + 1\right) C_{AB}(\omega) \; . \tag{4.14}$$

4.1 Adiabatic centroid path integral molecular dynamics (ACPIMD)

The calculation of quantum dynamics with centroid molecular dynamics described above requires the motion of a variable on a quantum mechanical potential of mean force. The adiabatic path integral molecular dynamics method (ACPIMD), capable for generating this dynamics, is formed by the combination of an effective path integral MD method and a classical adiabatic principal [CM96]. The implementation of ACPIMD can be done very similar to a path integral molecular dynamics simulation which uses the normal mode representation of the ring polymers. The modes u_k^0 are proportional to the particles' centroid coordinates $u_k^0 = \sqrt{P} x_{\mathrm{cent}}$. One can choose the dynamical masses in a way that an adiabatic separation of the centroid modes and the ring polymer's internal modes cause the slow centroid modes to move on the potential of mean force of the fast variables. To achieve that, the dynamic masses of the centroid modes u_k^0 have to be small compared to the internal modes' masses. The choice

$$m_k^0 = m \ ; \ \text{and} \qquad m_k^s = ma^2 \frac{K_V + K^{\text{eff}}_{\text{i-time}}(s)}{K_V} \ , \quad s \neq 0 \tag{4.15}$$

collapses the time scales of all internal modes of the ring polymers at a time scale which is a times faster than the one of the centroid. In a simulation a has to be determined empirically: One has to increase a until the dynamics of the centroids becomes invariant under further growth of the time scale gap. For the simulations in this work, $a = 6$ turned out to be sufficient (see Fig. 4.1). The average over the initial conditions in Eq. (4.8) can be done in different ways.

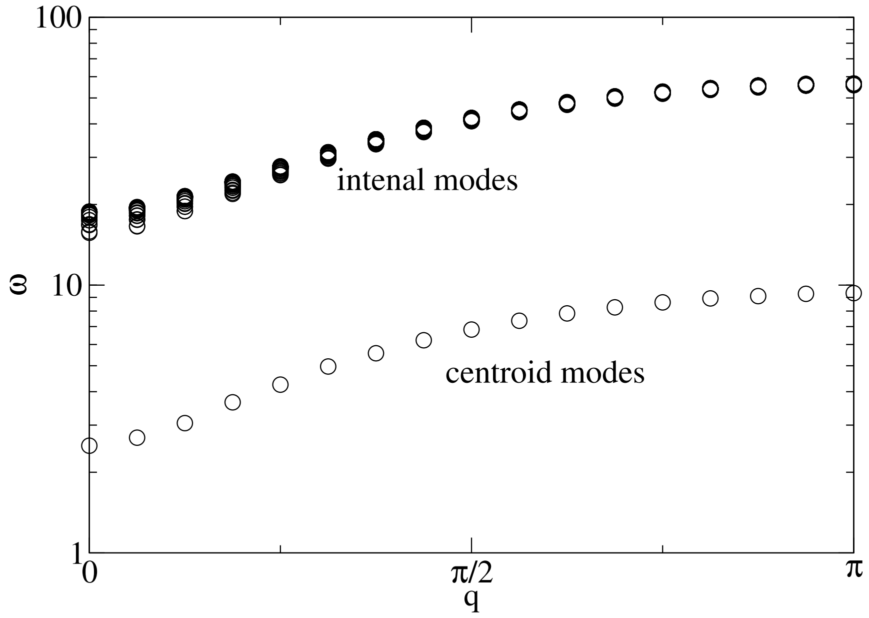

Figure 4.1: Time scales of the centroid and internal modes of the ring polymers. The dynamical masses are adjusted such that the internal modes are six times faster than the centroid modes.

The straight forward method is to use a Monte Carlo algorithm to sample centroid phase space configurations which are distributed according to the centroid density

$$\rho\left(x_c, p_c\right) \propto \rho_c(x_c) \exp\left(-\frac{p_c^2}{2m}\right) \ . \tag{4.16}$$

A more indirect method is used in this work: The centroid variables are coupled to a Langevin thermostat. Because the thermostat may not disturb the observed dynamics, the thermostat's time scales must be large compared to the decay time of the correlation functions. It must be checked that the centroid correlation functions converge as the time scale $1/\gamma$ of the thermostat is increased. This is shown explicitly for the square displacement and the force velocity relation calculated in chapter 5. (see Fig. 5.11 right and Fig 5.15). The equations of motion for the centroid variables coupled to a Langevin thermostat are

$$m\ddot{x}_c = F_{x_c}(x_c) - m\gamma\dot{x}_c - F^{x_c}_{\text{rand}}(t), \tag{4.17}$$

with the delta correlated fluctuating force

$$\langle F^{x_c}_{\text{rand}}(t)F^{x_c}_{\text{rand}}(t')\rangle = 2dm\gamma k_B T\delta(t-t')\ . \tag{4.18}$$

In terms of the normal modes $u^0 = \sqrt{P}x_\text{c}$, with the centroid force on the normal mode coordinates $F_{u_c}(u_c) = \sqrt{P}F_{x_c}(x_c)$, these equations read

$$m\ddot{u}_c = F_{u_c}(u_c) - m\gamma\dot{u}_c - F^{u_c}_{\text{rand}}(t) \tag{4.19}$$

and

$$\langle F^{u_c}_{\text{rand}}(t)F^{u_c}_{\text{rand}}(t')\rangle = 2dm\gamma k_B(TP)\delta(t-t')\ . \tag{4.20}$$

Equation (4.20) shows that the modes u^0 have to be coupled to a heat bath with the temperature TP.

4.2 Static and dynamic properties

In principle, is is possible to calculate static and dynamic properties of a system within the same simulation run. As discussed above, for the calculation of dynamic properties, the dynamical masses have to be chosen according to Eq. (4.15) which makes the calculation of static properties inefficient. Expecially, if one is interested in a high precision measurement of static properties, it is wise to perform different simulation runs for the calculation of static and dynamic properties which use the optimal choice of dynamical masses.

4.3 Non-primitive centroid molecular dynamics methods

The most CPU time consuming task of a CMD simulation is the calculation of the restricted path integral for the effective force given in equation (4.7). For this task all improved algorithms described above can be applied. If HOA techniques are applied, the correct estimator for the centroid force has to be used.

Chapter 5

The discrete quantum sine-Gordon chain (DQSGC)

The discrete sine-Gordon (SG) chain, also known as Frenkel-Kontorova (FK) model [FK38, BK98], is a generic model for the motion of an elastic object composed of discrete degrees of freedom through an external potential. It is given by a discrete, elastic chain which is commensurate with the underlying potential or substrate. The Hamiltonian $\hat{H}$ reads

$$\hat{H} = \sum_{j=0}^{N-1} \left(\frac{\hat{p}_j^2}{2m} + \frac{1}{2} K (\hat{x}_j - \hat{x}_{j+1})^2 - V_0 \cos(\hat{x}_j / b) \right) , \tag{5.1}$$

where $\hat{p}_j$ and $\hat{x}_j$ are the momentum and position of particle j, K is the stiffness of the spring

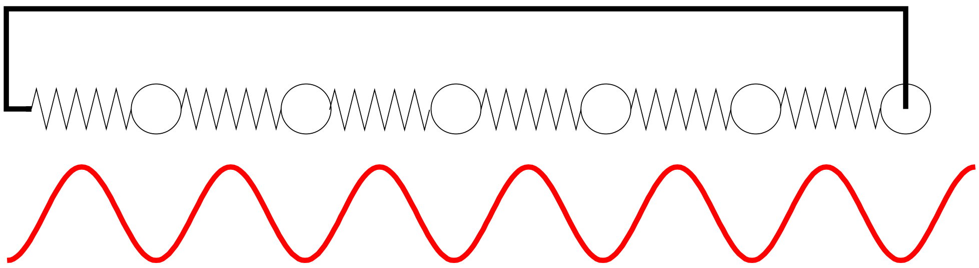
Figure 5.1: Illustration of the Hamiltonian (Eq.5.1) for the discrete sine-Gordon chain

connecting two neighbored particles, V_0 is the coupling strength to the embedding system, and $2\pi b$ is the substrate's lattice constant. The periodic boundary condition $x_{j+N} = x_j + 2\pi b N$ makes the chain commensurate with the substrate. The system of units is defined by V_0, b, $\hbar$, and Boltzmann's constant k_B. The units for the quantities considered are given in appendix C. In the following, the units will not be given explicitly. Unless otherwise noted, the mass m is varied, and the harmonic intra-chain coupling $K = 0.1$ is left constant. This value of K is much smaller than the maximum curvature of the potential $\max(\partial_x^2 V(x)) = 1$ which challenges the validity of continuum approximations that assume slow variations of the reduced positions $x_j - 2\pi jb$ with index j.

To name a few of its applications, the SG model is used for the description of driven charge-density waves (CDW) in solids, coupled Josephson junctions [FM96], the sliding motion of an adsorbed layer of atoms over a substrate [MUR03], and most recently electronic conductance in nanotubes [LT03]; see also the reviews [AA98, SCP00, Zs99] on electronic transport in one-dimensional structures.

A lot of attention has been devoted to the classical FK model both at zero and finite temperatures [BK98, FM96, HMR90] and the continuum quantum-mechanical SG model [Zs99, STF79, STF]. However, less is known about the quantum-mechanical properties of the discrete quantum FK (QFK) variant which concerns in particular its *dynamical* properties. Numerical approaches have lead to a clear picture how quantum fluctuations renormalize the thermal equilibrium structure, but results are often limited to zero (or small) external fields [HL99, TLH02] or to variational approaches [HC01]. While quantum Monte Carlo (QMC) simulations yield quasi-exact results for static properties, they only allow the indirect calculation of small-frequency dynamical properties. Conclusions on the existence of a phonon gap are drawn by studying the temperature dependence of the internal energy [BGS89, BB94]. In some cases, more dynamical information can be withdrawn from QMC if the functional form of the low-energy spectrum is known [AG02].

It is well established for the continuous SG model that the effects due to thermal and those due to quantum fluctuations differ qualitatively. Thermal fluctuations automatically lead to a creep motion if an external force is applied, while quantum fluctuations do not. At finite temperature, kink/anti-kink pairs will be activated and a small external force will eventually be able to drive the pairs apart, resulting in net mass transport. In order for the 1-d quantum sine-Gordon model to creep, it is not sufficient to have arbitrarily small quantum fluctuations, but the effective masses m (defined as density times period of substrate potential) must be less than a certain critical value m_c (at fixed momentum cut-off and fixed substrate strength) [STF79, STF]. For $m > m_c$, thermal fluctuations and/or finite external forces are required to initiate mass transport. Recent renormalization group (RG) studies [GFB02] suggest that m_c is also finite in higher dimensions if the elastic manifold is pinned through an external random potential.

In many of the above mentioned cases, including the quantum SG model, it is necessary to go beyond standard RG theories [CGD98, GFB02, Keh99] because otherwise the effect of creep or alternatively the effect of a vanishing excitation gap at zero wave vectors might be artificially removed. It is thus desirable to have numerical techniques which allow one to (i) verify the results of RG studies, (ii) obtain results beyond continuum approximations allowing an accurate determination of critical values such as m_c for discrete systems, and (iii) obtain dynamical responses of the system for arbitrary external forces. In this chapter, it is shown that adiabatic centroid path integral molecular dynamics (ACPIMD) [TBMK93, CM96], is a well-suited technique to tackle the many-body, non-equilibrium/far-from-equilibrium dynamics of elastic manifolds moving through embedding systems. The study includes the exact calculation of the phonon dispersion relation for $T = 0$ and an analysis of the dynamics at the presence of an external field in both, the linear response regime and the regime far from equilibrium.

5.1 The dispersion relation for the discrete sine-Gordon chain

The phonon dispersion relation of the discrete sine-Gordon chain is a quantity of particular interest. To the authors knowledge, it has not yet been calculated exactly. For the calculation, the ACPIMD method is used, which is a special variant of the centroid dynamics method explained in detail in chapter 4. Classically, the zero-temperature dispersion relation of the chain is simply given by

$$\omega^2(q) = \frac{1}{m}\left[K_0 + 4K\sin^2(qb/2)\right], \tag{5.2}$$

where q denotes the phonon's wave number and $K_0 = 1$. Eq. (5.2) can be found by a simple harmonic approximation of the sin potential around the minima. The quantum dispersion relation

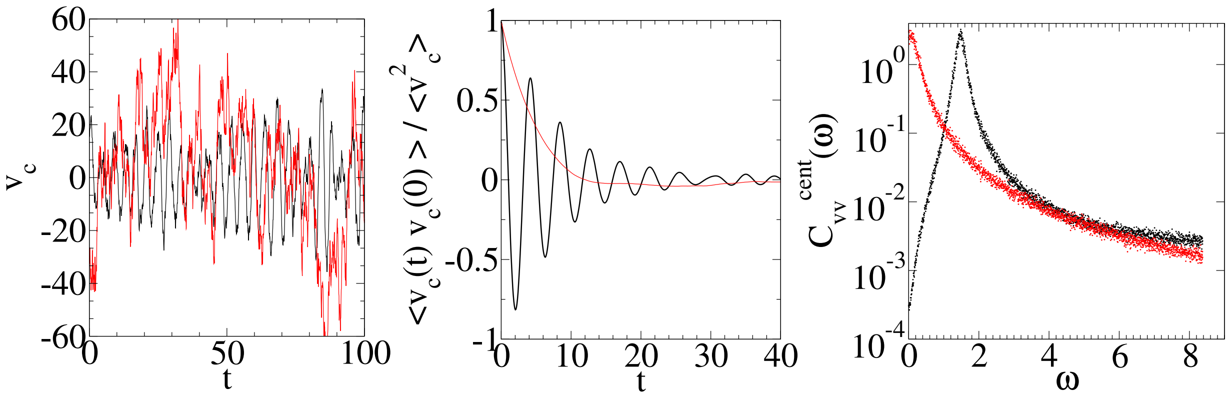

Figure 5.2: Steps in the calculation of the dispersion relation. Left: time dependence of the centroid velocity v_c. Middle: centroid velocity autocorrelation function. Right: Fourier transformed autocorrelation function. The data correspond to the zero wave vector mode for a system with $m = 0.1$ (black curves) and $m = 0.02$ (red curves). The system size is $N = 128$ and the temperature is set to $T = 0.032$. At this temperature the system is dominated by the ground state wave function (compare Fig. 5.3)

can be found from the centroid velocity autocorrelation function. Fig. 5.2 illustrates the three steps in the calculation. First the centroid trajectory is observed during the simulation. From the data $v_c(t)$, the centroid velocity autocorrelation function $\langle v_c(t)v_c(0)\rangle$ is calculated. This is done for every normal mode separately. The Fourier transformed correlation functions are defined as:

$$C_{vv}\left(\omega, q = \frac{2\pi k}{Nb}\right) = \frac{1}{2\pi}\int dte^{i\omega t}\frac{\langle v_c^k(t)v_c^k(0)\rangle}{\langle v^2\rangle}. \tag{5.3}$$

Here k is the index that labels the normal modes u_k^0 and their velocities v_c^k, and $q = 2\pi k/Nb$ is the corresponding phonon wave vector. The correlation function $C_{vv}(\omega, q)$ has the shape of a Lorenz profile for ω values larger or of the order of the value at which $C_{vv}(\omega, q)$ has it's maximum $\omega_{\max}(q)$. For $\omega \ll \omega_{\max}(q)$ the correlation functions $C_{vv}(\omega, q)$ decay exponentially with the exception of the zero wave vector mode in the gapless phase. The frequency $\omega_{\max}(q)$ at which the correlation function $C_{vv}(\omega, q)$ becomes maximal is shown in the dispersion relations for different

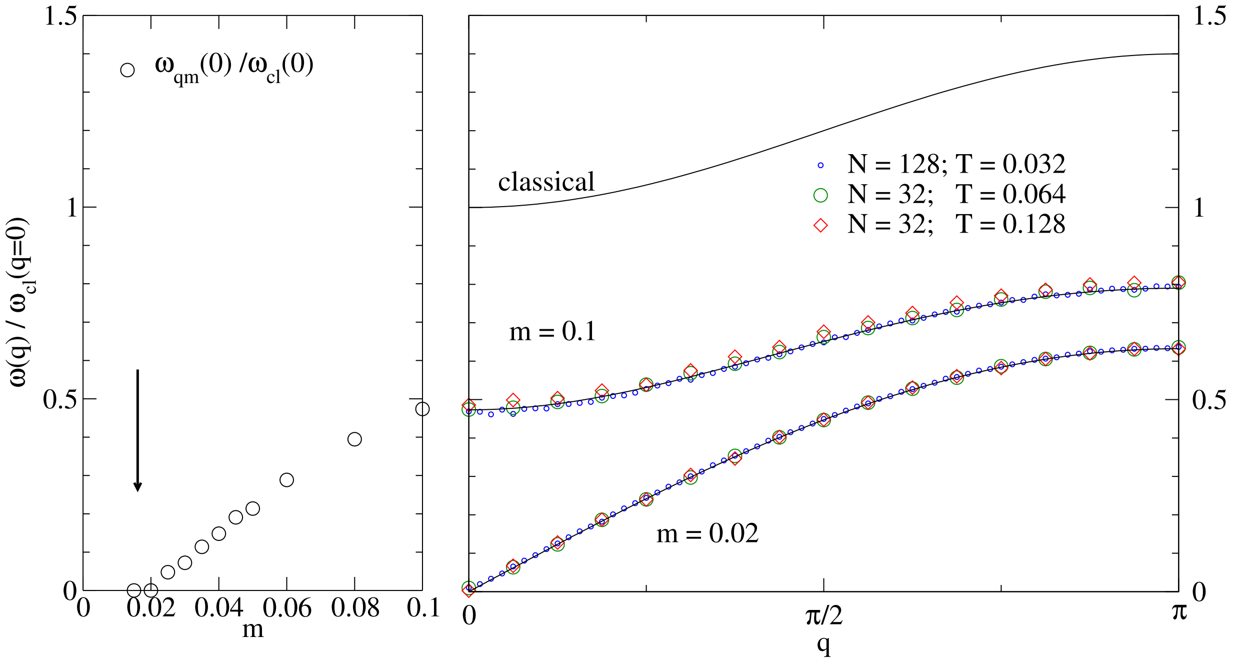

Figure 5.3: Left: Phonon excitation gap at $q = 0$ as a function of the particle mass m. The arrow indicates the value of the critical mass in the continuum limit. Right: Centroid dispersion relations calculated with ACPIMD for two systems with mass $m = 0.1$ and $m = 0.02$. The classical zero-temperature dispersion relation is shown for comparison. Variation of the particle number N and the inverse temperature β (with sufficiently small but fixed (β/P)) allows one to conclude that the curves above represent the thermodynamic and zero-temperature limit.

masses (see Fig. 5.3 right). The right-hand side of Fig. 5.3 shows the phonon dispersion relation for two different masses. The classical zero-temperature dispersion relation which corresponds to the $m \to \infty$ limit of the quantum dispersion relation is plotted for comparison.

The dispersion relation does not change when particle number N and inverse temperature β (sufficiently small but fixed β/P) are increased. This allows one to conclude that the dispersion relations given in Fig. 5.3 represent the thermodynamic and zero-temperature limit within error bars smaller than the used symbols. The quantum dispersion relations can be fitted with a function similar to the classical result:

$$\omega^2(q) = \frac{1}{m}\left[\tilde{K}_0 + 4K\sin^2(qb/2)\right] . \tag{5.4}$$

Each curve requires only one fit parameter, namely the value for $\tilde{K}_0$. The left-hand side of Fig. 5.3 shows the phonon excitation gap at zero wave vector. The gap size is determined from the fit parameter $\tilde{K}_0$ in Eq. (5.4) which is fitted to the data for the dispersion relation. Early calculations [TT85, MT79] suggested that the classical zero-temperature value for K_0 is renormalized to a reduced effective coupling $\tilde{K}_0$ due to to zero-point quantum fluctuations and/or thermal fluctuations. The phonon excitation gap apparently becomes zero at a value $m_c \approx 0.02$, as shown in the left-hand side of Fig. 5.3. This in turn implies that sliding can be induced at arbitrarily small external driving forces for $m \leq 0.02$. As will be shown later, the system creeps in the zero-gap regime when subjected to a small external driving force. The discreteness of the chain alters the value of the mass m_c at which the transition from finite gap (no creep) to zero gap (creep) takes place. The continuum model predicts this transition to occur at $m_c = 0.016$ [Col75]. This is in close agreement with the value of $m_c = 0.02$ obtained for the discrete model in this work, which is valid for the choice of $K = 0.1$. The discrepancy between continuum and discrete model will decrease further as the spring stiffness within the chain increases as compared to the maximum curvature $K_0 = 1$ of the embedding potential.

5.2 Centroid and imaginary-time dynamics

Standard path integral simulations make it possible to calculate the imaginary-time evolution along closed paths in imaginary-time. It is almost impossible to determine real-time evolution properties from the imaginary-time correlation function because the inverse Laplace transformation which maps imaginary-time correlation functions to the real-time correlation functions is numerical highly unstable [Cep95]. This is a reflection of the sign problem which appears in the numerical treatment of the real-time evolution operator $U(t) = e^{-it\hat{H}/\hbar}$, while the imaginary-time evolution operator $e^{\tau\hat{H}/\hbar}$ is numerically unproblematic. To check the results obtained from centroid dynamics for the real-time correlation functions, one can calculate imaginary-time properties from the real-time properties obtained with ACPIMD and compare them with the imaginary-time evolution obtained in a PIMD simulation. As a test quantity for the results obtained with centroid dynamics, the imaginary-time mean square displacement $[\Delta x(\tau)]^2$ is considered. It is given by

$$[\Delta x(\tau)]^2 = \langle (\hat{x}(0) - \hat{x}(\tau))^2 \rangle = \frac{\operatorname{tr}\left(e^{-\beta\hat{H}} \left(\hat{x} - e^{-\tau\hat{H}/\hbar}\hat{x}e^{\tau\hat{H}/\hbar} \right)^2 \right)}{\operatorname{tr} e^{-\beta\hat{H}}} . \tag{5.5}$$

Fig. 5.4 shows the imaginary-time mean square displacement calculated with a PIMD simulation

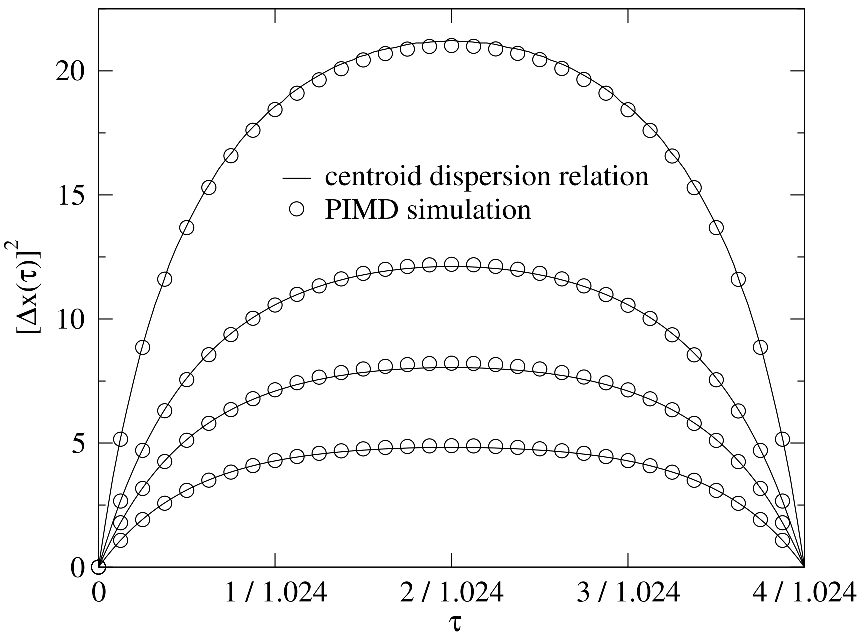

Figure 5.4: Mean square displacements in imaginary-time calculated with a PIMD simulation and from the dispersion relation obtained with a ACPIMD. The particle masses are set to $m = 0.02,\ 0.04,\ 0.06,\ 0.1$ (from above)

and from the dispersion relation obtained with ACPIMD. This confirms that ACPIMD allows one to calculate real-time quantum dynamics for the quantum sine-Gordon chain. The imaginary-time mean square displacement is calculated from the centroid dispersion relation as follows: One considers a harmonic chain with each particle trapped in a harmonic oscillator:

$$\begin{aligned} \hat{H} &= \sum_{j=0}^{N-1} \left(\frac{\hat{p}_j^2}{2m} + \frac{1}{2}K\left(\hat{x}_j - \hat{x}_{j+1}\right) + \frac{1}{2}\tilde{K}_0\left(\hat{x}_j - 2\pi b j\right)^2 \right) \\ &= \sum_{k=0}^{N-1} \left(\frac{\hat{p}_k^2}{2m} + \frac{1}{2}m\omega_k^2\hat{q}_k^2 \right) + \text{const} \end{aligned} \tag{5.6}$$

with the dispersion relation

$$\omega_k^2 = \frac{1}{m}\left(\tilde{K}_0 + 4K\sin^2\left(\frac{\pi}{N}k\right) \right) . \tag{5.7}$$

Here $\hat{q}_k$ and $\hat{p}_k$ are the space and the momentum operators of the normal modes defined in appendix A. The effective strength of the external oscillator $\tilde{K}_0$ is given by the fit parameter for the centroid dispersion relations. For a simple harmonic oscillator the imaginary-time correlation function is given by [1]

$$[\Delta x_\omega(\tau)]^2 = \frac{\hbar}{m\omega}\left(\coth\left(\hbar\omega\beta/2\right) - \frac{\cosh\left[\hbar\omega\left(\tau/\hbar - \beta/2\right)\right]}{\sinh\left[\hbar\omega\beta/2\right]}\right) . \tag{5.10}$$

The imaginary-time correlation function for the harmonic chain $[\Delta x_{\mathrm{harm}}(\tau)]^2$ is the sum over all normal modes correlation functions $[\Delta x_{\omega_k}(\tau)]^2$

$$[\Delta x_{\mathrm{harm}}(\tau)]^2 = \frac{1}{N}\sum_{k=0}^{N-1}[\Delta x_{\omega_k}(\tau)]^2 \tag{5.11}$$

with the frequencies ω_k given by the dispersion relation Eq. (5.7).

5.3 Correlation functions

The phase transition from the phase with a gap to the gapless phase can also be characterized in terms of the imaginary-time correlation function given by

$$G(\Delta n, \Delta\tau) = \langle\{x_{j+\Delta n}(\tau+\Delta\tau) - x_j(\tau) - 2\pi b\Delta n\}^2\rangle . \tag{5.12}$$

For $m > m_c$, $G(\Delta n, \Delta\tau)$ remains finite in the limits $\Delta n \to \infty$ and $\Delta\tau \to \infty$ (see. Fig. 5.5). $G(\Delta n, \Delta\tau)$ increases logarithmically with $\Delta\tau$ and Δn for $m < m_c$. An accurate determination of the critical mass m_c would yet remain much more difficult in terms of an imaginary-time analysis as compared to the one based on centroid dynamics.

5.4 Particle delocalization and tunneling

In the previous paragraphs, it was shown that the ground state of the quantum sine-Gordon chain undergoes a phase transition from a phase with a gap to a gapless phase as the mass is decreased

[1] The imaginary-time correlation function of an harmonic oscillator can be calculated as

$$\begin{aligned}\langle\hat{x}(0)\hat{x}(\tau)\rangle &= \frac{\mathrm{Tr}\left(e^{-\beta H}\hat{x}e^{\tau H/\hbar}\hat{x}e^{-\tau H/\hbar}\right)}{\mathrm{Tr}e^{-\beta H}} = \frac{\sum_{nn'} e^{-(\beta+\tau/\hbar)E_n}e^{\tau E_{n'}}|\langle n|\hat{x}|n'\rangle|^2}{\sum_n e^{-\beta E_n}} \\ &= \frac{\hbar}{2m\omega}\frac{\sum_n n e^{-\beta\hbar\omega n}}{\sum_n e^{-\beta\hbar\omega(n+1/2)}}\cosh\left[\hbar\omega\left(\tau/\hbar-\beta/2\right)\right] = \frac{\hbar}{2m\omega}\frac{\cosh\left[\hbar\omega\left(\tau/\hbar-\beta/2\right)\right]}{\sinh\left[\hbar\omega\beta/2\right]}\end{aligned} \tag{5.8}$$

with matrix elements of the space operator given by

$$\langle n|\hat{x}|n'\rangle = \sqrt{\frac{\hbar}{2m\omega}}\left(\delta_{n,\,n'-1}\sqrt{n'} + \delta_{n,\,n'+1}\sqrt{n'+1}\right) . \tag{5.9}$$

From $\langle\hat{x}(0)\hat{x}(\tau)\rangle$, the imaginary-time mean square displacement of a harmonic oscillator $[\Delta x_\omega(\tau)]^2$ follows.

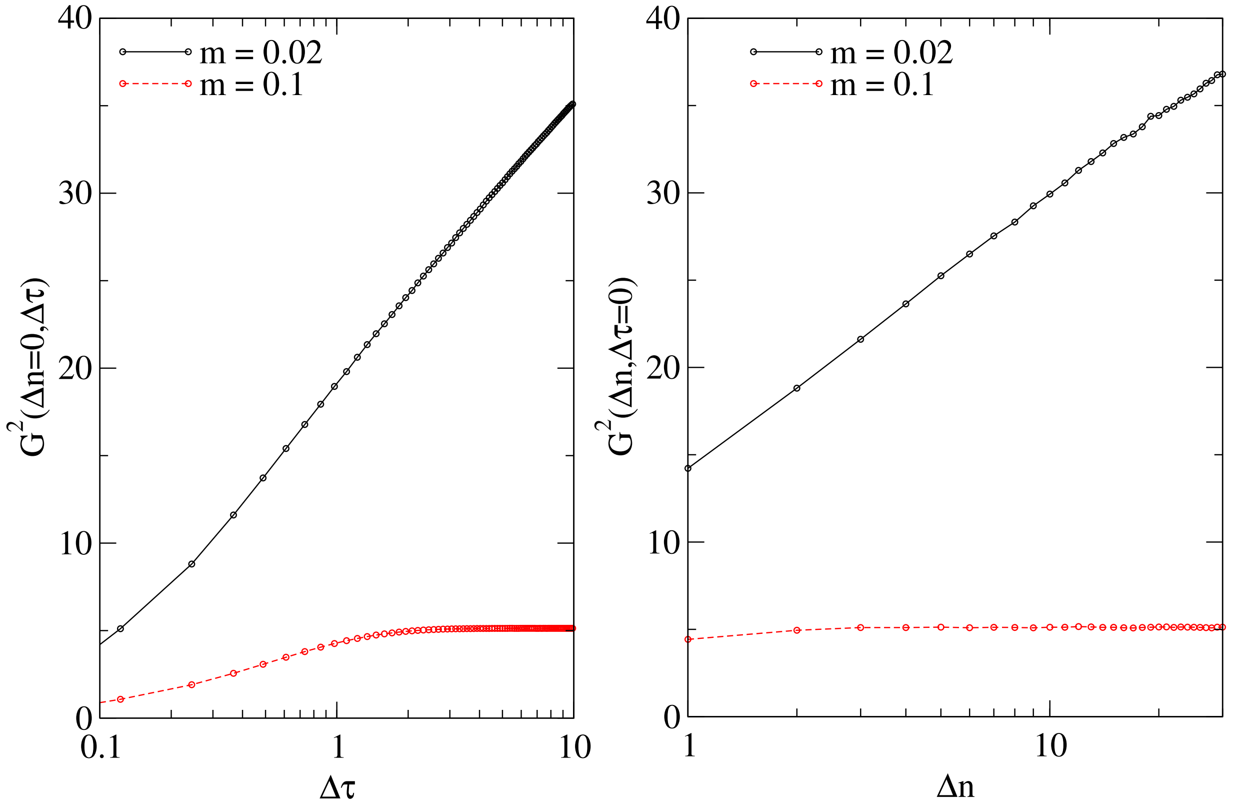

Figure 5.5: Imaginary-time correlation function. Calculated with $T = 0.016$ and $N = 128$.

below a critical value m_c. The different structures of the two ground states in the two phases were characterized in terms of the phonon dispersion relation (Fig. 5.3) and the imaginary-time correlation function $G(\Delta n, \Delta\tau)$ (Fig. 5.6). For a more physical understanding of the processes that lead to this behavior, the gyration radius of the imaginary-time path and the imaginary-time minima correlation function are analyzed in this paragraph. It will be shown that tunneling in imaginary-time is the process that drives the phase transition and that the importance of imaginary-time tunneling with decreasing temperature is qualitatively different in the two phases.

The gyration radius R_g can be considered as a measure for the delocalization of a particle. It is defined as

$$R_g(T) \equiv [\Delta x(\hbar\beta/2)]^2 = \left\langle (x(0) - x(\hbar\beta/2))^2 \right\rangle = \frac{\mathrm{tr}\left(e^{-\beta\hat{H}}\left(\hat{x} - e^{-\beta\hat{H}/2}\hat{x}e^{\beta\hat{H}/2}\right)^2\right)}{\mathrm{tr}\ e^{-\beta\hat{H}}} \tag{5.13}$$

with $\beta = 1/(k_\mathrm{B}T)$. The temperature dependence of the gyration radius R_g is shown in Fig. 5.6 for particle masses $m = 0.02, 0.04, 0.06$ and 0.1. In addition, the corresponding gyration radii for ensembles of harmonic oscillators with the frequencies given by the centroid spectra (Fig. 5.3) are shown. The gyration radii for the ensemble of harmonic oscillators can be calculated as explained in paragraph (5.2) if one takes into account that $R_g(T) \equiv [\Delta x(\hbar\beta/2)]^2$. The agreement between

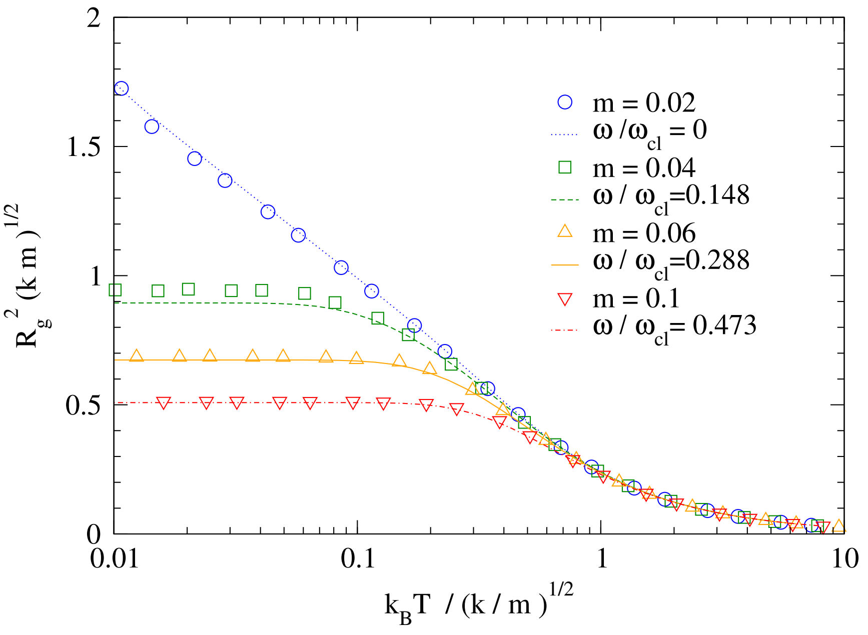

Figure 5.6: Temperature dependence of the gyration radius. The symbols are the results from a PIMD simulation and the lines correspond to harmonic ensembles given by the centroid dispersion relations.

gyration radii calculated from the centroid dispersion relations and the results stemming from the PIMD simulation is very good in the weak quantum region at masses $m \geq 0.06$ and in the unpinned phase for $m \leq m_c = 0.02$. In the region close above the critical mass, the consistence of ACPIMD and PIMD results is less accurate but still good. One reason for this might be that in this region one has to use very large system sizes to approximate the thermodynamic limit well because the energy barriers stemming from the small but finite excitation gap at zero phonon wave vector become very small. In the phase with the gap where $\tilde{K}_0 > 0$, the gyration radius R_g levels at a finite value as the temperature is decreased, while in the gapless phase with $\tilde{K}_0 = 0$ the gyration radius diverges logarithmically with $\beta = 1/k_\mathrm{B}T$. This means that in the gapless phase the delocalization of a particle becomes increasingly important with decreasing temperature, while a particle's delocalization remains finite in the phase with the gap as the temperature goes to zero.

Another quantity which can characterize the structure of the different discrete quantum sine-Gordon chain's ground states can be found if one observes the importance of the tunneling process in imaginary-time. Fig. 5.7 shows typical imaginary-time trajectories at different masses. One for $m = 0.02$ in the gapless region and the other for $m = 0.1$ in the phase with a gap. While the particles are mainly located at one minimum in the phase with the gap, particles are delocalized over various minima in the gapless phase.

For a more qualitative understanding of the importance of the tunneling process, the imaginary-time minima correlation function $\langle s(0)s(\tau)\rangle$ is considered. Here the function $s(x)$ can be either 1 or (-1) and it changes its sign between neighbored minima as illustrated in Fig. 5.8. The be-

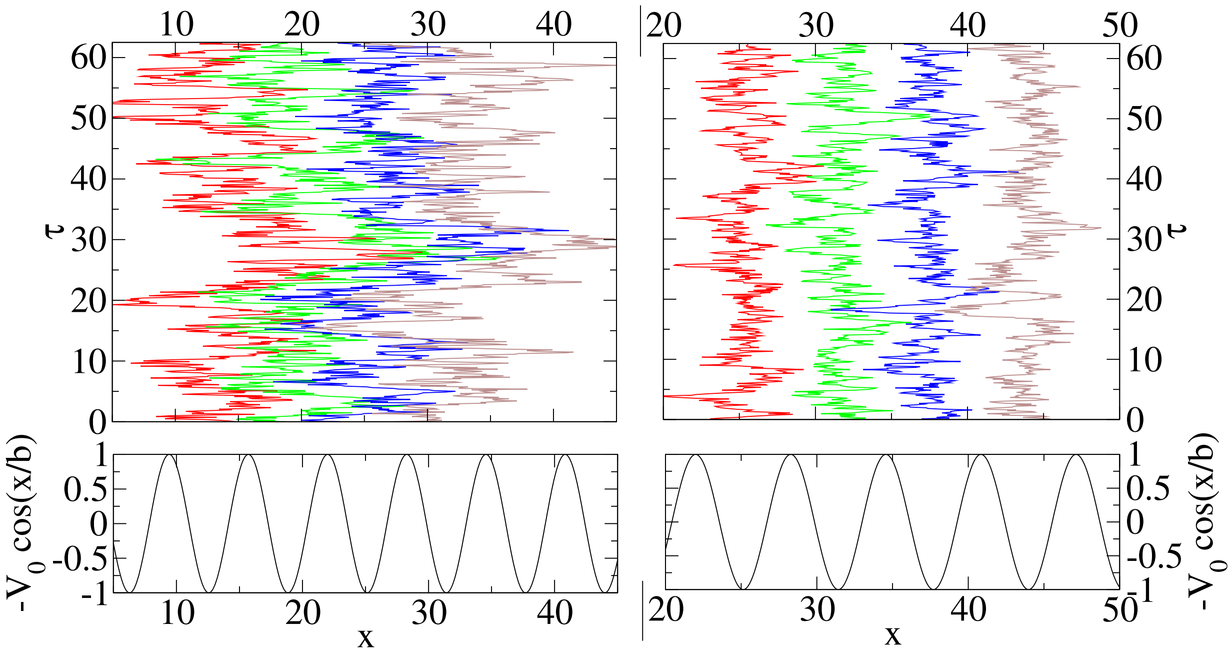

Figure 5.7: Above: Imaginary-time trajectories of typical configurations, left: m = 0.02, right m = 0.1, calculated with $T = 0.016$, $N = 128$ and $P = 512$. Below: substrate potential

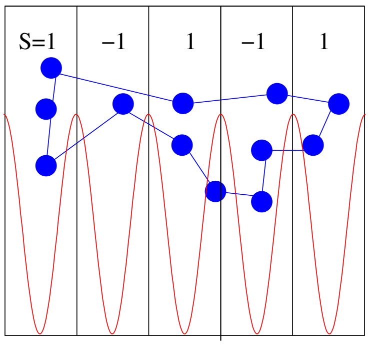

Figure 5.8: Illustration of the alternating sign function $s(X)$ which has opposite signs for neighbored minima.

havior of the correlation function $\langle s(0)s(\tau)\rangle$ is a measure for the importance of tunneling along the imaginary-time axis. For $\tau = 0$, the minima correlation function is always unity by construction. If tunneling never occurs, $\langle s(0)s(\tau)\rangle$ would remain constantly 1 along the imaginary-time axis. Fig. 5.9 shows the imaginary-time minima correlation functions for different temperatures and particle masses $m = 0.1$ and 0.02. The system size is chosen such that finite size effects are smaller then the line thickness. For the higher temperatures a system size of $N = 64$ particles turned out to be sufficient while below $T = 0.032$ $N = 128$ was used. As shown in Fig. 5.10, $\langle s(0)s(\tau)\rangle$ decays exponentially to finite value in the phase with the gap $(m = 0.1)$. The minima

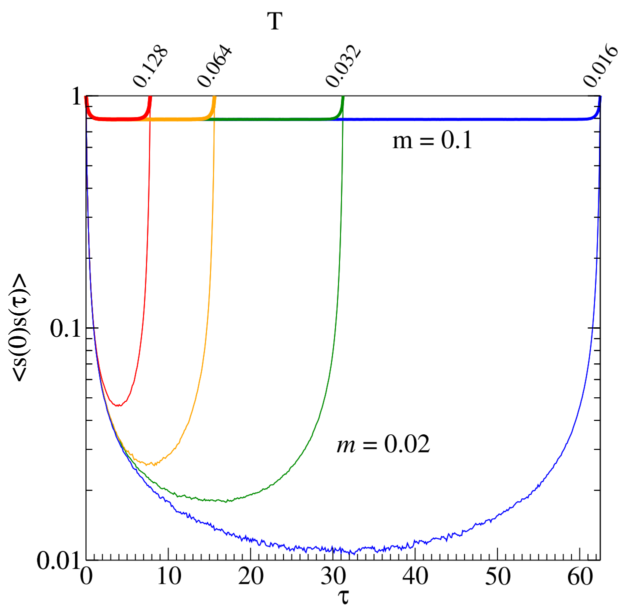

Figure 5.9: The imaginary-time minima correlation function in the phase with and without gap. In the phase with a gap $(m = 0.1)$ the correlation function levels to a finite value as the temperature is decreased, while in the gapless phase the correlation function decays to zero.

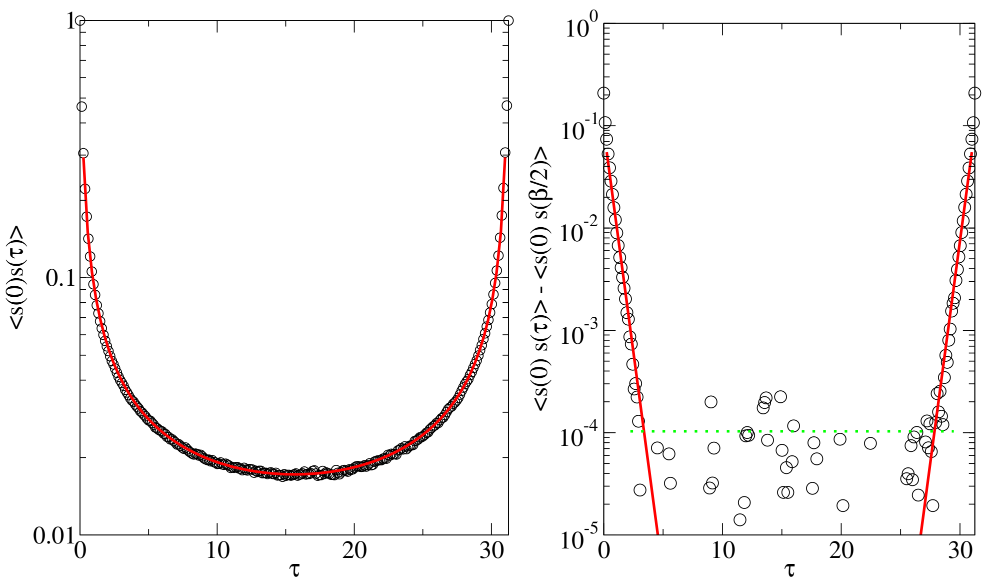

Figure 5.10: Left: algebraic fit of the minima correlation function for $m = 0.02$ with $\kappa = 0.085$ according to Eq. (5.15). Right: exponential fit of the minima correlation function for $m = 0.1$ with $\tau_0 = 2.0$ according to Eq. (5.14). The dotted line indicates the size of the statistical noise.

correlation function can be fitted well with the functional form:

$$\langle s(0)s(\tau)\rangle \propto C + \left\{ \exp\left(\frac{-\tau}{\tau_0}\right) + \exp\left(\frac{+\tau - \hbar\beta}{\tau_0}\right) \right\} . \tag{5.14}$$

Because of the finite constant term C one can conclude that each particle is located mainly in one minimum of the substrate potential. In contrast to that, $\langle s(0)s(\tau)\rangle$ decreases algebraically in the gapless phase with the power law:

$$\langle s(0)s(\tau)\rangle \propto \left[\left(\frac{\tau}{\hbar\beta}\right)^{-\kappa} + \left(1 - \frac{\tau}{\hbar\beta}\right)^{-\kappa} \right] . \tag{5.15}$$

Within the accuracy of the simulation there was no evidence of a finite offset in Eq. (5.15) to which $\langle s(0)s(\tau)\rangle$ converges as $\tau \to \infty$ and $\beta = 2\tau/\hbar$. The different behavior of the correlation function in the two phases is in accordance with the predictions for the continuum model [GNT98].

5.5 The mean square displacement

Because of the ground state's different characters in the phases with and without gap, one can expect that the mean square displacement of the center of mass mode $\langle(\Delta u_0)^2\rangle$ behaves qualitatively different in the low temperature limit for the two phases. Of special interest is the temperature dependence of the center of mass mode's diffusion constant as the temperature T goes to zero. The central mode is defined as

$$u_0 = \sqrt{\frac{1}{N}} \sum_{j=0}^{N-1} x_j . \tag{5.16}$$

The central mode's mean square displacement remains finite in the thermodynamic limit, while the mean square displacement of the center of mass $X_{\mathrm{CMS}} \equiv \sqrt{1/N} u_0$ is zero for $N \to \infty$.

The mean square displacements of the central mode for different temperatures are shown in Fig. 5.11 for the mass $m = 0.02$ and in Fig. 5.12 for $m = 0.1$. Variation of the number of particles N confirmed that finite size effects are much smaller than the statistical errors for systems with $N = 32$ or more particles. As explained in chapter 4, a thermostat is coupled to the centroid modes to average over the centroid equations' starting conditions according to the centroid density (Eq. (4.16)). It was confirmed that the calculated mean square displacements do not depend on the thermostat's time scales: reduction of the time scales for the thermostat coupled to the internal and/or the central mode's time scale (γ_{int}/γ_{CMS}) leaves the results for the square displacement invariant (Fig. 5.11 right). This confirms that the time scales belonging to the centroids' thermostats are large compared to the characteristic time scales of the system which are observed. The simulation time is chosen to be 20 times larger than the maximal time plotted in Fig. 5.11 and 5.12, such that the average over the equation of motions' initial conditions is sufficiently accurate. Each curve is an average over eight independent simulation runs.

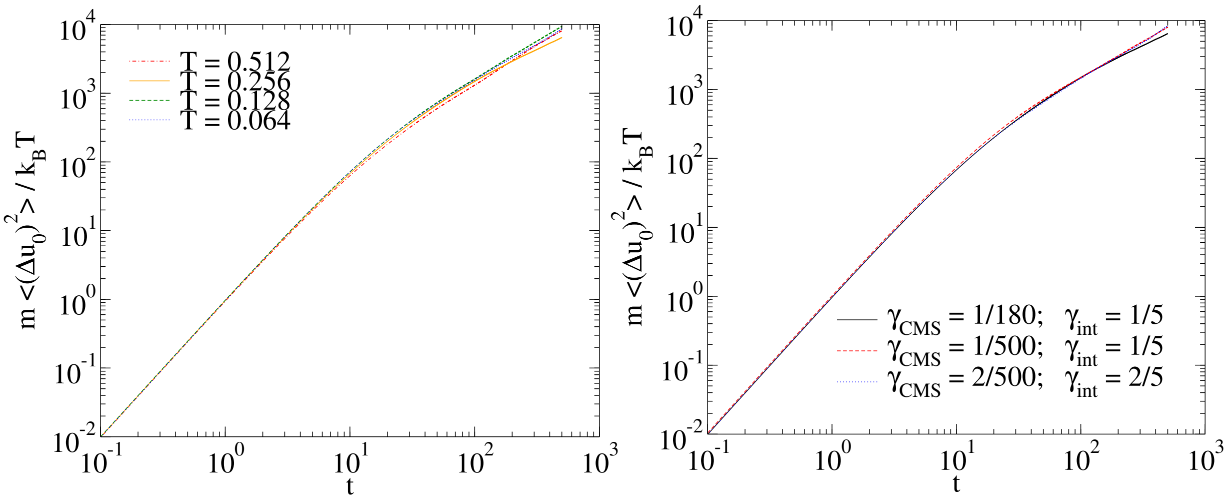

Figure 5.11: Left: Centroid square displacement of the central mode in the gapless phase ($m = 0.02$). The trivial temperature dependence is factored out. Right: Mean square displacement for N = 32 and T = 0.256. The centroid thermostat's time scales are varied.

The gapless phase

In Fig. 5.11 the results for the mean square displacement for $m = 0.02$ are shown. This is the value for m where the phonon excitation gap closes. The curves are normalized in a way that the trivial temperature and mass dependence is factored out and all curves are equal in the ballistic regime where $\langle(\Delta u_0)^2)\rangle = k_\mathrm{B}Tt^2/m$. In this normalization, the mean square displacements for all temperatures are scaled on one curve. To get a better understanding of mean square displacement's low temperature behavior the diffusion constant is considered, which can be defined through Einstein's relation:

$$D = \lim_{t\to\infty} \frac{\langle(\Delta u_0)^2\rangle}{2t} \ . \tag{5.17}$$

In the gapless phase, the diffusion constant D shows an Einstein diffusion behavior and the temperature dependence of the diffusion constant is of the form

$$D_\mathrm{gapless}(T) = \frac{k_\mathrm{B}T}{f_\mathrm{creep}} \tag{5.18}$$

in the low temperature regime. The numerical value for the effective friction constant defined in Eq. (5.18) is $f_\mathrm{creep} = 0.0025 \pm 0.0005$ for $m = 0.02$. This value for f_creep is in agreement with the results for the driven sine-Gordon chain presented in the following paragraph.

The phase with the gap

In the phase with a gap, the low temperature dependence of the diffusion constant differs from the Einstein diffusion behavior found for the gapless case. The results for the mean square

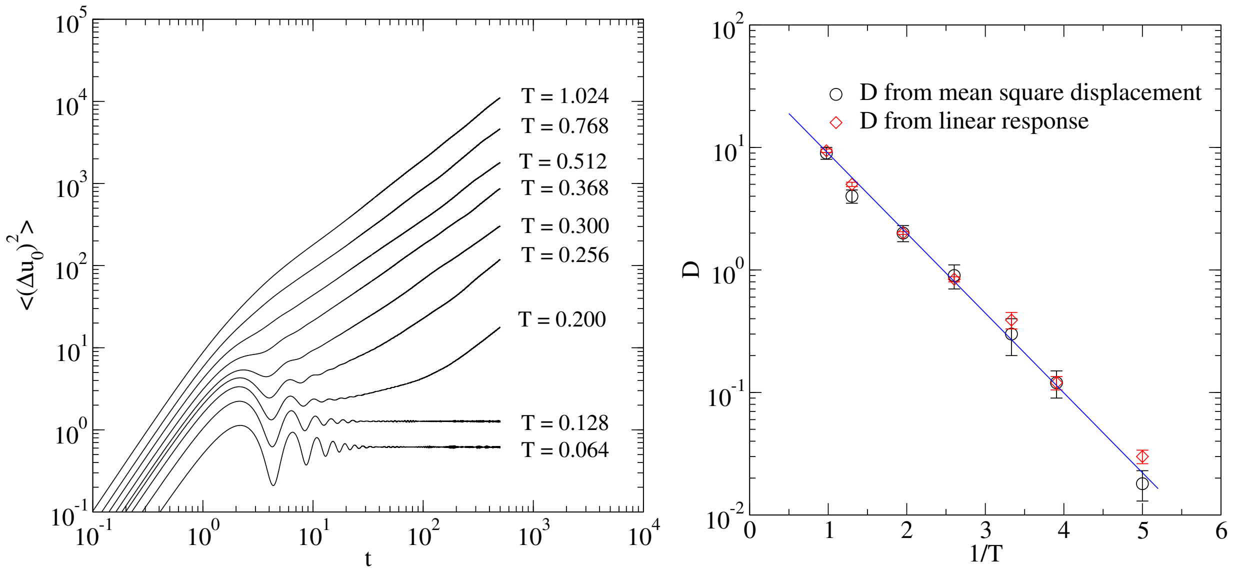

Figure 5.12: Left: Mean square displacement of the central mode in the phase with a gap (m = 0.1) Right: Arrhenius plot of the diffusion constant; the diffusion constant is determined from the mean square displacement (circles) and the linear response to a small external force (diamonds).

displacement for various temperatures are shown on the left hand side of Fig. 5.12. The particle mass is set to $m = 0.1$. At this m value, the phonon dispersion relation has a finite excitation gap. For the larger temperatures, the central mode's mean square displacement shows a simple crossover from the ballistic regime to the diffusive regime. As the temperature is decreased, a plateau develops between the ballistic and diffusive regimes. The size of the plateau increases as the temperature is decreased further. For $T \leq 0.128$, no finite diffusion can be measured during the simulation time. For very small temperatures, the mean square displacement shows a behavior similar to that of a harmonic oscillator. On the right hand side of Fig. 5.12, the temperature dependence of the diffusion constant is shown in an Arrhenius plot. The results for the diffusion constant $D_{\text{gap}}(T)$ fulfill the Arrhenius law:

$$D_{\text{gap}}(T) \propto \exp\left(-\frac{E}{k_{\text{B}}T}\right) . \tag{5.19}$$

The energy E is a barrier that has to be crossed by thermal activation if the central mode moves. Mass transport is induced by kink motion. The barrier E can be interpreted as the sum of the energy that is necessary to create a kink/anti-kink pair, the smallest energy barrier a kink has to overcome to move through the system which is called Peierls-Nabarro barrier and higher order multi-kink interactions. Thermal activated barrier crossing leads to the Arrhenius law behavior given in Eq. (5.19). For $m = 0.1$ the numerical value for the energy barrier is $E = 1.5 \pm 0.2$.

The diffusion constant can also be determined from the linear response of the system through a fluctuation dissipation theorem. This is done in the following paragraph (Fig. 5.17). The results are shown in the Arrhenius plot given in Fig. 5.12. They are in very good agreement with the results for the diffusion constant obtained from the mean square displacement of the central mode.

5.6 The driven discrete sine-Gordon chain

One of the main differences of the phases with and without gap is their response to a small external force F at $T = 0$. This is of particular interest for the description of transport phenomena. Here the velocity vs. applied force characteristics can be directly related to the transport of magnetic domain walls, vortex systems and charge density waves. The reaction of a discrete sine-Gordon chain on an external driving force is the subject of this paragraph. The analysis focuses on the ground state properties which can be determined by the extrapolation from finite T to zero T. Special emphasis is given to the fluctuation dissipation relation between the results obtained from the mean square displacement in the last paragraph and the linear response of the system to a small force.

To study the response to an externally applied field, a homogeneous force F is applied to each particle by adding a term $-F\sum_{j=0}^{N-1}\hat{x}_j$ to $\hat{H}$. The thermostats coupled to the centroid variables are needed to dissipate the energy added to the system by the external driving force F. The Hamiltonian which is considered has the form

$$\hat{H} = \sum_{j=0}^{N-1}\left(\frac{\hat{p}_j^2}{2m} + \frac{1}{2}K(\hat{x}_j - \hat{x}_{j+1})^2 - V_0\cos(\hat{x}_j/b) - F\hat{x}_j\right) + H_{\mathrm{bath}} \qquad (5.20)$$

where the bath is modeled as a Langevin equation with a classical fluctuating force and a friction term as discussed in chapter 4. It will be shown that the particular choice of the time constants belonging to the centroid thermostat does not affect the results in the creep regime if the time scales are sufficiently large.

The classical case

Before one studies the quantum mechanical case, it is instructive to look at the classical system. Chauve et al. analyzed interfaces of a harmonic manifold on a disordered substrate potential with a functional renormalization group approach [CGD98]. They found that the interface remains pinned until a critical force F_c is reached at zero temperature. As the external force reaches the depinning threshold, the system undergoes the depinning transition and starts moving. For large external driving forces, the interface moves with a velocity proportional to the force (Fig. 5.13). At finite temperature, the interface moves by thermal activation below F_c. This leads to a rounding of the depinning transition when $F \sim F_c$.

The gapless case

The force-velocity relation of a quantum sine-Gordon chain with the particle mass $m = 0.02$ is shown in Fig. 5.14. This is the value of m where the phonon excitation gap closes. A finite size analysis confirms that size effects are not important for $N \geq 32$ in the creep regime. For

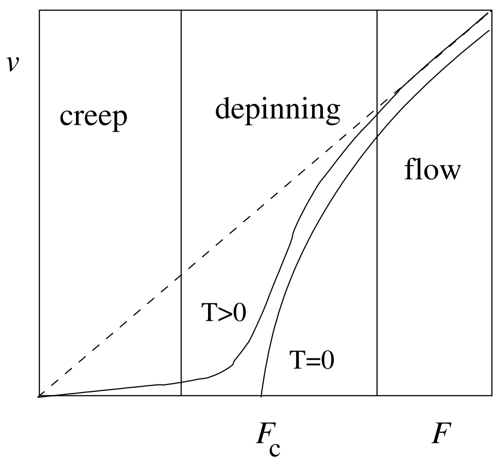

Figure 5.13: Typical force velocity curves for a classical system [CGD98]

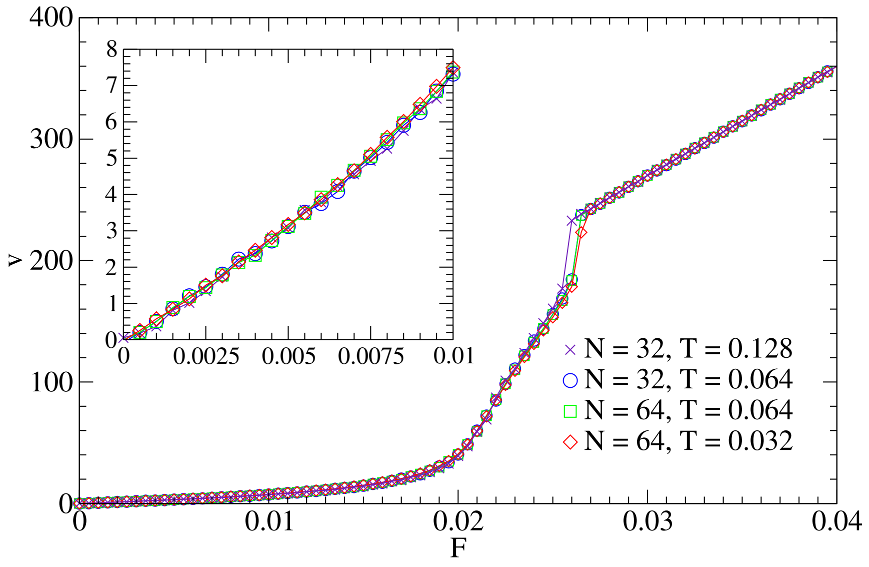

Figure 5.14: Sliding velocity v as a function of the externally applied force F for mass $m = 0.02$, system sizes $N = 32$ and $N = 64$ and temperatures $T = 0.064$ and $T = 0.032$, starting with a system at rest and slowly increasing F. The inset is an magnification of the small-v regime of the main figure.

$F < 0.01$, increasing or decreasing the thermostat (e.g. the externally imposed damping) has almost no effect on the $v(F)$ curve (Fig. 5.14). It is found that the response in v is linear with F at very small and very large F with different values for the effective damping coefficient $f_{\text{eff}} = F/v$. While (quantum) continuum approximations predict f_{eff} to be zero [Zs99], the chain's discreteness is known to change this property in classical systems because kink-phonon interactions damp solitons [CY83, PK83]. Interestingly, f_{eff} is independent of the temperature as T approaches zero in both linear regimes. This rules out the possibility that thermal fluctuations

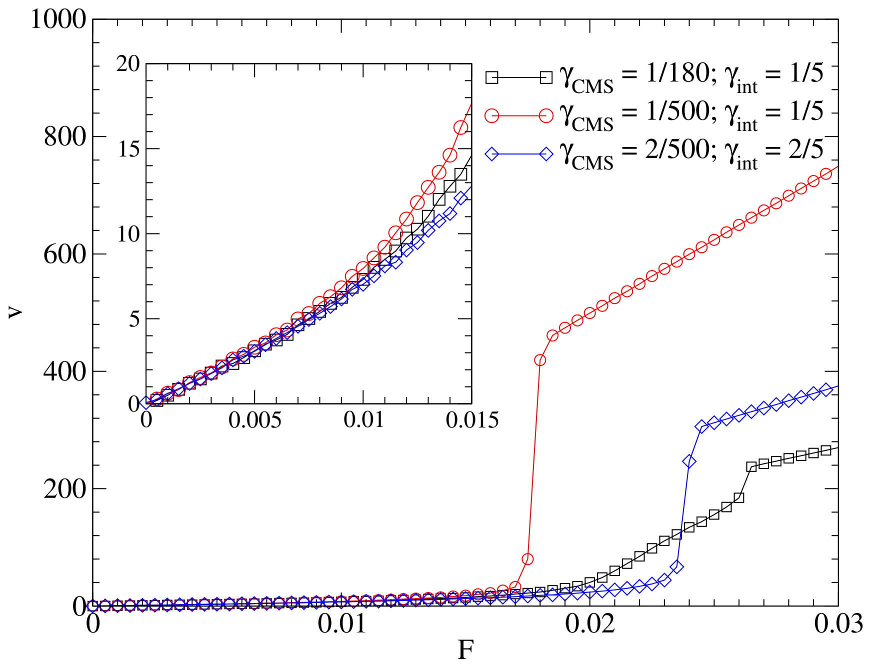

Figure 5.15: Sliding velocity v as a function of the applied force F for mass $m = 0.02$, $N = 32$ and $T = 0.256$. The time constant of the thermostat coupled to the central mode γ_{CMS} is varied.

assist the system to overcome energy barriers at small F.

Another similarity with the dynamics of the classical (discrete) sine-Gordon chain is seen at an external force $F = 0.02$, where the friction-velocity relation exhibits a cusp, separating a high-friction, low-velocity from a low-friction, high-velocity regime [CY83, PK83]. Finally, at external forces $F > 0.025$, the response crosses over into a completely unpinned regime in which the motion of the sine-Gordon chain is mainly opposed by the drag coefficient associated with the heat bath. In that regime, soliton-related dissipation mechanisms become unimportant.

In the linear response regime at very small external forces F, one can define an effective friction constant as

$$f_{\text{eff}} = \lim_{F \to 0} \frac{F}{v} \ . \tag{5.21}$$

This friction constant is connected to the diffusion constant by a fluctuation dissipation theorem:

$$f_{\text{eff}}(T) = \frac{D}{k_{\text{B}}T} \ . \tag{5.22}$$

The inset of Fig. 5.14 shows that the creep motion is temperature independent. This implies that in the low temperature regime the effective friction constant $f_{\text{eff}}^{\text{gapless}}$ is temperature independent. This behavior is in accordance with the results found from the mean square displacements which have an Einstein diffusion like temperature dependence in the gapless phase. The zero temperature limit of the effective friction constant is finite:

$$\lim_{T \to 0} f_{\text{eff}}^{\text{gapless}}(T) =: f_{\text{creep}}^{\text{gapless}} \ . \tag{5.23}$$

For $m = 0.02$, the numerical value for the friction coefficient in the creep regime is $f_{\text{creep}}^{\text{gapless}} = 0.0016 \pm 0.0008$. The corresponding value for $f_{\text{creep}}^{\text{gapless}}$ determined from the mean square displacement is $f_{\text{creep}}^{\text{gapless}} = 0.0025 \pm 0.0008$. The finite zero temperature limit of $f_{\text{eff}}^{\text{gapless}}(T)$ leeds to a creep motion which is temperature independent and does not vanish at $T = 0$ (Fig. 5.14). Zero temperature creep motion in the gapless phase is an example for a macroscopic property that is qualitatively changed by quantum fluctuations with respect to the classical system.

The phase with a gap

In the phase with a gap, the reaction of the system to an external force at $T = 0$ does not differ qualitatively from the classical system. Fig. 5.16 shows the force-velocity relation for $m = 0.1$. The system remains pinned until the force reaches a critical value $F_c \approx 0.05$ where the depinning transition occurs. At finite temperature, the system creeps thermally activated. The

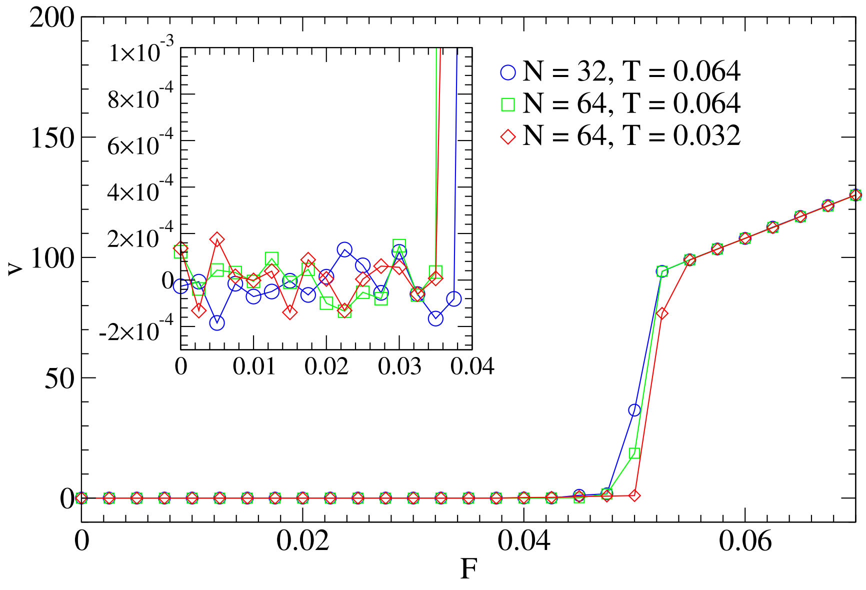

Figure 5.16: Force velocity relation in the phase with a gap $(m = 0.1)$; the calculation is started with a system at rest and the force F is increased in small steps.

response of the system to a small driving force at finite temperature is shown in Fig. 5.17. The creep motion is suppressed exponentially as the temperature is decreased. This can be described phenomenologically by a divergence of the effective friction constant in the creep regime. The temperature dependence is described by the Arrhenius law:

$$\frac{1}{f_{\text{eff}}^{\text{gap}}(T)} \propto \frac{1}{T} \exp\left\{-\frac{E}{k_{\text{B}}T}\right\} . \tag{5.24}$$

Numerical results for $D = T/f_{\text{eff}}^{\text{gap}}$ determined form the data presented in Fig. 5.17 are plotted in the Arrhenius plot (Fig. 5.12) together with the corresponding results from the mean square displacements. They are in good agreement with Eq. (5.24). As already mentioned, $E = 1.5 \pm 0.2$ is

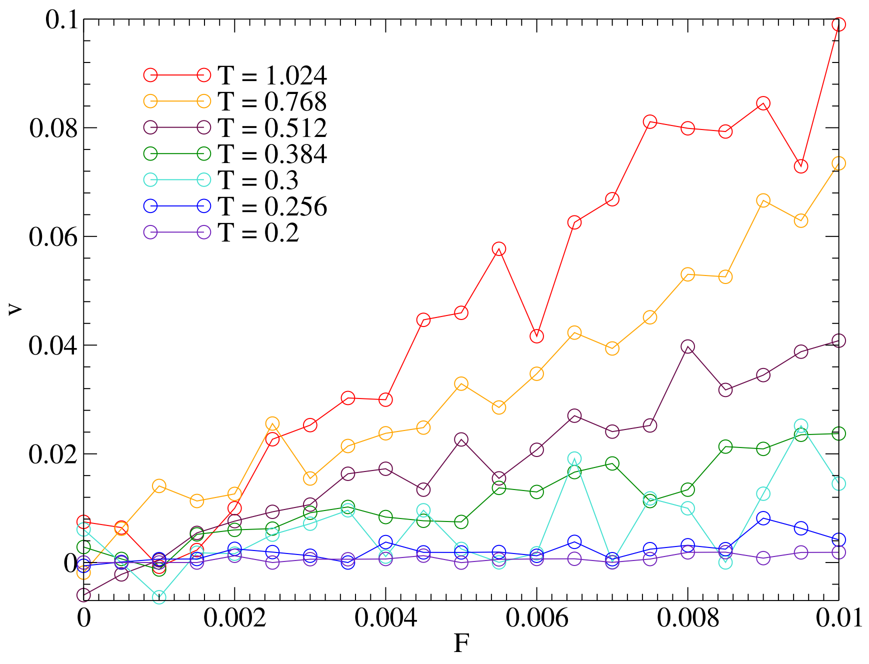

Figure 5.17: Linear response of the sine-Gordon chain to an external driving force F at various temperatures. The particle mass is set to $m = 0.1$ and the system size is $N = 32$. The slope of the curves is the inverse effective friction constant $1/f_{\text{eff}}^{\text{gap}}$.

found for $m = 0.1$ from both, mean square displacement and linear response. This confirms that the non-equilibrium analysis presented in this paragraph is in accordance with the equilibrium results. Eq. (5.24) leads to a pinning of the system as the temperature approaches zero (Fig. 5.17) because

$$\lim_{T\to 0} f_{\text{eff}}^{\text{gap}}(T) = \infty \ . \tag{5.25}$$

One can conclude that the transport of chains consisting of beads with masses $m > m_c$ requires either finite temperatures and/or finite forces. For example, for $m = 5m_c$, the numerical data provides an upper bound for the inverse mobility or effective damping $f_{\text{eff}}^{m=0.1} > 0.5{\cdot}10^6 f_{\text{eff}}^{m=0.02}$.

Chapter 6

Disordered substrate potentials

The previous analysis shows that path integral molecular dynamics can distinguish well between systems with a finite gap in the phonon dispersion relation and systems without excitation gap at phonon wave vector $q = 0$. It is now possible to analyze more complicated systems for which the quantum mechanical ground state of the continuum model is not known analytically. One of the topics of current scientific interest is the interplay of disorder and quantum fluctuations [GFB02]. In the following, the author investigates the effect of disorder on the spectral properties of the discrete quantum sine-Gordon chain (DQSGC). This is either done by replacing the external potential $V_0 \cos(x_n/b)$ of the discrete sine-Gordon chain with a random potential, or by introduction of a fluctuating bond length a_j. In the latter case, the harmonic interaction of next neighbored particles is set to be $(K/2)(x_j - x_{j+1} - a_j)^2$ with randomly chosen parameters a_j, distributed according the probability distribution shown in Fig. 6.5. This kind of harmonic interaction can be also interpreted as a substrate potential $\cos(x_j/b + \phi_j)$ with uniformly distributed random phases ϕ_j. The disordered potentials are classified by the roughness exponent H which is defined as

$$\langle \{V(x + \Delta x) - V(x)\}^2 \rangle \propto \Delta x^{2H} \ . \tag{6.1}$$

In a simulation with periodic boundary conditions ($V(x) = V(x + 2\pi N)$), Eq. (6.1) holds for $b \ll \Delta x \ll \pi b N$. As in the previous analysis, the unit system is defined by V_0, b, $\hbar$ and k_B (see appendix C). The harmonic coupling is set to $K = 0.1$ for all simulations and the mass m is varied. For the calculation of the quantum dispersion relation in the limit $T \to 0$, the numerical value for the temperature $T = 0.064$ turned out to be sufficiently small. For this temperature, the Trotter number is set to $P = 256$.

6.1 Roughness exponent H = 0

In order to construct a random substrate potential with $H = 0$, the potential is either zero on a length of πb or - with same probability - it takes the functional form of $V_0\{1 + \cos(x/b)\}$ on the interval $-\pi \leq x/b < \pi$. The numerical analysis of the results for the case $H = 0$ suggests a finite gap for a mass $m > m_c^{(H=0)}$ and zero gap for $m < m_c^{(H=0)}$ (see Fig. 6.1) in agreement with the predictions by Gorokhov et al. [GFB02]. The critical value for the mass $m_c^{H=0}$ is larger than

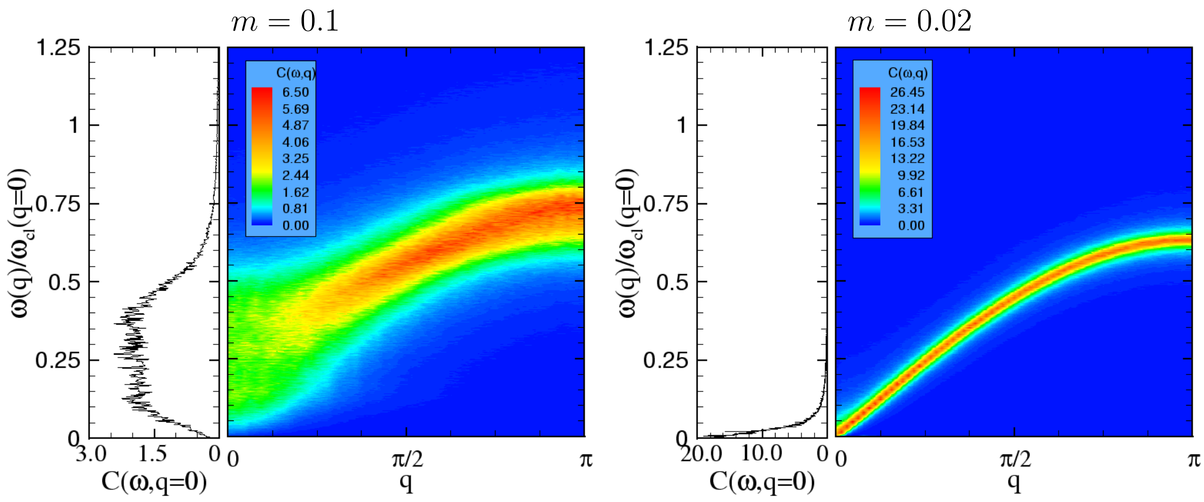

Figure 6.1: Quantum ground state phonon dispersion relations for a system with disordered potential with roughness exponent $H = 0$, The results are averages over 12 different realizations of the potential. Left: $m = 0.1$, right: $m = 0.02$. For each dispersion relation, the zero wave vector spectrum $C(\omega, q = 0)$ is shown separately on the left. Here $\omega_{\text{cl}}(q = 0)$ stands for the classical excitation gap of the corresponding sine-Gordon chain, which is given by $\omega_{\text{cl}}(q = 0) = \sqrt{V_0/(b^2 m)}$.

the value for the DQSGC found in chapter 5. For the model potential under consideration, one phonon branch is observed. For masses larger than but in the order of m_c, it forms a broad band. At $m < m_c$, the substrate potential becomes irrelevant and the system shows the same dispersion relation as the commensurate discrete sine-Gordon model in the gapless phase: there is only one relatively narrow branch, which can be well described with Eq. (5.4) and $\tilde{K}_0 = 0$.

6.2 Roughness exponent $\mathbf{H = 1/2}$

The disordered potential with roughness exponent $H = 1/2$ is constructed in the following way: Patches of the functional form $V_0 \cos(x/b)$ and of the length πb are added where the underlying domain is chosen randomly to be either $[0, \pi b]$ or $[\pi b, 2\pi b]$. The patches are shifted by a constant in a way that no discontinuity in the potential occurs. This leads to a potential which does a random walk in it's magnitude. The mean square difference of the potential at two different values for x which are separated by Δx behaves as $\langle \{V(x + \Delta x) - V(x)\}^2 \rangle \propto \Delta x^1$ on scales $b \ll \Delta x \ll \pi b N$ for large N. The corresponding surface roughness exponent H takes the value $H = 1/2$. Fig. 6.2 shows a typical random potential and a snapshot of the imaginary-time trajectories of the particles.

The results for the classical and quantum dispersion relations for three different values of the mass m ($m = 0.1, 0.02, 0.01$) are depicted in Fig. 6.3. For each dispersion relation, the zero wave vector spectrum is shown separately on the left. The shown quantum dispersion relations correspond to the ground state while for the classical dispersion relations the temperature T

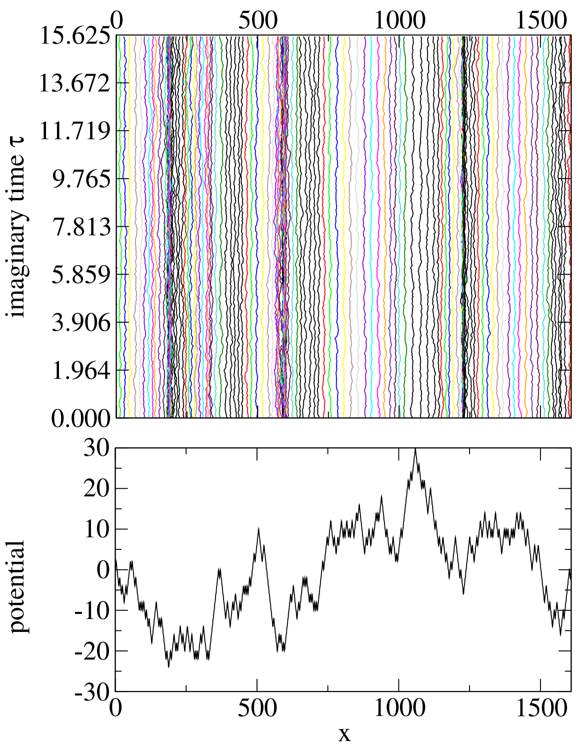

Figure 6.2: Above: Snapshot of the imaginary-time trajectories of a typical configuration, calculated with $N = 256$, $T = 0.064$, $P = 256$, $m = 0.02$. Below: Typical realization of the disordered substrate potential with roughness exponent $H = 1/2$ belonging to the configuration shown above.

was chosen in a way that the classical kinetic energy is equal to the expectation value for the kinetic energy of the corresponding quantum ground state ($T = \frac{2}{k_\mathrm{B}}\langle \hat{E}_\mathrm{kin}\rangle_\mathrm{qm}$). It is found that the situation is more complex for disordered potentials with roughness $H = 1/2$ than it is for the case $H = 0$. While for $H = 0$ the system behaves qualitatively equal to the DQSGC, the observations for the $H = 1/2$ system differ qualitatively from the DQSGC. In agreement with the predictions by Chauve et al. [CGD98], there is no indication for the classical mobility to become zero at finite temperatures. In contrast, the quantum mechanical mobility at $T = 0$ always appears to be zero for $H = 1/2$, no matter how small m is. The zero wave vector spectrum $C(q\!=\!0, \omega)$ for classical and quantum mechanical chains shows a qualitative difference in the limit $(q, \omega) \to (0, 0)$. While the classical value for $C(0, 0)$ is finite, $C(0, 0)$ vanishes for the quantum systems. (Note that due to statistical uncertainties and finite system size, one can never observe $C(0, 0) = 0$ in a computer simulation.) This is confirmed by the finite size analysis shown in Fig. 6.4. While thermal fluctuations can lead to mobility, the system remains pinned if only quantum fluctuations and no classical fluctuations are present.

The spectrum at finite wavelengths of the quantum system shows two phonon branches. The numerical results are the average over 12 different disorder realizations. A single realization shows much sharper lines. It appears that the system is not self-averaging.

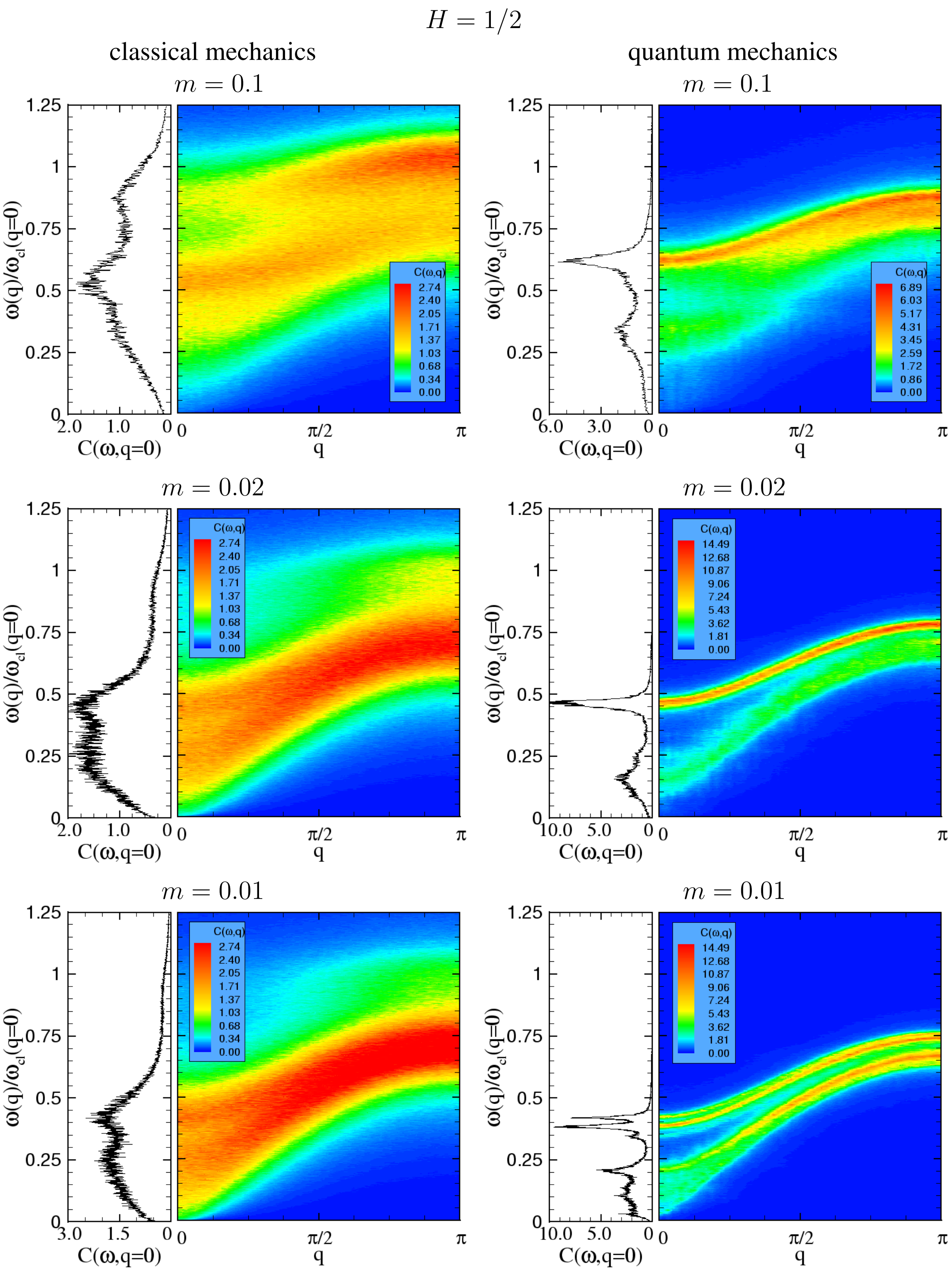

Figure 6.3: Phonon dispersion relations for systems with disordered substrate potential with roughness exponent $H = 1/2$ for three different masses: above: $m = 0.1$, middle: $m = 0.02$, below: $m = 0.01$. Left: classical systems at finite temperature $T = \frac{2}{k_{\rm B}}\langle \hat{E}_{\rm kin}\rangle_{\rm qm}$, above $T = 0.987$, middle: $T = 1.635$, below $T = 1.980$. Right: quantum system in the ground state. For each dispersion relation the zero wave vector spectrum is shown separately on the left. Here $\omega_{\rm cl}(q{=}0)$ stands for the zero wave vector excitation gap of the corresponding classical sine-Gordon chain, which is given by $\omega_{\rm cl}(q{=}0) = \sqrt{V_0/(b^2 m)}$.

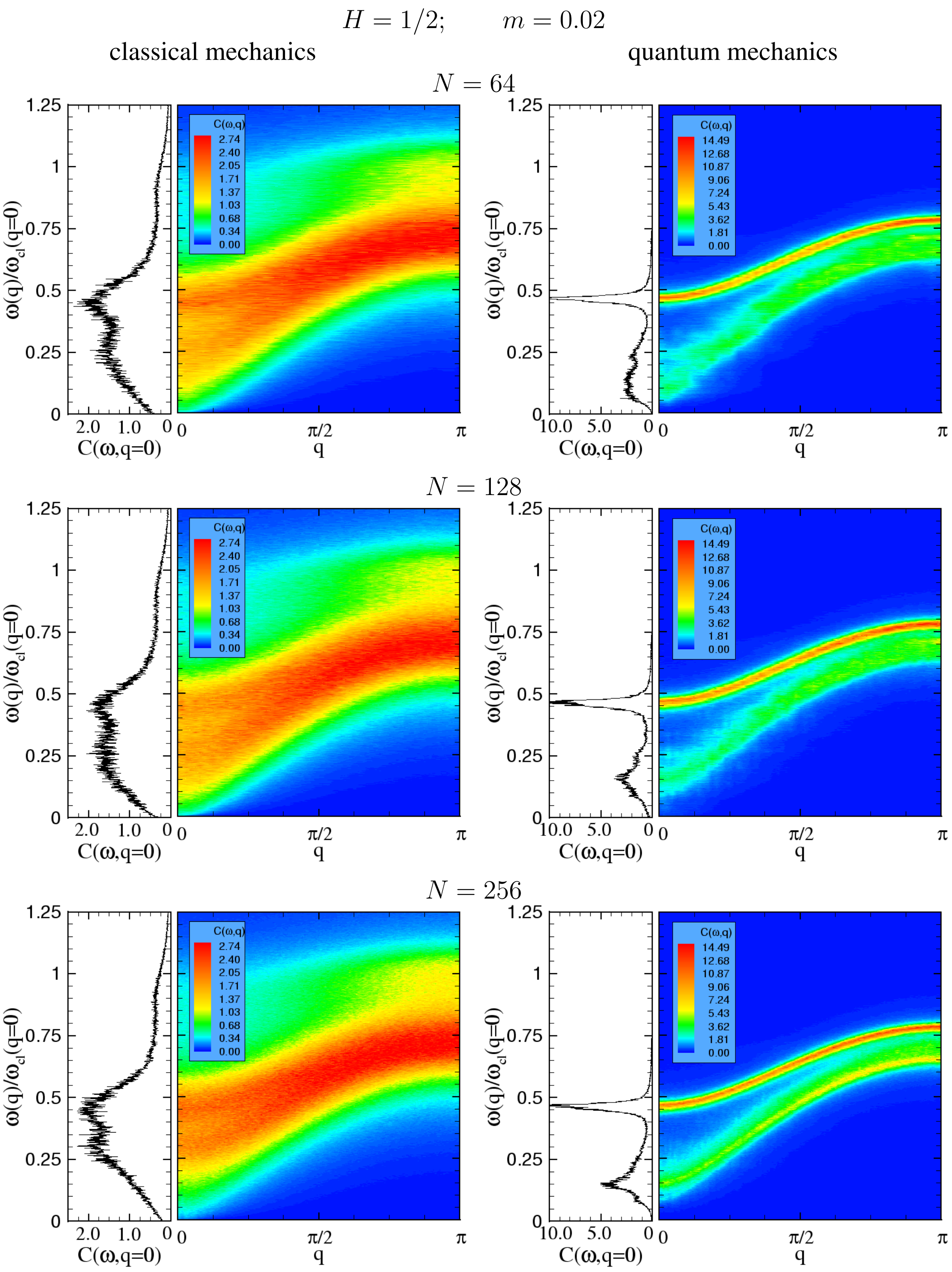

Figure 6.4: Finite size analysis for the disordered systems with $H = 1/2$ for $m = 0.02$. The system sizes are: above: $N = 64$, middle: $N = 128$, below: $N = 256$. Left: classical systems at finite temperature $T = \frac{2}{k_B}\langle \hat{E}_{\text{kin}} \rangle_{\text{qm}}$, $T = 1.980$. Right: quantum system in the ground state. For each dispersion relation the zero wave vector spectrum is shown separately on the left. Here $\omega_{\text{cl}}(q = 0)$ stands for the classical excitation gap of the corresponding sine-Gordon chain, which is given by $\omega_{\text{cl}}(q = 0) = \sqrt{V_0/(b^2 m)}$.

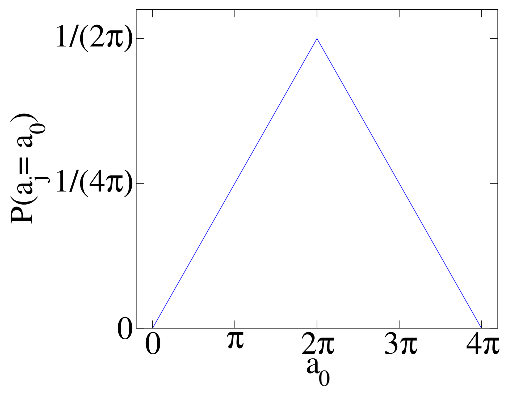

Figure 6.5: Bond length distribution

6.3 Disordered bond length or phase

A model system with disordered bond length a_j is described by the Hamiltonian

$$\hat{H} = \sum_{j=0}^{N-1} \left(\frac{\hat{p}_j^2}{2m} + \frac{1}{2} K (\hat{x}_j - \hat{x}_{j+1} - a_j)^2 - V_0 \cos(\hat{x}_j / b) \right) . \tag{6.2}$$

With the transformation

$$x_j = y_j + (2\pi j + \phi_j)\, b \; , \tag{6.3}$$

this system can be mapped on a discrete sine-Gordon chain with the random phases ϕ_j in the substrate potential, given by

$$\hat{H} = \sum_{j=0}^{N-1} \left(\frac{\hat{p}_j^2}{2m} + \frac{1}{2} K (\hat{y}_j - \hat{y}_{j+1})^2 - V_0 \cos(\hat{y}/b + \phi_j) \right) . \tag{6.4}$$

For the numerical analysis, the random phases ϕ_j are chosen to be uniformly distributed in the interval $[-\pi, \pi]$. They are connected with the random bond length as

$$a_j = b\,(2\pi + \phi_j - \phi_{j+1}) \; . \tag{6.5}$$

This leads to the probability distribution for the bond length a_j shown in Fig. 6.5. The simulations show that the model system with disordered bond length behaves similar to the one with a disordered potential with roughness exponent $H = 0$. A broad band can be seen in the dispersion relation with a finite excitation gap for large masses (e.g. $m = 0.1$). As the mass is decreased, the excitation gap becomes smaller and vanishes at a the critical value $m_{\mathrm{c}}^{\mathrm{bond}}$. Fig. 6.6 shows phonon dispersion relations for the masses $m = 0.1$, 0.08, 0.06, 0.04 and 0.02. From the simulation results, the critical value for the mass m_c^{bond} can be estimated to be slightly smaller than $m = 0.04$. At $m = 0.02$, the system is already in the gapless phase.

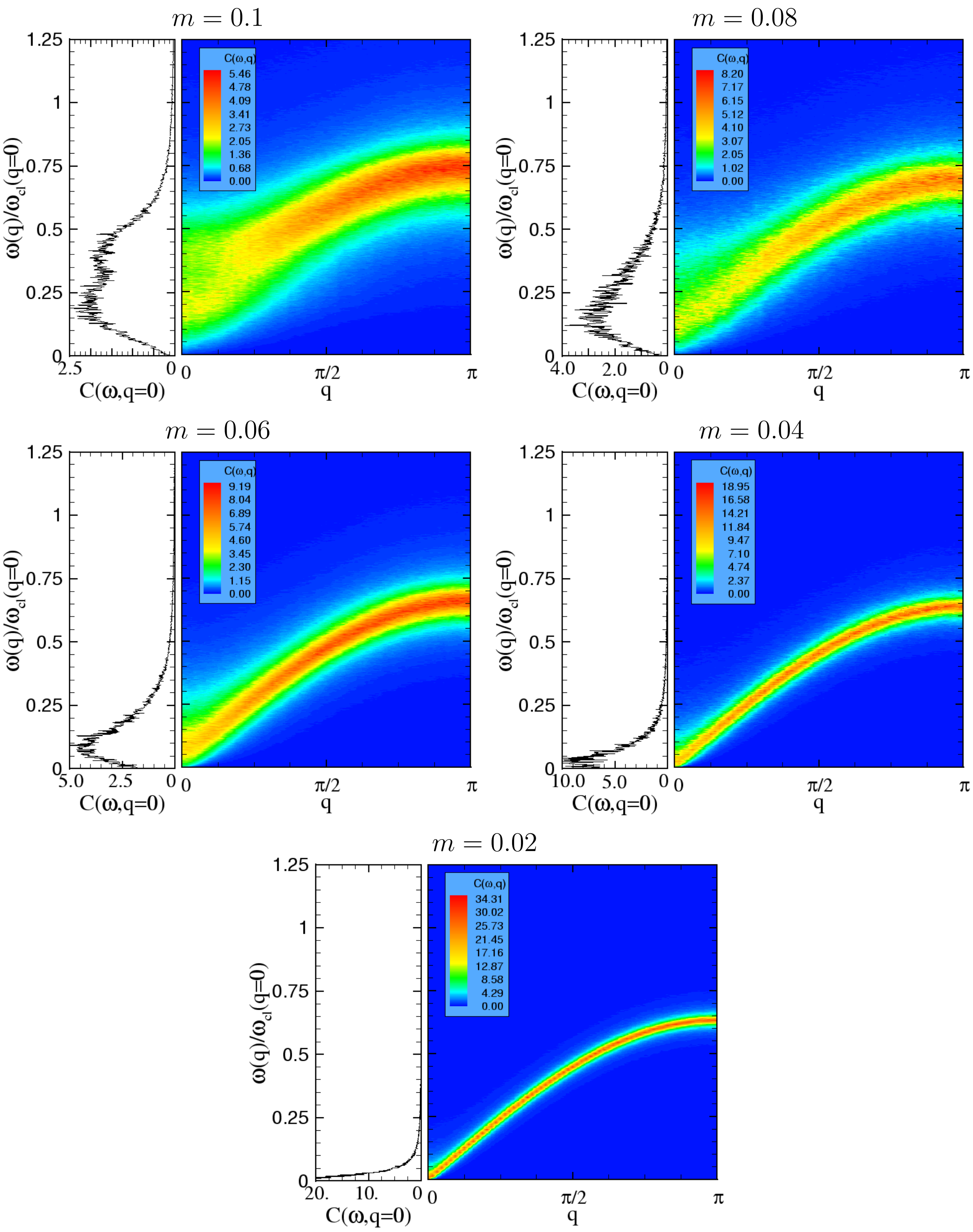

Figure 6.6: Dispersion relations for DQSGC with disordered bond length. The particle mass is varied. For each dispersion relation the zero wave vector spectrum is shown separately on the left. Here $\omega_{\rm cl}(q\!=\!0)$ stands for the zero wave vector excitation gap of the corresponding classical sine-Gordon chain which is given by $\omega_{\rm cl}(q\!=\!0) = \sqrt{V_0/(b^2 m)}$.

Chapter 7

Conclusion and outlook

A path integral simulation algorithm which includes a higher-order Trotter approximation (HOA) is analyzed and compared to an approach which includes the correct quantum mechanical pair interaction (EPr). It is found that the HOA algorithm converges to the quantum limit with increasing Trotter number P as $1/P^4$, while the EPr algorithm converges as $1/P^2$ but starts with a better estimate for very low Trotter numbers. For the comparative study, a harmonic chain is considered as a test model. This model has the advantage that all calculations can be performed analytically. The convergence rate of the HOA algorithm is analyzed for physical systems as a particle in a double well potential, gaseous argon, gaseous helium and crystalline argon. It is found that the $1/P^4$ convergence holds also in the case that the interaction potential has strong repulsive divergences as the r^{-12} divergence of the Lennard Jones potential. New simulation techniqus for the HOA algorithm are developed: The estimators for the pair correlation function and for the particle position and distance are derived and it is shown for a particle in the double-well potential and crystalline argon that the HOA approach allows one to reduce strongly the systematic error of these quantities at a fixed Trotter number. As a further result, it is shown that the estimators for the volume or the lattice constant in a simulation in the constant pressure ensemble do not receive higher-order corrections in the HOA algorithm.

A new path integral algorithm, the hybrid algorithm is developed which combines an exact treatment of the quadratic part of the Hamiltonian and the higher-order Trotter expansion techniques. For the discrete quantum sine-Gordon chain (DQSGC), it is demonstrated that this algorithm works more efficiently than all other improved path integral algorithms discussed in this work. The hybride algorithm shows a $1/P^4$ convergence to the quantum limit and starts at small Trotter numbers P with a good estimate for the quantum system. The sampling efficency can be improved by an optimization of the dynamical masses, which allows to adjust all time scales of the system to one typical sampling frequency. This is possible if a harmonic approximation of the Hamiltonian exists and the simulation is performed in the normal coordinates of this approximation.

The centroid dynamics method, a method which allows the efficient and exact calculation of several correlation functions is applied to the DQSGC. In this work, a special variant, the adiabatic centroid path integral molecular dynamics (ACPIMD) method is applied. Within this method, the time scales of the system have to be adjusted in a way that the centroid variables move adiabatically on a mean potential. The correct choice for the dynamic masses which determine the time scales occurring in the simulation is deeply discussed and a concrete implementation of the ACPIMD method with a Langevin thermostat is presented. This implementation has the advantage that it also allows the treatment of non-ergodic systems.

The newly developed simulation techniques allow the analysis of the DQSGC and related disordered model systems in the highly quantum mechanical regime. The ground state phonon dispersion relation is calculated for the DQSGC within the ACPIMD method. It is found that the excitation gap at zero wave vector is reduced by quantum fluctuations. The size of the excitation gap can be used as an order parameter to distinguish two different phases. A phase which has an excitation gap at $q = 0$ and a gapless phase without excitation gap a $q = 0$. The real-time correlation functions obtained with ACPIMD are found to be in good agreement with the imaginary-time dynamics obtained from PIMD. The correlation function $G(\Delta n, \Delta\tau)$ which measures the correlation of two particles separated by Δn in the topology of the chain or $\Delta\tau$ in imaginary-time is calculated in the phase with gap and the gapless phase. G diverges in the gapless phase for Δn or $\Delta\tau \to \infty$, while it converges to a finite value in the other phase. Tunneling of the imaginary-time paths is found to be the process that drives the phase transition from the phase with a gap to the gapless phase. The importance of imaginary-time tunneling is analyzed qualitatively with the minimum correlation function which vanishes analytically in the gapless phase and converges exponentially to a finite value in the phase with a gap for $\beta \to \infty$, $\Delta\tau = \hbar\beta/2$. The mean square displacement of the center of mass mode is calculated for both phases. From the mean square displacement the diffusion constant D can be determined. The temperature dependence of the diffusion constant differs qualitatively in the phase with a gap and the gapless phase: While in the phase with a gap the diffusion constant shows an Einstein diffusion like behavior and is proportional to the temperature, it vanishes exponentially in the gapless phase by the Arrhenius law. The reaction of the DQSGC to the presence of an external driving force is calculated for the ground state. In the gapless phase, the system starts to creep when an arbitrarily small driving force is applied. At a critical value of the force, the system undergoes a depinning transition and enters a flow regime. In contrast to that, the DQSGC chain is pinned at $T = 0$ in the phase with a gap. The system shows no reaction to a small external force until a critical value for the force is reached and the system undergoes a depinning transition. For finite temperature T, the mobility of the DQSGC goes to zero exponentially in the phase with a gap as the temperature is decreased. The diffusion constant is determined form the linear response of the chain to a small external force. The results are in good accordance with the diffusion constants obtained from the mean square displacements.

The analysis of the DQSGC is generalized to models with disordered substrate potentials. Three different cases are analyzed: Disordered substrate potentials with roughness exponents $H = 0$, $H = 1/2$, and a system with disordered bond length. For all models, the ground state phonon dispersion relation is calculated. It is found that the disordered system with $H = 0$ behaves qualitatively similar to the DQSGC. A phaes with a gap and a gapless phase are found. In the phase with a gap, the phonon dispersion relation has one broad branch while the dispersion relation in the gapless phase is equal to that of the DQSGC. Disordered systems with roughness $H = 1/2$ show a more complicated behavior. For these systems the chain is always pinned and no mobility due to quantum fluctuations can be observed. Here, the effect of quantum and classical fluctuations on the mobility is different. Classical systems at finite temperature show a creep motion while the quantum system is pinned at $T = 0$. For finite wave vectors, two different phonon branches are found for the quantum system. The corresponding classical system shows only one broad phonon brach in the dispersion relation. The DQSGC with disordered bond length behaves similarly to the disordered case with $H = 0$. A transition between a phase with a gap and a gapless phase is found. For the gapless phase, the dispersion relation is equal to the one found for the ordered system while in the phase with a gap, the phonon branch is much broader than for the ordered system.

For further studies, it would be interesting to extend the study of the disordered systems to models with continuous roughness exponent H. Here the question arises whether the mobility of the quantum ground state vanishes for any $H > 0$ or if a finite critical value for H exists. With the techniques developed, it is possible to calculate a complete phase diagram for the DQSGC. Here, the highly discrete region is of special interest because large differences might be expected to the continuum model which is analytically solvable.

With increasing computer power, it is also possible to analyze higher dimensional systems. For these systems, it can be expected that the correlation lengths are smaller than for the one dimensional system. This allows to reduce the system sizes used in the simulation with respect to the calculations in one dimension.

Bibliography

[AA98] T. Ando and Y. Arakawa, editors. *Mesoscopic Physics and Electronics*. Springer, Berlin, 1998. 3, 60

[AAB+84] H. C. Anderson, M. P. Allen, A. Bellemans, J. Board, J. H. R. Clarke, M. Ferrario, J. M. Haile, S. Nosé, J. V. Opheusden, and J. P. Ryckaert. New molecular dynamics methods for various ensembles. *Rapport d'activités scientifique du CECAM*, pages 82–115, 1984. 17

[ACW87] R. A. Aziz, F. R. W. Court, and C. C. K. Wong. A new determination of the ground state interatomic potential of He_2. *Mol. Phys.*, 61:1487, 1987. 14

[AG02] J. V. Alvarez and C. Gros. Conductivity of quantum spin chains: A quantum Monte Carlo approach. *Phys. Rev. B*, 66:094403, 2002. 60

[Bar79] J. A. Barker. A quantum-statistical Monte Carlo method; path integrals with boundary conditions. *J. Chem. Phys.*, 70:2914, 1979. 8

[BB94] G. P. Berman and E. N. Bulgakov. Coherent structures in the ground-state of the quantum Frenkel-Kontorova model. *Phys. Rev. B*, 49:8212–8226, 1994. 60

[BGS89] F. Borgonovi, I. Guarneri, and D. Shepelyansky. Destruction of classical cantori in the quantum Frenkel-Kontorova model. *Phys. Rev. Lett.*, 63:2010–2012, 1989. 60

[BK98] O. M. Braun and Yu. S. Kishvar. Dynamic properties of the Frenkel-Kontorova model. *Phys. Rep.*, 306:1–108, 1998. 3, 59, 60

[Cep95] D. M. Ceperley. Path integrals in the theory of condensed helium. *Rev. Mod. Phys.*, 67:279, 1995. 3, 5, 6, 8, 9, 13, 28, 63, 99

[CGC98] C. Chakravarty, M. C. Gordillo, and D. M. Ceperley. A comparison of the efficiency of fourier- and discrete time-path integral Monte Carlo. *J. Chem. Phys.*, 109:2123, 1998. 2

[CGD98] P. Chauve, T. Giamarchi, and P. Le Doussal. Creep via dynamical functional renormalization group. *Europhys. Lett.*, 44:110–115, 1998. 4, 60, 73, 74, 81

[CM96] J. Cao and G.J. Martyna. Adiabatic path integral molecular dynamics methods. II algorithms. *J. Chem. Phys.*, 104:2028, 1996. 4, 55, 60

[CMTV93] A. Cuccoli, A. Macchi, V. Tognetti, and R. Vaia. Monte Carlo computations of the quantum kinetic energy of rare-gas solids. *Phys. Rev. B*, 47:14923, 1993. 3

[Col75] S. Coleman. Quantum sine-Gordon equation as the massive Thirring model. *Phys. Rev. D*, 11:2088, 1975. 63

[CV93] J. Cao and G. A. Voth. A new perspective on quantum time correlation functions. *J. Chem. Phys.*, 99:10070, 1993. 2, 53

[CY83] J. A. Combs and S. Yip. Single-kink dynamics in a one-dimensional atomic chain: A nonlinear atomistic theory and numerical simulation. *Phys. Rev. B*, 28:6873, 1983. 74, 75

[Fey72] R. P. Feynman. *Statistical Mechanics*. W. A. Benjamin, Inc., 1972. 33, 41

[FK38] Y. I. Frenkel and T. Kontorova. On the theory of plastic deformation and twinning. *Zh. Eksp. Teor. Fiz.*, 8:1340, 1938. 3, 59

[FM96] L. M. Floria and J. J. Mazo. Dissipative dynamics of the Frenkel-Kontorova model. *Adv. Phys.*, 45:505–598, 1996. 3, 60

[GFB02] D. A. Gorokhov, D. S. Fisher, and G. Blatter. Quantum collective creep: A quasi-classical Langevin equation approach. *Phys. Rev. B*, 66:214203, 2002. 4, 60, 79

[GNT98] A.O. Gogolin, A. A. Nersesyan, and A. M. Tsvelik. *Bosonization and Strongly Correlated Systems*. Cambridge University Press, 1998. 70

[GT86] R. Giachetti and V. Tognetti. Quantum corrections to the thermodynamics of non-linear systems. *Phys. Rev. B*, 33:7647, 1986. 3

[HBB82] M. F. Herman, E. J. Bruskin, and B. J. Berne. On path integral Monte Carlo simulations. *J. Chem. Phys.*, 76:5150, 1982. 8, 99

[HC01] C.-L. Ho and V. C.-I. Chou. Simple variational approach to the quantum Frenkel-Kontorova model. *Phys. Rev. E*, 63:016203, 2001. 60

[HL99] B. Hu and B. Li. Ground-state wave function of the quantum Frenkel-Kontorova model. *Europhys. Lett.*, 46:655–660, 1999. 60

[HMR90] P. Hänggi, F. Marchesoni, and P. Riseborough. Dissipative tunneling in a sine-Gordon chain at high temperatures. *Europhys. Lett.*, 13:217, 1990. 60

[JV99] S. Jang and G. A. Voth. Path integral centroid variables and the formulation of their exact real time dynamics. *J. Chem. Phys.*, 111:2357, 1999. 2, 53

[Keh99] S. Kehrein. Flow equation solution for the weak- to strong-coupling crossover in the sine-Gordon model. *Phys. Rev. Lett.*, 83:4914, 1999. 4, 60

[Kin98] K. Kinugawa. Path integral centroid molecular dynamics study of the dynamic structure factors of liquid para-hydrogen. *Chem. Phys. Lett.*, 292:454, 1998. 53

[KM02] F. R. Krajewski and M. H. Müser. Comparison of two non-primitive methods for path integral simulations: Higher-order corrections versus an effective propagator approach. *Phys. Rev. B*, 65:174304, 2002. 2, 3, 11

[KMK97] K. Kinugawa, P.B. Moore, and M. L. Klein. Centroid path integral molecular dynamics simulation of lithium para-hydrogen clusters. *J. Chem. Phys.*, 106:1154, 1997. 53

[LB87] X.-P. Li and J. Q. Broughton. High-order correction to the Trotter expansion for use in computer simulation. *J. Chem. Phys.*, 86:5094, 1987. 2, 9, 11, 23

[LT03] L. S. Levitov and A. M. Tsvelik. Narrow-gap Luttinger liquid in carbon nanotubes. *Phys. Rev. Lett.*, 90:016401, 2003. 3, 60

[LV96] J. Lobaugh and G. A. Voth. The quantum dynamics of an excess proton in water. *J. Chem. Phys.*, 104:2056, 1996. 53

[LV97] J. Lobaugh and G. A. Voth. A quantum model for water: Equilibrium and dynamical properties. *J. Chem. Phys.*, 106:2400, 1997. 53

[ML02] M. H. Müser and E. Luijten. On quantum effects at the liquid fluid transition in helium. *J. Chem. Phys.*, 116:1621, 2002. 12, 14

[MNB95] M. H. Müser, P. Nielaba, and K. Binder. Path-integral Monte Carlo study of crystalline Lennard-Jones systems. *Phys. Rev. B*, 51:2723, 1995. 22

[MOK99] S. Miura, S. Okazaki, and K. Kinugawa. A path integral centroid molecular dynamics study of nonsuperfluid liquid helium-4. *J. Chem. Phys.*, 110:4523, 1999. 53

[MT79] K. Maki and H. Takayama. Quantum-statistical mechanics of extended objects. I. kinks in the one-dimensional sine-gordon system. *Phys. Rev. B*, 20:3223, 1979. 63

[MUR03] M. H. Müser, M. Urbakh, and M. O. Robbins. Statistical mechanics of static and low-velocity kinetic friction. *Adv. Chem. Phys.*, 126:187, 2003. 3, 60

[Müs96] M. H. Müser. The path-integral Monte Carlo of rigid linear molecules in three dimensions. *Molecular Simulation*, 17:131, 1996. 14

[Müs02] M. H. Müser. On new efficient algorithms for PIMC and PIMD. *Comp. Phys. Comm.*, 147:83, 2002. 9

[PC84] E. L. Pollock and D. M. Ceperley. Simulation of quantum many-body systems by path-integral methods. *Phys. Rev. B*, 30:2555, 1984. 12, 13, 29

[PCL$^+$97] M. Paves, S. Chawla, D. Lu, J. Lobaugh, and G. A. Voth. Quantum effects and the excess proton in water. *J. Chem. Phys.*, 107:7428, 1997. 53

[PK83] M. Peyrard and M. D. Kruskal. Kink dynamics in the highly discrete sine-Gordon system. *Physica D*, 14:88, 1983. 74, 75

[Pol88] E. L. Pollock. Properties and computation of the coulomb pair density matrix. *Comput. Phys. Comm.*, 52:49, 1988. 3, 28

[PR80] M. Parrinello and A. Rahman. Chrystall structures and pair potentials: a molecular dynamics study. *Phys. Rev. Lett.*, 45:1196–99, 1980. 17

[PTVF92] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in FORTRAN.* Cambridge University Press, New York, 1992. 113

[RR83] H. De Raedt and B. De Raedt. Applications of the generalized Trotter formula. *Phys. Rev. A*, 28:3575, 1983. 9, 14

[SCP00] H. J. Schulz, G. Cuniberti, and P. Pieri. *Field Theories for Low-Dimensional Condensed Matter Systems: Spin Systems and Strongly Correlated Electrons.* Springer Series in Solid-State Sciences. Springer, Berlin, 2000. 60

[STF] E. K. Sklyanin, L. A. Takhtadzhyan, and L. D. Faddeev. Quantum inverse problem method I. *Theor. Math. Phys.*, 40. 3, 60

[STF79] E. K. Sklyanin, L. A. Takhtadzhyan, and L. D. Faddeev. *Teor. Mat. Fiz.*, 40:194, 1979. 3, 60

[Sto68] R. G. Storer. Path integral calculation of the quantum-statistical density matrix for attractive coulomb forces. *J. Math. Phys.*, 9:964, 1968. 12, 13

[Suz87] M. Suzuki. *Quantum Monte Carlo methods.* Springer Series in Solid-State Sciences. Springer, Berlin, 1987. 14

[TBB83] D. Thirumalai, E. J. Bruskin, and B. J. Berne. An iterative scheme for the evaluation of discretized path integrals. *J. Chem. Phys.*, 79:5063, 1983. 12, 13

[TBMK93] M. E. Tuckerman, B. J. Berne, G. J. Martyna, and M. L. Klein. Efficient molecular dynamics and hybrid Monte Carlo algorithms for path integrals. *J. Chem. Phys.*, 99:2796, 1993. 2, 4, 8, 60

[THB84] D. Thirumalai, R. W. Hall, and B. J. Berne. A path integral Monte Carlo study of liquid neon and the quantum effective pair potential. *J. Chem. Phys.*, 81:2523, 1984. 29

[TI84] M. Takahashi and M. Imada. Monte Carlo calculation of quantum systems II. higher order correction. *J. Phys. Soc. Jpn.*, 53:3765, 1984. 2, 9, 11, 14

[TLH02] P. Tong, B. Li, and B. Hu. Electronic properties of the 1D Frenkel-Kontorova model. *Phys. Rev. Lett.*, 88:046804, 2002. 60

[TM01] M. E. Tuckerman and D. Marx. Heavy-atom skeleton quantization and proton tunneling in "intermediate barrier" hydrogen bonds. *Phys. Rev. Lett.*, 86:4946, 2001. 1

[TT85] R. Tognetti and V. Tognetti. Variational approach to quantum statistical mechanics of nonlinear systems with application to sine-Gordon chains. *Phys. Rev. Lett.*, 55:912, 1985. 63

[Zs99] X. Zotos and P. Prelovšek. *Physics and Chemistry of Materials with Low Dimensional Structures*. Kluwer Academic Publishers, Dordrecht, in press, 1999. 3, 60, 74

Appendix A

The normal mode transformation

The normal mode transformation is an orthogonal transformation which is closely related to the Fourier transformation. If $\{x_j\}$ is a set of N coordinates, the normal mode transformed coordinates $\{\tilde{x}_j\}$ can be calculated in the following way: The complex Fourier transformed coordinates c_k are given by

$$c_k = \sqrt{\frac{1}{N}} \sum_{j=0}^{N-1} e^{2\pi ijk/N} x_j \tag{A.1}$$

and fulfill the relation

$$c_k = c^\star_{-k} = c^\star_{N-k} \; . \tag{A.2}$$

The normal modes $\tilde{x}_k$ are real numbers and are defined over the c_k as

$$\begin{aligned}
c_0 &= \tilde{x}_0 \\
c_1 &= \sqrt{\frac{1}{2}}\left(\tilde{x}_1 - i\tilde{x}_{N-1}\right) \\
c_1 &= \sqrt{\frac{1}{2}}\left(\tilde{x}_2 - i\tilde{x}_{N-2}\right) \\
&\vdots \\
c_{N/2-1} &= \sqrt{\frac{1}{2}}\left(\tilde{x}_{N/2-1} - i\tilde{x}_{N/2+1}\right) \\
c_{N/2} &= \tilde{x}_{N/2} \\
c_{N/2+1} &= \sqrt{\frac{1}{2}}\left(\tilde{x}_{N/2-1} + i\tilde{x}_{N/2+1}\right) \\
&\vdots \\
c_{N-2} &= \sqrt{\frac{1}{2}}\left(\tilde{x}_2 + i\tilde{x}_{N-2}\right) \\
c_{N-1} &= \sqrt{\frac{1}{2}}\left(\tilde{x}_1 + i\tilde{x}_{N-1}\right)
\end{aligned} \tag{A.3}$$

The normal mode transformation is orthogonal

$$\sum_{j=0}^{N-1} x_j^2 = \sum_{k=0}^{N-1} \tilde{x}_k^2 \tag{A.4}$$

and diagonalizes the harmonic chain because

$$\sum_{j=0}^{N-1} (x_j - x_{j+1})^2 = \sum_{k=0}^{N-1} 4\sin^2\left(\frac{\pi}{N}k\right)\tilde{x}_k^2 \ . \tag{A.5}$$

Appendix B

The virial estimator

Several derivations for the virial estimator can be found in the literature. In the original work by Hermann et al.[HBB82], an integration by parts of the partition function is used for the derivation. A simpler derivation is given by David Ceperley which uses rescaled, dimensionless variables. In this work, it was found that writing the partition function in terms of normal mode coordinates leads to a further simplification of the derivation.

Hermann et al. applied the virial theorem to all modes of the imaginary time path, including the center of mass (CMS) mode of the path. This leads to problems if particles are not bound by an external potential, what is reflected in the well known fact that the virial theorem applies only for particles which have trajectories that are confined in a finite region for $t \to \infty$. In a more sophisticated version, the virial theorem is applied to the internal modes of the imaginary time path [Cep95] only which have always bound trajectories due to the kinetic term in the action.

In the following, a derivation of the virial theorem is given for a one particle system in one dimension. A generalization to a N particle system in three dimensions is trivial. The starting point is the partition function of one particle

$$\begin{aligned} Z(\beta) = & \int \prod_{t=0}^{P-1} \left(\sqrt{\frac{m}{2\pi\hbar^2(\beta/P)}} dx^s \right) \\ & \exp\left\{ -\frac{\beta}{P} \sum_{t=0}^{P-1} \left[\frac{m}{\hbar^2(\beta/P)} \left(x^{t+1} - x^t\right)^2 - V\left(x^t\right) \right] \right\} \\ = & \int \prod_{s=0}^{P-1} \left(\sqrt{\frac{m}{2\pi\hbar^2(\beta/P)}} du^s \right) \\ & \exp\left\{ -\sum_{s=1}^{P-1} 4\sin^2\left(\frac{\pi}{P}s\right) \frac{m}{\hbar^2(\beta/P)} \left(u^s\right)^2 - \frac{\beta}{P} V\left(u^0, \dots, u^{P-1}\right) \right\} \end{aligned} \tag{B.1}$$

with the normal mode coordinates $u^s = U_{st} x^t$ defined in Appendix A and the abbreviation

$V\left(u^0,\ldots,u^{P-1}\right)=\sum_{t=0}^{P-1}V\left(x^t\left(u^0,\ldots,u^{P-1}\right)\right)$. Rescaling the coordinates according to

$$\begin{aligned} v^0 &= u^0 \\ v^s &= u^s\sqrt{\frac{m}{\hbar^2\beta/P}} \qquad\qquad s=1,\ldots(P-1) \end{aligned} \tag{B.2}$$

leads to

$$\begin{aligned} Z &= \int\sqrt{\frac{m}{2\pi\hbar^2(\beta/P)}}dv^0\prod_{s=1}^{P-1}\left(\sqrt{\frac{1}{2\pi}}dv^s\right) \\ &\quad \exp\left\{-\sum_{s=1}^{P-1}\left(v^s\right)^2 4\sin^2\left(\frac{\pi}{P}s\right)-\frac{\beta}{P}V\left(v^0,\sqrt{\frac{\hbar^2\beta/P}{m}}v^1,\ldots,\sqrt{\frac{\hbar^2\beta/P}{m}}v^{P-1}\right)\right\} . \end{aligned} \tag{B.3}$$

Please, note that the CMS mode $u^0=(1/\sqrt{P})\sum_{t=0}^{P-1}x^t$ is treated separately from the internal modes. The thermodynamic average of the kinetic energy is

$$\begin{aligned} \langle T\rangle &= \frac{m}{\beta}\frac{d}{dm}\ln(Z) \\ &= \frac{1}{2}\frac{1}{\beta}+\frac{1}{2P}\sum_{s=1}^{P-1}\left\langle u^s\frac{d}{du^s}V\left(u^0,u^1,\ldots,u^{P-1}\right)\right\rangle_{U_{\rm rp}} \\ &=: \left\langle T_{\rm est}^{\rm PA(vir)}\{u^s\}\right\rangle_{U_{\rm rp}} \end{aligned} \tag{B.4}$$

what leads to the virial estimator in the normal mode representation

$$T_{\rm est}^{\rm PA(vir)}\left(\{u^s\}\right)=\frac{1}{2}\frac{1}{\beta}+\frac{1}{2P}\sum_{s=1}^{P-1}u^s\frac{d}{du^s}V\left(u^0,u^1,\ldots,u^{P-1}\right) . \tag{B.5}$$

Eq. (B.5) can be re expressed in terms of the coordinates x^t. For the derivative one finds the relation:

$$\sum_{s=1}^{P-1}u^s\frac{d}{du^s}=\sum_{s=0}^{P-1}u^s\frac{d}{du^s}-u^0\frac{d}{du^0}=\sum_{t=0}^{P-1}x^t\frac{d}{dx^t}-u^0\sum_{t=0}^{P-1}\frac{dx^t}{du^0}\frac{d}{dx^t} . \tag{B.6}$$

In the last step the orthogonality of the normal mode transformation was used. The final result for the virial estimator is

$$T_{\rm est}^{\rm PA(vir)}\left(x^t\right)=\frac{1}{2}\frac{1}{\beta}+\frac{1}{2P}\sum_{t=0}^{P-1}\left(x^t-x_{\rm c}\right)\frac{d}{dx^t}V\left(x^t\right) \tag{B.7}$$

with $x_{\rm c}=(1/P)\sum_{t=0}^{N-1}x^t$.

Appendix C

Natural units for the discrete sine-Gordon chain

For the discrete sine-Gordon chain and related disordered models, the Hamiltonian can be written as

$$\hat{H} = \sum_{j=0}^{N-1} \left(\frac{\hat{p}_j^2}{2m} + \frac{1}{2} K (\hat{x}_j - \hat{x}_{j+1})^2 - V_0 \cos(\hat{x}_j / b) \right) , \tag{C.1}$$

where the $\cos(\hat{x}_j/b)$ term is replaced by a random function $f(\hat{x}_j/b)$ for the disordered systems. The system of units is defined by V_0, b, $\hbar$, and Boltzmann's constant k_B. This leads to the following units for the physical quantities considered in this work:

Energy:	V_0
Action:	$\hbar$
Space:	b
Time:	$\hbar/V_0$
Force:	V_0/b
Mass:	$\hbar^2/(V_0 b^2)$
Temperature:	V_0/k_B

Appendix D

Source code of the HOA pair potential force routine

The Fortran 77 source code of the force routine of the PIMD code for the simulation of crystalline Argon in the NPT ensemble is given in the following. This code was used to obtain the results on crystalline Argon given in chapter 1.

```
      subroutine tak_imada_potential
      include "pimd.com"
      real r_2
      real scal_prod
      real delta_f(n_dim)

      real square_f

      v_inter = 0.
      u_c_tak = 0.
      do i_dim = 1, n_dim+3
       force_scal_inter(i_dim) = 0.
      end do


! ---------- calculate classical Forces ---------------------------------

!     initialize cl_force
      do i_part = 1 , n_part
       do i_trot = 0 , n_trot-1
        do i_dim = 1, n_dim
         cl_force(i_dim,i_trot,i_part) = 0.
        end do
       end do
      end do



!     Sum over all particles
      do i_part = 1,n_part
       i_type = type(i_part)


!     Sum over the particles neighbors
       do i_neigh = 1,neighbor(0,i_part)
        j_part = neighbor(i_neigh,i_part)
        j_type = type(j_part)


!     Sum over Trotter number
        do i_trot = 0,n_trot-1

!     Calculate the square distance r_2 of particles with labels i_part and j_part
!     at Trotter time i_trot
         call calc_distance(r_2,i_trot,i_part,j_part)


!     Check if particles with lables i_part and j_part are closer
!     to each other than the cutoff radius.
         if(r_2.lt.r_cutoff_2(i_type,j_type)) then

!     delta_r is the vector that connects i_part and j_part in
!     physical coordinates.
!     scal_0  is the vector that connects i_part and j_part in
!     scaled coordinates.
!     delta_cell  is the vector with the edges of the simulation box.

          do i_dim = 1, n_dim
           delta_r(i_dim) = scal_0(i_dim)*delta_cell(i_dim)
          end do
          if (e_dim.eq.3) then
           delta_r(1) = delta_r(1)
     &      + scal_0(4)*delta_cell(2) + scal_0(6)*delta_cell(3)
           delta_r(2) = delta_r(2)
     &      + scal_0(5)*delta_cell(3) + scal_0(4)*delta_cell(1)
           delta_r(3) = delta_r(3)
     &      + scal_0(6)*delta_cell(1) + scal_0(5)*delta_cell(2)
          end if


          i_potential=int(n_potential*r_2/r_cutoff_2(i_type,j_type)+0.5)
```

```
          r_dummy = r_pot_d(i_type,j_type,i_potential)


!        calculate classical force and the inteaction with the box
          do i_dim = 1, n_dim
           force_loc(i_dim) =  r_pot_d(i_type,j_type,i_potential) *
     &     delta_cell(i_dim)
           cl_force(i_dim,i_trot,i_part) = - force_loc(i_dim) +
     &     cl_force(i_dim,i_trot,i_part)
           cl_force(i_dim,i_trot,j_part) = + force_loc(i_dim) +
     &     cl_force(i_dim,i_trot,j_part)
           force_scal_inter(i_dim) = force_scal_inter(i_dim) -
     &        force_loc(i_dim) * delta_r(i_dim)
          end do

!---   symmetrized "stress" field
           if (e_dim.eq.3) then
            force_scal_inter(4) = force_scal_inter(4) -
     &      ( force_loc(1)*delta_r(2) + force_loc(2)*delta_r(1) ) / 2
            force_scal_inter(5) = force_scal_inter(5) -
     &      ( force_loc(2)*delta_r(3) + force_loc(3)*delta_r(2) ) / 2
            force_scal_inter(6) = force_scal_inter(6) -
     &      ( force_loc(3)*delta_r(1) + force_loc(1)*delta_r(3) ) / 2
           end if

         end if

        end do ! n_trot
       end do ! i_neigh
      end do ! i_part
!-----------------------------------------------------------------------------


!     Sum over all particles
      do i_part = 1,n_part
       i_type = type(i_part)

!     Sum over Trotter number
       do i_trot = 0,n_trot-1

!     Sum over the particles neighbors
        do i_neigh = 1,neighbor(0,i_part)
         j_part = neighbor(i_neigh,i_part)
         j_type = type(j_part)

!      Calculate Quantum Force

         call calc_distance(r_2,i_trot,i_part,j_part)

         if(r_2.lt.r_cutoff_2(i_type,j_type)) then
          i_potential=int(n_potential*r_2/r_cutoff_2(i_type,j_type)+0.5)

!------------Quantum Force----------------------

!         delta_r connects i_part and j_part in physical Coordinates
          do i_dim = 1, n_dim
           delta_r(i_dim) = scal_0(i_dim)*delta_cell(i_dim)
          end do
          if (e_dim.eq.3) then
           delta_r(1) = delta_r(1)
     &      + scal_0(4)*delta_cell(2) + scal_0(6)*delta_cell(3)
           delta_r(2) = delta_r(2)
     &      + scal_0(5)*delta_cell(3) + scal_0(4)*delta_cell(1)
           delta_r(3) = delta_r(3)
     &      + scal_0(6)*delta_cell(1) + scal_0(5)*delta_cell(2)
          end if

!           Difference of forces/m
          do i_dim = 1 ,n_dim
           delta_f(i_dim)=cl_force(i_dim,i_trot,j_part)/mass(0,j_type)
     &                 -  cl_force(i_dim,i_trot,i_part)/mass(0,i_type)
```

```
          end do

!           Scalar Product
          scal_prod = 0.
          do i_dim = 1, n_dim
           scal_prod = scal_prod
     &      + met_ten_voi(i_dim) * delta_cell(i_dim)* delta_f(i_dim)
          end do
          if (e_dim.eq.3) then
           scal_prod  = scal_prod
     &       + met_ten_voi(4)/2. * delta_cell(1) * delta_f(2)
     &       + met_ten_voi(5)/2. * delta_cell(2) * delta_f(3)
     &       + met_ten_voi(6)/2. * delta_cell(3) * delta_f(1)
     &       + met_ten_voi(4)/2. * delta_cell(2) * delta_f(1)
     &       + met_ten_voi(5)/2. * delta_cell(3) * delta_f(2)
     &       + met_ten_voi(6)/2. * delta_cell(1) * delta_f(3)
          end if

!         Quantum Correction to the force
          scal_prod = scal_prod * r_pot_2d(i_type,j_type,i_potential)

          do i_dim = 1 , n_dim
           force_loc(i_dim)=
     &             + scal_prod * delta_cell(i_dim)
     &             + r_pot_d(i_type,j_type,i_potential)
     &             * delta_f(i_dim) * tak_ima_const

           force(i_dim,i_trot,i_part) = - force_loc(i_dim) +
     &     force(i_dim,i_trot,i_part)
           force(i_dim,i_trot,j_part) = + force_loc(i_dim) +
     &     force(i_dim,i_trot,j_part)
           force_scal_inter(i_dim) = force_scal_inter(i_dim) -
     &     force_loc(i_dim) * delta_r(i_dim)
          end do

!---   symmetrized "stress" field
           if (e_dim.eq.3) then
            force_scal_inter(4) = force_scal_inter(4) -
     &      ( force_loc(1)*delta_r(2) + force_loc(2)*delta_r(1) ) / 2
            force_scal_inter(5) = force_scal_inter(5) -
     &      ( force_loc(2)*delta_r(3) + force_loc(3)*delta_r(2) ) / 2
            force_scal_inter(6) = force_scal_inter(6) -
     &      ( force_loc(3)*delta_r(1) + force_loc(1)*delta_r(3) ) / 2
           end if

!         Classical interaction energy
           v_inter = v_inter + r_pot(i_type,j_type,i_potential)

         end if

        end do ! i_neigh

!         Quantum Correction U_c

!         find cl_force**2
           square_f = 0.
          do i_dim = 1, n_dim
           square_f = square_f
     &      + met_ten_voi(i_dim) * cl_force(i_dim,i_trot,i_part)**2
          end do
          if (e_dim.eq.3) then
           square_f = square_f
     &       + met_ten_voi(4) * cl_force(1,i_trot,i_part)
     &                        * cl_force(2,i_trot,i_part)
     &       + met_ten_voi(5) * cl_force(2,i_trot,i_part)
     &                        * cl_force(3,i_trot,i_part)
     &       + met_ten_voi(6) * cl_force(3,i_trot,i_part)
     &                        * cl_force(1,i_trot,i_part)
          end if
!         quantumcorrection to the force
          u_c_tak = u_c_tak
     &             + tak_ima_const * square_f/ mass(0,i_type)/2.
```

```
!         Add classical Force
          do i_dim = 1 , n_dim
           force(i_dim,i_trot,i_part)=   force(i_dim,i_trot,i_part)
     &      +   cl_force(i_dim,i_trot,i_part)
          end do

       end do   ! i_trot
      end do    ! i_part



      call force_on_scal

      end subroutine

! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !

      subroutine calc_distance(r_2,i_trot,i_part,j_part)
      include "pimd.com"
      real r_2

      do i_dim = 1,n_dim
         delta_cell(i_dim) =
     &        r0(i_dim,i_trot,i_part) - r0(i_dim,i_trot,j_part)
         if (delta_cell(i_dim).gt.0.5) then
            delta_cell(i_dim) = delta_cell(i_dim) - 1.
         else if(delta_cell(i_dim).lt.-0.5) then
            delta_cell(i_dim) = delta_cell(i_dim) + 1.
         end if
      end do
!---- term square
      r_2 = 0.
      do i_dim = 1, n_dim
         r_2 = r_2 + met_ten_voi(i_dim) * delta_cell(i_dim)**2
      end do
      if (e_dim.eq.3) then
         r_2 = r_2
     &        + met_ten_voi(4) * delta_cell(1) * delta_cell(2)
     &        + met_ten_voi(5) * delta_cell(2) * delta_cell(3)
     &        + met_ten_voi(6) * delta_cell(3) * delta_cell(1)
      end if

      end subroutine

! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !

      subroutine force_on_scal
      include "pimd.com"

!---- change transform
      do i_dim = 1,n_dim + e_dim
       force_scal(i_dim) = force_scal(i_dim) + force_scal_inter(i_dim)
      end do

!---  add external pressure
!---  term press
      v_press = v_press + n_trot * press * volume
      do i_dim = 1, n_dim
       force_scal(i_dim) = force_scal(i_dim) -
     & n_trot * press *volume * h_mat_inv(i_dim,i_dim)
      end do
      if (e_dim.eq.3) then
       force_scal(4) = force_scal(4) -
     & n_trot * press *volume * h_mat_inv(1,2)
       force_scal(5) = force_scal(5) -
     & n_trot * press *volume * h_mat_inv(2,3)
       force_scal(6) = force_scal(6) -
     & n_trot * press *volume * h_mat_inv(3,1)
      end if
      end subroutine
! ! ! ! ! ! ! ! ! ! !
```

Appendix E

Source code of the numerical matrix multiplication program

The numerical matrix multiplication Fortran 77 code is given in the following. This code was used to obtain the results given in chapter 1.

```
      program Matrix_Multiplication
      implicit none

      real*8 pi, mass_amu
      parameter (mass_amu=0.02061 )
      parameter (pi=3.1415926536d0)

      real*8 tk, mass, beta, betap, press, volume, ideal_volume
      real*8 lenjo, harm, d_well
      real*8 cur_at_min
      real*8 alpha, besselseries
      integer nm
      integer i_r, j_r, k_r, n_r, i_l, n_l, i_run, n_run, i_min
      integer f_potential, f_tak_ima

!     f_potential = 1: Harmonic Oszillator
!     f_potential = 2: Double well
!     f_potential = 3: Lenard Jones Potential

      parameter (f_potential = 3)
      parameter (f_tak_ima   = 0)

      parameter (n_r=800)

      real*8 r_box

      real*8 r_r(-n_r:n_r), pot_r(-n_r:n_r)
      real*8 pot_rl(-n_r:n_r), dr(-n_r:n_r)
      real*8 g_r_pot(-n_r:n_r), g_r_free(-n_r:n_r), vir_coef2, pot_coef2
      real*8 g_r_pot_l(-n_r:n_r), g_r_free_l(-n_r:n_r)
      real*8 mat(2,-n_r:n_r,-n_r:n_r), norm
      real*8 h_tak_cor(-n_r:n_r), g_r_tak_cor(-n_r:n_r)
      real*8 pot_est(-n_r:n_r)

      integer i_log_trot, n_log_trot, n_trot
      real*8 loc_action, l_debr, l_debr_l

      real sigma, epsilon, joule

      parameter (joule = 7.2432e22)
      parameter (sigma = 3.405)
      parameter (epsilon = 1.67e-21*joule)

      real*8 p_norm, p_norm_2


      r_box = 12.


! --- absolute mass
!      mass = 4 * mass_amu ! He4
!      mass = 40 * mass_amu !Ar40
      mass = 1 ! Harm Oszillator / double well
! --- reduced mass
      if (f_potential .eq. 3 )  mass = mass / 2

      n_run = 5
      do i_run = 2 , n_run
!       tk = 1./8. ! harm oszillator
        tk = 10.

        write(*,*) "tk" , tk
        beta = 1. / tk

        n_trot = 2**i_run
        n_log_trot = i_run


        betap = 1. / (n_trot*tk)
        l_debr = sqrt( 2*pi*betap / mass )
        l_debr_l = sqrt(2*pi/mass/tk)
        alpha = 1./12. * 1./(mass) * betap**2.
```

```
!---   initialize r_r(i_r), dr(i_r)
         do i_r = -n_r , n_r
            r_r(i_r) =  real(i_r)/n_r  * r_box
            dr(i_r) =  r_box /n_r
            g_r_pot_l(i_r) = 0.
            g_r_free_l(i_r) = 0.
         end do


!---   Harmonic potential
         if (f_potential.eq.1) then
            do i_r = -n_r , n_r
               pot_r(i_r) = harm(r_r(i_r),0)
               if ( f_tak_ima .eq. 1 ) then
                  pot_r(i_r) = pot_r(i_r) + alpha/2. *
     &                 harm(r_r(i_r),1)**2
               end if
            end do
         end if

!---   Double well potential
         if (f_potential.eq.2) then
            do i_r = -n_r , n_r
               pot_r(i_r) = d_well(r_r(i_r),0)
               if ( f_tak_ima .eq. 1 ) then
                  pot_r(i_r) = pot_r(i_r) + alpha/2. *
     &                 d_well(r_r(i_r),1)**2
               end if
            end do
         end if


!---   Lennard Jones potential
         if (f_potential.eq.3) then
            do i_r = -n_r , n_r
               pot_r(i_r) = lenjo(r_r(i_r),0)
               if ( f_tak_ima .eq. 1 ) then
                  pot_r(i_r) = pot_r(i_r) + alpha/2.*
     &                 lenjo(r_r(i_r),1)**2
               end if
          end do
        end if


        write(*,*) "eff. Trot #", n_trot
        write(*,*) "radial incr. / max. r:" , dr(3), r_r(n_r)
        write(*,*) "de Br. w.l.  / real l:", l_debr, l_debr_l

!       init High Temp density Matrix
        do i_r = -n_r , n_r
          do j_r =  -n_r , n_r
             loc_action =  betap * (pot_r(i_r)+pot_r(j_r)) / 2
             loc_action = loc_action +
     &            mass * (r_r(i_r) -  r_r(j_r))**2 / (2*betap)
             mat(2,i_r,j_r) = dexp(-loc_action)/l_debr
          end do
        end do


! ---   compute low-T dens mat by squaring
        do i_log_trot = 1, n_log_trot
          do j_r = -n_r, n_r
             do i_r = -n_r, n_r
                mat(1,i_r,j_r) = mat(2,i_r,j_r)
                mat(2,i_r,j_r) = 0.d0
             end do
          end do
          do i_r = -n_r, n_r
             do k_r = -n_r, n_r
                if (mat(1,i_r,k_r).gt.0.) then
                   do j_r = -n_r, n_r
```

```
                     mat(2,i_r,j_r) = mat(2,i_r,j_r) +
     &                    mat(1,i_r,k_r) * mat(1,k_r,j_r)* dr(k_r)
                  end do
               end if
            end do
         end do
      end do

!---   save diagonal elements
      do i_r = -n_r, n_r
         g_r_pot_l(i_r) = mat(2,i_r,i_r)
      end do


!---   init free high-T dens mat for i_l
      do i_r = -n_r, n_r
         do j_r =  -n_r, n_r
            loc_action =
     &           mass * (r_r(i_r) -  r_r(j_r))**2 / (2*betap)
            mat(2,i_r,j_r) = dexp(-loc_action)/l_debr
         end do
      end do

!---   compute low-T dens mat by squaring
      do i_log_trot = 1, n_log_trot
         do j_r = -n_r, n_r
            do i_r = -n_r, n_r
               mat(1,i_r,j_r) = mat(2,i_r,j_r)
               mat(2,i_r,j_r) = 0.d0
            end do
         end do
         do i_r = -n_r, n_r
            do k_r = -n_r, n_r
               if (mat(1,i_r,k_r).gt.0.) then
                  do j_r = -n_r, n_r
                     mat(2,i_r,j_r) = mat(2,i_r,j_r) +
     &                    mat(1,i_r,k_r) * mat(1,k_r,j_r)* dr(k_r)
                  end do
               end if
            end do
         end do
      end do

!---   save diagonal elements
      do i_r = -n_r, n_r
         g_r_free_l(i_r) = mat(2,i_r,i_r)
      end do


! --- calculate HOA correction to g_r
      do i_r = -n_r , n_r

         if (f_potential .eq. 1) then
            h_tak_cor(i_r) = g_r_pot_l(i_r) *
     &           harm(r_r(i_r),1)
         else if (f_potential .eq. 2) then
            h_tak_cor(i_r) = g_r_pot_l(i_r) *
     &           d_well(r_r(i_r),1)
         else if (f_potential .eq. 3) then
            if ( i_r .ne. 0 ) then
               h_tak_cor(i_r) = g_r_pot_l(i_r) *
     &              lenjo(r_r(i_r),1)
            else
               h_tak_cor(i_r) =0.
            end if
         end if
      end do

      do i_r =  -n_r+1 , n_r-1
         if ( (i_r**2 .gt.  1) .or. f_potential .ne. 3 ) then
            g_r_tak_cor(i_r) = - alpha *
     &           ( h_tak_cor(i_r+1) - h_tak_cor(i_r-1))
```

```
     &         /( r_r(i_r+1)- r_r(i_r-1) )
         else
            g_r_tak_cor(i_r) = 0.
         end if
      end do

      do i_r = 0, 0
       g_r_tak_cor( -n_r + i_r) = 0.
       g_r_tak_cor(  n_r - i_r) = 0.
      end do

! ---- output of g_r
      do i_r = -n_r , n_r
         if ( f_tak_ima .eq. 0 ) then
            write(50+i_run,*) r_r(i_r),
     &           g_r_free_l(i_r) * l_debr_l,
     &           g_r_pot_l(i_r) * l_debr_l,
     &           dr(i_r)
         else if (f_tak_ima .eq. 1) then
            write(50+i_run,*) r_r(i_r),
     &           g_r_free_l(i_r) * l_debr_l
     &           ,g_r_pot_l(i_r) * l_debr_l
     &           ,g_r_tak_cor(i_r) * l_debr_l
         end if
      end do



! --- calculate vir_coeff2 , pot_coef2
      vir_coef2 = 0.
      pot_coef2 = 0.
      p_norm   = 0.
      p_norm_2 = 0.

      do i_r = -n_r  , n_r

         if (f_potential .eq. 1) then
            pot_est(i_r) = harm(r_r(i_r),0)
            if ( f_tak_ima .eq. 1 ) then
               pot_est(i_r) = pot_est(i_r) + alpha *
     &              harm(r_r(i_r),1)**2
            end if
         else if (f_potential .eq. 2) then
            pot_est(i_r) = d_well(r_r(i_r),0)
            if ( f_tak_ima .eq. 1 ) then
               pot_est(i_r) = pot_est(i_r) + alpha *
     &              d_well(r_r(i_r),1)**2
            end if
         else if (f_potential .eq. 3) then
            if (i_r .ne. 0 ) then
               pot_est(i_r) = lenjo(r_r(i_r),0)
               if ( f_tak_ima .eq. 1  ) then
                  pot_est(i_r) = pot_est(i_r) + alpha *
     &                 lenjo(r_r(i_r),1)**2
               end if
            end if
         end if

         if ( f_potential .eq. 3 ) then
            pot_coef2 = pot_coef2 +
     &           ( g_r_pot_l(i_r) - g_r_free_l(i_r) + 1 ) ! free potentials
     &           * pot_est(i_r) * dr(i_r)
            vir_coef2 = vir_coef2 +
     &           ( g_r_pot_l(i_r) - g_r_free_l(i_r) ) * dr(i_r)
         else
            pot_coef2 = pot_coef2 +
     &           g_r_pot_l(i_r) ! bound potentials
     &           * pot_est(i_r) * dr(i_r)
         end if


         p_norm = p_norm +  g_r_pot_l(i_r)* dr(i_r)
```

```
          if ( f_tak_ima  .eq. 1 ) then
             vir_coef2 = vir_coef2 + g_r_tak_cor(i_r) * dr(i_r)
             p_norm_2 = p_norm_2   + g_r_tak_cor(i_r) * dr(i_r)
          end if

        end do

        if ( f_potential .eq. 3 ) then
           write(20,*) i_run, 2**i_run,
     &          -1. * vir_coef2/2. ,
     &          pot_coef2/2. ,p_norm, p_norm_2
        else
           write(20,*) i_run, 2**i_run,
     &          pot_coef2/2. ,p_norm, p_norm_2
        end if

       end do

       end program

! ! ! ! ! ! ! ! ! !

       real*8 function lenjo(r,der)
       implicit none

       real*8 r, cutoff, v0, pi
       real*8 sigma, epsilon, joule
       integer der
       parameter ( cutoff = 6 )
       parameter (pi=3.1415926536d0)

       parameter (joule = 7.2432e22)

!      Argon
       parameter (sigma = 3.405)
       parameter (epsilon = 1.67e-21*joule)

!      Helium
!       parameter (sigma = 2.556)
!       parameter (epsilon = 10.22)

       v0 =  4 * epsilon * ( (sigma/cutoff)**6 - 1 )
     &        * (sigma/cutoff)**6

       if (der .eq. 0 ) then
        lenjo =   4 * epsilon * ( (sigma/r)**6 - 1 )* (sigma/r)**6
        lenjo = lenjo - v0
       else if ( der .eq. 1 ) then
        lenjo = -24./r * epsilon * ( 2*(sigma/r)**6 - 1)* (sigma/r)**6
       end if

       if (r**2 .gt. cutoff**2) then
        lenjo = 0.
       end if

       end function

! ! ! ! ! ! ! ! ! !


       real*8 function harm(r,der)
       implicit none

       real*8 r
       real*8 k
       integer der

       parameter (k = 1./20.)

       if ( der .eq. 0 ) then
        harm = 1. / 2. * k * r**2
       else if (der .eq. 1) then
        harm =  k * r
       end if

       end function

! ! ! ! ! ! ! ! ! !


       real*8 function d_well(r,der)
       implicit none

       real*8 r
       integer der


       if ( der .eq. 0 ) then
        d_well = - 1. / 2. * r**2 + 1./4. * 1. / 5. * r**4
       else if (der .eq. 1) then
        d_well =  -  r + 1./5. * r**3
       end if

       end function

! ! ! ! ! ! ! ! ! !
```

Appendix F

Source code for the numerical summation of the eigenmodes

In the following the Mathematica source code for the calculations presented in chapter 2 is given.

```
Aa[p_] := 2 Tanh[f[p]/2] /f[p]
Cc[p_] := 2 / f[p]^2 ( 1-  f[p]/ Sinh[f[p]] )
f[p_] := hbar Sqrt[2 k /m] beta / p

kst[q_,om_,p_,n_] := 4 ( m p^2 / beta^2 hbar^2 Sin[Pi/p om]^2            \
                                     + k Sin[Pi/n q]^2 )

ktak[q_,om_,p_,n_] := 4 ( m p^2 / beta^2 hbar^2   Sin[Pi/p om]^2           \
                          + ( k+ 1/3 hbar^2 / m  (beta/p)^2 k^2  )         \
                                                  Sin[Pi/n q]^2            \
                           -  1/12 hbar^2 / m  (beta/p)^2 k^2              \
                                                  Sin[Pi/n 2 q]^2 )

knd[q_,om_,p_,n_] :=  4( ( m p^2 / beta^2 hbar^2 + - k Cc[p] )             \
                                            Sin[Pi/p om]^2                 \
                         + ( k + k ( Aa[p]-Cc[p]-1) )                      \
                                            Sin[Pi/n q]^2                  \
                         + k Cc[p] /2 (     Sin[Pi  ( q/n - om / p)]^2     \
                                          + Sin[Pi  ( q/n + om / p)]^2 ) )

rknd[q_,om_,p_,n_] :=  4( m p^2 / beta^2 hbar^2 Sin[Pi/p om]^2             \
                           +  k  Aa[p]  Sin[Pi/n q]^2 )


values = { n -> 5 , k -> 1 , m -> 1 , beta -> 64 , hbar -> 1 }


stsummand[q_,om_,p_,n_]  := 4 Sin[Pi/n q]^2 k / (2 beta) / kst[q,om,p,n]

ndsummand[q_,om_,p_,n_]  := 4 Sin[Pi/n q]^2 k / (2 beta) / knd[q,om,p,n]
```

```
rndsummand[q_,om_,p_,n_] := 4 Sin[Pi/n q]^2 k / (2 beta) /rknd[q,om,p,n]

taksummand[q_,om_,p_,n_] := D[ktak[q,om,p,n],k] * k                        \
                                  / ( 2 beta) / ktak[q,om,p,n]



sts[p_,n_]  := Sum[Sum[N[  stsummand[q,om,p,n]  /. values ],               \
                       {om,1,p}],{q,1,n-1}]/.values

taks[p_,n_] := Sum[Sum[N[ taksummand[q,om,p,n]  /. values ],               \
                       {om,1,p}],{q,1,n-1}]/.values

nds[p_,n_]  := Sum[Sum[N[ ndsummand[q,om,p,n]  /. values ],                \
                       {om,1,p}],{q,1,n-1}]/.values

rnds[p_,n_]  := Sum[Sum[N[ rndsummand[q,om,p,n]  /. values ],              \
                       {om,1,p}],{q,1,n-1}]/.values

stream=OpenWrite["result.out"]
Do[ WriteString[stream,FortranForm[i], "\t", FortranForm[ sts[i,5]] ,"\t",  \
                                             FortranForm[ taks[i,5]],"\t",  \
                                             FortranForm[ nds[i,5]] ,"\t",  \
                                             FortranForm[ rnds[i,5]],"\n" ],\
   {i,1,1000,5}]
Close[stream]
```

Appendix G

Source code of the path integral molecular dynamic program

The Fortran 90 source code for the path integral molecular dynamics simulation is given in the following. This program was used for the calculation of the results given in the chapters 3, 4, 5 and 6. The random number generator and the fast Fourier transform are take from [PTVF92].

G.1 Main simulation routine

```
include "modules.f90"
include "normal_mode_trafo.f90"
include "md_routines.f90"
include "observation.f90"
include "random.f90"

!===============================================================================
!----------------- Main Program ------------------------------------------------
!===============================================================================
program harm_chain
  use input_parameters
  use observation_data
  use part_coordinates ! test test
  use r250_data
  implicit none
  integer :: iseed
  logical :: stop_flag

  real x_warm_up
  integer i_warm_up
  integer :: count, c1, c2, crate, cmax

  integer :: i_ex_force , n_ex_force                        !--driving Force
  real(kind=dp) :: d_ex_force, ex_force_start               !--driving Force

  !-----------------------------------------------------------------------------
  ! For the non equilibrium simulation with external driving potential      !
  ! all comment lines which have !--driving Force in the end                !
  ! must be commented in.                                                   !
  ! For equilibrium simulations they have to be commented out.              !
  ! For then calculation of the reaction of a systen due to a small driving !
  ! force all lines with !--displacement must be commented in. For          !
  ! equilibrium cacluations they have to be commented out as well           !
  ! These lines are  only in the main program.                              !
  !-----------------------------------------------------------------------------


  ! read harm_chain.inp
  call read_input

  !-----------------------------------------------------------------------------
  ! define the number of threads used for openMP                             !
  ! this statement will be ignored if                                       !
  ! the compiler does not support openMP                                     !
  !$  call omp_set_num_threads(n_thred)                                     !
  ! ----------------------------------------------------------------------------

  !  initialize random number generator
  iseed = seed
  KPTR = 1
  call INR250(MZ,ISEED,KPTR)

  !-----------------------------------------------------------------------------
  !r250 warm up                                                             !
  do i_warm_up =  0 , 10000                                                 !
     call r250(mz,x_warm_up,1,kptr)                                         !
  end do                                                                    !
  !-----------------------------------------------------------------------------

  !-----------------------------------------------------------------------------
  ! read conf.inp or initializs  x,d,n_win with default values              !
  if (f_conf == 0 ) then                                                    !
     call def_conf ! needs n_vol , n_winding, f_potential from read_input   !
  else if (f_conf == 1 ) then                                               !
     call io_conf(0)  ! read configuration                                  !
  else                                                                      !
     stop 'f_conf makes no sense'                                           !
  end if                                                                    !
  !-----------------------------------------------------------------------------
```

```
  call init_observation
  call init_normal_mode ! needs d, and n_win from def_conf or io_conf
  call init_md_data
  call trans_recip
  call init_derivatives


  ! initialize ex_force !!!!! important for eq. calc. !!!!!!
  ex_force = 0.
  !---------------------------------------------------------------------------
  !d_ex_force       = 0.01                                 !--driving Force !
  !ex_force_start   = 0.                                   !--driving Force !
  !n_ex_force = 40                                         !--driving Force !
  !do i_ex_force =  0 , n_ex_force                         !--driving Force !
  !ex_force   =  ex_force_start +  i_ex_force * d_ex_force !--driving Force !
  !---------------------------------------------------------------------------

  !-----------------------------------------------------------------------------
  !-----------------------------------------------------------------------------
  ! main simulation loop
  do i_md = 1 , n_relax+n_obs ! main simulation loop


     !goto 701                                              !-- driving Force


     !-----------------------------------------------------------------------
     !  timing                                                              !
     if ( i_md .eq. 400 ) call system_clock( count=c1, count_rate=crate, &  !
          & count_max=cmax )                                                !
     if ( i_md .eq. 900 )  then                                             !
        call system_clock( count=c2, count_rate=crate, count_max=cmax )     !
        performance = (c2-c1)/real(crate)                                   !
     end if                                                                 !
     !-----------------------------------------------------------------------

701  continue

     call predict  ! Predictor of the Gear Predictor corrector routine

     !-----------------------------------------------------------------------
     ! claculate Force on the u(i_trot,i_part)                              !
     call trans_real                                                        !
     call calc_force                                                        !
     call friction_force                                                    !
     !-----------------------------------------------------------------------

     call correct ! Correctop of the Gear Predictor corrector routine



     !-----------------------------------------------------------------------
     ! Observation                                                          !
     if ( (mod(i_md,obs_step) == 0) .and. (i_md > n_relax) ) then           !
        !goto 702                                        !--dirving Force   !
        n_o = n_o + 1                                                       !
        call trans_real ! observe corrected coordinates                     !
        call transform_cl_force                                             !
        call observation                                                    !
702     continue                                                            !
                                                                            !
        !write(23,fmt='(4e20.8))') dt * (i_md-n_relax), & !--driving Force   !
        !     &  u(0,0,1)/dt,  u(0,0,0), ex_force         !--driving Force   !
        !ex_force = 0.05                                  !--displacement  !
     end if                                                                !
     !-----------------------------------------------------------------------


     !-----------------------------------------------------------------------
     ! write state file                                                     !
     if ( mod(i_md , max( (n_relax + n_obs)/ 100  , 1) ) == 0 ) then        !
```

```
        open ( unit=10, file="state.out",status="unknown" )               !
        write(unit=10,fmt='(a17,2i20)') 'n_relax , n_obs  ', n_relax, n_obs !
        write(unit=10,fmt='(a4,i20)') 'i_md' , i_md                       !
        write(unit=10,fmt='(a15,i20)') 'remaining steps' , &              !
             &    n_relax + n_obs - i_md                                  !
        write(unit=10,fmt='(a14,f4.0,a1)') 'calculated    ' , &           !
             & (real(i_md,kind=dp)*100.)/(n_relax + n_obs), "%"           !
        close(10)                                                         !
        inquire (file="stop.end" , EXIST = stop_flag )                    !
        if ( stop_flag ) exit                                             !
     end if                                                               !
     !---------------------------------------------------------------------


  end do ! i_md
  !end of the main simulation loop
  !--------------------------------------------------------------------------
  !--------------------------------------------------------------------------
  !end do ! i_ex_force                                    !--driving Force

  call trans_real
  call io_conf(1)   ! write configuration conf.end

  !goto 703                                               !--driving Force

  call observation_out
  call write_output ! write output file harm_chain.out
                    ! must be called after observation_out

703 continue

end program harm_chain
```

G.2 Input/Output, default configuration and substrate potential

```
subroutine read_input
  use constants
  use input_parameters
  use part_coordinates
  implicit none

  open ( unit=10, file="harm_chain.inp",status="old" )

  read(unit=10,fmt=*)
  read(unit=10,fmt='(i20)') n_relax
  read(unit=10,fmt='(i20)') n_obs
  read(unit=10,fmt='(E20.8)') tk
  read(unit=10,fmt='(E20.8)') mass
  read(unit=10,fmt='(E20.8)') k
  read(unit=10,fmt='(19xi1)') f_conf
  read(unit=10,fmt=*)
  read(unit=10,fmt=*)
  read(unit=10,fmt='(i20,a40)') n_vol
  read(unit=10,fmt='(i20,a40)') n_winding
  read(unit=10,fmt='(i20,a40)') f_roughness
  read(unit=10,fmt='(i20,a40)') f_rand_phase

  read(unit=10,fmt='(19xi1)') f_potential
  read(unit=10,fmt=*)
  read(unit=10,fmt=*)
  read(unit=10,fmt=*)
  read(unit=10,fmt=*)
  read(unit=10,fmt=*)
  read(unit=10,fmt='(E20.8)') pot_strength
  read(unit=10,fmt=*)
```

```
read(unit=10,fmt='(19xi1)') f_HOA
read(unit=10,fmt=*)
read(unit=10,fmt=*)
read(unit=10,fmt='(19xi1)') f_decomp
read(unit=10,fmt=*)
read(unit=10,fmt='(19xi1)') f_cms
read(unit=10,fmt=*)

read(unit=10,fmt='(E20.8)') x_cms
read(unit=10,fmt='(19xi1)') f_fic_mass
read(unit=10,fmt=*)
read(unit=10,fmt='(E20.8)') v_renorm
read(unit=10,fmt='(E20.8)') omega_sample

read(unit=10,fmt=*)
read(unit=10,fmt='(E20.8)') dt
read(unit=10,fmt='(E20.8)') friction
read(unit=10,fmt='(i20)') seed
read(unit=10,fmt='(i20)') obs_step
read(unit=10,fmt='(i20)') n_thred

read(unit=10,fmt='(i20)') n_trot
read(unit=10,fmt='(i20)') n_part
read(unit=10,fmt='(i20)') n_order

close(10)

if (n_trot .gt. n_trot_max) then
   stop "n_trot is to big"
end if

if (n_part .gt. n_part_max  ) then
   stop "n_part is to big"
end if

if (n_order .gt. n_order_max  ) then
   stop "n_order is to big"
end if

if (n_vol .gt. n_vol_max  ) then
   stop "n_vol is to big"
end if


beta = 1/ tk
beta_p = beta / n_trot
dt_2 = dt**(2.d0)
in_dt = 1.d0 / dt
c_hoa = 1.d0 / 12.d0 * 1.d0 / mass * beta_p**2

if ( obs_step <= 0 )  stop 'obs. step makes no sense'

do i_part = 0 , n_part-1
   if ( f_decomp == 0 ) then
      f(i_part) = 0.d0
      A_QV(i_part) = 1.d0
      B_QV(i_part) = 1.d0
      dA_QV(i_part) = 0.d0
      dB_QV(i_part) = 0.d0
   else if ( f_decomp == 1 ) then
      f(i_part) = beta_p *  2 * Sin( ( pi/ real(n_part) ) * i_part ) &
                       & * (  k / mass )**.5d0
      if ( f(i_part) == 0.d0 ) then
         A_QV(i_part) = 1.d0
         B_QV(i_part) = 1.d0
         dA_QV(i_part) = 0.d0
         dB_QV(i_part) = 0.d0
      else
         A_QV(i_part) = f(i_part) / sinh(f(i_part))
         B_QV(i_part) = 2.d0 * tanh(f(i_part)/2.d0) / f(i_part)
         dA_QV(i_part) = (1 - f(i_part) / tanh(f(i_part)) ) / sinh(f(i_part))
         dB_QV(i_part) = ( f(i_part) - sinh(f(i_part)) ) / &
```

```
              & ( f(i_part)**2 * cosh(f(i_part)/2.d0)**2 )
        end if
     else
        stop ' f_decomf makes no sense '
     end if
  end do

  if ( (f_hoa .eq. 2)  .and. (f_decomp .eq. 1 ) ) then
     stop 'f_decomp = 1 and f_hoa =2  makes no sense '
  end if

  x_cms = x_cms * (n_part * n_trot )** .5d0

end subroutine read_input
!===============================================================================
subroutine write_output
  use constants
  use input_parameters
  use part_coordinates
  use md_data
  use observation_data
  use normal_mode_data
  use functions
  implicit none
  real(kind=dp) disorder_para


  open ( unit=10, file="harm_chain.out",status="unknown" )

  write(unit=10,fmt=*) "------Read DATA--------------------"
  write(unit=10,fmt='(i20,a30)') min( n_relax, i_md) ,   &
       &            "      Relaxation Steps       "
  write(unit=10,fmt='(i20,a30)') max(min(n_obs ,i_md-n_relax),0), &
       &            "      Observation Steps      "
  write(unit=10,fmt='(E20.8,a30)') tk,  &
       &            "      Temperature [K]        "
  write(unit=10,fmt='(E20.8,a30,E20.8,a4)') mass, &
       &            "      Mass               ( = ",  mass/0.02062 ,"amu)"
  write(unit=10,fmt='(E20.8,a34)') k, &
       &            "      Spring Constant  [k_B K/A^2]"
  write(unit=10,fmt='(a19,i1,a30)') "                    ",f_conf, &
       &            "      0 : default conf       "
  write(unit=10,fmt='(a51)')   &
       &            "                    1 : read conf.sta       "
  write(unit=10,fmt=*) "--- Data for def configuration    "
  write(unit=10,fmt='(i20,a40)') n_vol,  &
       &            "       d = 2 pi n_vol, for def. conf.   "
  write(unit=10,fmt='(i20,a40)') n_winding, &
       &            "       n_winding                        "
  write(unit=10,fmt='(i20,a50)') f_roughness, &
       &            "       f_roughness  0: alpha=0; 1: alpha =.5      "
  write(unit=10,fmt='(i20,a50)') f_rand_phase, &
       &            "       f_rand_phase 0: det phase , 1:rand phase   "
  write(unit=10,fmt='(a19,i1,a30)') "                    ",f_potential, &
       &            "      0 : no potential       "
  write(unit=10,fmt='(a51)')   &
       &            "                    1 : harm. osz.          "
  write(unit=10,fmt='(a51)')   &
       &            "                    2 : - cos( x )          "
  write(unit=10,fmt='(a51)')   &
       &            "                    3 : (x^2-1)^2           "
  write(unit=10,fmt='(a51)')   &
       &            "                    4 : random              "
  write(unit=10,fmt='(a51)')   &
       &            "                    5 : (x-i_part 2 pi)^2   "
  write(unit=10,fmt='(E20.8,a34)') pot_strength,  &
       &            "      Potential strength         "
  write(unit=10,fmt=*) "------Algorithm------------------"
  write(unit=10,fmt='(a19,i1,a30)') "                    ", f_HOA, &
       &            "      0 : Primitive Algorithm"
  write(unit=10,fmt='(a51)')   &
       &            "                    1 : use HOA for ex. pot "
  write(unit=10,fmt='(a51)')   &
       &            "                    2 : use HOA for all     "
  write(unit=10,fmt='(a19,i1,a30)') &
       &            "                    ", f_decomp, &
       &            "      0 : T V Decomposition  "
  write(unit=10,fmt='(a51)')   &
       &            "                    1 : Q V Decomposition   "
  write(unit=10,fmt='(a19,i1,a30)')                 &
       &            "                    ", f_cms, &
       &            "      0 : x_cms moves        "
  write(unit=10,fmt='(a51)')   &
       &            "                    1 : x_cms is fixed      "
  write(unit=10,fmt='(E20.8,a34)') x_cms,    &
       &            "      x_cms                      "
  write(unit=10,fmt='(a19,i1,a30)') "                    ", f_fic_mass, &
       &            "      0 : mass for coll. h.c."
  write(unit=10,fmt='(a51)')    &
       &            "                    1 : cent. dynamic. mass."
  write(unit=10,fmt='(E20.8,a34)') v_renorm, &
       &            "      ren. gap for fic. masses   "
  write(unit=10,fmt='(E20.8,a34)') omega_sample, &
       &            "      sampling frequency         "

  write(unit=10,fmt=*) "Simulation Parameters:"
  write(unit=10,fmt='(E20.8,a30,E20.8,a2)') dt,   &
       &            "      Time Step          ( = ", dt*7.6386d-12,"s)"
  write(unit=10,fmt='(E20.8,a30)') friction, "      Friction                "
  write(unit=10,fmt='(i20,a31)') seed,      &
       &            "       issed (random gen. init)     "
  write(unit=10,fmt='(i20,a30)') obs_step,  &
       &            "       Observation Step             "
  write(unit=10,fmt='(i20,a30)') n_thred,   &
       &             "       Number of Threads            "

  write(unit=10,fmt='(i20,a38)') n_trot,    &
       &             "       Trotter Number          ( 2^x )"
  write(unit=10,fmt='(i20,a38)') n_part,    &
       &             "       Number of Particles     ( 2^x )"
  write(unit=10,fmt='(i20,a38)') n_order,   &
       &             "       Order of the Integrator ( <=5 )"
  write(unit=10,fmt=*) "------Calculated Results----------"
  !-- classical kinetic Energy
  t_class  = t_class  / n_o
  t_class2 = t_class2 / n_o - t_class**2
  t_class2 = t_class2**.5d0
  write(unit=10,fmt='(2E20.8,a40)')  t_class , t_class2 , &
       &"      class. Kinetic Energy                "
  !-- harmonic Energy
  e_harm  = e_harm  / n_o
  e_harm2 = e_harm2 / n_o - e_harm**2
  e_harm2 = e_harm2**.5d0
  !-- QT decomposition
  if (f_decomp .eq.1 ) then
     do i_part = 0, n_part-1
        if (f(i_part) .ne. 0 ) then
           e_harm = e_harm                                              &
                &   + .25d0 * ( f(i_part) / Tanh(f(i_part)) - 1  ) &
                &   / (beta_p * n_part)
        end if
     end do
  end if
  write(unit=10,fmt='(2E20.8,a40)')  e_harm , e_harm2 , &
       &"      Pot. Energy in the springs           "
  write(unit=10,fmt='(2E20.8,a40)')  k / 2.d0 * d**2 * c_const / n_part  , 0 , &
       &"      minimal Pot. Energy in the springs"
  !-- interaction Energy with external Potential
  v_ex_pot  = v_ex_pot  / n_o
  v_ex_pot2 = v_ex_pot2 / n_o - v_ex_pot**2
  v_ex_pot2 = v_ex_pot2**.5d0
  write(unit=10,fmt='(2E20.8,a50)')  v_ex_pot, v_ex_pot2 , &
       &"      Pot. Energy with external potential          "
  write(unit=10,fmt=*)
```

```
!-- kinetic Energy
!-- primitive estimator
t_qm  = t_qm  / n_o
t_qm2 = t_qm2 / n_o - t_qm**2
t_qm2 = t_qm2**.5d0
!-- TV decomposition
if (f_decomp .eq. 0 ) then
   t_qm = t_qm + t_class
else if (f_decomp .eq.1 ) then
   !-- QT decomposition
   do i_part = 0, n_part-1
      if (f(i_part) .ne. 0 ) then
         t_qm = t_qm   &
              &  + .25d0 *  f(i_part) / Tanh(f(i_part))  / (beta_p * n_part)
      else
         t_qm = t_qm   + 0.25d0 / (beta_p *n_part)
      end if
   end do
   t_qm = t_qm + t_class / 2.d0
end if
write(unit=10,fmt='(2E20.8,a50)')  t_qm , t_qm2 , &
     &"      Kinetic Energy (Primitive Estimator)        "
!-- virial estimator
t_qm_vir  = t_qm_vir  / n_o
t_qm_vir2 = t_qm_vir2 / n_o - t_qm_vir**2
t_qm_vir2 = t_qm_vir2**.5d0
t_qm_vir  = t_qm_vir + t_class / n_trot
if (f_decomp .eq.1 ) then
   do i_part = 0, n_part-1
      if (f(i_part) .ne. 0 ) then
          t_qm_vir = t_qm_vir                                    &
               &    + .25d0 * ( f(i_part) / Tanh(f(i_part)) - 1  )  &
               &                  / (beta_p * n_part)
      end if
   end do
end if
write(unit=10,fmt='(2E20.8,a50)')  t_qm_vir , t_qm_vir2 , &
     &"      Kinetic Energy (Virial Estimator)           "



write(unit=10,fmt=*)

!-- HOA correction
if (f_hoa == 0 ) then
   U_c_HOA = 0.d0
   U_c_HOA2 = 0.d0
else if (f_hoa .ge. 1 ) then
   U_c_HOA = U_c_HOA * c_HOA  / 2.d0
   U_c_HOA2 = U_c_HOA2 * (c_HOA  / 2.d0)**2.
   U_c_HOA = U_c_HOA / n_o
   U_c_HOA2 = U_c_HOA2 / n_o - U_c_HOA**2
   U_c_HOA2 = U_c_HOA2**.5d0
end if
write(unit=10,fmt='(2E20.8,a50)')  U_c_HOA , U_c_HOA2 , &
     &"      Correction to Pot(*2) and Kin(*1) Energy    "



write(unit=10,fmt='(E20.8,a50)')  real(n_top)/(n_part * n_o )  , &
     &"      prob. for a Particle to be on top of the potential "

 disorder_para = abs(  mod_2pi(hull_function(0)) - pi)
 do i_part = 0 , n_part -1
    disorder_para =  &
         & min( abs(disorder_para), abs( mod_2pi(hull_function(i_part)) - pi))
 end do
write(unit=10,fmt='(E20.8,a50)')  disorder_para  , &
     &"      Fisher's disorder parameter                      "

gyration_radius  = gyration_radius  / n_o
gyration_radius2 = gyration_radius2 / n_o - gyration_radius**2
```

```
  gyration_radius2 = gyration_radius2**.5d0
  write(unit=10,fmt='(2E20.8,a25)')  &
       & gyration_radius ,gyration_radius2,   &
       & "      Gyration radius    "

  write(unit=10,fmt='(E20.8,a50)')  performance , &
       &"      Time needed from i_md = 400 to i_md = 900           "



  close(10)
end subroutine write_output
!-------------------------------------------------------------------------------
subroutine def_conf
  use constants
  use part_coordinates
  use input_parameters
  use r250_data
  implicit none
  real  ran_loc
  integer i_vol, ud_sign, i_dummy

  !  Volumen
  d = n_vol * 2.d0 * pi
  !  part coordinates
  do i_part = 0 , n_part-1
     do i_trot = 0 , n_trot-1
        x(i_trot,i_part) =    ( i_part * d / n_part ) * n_winding
     end do
  end do


  !-- initialize random Potential
  if ( f_potential .eq. 4 ) then
     if (f_roughness .eq. 0 ) then

        i_vol = 0
33      continue
        call r250(mz,ran_loc,1,kptr)
        ud_sign = nint(ran_loc-.0)  ! prob for hill : ran_loc - (-p_hill +.5)
        rand_pot_data( i_vol ) = ud_sign
        rand_pot_data( i_vol+1 ) = - ud_sign
        if ( ud_sign .eq. 1 ) then
           i_vol = i_vol + 2
        else
           i_vol = i_vol +1
        end if
        if (i_vol .ge. (2 * n_vol) ) goto 44
        goto 33
44      continue

     else if  (f_roughness .eq. 1 ) then
        rand_pot_data = +1
        do i_dummy = 1 , n_vol
100        continue
           call r250(mz,ran_loc,1,kptr)
           i_vol = mod(nint( ran_loc * n_vol * 2 ),n_vol*2)
           if ( rand_pot_data(i_vol) .eq. 1) then
              rand_pot_data( i_vol ) = -1
           else
              goto 100
           end if

        end do
     else
        stop 'f_roughness makes no sense'
     end if
     do i_vol = 0 ,2* n_vol -1
        rand_pot_data(i_vol) = rand_pot_data(i_vol) * (-1)**i_vol
     end do
  end if
end subroutine def_conf
!-------------------------------------------------------------------------------
subroutine io_conf(f_io)
  use constants
  use part_coordinates
  use input_parameters
  implicit none
  integer f_io, i_dummy, n_trot_dat, a_trot, b_trot, test_f_potential, i_vol

  if ( f_io == 0 ) then
     !--------- read conf.sta-------------------------------------
     open ( unit=11, file="conf.sta",status="old" )
     read (unit=11,fmt='(3i10)') i_dummy, n_trot_dat, test_f_potential
     if ( test_f_potential  .ne. f_potential ) &
          &   stop 'potential of the data dose no fit'
     if (n_trot_dat == n_trot ) then
        a_trot = 1
        b_trot = 1
      else if ( n_trot == 2*n_trot_dat )then
        a_trot = 1
        b_trot = 2
     else if (n_trot * 2 == n_trot_dat ) then
        a_trot = 2
        b_trot = 1
     else
        stop 'data has wrong trotter number'
     end if

     read (unit=11,fmt='(2i10)') n_vol, n_winding
     read (unit=11,fmt=*)

     if (n_vol .gt. n_vol_max  ) then
        stop 'n_vol is to big'
     end if

     d = n_vol * 2 * pi

     if (f_potential .eq. 4 ) then
        rand_pot_data = 0
        read (unit=11,fmt=*)
        do i_vol = 0 , n_vol *2 -1
           read (unit=11,fmt='(i2)')  rand_pot_data(i_vol)
        end do
     end if
     read (unit=11,fmt=*)
     do i_part = 0 , n_part-1
        do i_trot = 0 , n_trot/b_trot-1 , a_trot
           if (b_trot == 1 ) then
              read (unit=11,fmt='(E20.8)') x(i_trot,i_part)
           else if ( b_trot == 2 ) then
              read (unit=11,fmt='(E20.8)') x(2 * i_trot,i_part)
              x(2 * i_trot+1,i_part) = x(2 * i_trot,i_part)
           end if
        end do
     end do
  else if (f_io == 1 ) then
     !-------- write conf,end -------------------------------------
     open ( unit=11, file="conf.end",status="unknown" )
     write (unit=11,fmt='(3i10,a40)' ) n_part, n_trot,f_potential , &
          &     '     n_part, n_trot, f_potential        '
     write (unit=11,fmt='(2i10,a30)') n_vol, n_winding,   &
          & '     volumen, winding number  '
     write (unit=11,fmt=*)
     if (f_potential .eq. 4 ) then
        write (unit=11,fmt='(a30)') 'potential data:               '
        do i_vol = 0 , n_vol *2 -1
           write (unit=11,fmt='(i2)')  rand_pot_data(i_vol)
        end do
     end if
        write (unit=11,fmt='(a30)') 'particle coordinates:         '
     do i_part = 0 , n_part-1
        do i_trot = 0 , n_trot-1
           write (unit=11,fmt='(E20.8)') x(i_trot,i_part)
        end do
     end do
  else
     stop 'io flag makes no sense'
  end if

  close(11)

end subroutine io_conf
!-------------------------------------------------------------------------------
subroutine calc_force
  use constants
  use part_coordinates
  use input_parameters
  use md_data
  use functions
  implicit none

  real(kind=dp) :: dummy

  !$OMP PARALLEL DO  PRIVATE(i_part,i_trot,dummy)  &
  !$              &  SHARED(force,cl_force,c_hoa,n_part,n_trot)
  do i_part = 0 , n_part-1
     do i_trot = 0, n_trot-1
        !-- calculate external force on the x coordinates
        force(i_trot,i_part) = - potential(x(i_trot,i_part),1,i_part)
        cl_force(i_trot,i_part) =  force(i_trot,i_part)


        !-- calculate HOA - corrections for the x coordinates
        if ( f_hoa .ge. 1.) then
           dummy =    c_hoa *  force(i_trot,i_part)  &
                &   * potential(x(i_trot,i_part),2,i_part)
           force(i_trot,i_part) = force(i_trot,i_part) + dummy
        end if
     end do
  end do
  !$OMP END PARALLEL DO

!-- transform forces on the x coordinates to forces on the u coordinates
  call transform_force

!-- calculate harmonic force on the u coordinates
!   and f_HOA=2 correction included in k_ges
  !$OMP PARALLEL DO  PRIVATE(i_part,i_trot)  SHARED(force,u,k_ges)
  do i_part = 0 , n_part-1
     do i_trot = 0, n_trot-1
        force(i_trot,i_part) = force(i_trot,i_part) - &
             &   k_ges(i_trot,i_part) * u(i_trot,i_part,0)
     end do
  end do
  !$OMP END PARALLEL DO


  force(0,0) = force(0,0) + ex_force * sqrt( real(n_trot*n_part) )

end subroutine calc_force
```

G.3 Observation routines

```
!===============================================================================
!        Observation Routines
!=-=============================================================================
subroutine init_observation
  use constants
  use observation_data
  use input_parameters
  use part_coordinates
```

```
implicit none
real(kind=dp) ::dk_df

!-- counter
n_o = 0
!-- classical kin energy
t_class = 0.d0
t_class2 = 0.d0
!-- hull function
hull_function = 0.d0
!-- quantum kin. energy
t_qm =    0.d0
t_qm2 =   0.d0
!-- vir estimator for quantum kin. energy
t_qm_vir =    0.d0
t_qm_vir2 =   0.d0
!  harm. energy
e_harm =  0.d0
e_harm2 = 0.d0
!-- external Energy and HOA correction
v_ex_pot = 0.d0
v_ex_pot2 = 0.d0
U_c_HOA = 0.d0
U_c_HOA2 = 0.d0
!-- init spring constants for observation
do i_part = 0, n_part -1
   do i_trot = 0, n_trot-1
      k_kin_energy(i_trot,i_part) = - mass / beta_p**2 * A_QV(i_part) * &
           & 4 * sin(pi/n_trot * i_trot )**2
      k_harm_energy(i_trot,i_part) = k * B_QV(i_part)  * &
           & 4 * sin(pi/n_part * i_part )**2

      if (i_trot .ne. 0 ) then
         k_kin_vir_energy(i_trot,i_part) =  k * B_QV(i_part)  * &
              & 4 * sin(pi/n_part * i_part )**2
      else
         k_kin_vir_energy(i_trot,i_part) = 0.
      end if


      dk_df = f(i_part) / 2.d0 * &
           &(  k * dB_QV(i_part)  * 4 * sin(pi/n_part * i_part )**2 +  &
           &   mass / beta_p**2 * dA_QV(i_part) *                      &
           &    4 * sin(pi/n_trot * i_trot )**2                        &
           &)

      k_kin_energy(i_trot,i_part)  = k_kin_energy(i_trot,i_part) + dk_df
      k_harm_energy(i_trot,i_part) = k_harm_energy(i_trot,i_part)+ dk_df
      k_kin_vir_energy(i_trot,i_part) = k_kin_vir_energy(i_trot,i_part) &
           & + dk_df


   end do
end do

! Gyration radius
gyration_radius  = 0.
gyration_radius2 = 0.

part_dist = 0
g_ima_part = 0.d0
g_tun = 0.d0
g_ima_cms = 0.d0
g_tun_cms = 0.d0
n_top = 0
u_k_corr = 0.d0
g_r = 0

s_r = 0.
s_i = 0.
s_cent = 0.
n_cent_obs = 0
```

```
  n_cent_cor = obs_step * nint( real(n_obs) / (10*obs_step) )
!  n_cent_cor = obs_step * nint( real(n_obs) / (obs_step) )
  d_omega = 2 * pi / (n_cent_cor * dt )
  delta = d_omega * dt
  r2_corr = 0.
  part_gyration_radius = 0.

end subroutine init_observation
!==============================================================================
subroutine observation
  use constants
  use observation_data
  use part_coordinates
  use MD_data
  use input_parameters
  use functions
  use r250_data
  implicit none

  real(kind=dp) ::  t_class_part, t_class_trot, vel_loc, t_kin_loc
  real(kind=dp) ::  v_ex_pot_part, v_ex_pot_trot
  real(kind=dp) ::  U_c_HOA_part, U_c_HOA_trot, U_c_loc
  real(kind=dp) ::  t_qm_trot, t_qm_part, e_harm_trot, e_harm_part
  real(kind=dp) ::  t_qm_vir_trot, t_qm_vir_part
  real  ::  ran_loc  ! r250 needs a single precision real !!
  real(kind=dp) ::  x_i_cms_temp,  x_k_cms_temp
  integer :: k_trot, i_min, k_min, k_part
  real(kind=dp) ::  loc_real
  real(kind=dp) ::   phase, dummy
  real(kind=dp) ::   sigma_inst, pot_inst, kin_inst
  real(kind=dp) ::   k_qm_inst , E_pr_st_inst, E_pr_MDI_inst, E_vir_inst
  integer :: i_trot_ran, i_part_ran
  real(kind=dp) :: r_cent_loc, virial_inst
  real(kind=dp) :: cms_loc_i, cms_loc_j, bug_test
  real(kind=dp) :: gyration_radius_trot, gyration_radius_part

  call r250(mz,ran_loc,1,kptr)
  i_trot_ran = nint(ran_loc * (n_trot-1) )
  i_part_ran = nint(ran_loc * (n_part-1) )

  !---classical Kin Energy
  t_class_part = 0.d0
  !--external Potential Energy and HOA correction
  v_ex_pot_part = 0.d0
  U_c_HOA_part = 0.d0
  !--quantum kinetic energy and harmonic energy
  t_qm_part = 0.d0
  t_qm_vir_part = 0.d0
  e_harm_part = 0.d0
  !  Gyration radius
  gyration_radius_part = 0.

  !$OMP PARALLEL DO PRIVATE(i_trot,vel_loc,t_kin_loc,t_class_trot,        &
  !$ &                           V_ex_pot_trot,U_c_HOA_trot,U_c_loc,      &
  !$ &                           t_qm_trot,t_qm_vir_trot,e_harm_trot,     &
  !$ &                           j_trot,gyration_radius_trot )            &
  !$ &                REDUCTION(+:t_class_part,v_ex_pot_part,U_c_HOA_part,   &
  !$ &                          t_qm_part,t_qm_vir_part,e_harm_part,     &
  !$ &                          gyration_radius_part )
  do i_part = 0 , n_part -1

     !--------------------------------------------------------------------------
     ! calculate classical kinetic energy                                      !
     t_class_trot = 0.d0                                                       !
     do i_trot = 0 , n_trot -1                                                 !
        vel_loc =  u(i_trot,i_part,1) / dt                                     !
        t_kin_loc = .5d0 * f_mass(i_trot,i_part) * vel_loc*vel_loc             !
        t_class_trot = t_class_trot +  t_kin_loc                               !
     end do                                                                    !
     t_class_part = t_class_part + t_class_trot                                !
     !--------------------------------------------------------------------------
```

```
   !-------------------------------------------------------------------------
   ! calculate potential energy with substrate pot + Tak Ima correction     !
   v_ex_pot_trot = 0.d0                                                     !
   U_c_HOA_trot = 0.d0                                                      !
   do i_trot = 0, n_trot-1                                                  !
      v_ex_pot_trot = v_ex_pot_trot + potential(x(i_trot,i_part),0,i_part)  !
      if (f_HOA .eq.1 ) then                                                !
         U_c_loc = cl_rez_force(i_trot,i_part)**2                           !
      else if (f_HOA .eq.2 ) then                                           !
         U_c_loc = ( cl_rez_force(i_trot,i_part) -  &                       !
              &  4 * k * sin(pi/n_part *i_part)**2  &                       !
              &    * u(i_trot,i_part,0) )**2                                !
      else                                                                  !
         U_c_loc = 0.                                                       !
      end if                                                                !
      U_c_HOA_trot  =  U_c_HOA_trot + U_c_loc                               !
   end do                                                                   !
   v_ex_pot_part = v_ex_pot_part +  v_ex_pot_trot                           !
   U_c_HOA_part  = U_c_HOA_part  +  U_c_HOA_trot                            !
   !-------------------------------------------------------------------------

   !-------------------------------------------------------------------------
   ! calculate quantum kinetic energy and harmoic potential energy          !
   t_qm_trot = 0.d0                                                         !
   t_qm_vir_trot = 0.d0                                                     !
   e_harm_trot = 0.d0                                                       !
   do i_trot = 0, n_trot-1                                                  !
      t_qm_trot = t_qm_trot + &                                             !
           &    .5d0 * k_kin_energy(i_trot,i_part) * u(i_trot,i_part,0)**2  !
      t_qm_vir_trot =  t_qm_vir_trot + &                                    !
           &    .5d0 * k_kin_vir_energy(i_trot,i_part) &                    !
           &                  * u(i_trot,i_part,0)**2                       !
      e_harm_trot =  e_harm_trot + &                                        !
           &    .5d0 * k_harm_energy(i_trot,i_part)* u(i_trot,i_part,0)**2  !
      if (i_trot .ne. 0 ) then                                              !
         t_qm_vir_trot =  t_qm_vir_trot - &                                 !
              &     .5 * u(i_trot,i_part,0) * cl_rez_force(i_trot,i_part)   !
      end if                                                                !
   end do                                                                   !
   t_qm_part = t_qm_part +  t_qm_trot                                       !
   t_qm_vir_part = t_qm_vir_part +  t_qm_vir_trot                           !
   e_harm_part = e_harm_part +  e_harm_trot                                 !
   !-------------------------------------------------------------------------

   !-------------------------------------------------------------------------
   ! calculate gyration radius                                              !
   gyration_radius_trot = 0.                                                !
   do i_trot = 0 , n_trot-1                                                 !
      j_trot = mod(i_trot + n_trot/2 , n_trot)                              !
      dummy  = ( x_phys(i_trot,i_part) - x_phys(j_trot,i_part) )**2         !
      gyration_radius_trot = gyration_radius_trot + dummy                   !
      part_gyration_radius(i_part) =part_gyration_radius(i_part) + dummy    !
   end do                                                                   !
   gyration_radius_part = gyration_radius_part + gyration_radius_trot       !
   !-------------------------------------------------------------------------
end do
!$OMP END PARALLEL DO

kin_inst =  t_class_part / n_trot
t_class  = t_class  +   t_class_part / (n_part*n_trot)
t_class2 = t_class2 + ( t_class_part / (n_part*n_trot) )**2

pot_inst =  v_ex_pot_part
v_ex_pot  = v_ex_pot  +   v_ex_pot_part / (n_part*n_trot)
v_ex_pot2 = v_ex_pot2 + ( v_ex_pot_part / (n_part*n_trot) )**2
U_c_HOA   = U_c_HOA   +   U_c_HOA_part  / (n_part*n_trot)
U_c_HOA2  = U_c_HOA2  + ( U_c_HOA_part  / (n_part*n_trot) )**2

t_qm  = t_qm  +   t_qm_part / (n_part*n_trot)
t_qm2 = t_qm2 + ( t_qm_part / (n_part*n_trot) )**2

t_qm_vir  = t_qm_vir  +   t_qm_vir_part / (n_part*n_trot)
t_qm_vir2 = t_qm_vir2 + ( t_qm_vir_part / (n_part*n_trot) )**2

e_harm  = e_harm  +   e_harm_part / (n_part*n_trot)
e_harm2 = e_harm2 + ( e_harm_part / (n_part*n_trot) )**2

gyration_radius  = gyration_radius  + gyration_radius_part/(n_part*n_trot)
gyration_radius2 = gyration_radius2 + &
     &       ( gyration_radius_part / (n_part*n_trot) )**2


!----------------------------------------------------------------------------
! comment out the calcualtion of all correlation functions
! goto 999

!$OMP PARALLEL DO PRIVATE(i_trot,i_grid,loc_real,i_min,j_trot,k_trot,k_min, &
!$ &                       phase, i_phase, k_part, j_part, dummy, i_g_r,     &
!$ &                       i_omega)
do i_part = 0 , n_part-1

   !-------------------------------------------------------------------------
   ! Hull function                                                          !
   hull_function(i_part) = hull_function(i_part) +   x_expect(i_part)       !
   !-------------------------------------------------------------------------

   !-------------------------------------------------------------------------
   ! Particle Distribution                                                  !
   i_grid =  nint( x_phys(i_trot_ran,i_part)  / d * n_grid )                !
   if ( (i_grid .ge. 0) .and. (i_grid .le. n_grid )  )   &                  !
        &  part_dist(i_part,i_grid) = part_dist(i_part,i_grid) + 1          !
   !-------------------------------------------------------------------------

   !-------------------------------------------------------------------------
   ! Imaginary time correlation functions for one particle                  !
   ! calc i_min                                                             !
   loc_real =  x_phys(i_trot_ran,i_part) / (2 * pi)                         !
   i_min = nint( loc_real  )                                                !
                                                                            !
   do j_trot = 1 , n_trot-1                                                 !
      k_trot = mod( i_trot_ran + j_trot, n_trot)                            !
      g_ima_part(j_trot) = g_ima_part(j_trot) + &                           !
           & (x_phys(i_trot_ran,i_part)- x_phys(k_trot,i_part) )**2         !
                                                                            !
                                                                            !
      ! calc k_min                                                          !
      loc_real =  x_phys(k_trot,i_part) / (2 * pi)                          !
      k_min = nint( loc_real  )                                             !
                                                                            !
      if (f_potential .eq. 2 ) then                                         !
         g_tun(j_trot) = g_tun(j_trot) + (-1)**( i_min +  k_min)            !
      else if (f_potential .eq. 3 ) then                                    !
         g_tun(j_trot) = g_tun(j_trot)  &                                   !
              & + sign( real(1.d0,kind=dp) , &                              !
              &         x_phys(i_trot_ran,i_part) * x_phys(k_trot,i_part) ) !
      end if                                                                !
   end do                                                                   !
   !-------------------------------------------------------------------------

   !-------------------------------------------------------------------------
   ! phase distribution and top properbility                                !
   phase = mod_2pi( x_phys(i_trot_ran,i_part) )                             !
   phase = phase / ( 2 * pi )                                               !
                                                                            !
   if ( abs(phase - .5d0 ) .le. .005 )  n_top = n_top + 1                   !
                                                                            !
   i_phase = int( phase  *  n_phase )                                       !
   if (i_phase.gt. n_phase .or.  i_phase.lt.0 ) stop 'i_phase' ! test       !
   phase_dist(i_phase) = phase_dist(i_phase) + 1                            !
   !-------------------------------------------------------------------------

   !-------------------------------------------------------------------------
   !claculate g(r)                                                          !
   do k_part = 1 , min(n_part_g_r, n_part-1 )                               !
```

```
      j_part = mod(i_part + k_part, n_part)                                 !
      dummy = x_phys(i_trot_ran,j_part) - x_phys(i_trot_ran,i_part)+2. * pi !
      if (i_part .ge. n_part-k_part) dummy = dummy + d * n_winding          !
      i_g_r = nint( dummy / ( (n_vol+2) * 2 * pi ) *  n_g_r )               !
      if ( (i_g_r .ge. 0)  .and. (i_g_r .le. n_g_r) )  &                    !
           & g_r(i_g_r,k_part) =  g_r(i_g_r,k_part) + 1                     !
   end do                                                                   !
   !-------------------------------------------------------------------------

   !-------------------------------------------------------------------------
   !  centroid correlation function                                         !
   do i_omega =  0  , n_omega                                               !
      s_r(i_part,i_omega) = s_r(i_part,i_omega) + &                         !
           &   u(0,i_part,1) * &                                            !
           & cos(   delta * mod( (i_md-n_relax),n_cent_cor) * i_omega )  * &!
           & obs_step                                                       !
      s_i(i_part,i_omega) = s_i(i_part,i_omega) + &                         !
           &   u(0,i_part,1) * &                                            !
           & sin( - delta * mod( (i_md-n_relax),n_cent_cor) * i_omega )  * &!
           & obs_step                                                       !
   end do                                                                   !
   !-------------------------------------------------------------------------

   !-------------------------------------------------------------------------
   ! densety correlation                                                    !
   j_part = mod(i_part_ran + i_part , n_part)                               !
   r2_corr(i_part,1) = r2_corr(i_part,1) +  &                               !
        &  cos( x_phys(i_trot_ran,i_part_ran) ) *  &                        !
        &  cos( x_phys(i_trot_ran,j_part) ) + &                             !
        &  sin( x_phys(i_trot_ran,i_part_ran) ) * &                         !
        &  sin( x_phys(i_trot_ran,j_part) )                                 !
   r2_corr(i_part,2) = r2_corr(i_part,2) +  &                               !
        & ( x_phys(i_trot_ran,i_part_ran)- x_phys(i_trot_ran,j_part) &      !
        &   - 2 * pi * n_winding * ( i_part_ran - j_part) )**2              !
   !-------------------------------------------------------------------------

end do
!$OMP END PARALLEL DO

 if ( mod( (i_md-n_relax) ,n_cent_cor) .eq. 0 ) then
   do i_part = 0 , n_part-1
      do i_omega = 0 , n_omega
         s_cent(i_part,i_omega) =  s_cent(i_part,i_omega) + &
              &  s_r(i_part,i_omega)**2 + s_i(i_part,i_omega)**2
      end do
    end do
   s_r = 0.
   s_i = 0.
   n_cent_obs = n_cent_obs + 1
end if



!---------------------------------------------------------------------------
! Imaginary time korrelation funktions for the CMS mode                    !
                                                                           !
!$OMP PARALLEL DO PRIVATE(k_trot,x_i_cms_temp,x_k_cms_temp,loc_real, &      !
!$ &                        i_min,k_min,i_part)                            !
do j_trot = 1 , n_trot-1                                                   !
   k_trot = mod( i_trot_ran + j_trot, n_trot)                              !
   x_i_cms_temp = 0.d0                                                     !
   x_k_cms_temp = 0.d0                                                     !
   do i_part = 0, n_part-1                                                 !
      x_i_cms_temp = x_i_cms_temp + x_phys(i_trot_ran,i_part)              !
      x_k_cms_temp = x_k_cms_temp + x_phys(k_trot,i_part)                  !
   end do                                                                  !
   g_ima_cms(j_trot) = g_ima_cms(j_trot) + &                               !
        & (  x_i_cms_temp -  x_k_cms_temp )**2                             !
                                                                           !
   ! calc i_min                                                            !
   loc_real =    x_i_cms_temp  / (2 * pi)                                  !
   i_min = nint( loc_real  )                                               !
```

```
                                                                           !
   ! calc k_min                                                            !
   loc_real =   x_k_cms_temp  / (2 * pi)                                   !
   k_min = nint( loc_real  )                                               !
                                                                           !
   if (f_potential .eq. 2 ) then                                           !
      g_tun_cms(j_trot) = g_tun(j_trot) + (-1)**( i_min +  k_min)          !
   else if (f_potential .eq. 3 ) then                                      !
      g_tun_cms(j_trot) = g_tun(j_trot) &                                  !
           &  + sign( real(1.d0,kind=dp), x_i_cms_temp *  x_k_cms_temp  )  !
   end if                                                                  !
                                                                           !
end do                                                                     !
!$OMP END PARALLEL DO                                                      !
                                                                           !
!---------------------------------------------------------------------------

if (n_o == 1 ) call disp_start                   !--displacement


!  call moden_correlation

!  write velocty of then (0,0) mode and minimum sign of part 0
   ! calc i_min
   loc_real =  x_expect(0) / (2 * pi)
   i_min = nint( loc_real  )
  write(23,fmt='(4e20.8))') dt * (i_md-n_relax) , u(0,0,1)/dt, &
       &  u(0,0,0) , real((-1)**i_min)


!---------------------------------------------------------------------------
! MD-Improved primitive estmators
goto 999 !  comment_out
sigma_inst = 0.
virial_inst = 0.
do i_part = 0 , n_part -1
   j_part = mod(i_part +1 , n_part)
   k_part = mod(i_part - 1 + n_part , n_part)
   r_cent_loc = 0.
   do i_trot =  0 , n_trot-1
      j_trot = mod(i_trot +1 , n_trot)
      sigma_inst = sigma_inst + ( x(i_trot,i_part) - x(j_trot,i_part) )**2
      pot_inst = pot_inst + k/2.  * ( x(i_trot,i_part) - x(i_trot,j_part) )**2
      r_cent_loc = r_cent_loc +  x(i_trot,i_part)
   end do
   r_cent_loc = r_cent_loc / n_trot
   do i_trot = 0 , n_trot-1
      virial_inst  = virial_inst + ( - cl_force(i_trot,i_part) +   &
           &  k * ( 2 * x(i_trot,i_part) -  x(i_trot,j_part)  -    &
           &                                x(i_trot,k_part) ) ) * &
           &  ( x(i_trot,i_part) - r_cent_loc )
    end do
end do
pot_inst = pot_inst / n_trot
virial_inst = virial_inst / ( 2 * n_trot )


k_qm_inst = mass * n_trot/ beta
E_pr_st_inst = .5 / beta *n_trot * n_part  - &
     & .5 / beta *  k_qm_inst * sigma_inst !+ pot_inst
E_pr_st_inst =   E_pr_st_inst / n_part
E_pr_MDI_inst =  kin_inst -   .5 / beta *k_qm_inst * sigma_inst ! + pot_inst
E_pr_MDI_inst =  E_pr_MDI_inst / n_part
E_vir_inst =  .5 / beta * n_part +   virial_inst
E_vir_inst = E_vir_inst / n_part
pot_inst = pot_inst / n_part

write(22,fmt='(i10,5e20.8)') i_md ,  E_pr_st_inst ,  E_pr_MDI_inst , &
     & E_vir_inst, pot_inst

! end of MD-Improved primitive estmators
!---------------------------------------------------------------------------
```

```fortran
  999 continue ! comment_out

end subroutine observation
!===============================================================================
subroutine observation_out
  use constants
  use observation_data
  use normal_mode_data
  use part_coordinates
  use input_parameters
  use functions
  implicit none
  integer :: i_rad, i_vol, i_omega_max
  real(kind=dp) :: g_funk, cent_norm
  character(len=4) :: int_as_string
  character(len=15) :: pic_conf_format, pic_max_format
  character(len=11) :: u_corr_format,  g_r_format



  !-- write hull function-----------------------------------------------
  open ( unit=10, file="hull_function.out",status="unknown" )
  hull_function = hull_function / n_o
  do i_part = 0 , n_part -1
     if ( i_part .eq. 0 ) then
        g_funk =  hull_function(n_part-1)- 2. * hull_function(0) + &
             &    hull_function(1) - d
     else if (i_part .eq. n_part-1) then
        g_funk =  hull_function(n_part-2)- 2. * hull_function(n_part-1) + &
             &    hull_function(0) + d
     else
        g_funk =  hull_function(i_part-1)- 2. * hull_function(i_part) + &
             &    hull_function(i_part+1)
     end if
     g_funk = g_funk

     write(unit=10,fmt='(5e20.8)') mod_2pi( i_part * d/ n_part) , &
          &  mod_2pi (  hull_function(i_part) )   , g_funk, real(i_part),  &
          &  hull_function(i_part)
  end do
  close(10)


  !-- write particle distribution---------------------------------------
  call int_to_string(n_part+1,int_as_string)
  u_corr_format = '(' // int_as_string // 'e20.8)'
  open ( unit=10, file="part_dist.out",status="unknown" )
  do i_grid = 0   , n_grid
     write(unit=10,fmt=u_corr_format) real(i_grid) / real(n_grid) * d  , &
          & (real(part_dist(i_part,i_grid))/(  n_o * ( d / n_grid) ) , &
          &  i_part=0 , n_part-1 )
  end do
  close(10)

  !-- write g(r)
  call int_to_string( min(n_part_g_r,n_part-1)+1 , int_as_string)
  g_r_format = '(' // int_as_string // 'e20.8)'
  open ( unit=10, file="g_r.out",status="unknown" )
  do i_g_r = 0   , n_g_r
     write(unit=10,fmt=g_r_format) real(i_g_r) / real(n_g_r) *  &
          & ((n_vol+2) * 2 * pi)- 2. * pi  ,                    &
          & (real(g_r(i_g_r, i_part))                           &
          &     / ( (n_vol+2) *2* pi /n_g_r) / (n_o*n_part),    &
          & i_part = 1 , min(n_part_g_r,n_part-1) )
  end do
  close(10)



  !--Write picture------------------------------------------------------
call int_to_string( n_part , int_as_string)
pic_conf_format = '(i10,' // int_as_string // 'e20.8)'

open ( unit=10, file="conf.pic",status="unknown" )
 do i_trot = 0 , n_trot-1
    write (10,fmt=pic_conf_format) i_trot , &
         & (pbc(x(i_trot,i_part)),i_part=0,n_part-1)
end do
close(10)


call int_to_string( n_vol+1 , int_as_string)
pic_max_format = '(i10,' // int_as_string // 'e20.8)'

open ( unit=10, file="max.pic",status="unknown" )
 do i_trot = 0 , n_trot-1
    write (10,fmt=pic_max_format) i_trot , &
         & (pbc( pi * (2*i_vol+1) )  ,i_vol=0,n_vol )
end do
close(10)

open ( unit=10, file="pot.pic",status="unknown" )
do i_rad = 0 , 2000
   write (10,fmt='(4e20.8)') i_rad*d/2000.d0  ,          &
        & potential(real(i_rad*d/2000.d0,kind=dp),0,0), &
        & potential(real(i_rad*d/2000.d0,kind=dp),1,0), &
        & potential(real(i_rad*d/2000.d0,kind=dp),2,0)
end do
close(10)
!--Write ima korrelation Funktion--------------------------------------
open ( unit=10, file="g_ima.out",status="unknown" )

g_ima_part(0) = 0.d0
g_ima_part(n_trot) = 0.d0
g_ima_part = g_ima_part / (n_part * n_o)

g_ima_cms(0) = 0.d0
g_ima_cms(n_trot) = 0.d0
g_ima_cms = g_ima_cms /  n_o

g_tun = g_tun /  (n_part * n_o)
g_tun(0) = 1.d0
g_tun(n_trot) = 1.d0



g_tun_cms = g_tun_cms /  n_o
g_tun_cms(0) = 1.d0
g_tun_cms(n_trot) = 1.d0



do i_trot = 0 , n_trot
   write (10,fmt='(5e20.8)') i_trot*beta_p , g_ima_part(i_trot), &
        & g_ima_cms(i_trot)/n_part,                              &
        & g_tun(i_trot), g_tun_cms(i_trot)
end do
close(10)


!-- write eigenmoden korrelation
!open ( unit=10, file="u_corr_ima.out",status="unknown" )
!u_k_corr  = u_k_corr / n_o
!do i_part = 0 , n_part -1
!   u_k_corr(n_trot,i_part) = u_k_corr(0,i_part)
!end do
!
!do i_trot = 0 , n_trot
!   write (10,fmt=u_corr_format) i_trot*beta_p , &
!        & ( u_k_corr(i_trot,i_part), i_part = 0 , n_part-1 )
!end do
```

```fortran
  !close(10)

  !-- write centroid corr function
  !-- normalize
  do i_part = 0 , n_part-1
     cent_norm = 0.
     do i_omega = 0 , &
          &     min(n_omega, nint( 2 * pi / ( 2 * dt * obs_step * d_omega )) )
        cent_norm =  cent_norm + s_cent(i_part, i_omega) * d_omega
     end do
     do i_omega = 0 , &
          &     min(n_omega, nint( 2 * pi / ( 2 * dt * obs_step * d_omega )) )
        s_cent(i_part,i_omega) =  s_cent(i_part, i_omega)/cent_norm
     end do
  end do

  open ( unit=10, file="cent_corr.out",status="unknown" )
  do i_omega = 0 , &
       &     min(n_omega, nint( 2 * pi / ( 2 * dt * obs_step * d_omega )) )
     write (10,fmt=u_corr_format) i_omega * d_omega , &
          &  ( s_cent(i_part, i_omega)  , i_part = 0 , n_part-1 )
  end do
  close(10)

  !-- write centroid spectrum
  open ( unit=10, file="cent_spectrum.out",status="unknown" )
  do i_part = 0 , n_part-1
     i_omega_max = 0.
     ! maximal omega is half of the measure Frequ.
     do i_omega = 0 , &
          & min(n_omega, nint( 2 * pi / ( 2 * dt * obs_step * d_omega )) )
        if (   s_cent(i_part, i_omega) .gt. s_cent(i_part,i_omega_max) ) &
             &    i_omega_max = i_omega
     end do
     write(10,fmt='(i10,e20.8)') i_part , i_omega_max * d_omega
  end do
  close(10)



  !-- Write phase distribution
  open ( unit=10, file="phase_dist.out",status="unknown" )
  do i_phase = 0 , n_phase
     write(10,fmt='(2e20.8)') (i_phase * 2 * pi) / n_phase , &
          &  phase_dist(i_phase) * n_phase / (2 * pi * n_o * n_part)
  end do
  close(10)

  !-- Write density correlation function
  open ( unit=10, file="dense_corr.out",status="unknown" )
  do i_part = 0 , n_part-1
  write(10,fmt='(i10,2e20.8)') i_part , r2_corr(i_part,1) / n_o, &
       &  r2_corr(i_part,2) / n_o
  end do
  close(10)



  open ( unit=10, file="displacement.out",status="unknown" )
  do i_part = 0 , n_part-1
     write(10,fmt='(4e20.8)') translation(i_part),&
          &   pbc( displacement(i_part) ), &
          &   pbc( x(0,i_part) ), &
          &   part_gyration_radius(i_part)/n_o/n_trot
  end do
  close(10)



end subroutine observation_out
!=============================================================================
```

```fortran
subroutine int_to_string(int_in,string_out)
  implicit none
  integer :: int_in, diget , loc_dig
  character(len=4) :: string_out

  if (int_in .lt. 0 ) stop 'neg inp for int_to_string'
  if (int_in .gt. 9999 ) stop 'to bip inp fot int_to_string'

  do diget = 4 , 1 ,-1
        loc_dig = ( mod(int_in,10**diget) - mod(int_in,10**(diget-1) )) &
             & / 10**(diget-1)

     if (loc_dig .eq. 0 ) then
        string_out = string_out(:4-diget) // '0'
     else if   (loc_dig .eq. 1 ) then
        string_out = string_out(:4-diget) // '1'
     else if   (loc_dig .eq. 2 ) then
        string_out = string_out(:4-diget) // '2'
     else if   (loc_dig .eq. 3 ) then
        string_out = string_out(:4-diget) // '3'
     else if   (loc_dig .eq. 4) then
        string_out = string_out(:4-diget) // '4'
     else if   (loc_dig .eq. 5 ) then
        string_out = string_out(:4-diget) // '5'
     else if   (loc_dig .eq. 6 ) then
        string_out = string_out(:4-diget) // '6'
     else if   (loc_dig .eq. 7 ) then
        string_out = string_out(:4-diget) // '7'
     else if   (loc_dig .eq. 8 ) then
        string_out = string_out(:4-diget) // '8'
     else if   (loc_dig .eq. 9 ) then
        string_out = string_out(:4-diget) // '9'
     else
        stop 'int_to_string has a problem'
     end if
  end do

end subroutine int_to_string
!=============================================================================
subroutine moden_correlation
  use part_coordinates
  use input_parameters
  use normal_mode_data
  use observation_data
  use r250_data
  use functions
  implicit none

  integer k_trot, i_trot_ran
  real ::  ran_loc ! r250 needs single precision real !!!
  real(kind=dp), dimension( : ,  : ), allocatable :: x_u
  real(kind=dp), dimension( : ), allocatable :: four_dummy_part
  allocate( four_dummy_part( 1:2*n_part ) )
  allocate( x_u( 0:n_trot-1, 0:n_part-1)  )

  !-- Translation
  do i_part = 0 , n_part -1
     do i_trot =  0 , n_trot -1
        x_u(i_trot,i_part) =   x_phys(i_trot,i_part) -  translation(i_part)
     end do
  end do


  !--decouple different oscillators
  do i_trot = 0 , n_trot-1

     do i_part = 0 , n_part-1
        four_dummy_part(2*i_part+1) = x_u(i_trot,i_part) ! re part
        four_dummy_part(2*i_part+2) = 0.                ! im part
     end do
  call four1(four_dummy_part,n_part,1,n_part)


  x_u(i_trot,0) = four_dummy_part(1)
  if ( n_part .gt. 1 ) then
   x_u(i_trot,n_part/2) = four_dummy_part(n_part+1 )
  end if

  do i_part = 1 , n_part/2 - 1
   x_u(i_trot,i_part) = four_dummy_part(2*i_part+1)  * sqrt2
   x_u(i_trot,i_part+n_part/2) = four_dummy_part(n_part + 2*i_part+2)  * sqrt2
  end do

 end do ! i_trot


 do i_part = 0 , n_part-1
    do i_trot = 0 , n_trot-1
       x_u(i_trot,i_part) = x_u(i_trot,i_part) * in_sqrt_N
    end do
 end do

  call r250(mz,ran_loc,1,kptr)
  i_trot_ran = nint(ran_loc * (n_trot-1) )

 do i_part = 0 , n_part-1
    do j_trot = 0 , n_trot-1
       k_trot = mod( i_trot_ran + j_trot, n_trot)
       u_k_corr(j_trot,i_part) = u_k_corr(j_trot,i_part) + &
            &   x_u(i_trot_ran,i_part) *  x_u(k_trot,i_part)
    end do
 end do

end subroutine moden_correlation
!===============================================================================
subroutine disp_start
  use part_coordinates
  use observation_data
  implicit none

  do i_part = 0 , n_part-1
     displacement(i_part) = x(0,i_part)
  end do
end subroutine
!===============================================================================
```

G.4 Molecular dynamics routines

```
!===============================================================================
subroutine predict
  use part_coordinates
  use md_data
  use input_parameters



  !----  first positions
  i_order = 0
  do j_order = 1,n_order
  !$OMP PARALLEL DO
     do i_part = 0 , n_part-1
        do i_trot = 0 , n_trot-1
           u(i_trot,i_part,i_order) = u(i_trot,i_part,i_order) &
                &  + predict_coef(i_order,j_order) * u(i_trot,i_part,j_order)
        end do
     end do
  !$OMP END PARALLEL DO
  end do

  call random_force

  !----  then derivatives
  do i_order = 1, n_order-1
     do j_order = i_order + 1, n_order
        !$OMP PARALLEL DO
        do i_part = 0 , n_part-1
           do i_trot = 0 , n_trot-1
              u(i_trot,i_part,i_order) = u(i_trot,i_part,i_order) &
                   & + predict_coef(i_order,j_order) * u(i_trot,i_part,j_order)
           end do
        end do
        !$OMP END PARALLEL DO
     end do
  end do

  if ( f_cms == 1 ) then
     u(0,0,0) = x_cms
  end if

end subroutine predict
!===============================================================================
subroutine random_force
  use constants
  use part_coordinates
  use md_data
  use input_parameters
  use r250_data
  implicit none

  real(kind=dp) ::  f_rand
  real,  dimension(1:n_trot) :: x_rand
  ! for the r250 x_rand has to be an single precision real !!!

  !$OMP PARALLEL DO PRIVATE(f_rand,x_rand)
  do i_part = 0 , n_part-1
     !$OMP CRITICAL
     call r250(mz,x_rand,n_trot,kptr)
     !$OMP END CRITICAL
     do i_trot = 0 , n_trot-1
        !--- calculate random force
        f_rand = rand_force(i_trot,i_part) * ( 2.d0 * x_rand(i_trot+1) -1.d0 )
        !--- random acceleration times dt^2
        f_rand = f_rand / f_mass(i_trot,i_part) * dt_2

        ! ----- reduce random force for centroid Dynamics --------------------
        !        if ( (i_part.eq.0) .and. (i_trot .eq. 0 ) )then             !
        !        if ( (f_fic_mass .eq.1) .and. (i_trot .eq. 0 ) )then        !
        !           f_rand = f_rand / 6.                                     !
        !        end if                                                      !
        ! -------------------------------------------------------------------

        !--- add randon acceleration to x and xt * v
        u(i_trot,i_part,0) = u(i_trot,i_part,0) +  f_rand *.5d0
        u(i_trot,i_part,1) = u(i_trot,i_part,1) +  f_rand
     end do
  end do
  !$OMP END PARALLEL DO

end subroutine random_force
!===============================================================================
subroutine friction_force
  use constants
  use part_coordinates
  use md_data
  use input_parameters
  implicit none

  real(kind=dp) :: fric_loc
  !$OMP PARALLEL DO PRIVATE(i_trot,fric_loc)  &
  !$ &             SHARED(friction,f_mass,force,u,in_dt)
  do i_part = 0, n_part -1
     do i_trot = 0, n_trot -1
        fric_loc =  friction * f_mass(i_trot,i_part)

        ! ----- reduce friction for centroid Dynamics -----------------------
        !        if ( (i_part .eq.0) .and. (i_trot .eq. 0 ) )then           !
        !        if ( (f_fic_mass .eq.1) .and. (i_trot .eq. 0 ) )then       !
        !           fric_loc = fric_loc / 36.                               !
        !        end if                                                     !
        ! ------------------------------------------------------------------

        force(i_trot,i_part) = force(i_trot,i_part) - &
             &                  u(i_trot,i_part,1)*in_dt * fric_loc
     end do
  end do
  !$OMP END PARALLEL DO


end subroutine friction_force
!================================================================================
subroutine correct
  use constants
  use part_coordinates
  use md_data
  use input_parameters
  implicit none

  real(kind=dp), parameter :: cor_max_max = 1e-04
  real(kind=dp) ::  cor_max, rel_correction, v_therm_2, dummy
  real(kind=dp), dimension(0:n_trot_max-1,0:n_part_max-1) :: f0
  integer, dimension(1:2) :: failure


  !---  here, f0 has meaning of difference between predicted
  !     acceleration and measured acceleration


  !---check the size of the correction
  cor_max = 0.
  !$OMP PARALLEL DO PRIVATE(i_part,i_trot,v_therm_2,dummy,rel_correction)
  do i_part = 0, n_part-1
     do i_trot = 0, n_trot-1
        f0(i_trot,i_part) = -u(i_trot,i_part,2) +  &
             &  force(i_trot,i_part) / f_mass(i_trot,i_part)*dt_2 *.5d0
        v_therm_2  =  (1.d0) / ( beta_p  * f_mass(i_trot,i_part) )
        dummy =  f0(i_trot,i_part)*correct_coef(1) ! correction to the velocity
        rel_correction  = dummy * dummy /  v_therm_2
        if(rel_correction .gt. cor_max) then
           cor_max = rel_correction
           failure(1) = i_part
           failure(2) = i_trot
        end if
     end do
  end do
  !$OMP END PARALLEL DO

  if (cor_max.gt.cor_max_max) then
     if (i_md .lt. n_relax ) then
        dt =  dt * .98
        dt_2 = dt * dt
        in_dt = 1.d0 / dt

        call init_md_data
        call init_observation
     else
        write(*,'(a20,1e12.4,a20,2i5)') '(v_corr/c_therm)^2:', cor_max, &
             & '  i_part,i_trot :', failure
        stop"monomer move"
     end if
  end if
```

```
  !---correct the coordinates u
   do i_order = 0, n_order
      !$OMP PARALLEL DO PRIVATE(i_part,i_trot) SHARED(u,correct_coef,f0,i_order)
      do i_part = 0, n_part-1
         do i_trot = 0 , n_trot-1
            u(i_trot,i_part,i_order) =u(i_trot,i_part,i_order) +&
                 & correct_coef(i_order) * f0(i_trot,i_part)
         end do
      end do
      !$OMP END PARALLEL DO
  end do


  if (f_cms == 1 ) then
     u(0,0,0) = x_cms
  end if

end subroutine correct
!================================================================================
subroutine init_md_data
  use constants
  use part_coordinates
  use input_parameters
  use md_data
  implicit none

  real(kind=dp), dimension(0:n_order_max-1,0:n_order_max) :: pred_coef
  real(kind=dp), dimension(0:n_order_max) :: corr_coef

  !---  initialize predictor coefficients
  do i_order = 0,n_order_max-1
     do j_order = 0,n_order_max
        pred_coef(i_order,j_order) = 0.
     end do
     pred_coef(0,i_order+1) = 1.
     if(i_order.ge.1) pred_coef(1,i_order+1) = i_order+1
     pred_coef(i_order,i_order) = 1.
  end do
  pred_coef(2,3) =  3.
  pred_coef(2,4) =  6.
  pred_coef(2,5) = 10.
  pred_coef(3,4) =  4.
  pred_coef(3,5) = 10.
  pred_coef(4,5) =  5.

  do i_order = 0,n_order-1
     do j_order = 0,n_order
        predict_coef(i_order,j_order) = 0.
     end do
     do j_order = i_order,n_order
        predict_coef(i_order,j_order) = pred_coef(i_order,j_order)
     end do
  end do

  !---  initialize corrector
  do i_order = 1,n_order
     corr_coef(i_order) = 0.
  end do
  i_order = n_order
  if(i_order.eq.2) then
     !----  velocity Verlet algorithm
     corr_coef(0) = 0.
     corr_coef(1) = 1.
  else if (i_order.eq.3) then
     corr_coef(0) = 1./6
     corr_coef(1) = 5./6
     corr_coef(3) = 1./3
  else if (i_order.eq.4) then
     corr_coef(0) = 19./90
     !  corr_coef(0) = 19./120  only if thermostat absent
     corr_coef(1) = 3./4
```

```
    corr_coef(3) = 1./2
    corr_coef(4) = 1./12
  else if (i_order.eq.5) then
    corr_coef(0) = 3./16
    !  corr_coef(0) = 3./20    only if thermostat absent
    corr_coef(1) = 251./360
    corr_coef(3) = 11./18
    corr_coef(4) = 1./6
    corr_coef(5) = 1./60
  else
    stop 'Order for predictor corrector unavailable'
  end if
  corr_coef(2) = 1.
  do i_order = 0,n_order
    correct_coef(i_order) = corr_coef(i_order)
  end do

  ! Set k_ges, fric and ficticious masses
  do i_part = 0 , n_part-1
    do i_trot = 0 , n_trot-1

       k_ges(i_trot,i_part) =  k * B_QV(i_part)*  4.d0  &
            & * sin(pi / n_part * i_part)**2  &
            & + mass / beta_p**2 * A_QV(i_part)*  4.d0  &
            & * sin(pi / n_trot * i_trot)**2
       if (f_hoa .eq.2 ) then
          k_ges(i_trot,i_part) =  k_ges(i_trot,i_part) + &
               & c_hoa  *  &
               & ( k * 4. * sin(pi/n_part*i_part)**2 )**2
       end if

       ! set fictious masses
       if ( f_fic_mass .eq. 0 ) then
          ! collaps free chain
          f_mass(i_trot,i_part) = mass * &
               & (v_renorm +  k_ges(i_trot,i_part) ) / v_renorm
          f_mass(i_trot,i_part) =   f_mass(i_trot,i_part) / omega_sample**2 &
               & *  v_renorm / mass
       else if ( f_fic_mass .eq. 1 ) then
          ! centroid dynamics
          f_mass(i_trot,i_part) = mass * &
               & ( v_renorm +  k_ges(i_trot,i_part) ) / &
               & ( v_renorm +  k * B_QV(i_part)*  &
               & 4.d0  * sin(pi / n_part * i_part)**2 )
          if (i_trot .ne. 0 ) f_mass(i_trot,i_part) =  &
               &              f_mass(i_trot,i_part) / 36.
       else
          stop 'f_fic_mass makes no sense '
       end if
       ! end of mass init

       rand_force(i_trot,i_part) = &
            & (6.d0 * friction * f_mass(i_trot,i_part) / beta_p /dt )**.5d0
    end do
  end do

end subroutine init_md_data
!=========================================================================
```

G.5 Modules and functions

```
!-------------------------------------------------------------------------------
!-------------- Modules With shared Data ---------------------------------------
!-------------------------------------------------------------------------------

module constants
  implicit none
  save
  integer, parameter ::  dp = selected_real_kind(15,199)
!  integer, parameter ::  dp = selected_real_kind(6,30)
!  integer, parameter ::  dp = 8
  real(kind=dp), parameter :: pi =  3.14159265
end module constants
!==============================================================================
module r250_data
  implicit none
  save
  integer ::  kptr
  integer, dimension(1:250) :: mz
end module r250_data
!==============================================================================
module part_coordinates
  use constants, only : dp
  implicit none
  save

  integer  :: n_part, n_trot, n_order

  !-----------------------------------------------------
  integer, parameter :: n_part_max = 128, n_trot_max = 512 !
  !-----------------------------------------------------
  ! n_trot(_max) and n_part(_max) have to be powers of 2   !
  !-----------------------------------------------------

  integer, parameter :: n_order_max = 5, n_vol_max = 128

  integer :: i_trot, j_trot, i_part, j_part, i_order, j_order
  real(kind=dp), dimension(0:n_trot_max-1,0:n_part_max-1,0:n_order_max) :: u
  real(kind=dp), dimension(0:n_trot_max-1,0:n_part_max-1) :: x , force
  real(kind=dp), dimension(0:n_trot_max-1,0:n_part_max-1) :: cl_force
  real(kind=dp), dimension(0:n_trot_max-1,0:n_part_max-1) :: cl_rez_force
  integer, dimension(0:2*n_vol_max -1) :: rand_pot_data
  integer :: n_winding
  real(kind=dp) :: d
  real(kind=dp) :: ex_force

end module part_coordinates
!==============================================================================

module input_parameters
  use constants, only : dp
  use part_coordinates, only : n_part_max
  implicit none
  save

  integer :: n_relax, n_obs, obs_step, i_md
  real(kind=dp) :: tk, beta, beta_p, alpha
  real(kind=dp) :: k, a, k_qm, pot_strength
  real(kind=dp) :: c_hoa
  real(kind=dp) :: mass
  integer :: f_conf, f_potential, f_HOA, f_decomp, f_cms
  integer :: f_roughness, f_fic_mass, f_rand_phase
  real(kind=dp) :: x_cms
  integer :: seed
  real(kind=dp) :: dt, dt_2, in_dt, friction
  real(kind=dp) , dimension(0:n_part_max-1)  :: A_QV, dA_QV, B_QV, dB_QV, f
  integer :: n_vol, n_thred
  real(kind = dp ) :: performance
  real(kind=dp) :: v_renorm , omega_sample

end module input_parameters
!==============================================================================
module normal_mode_data
  use constants, only : dp
  use part_coordinates, only : n_part_max
  implicit none
  save

  real(kind=dp),  dimension(0:n_part_max-1) :: translation
  real(kind=dp) :: c_const, sqrt2, insqrt2, in_sqrt_N, in_sqrt_P,in_sqrt_PN
```

```
end module normal_mode_data
!==============================================================================
module observation_data
  use constants, only : dp
  use part_coordinates, only : n_part_max, n_trot_max
  implicit none
  save

  real(kind=dp) :: t_class, t_class2, v_ex_pot, v_ex_pot2
  real(kind=dp) :: e_harm, e_harm2, t_qm, t_qm2,  t_qm_vir, t_qm_vir2
  real(kind=dp) :: t_prim, t_prim2
  real(kind=dp) :: U_c_HOA , U_c_HOA2
  integer :: n_o
  real(kind=dp) , dimension(0:n_trot_max-1, 0:n_part_max-1) :: k_kin_energy
  real(kind=dp) , dimension(0:n_trot_max-1, 0:n_part_max-1) :: k_harm_energy
  real(kind=dp) , dimension(0:n_trot_max-1, 0:n_part_max-1) :: k_kin_vir_energy
  real(kind=dp) , dimension(0:n_part_max-1) :: hull_function
  integer , parameter :: n_grid = 200
  integer :: i_grid
  integer, dimension(0:n_part_max-1,0:n_grid) :: part_dist
  real(kind=dp), dimension(0:n_trot_max)  :: g_ima_part, g_ima_cms
  real(kind=dp), dimension(0:n_trot_max)  :: g_tun, g_tun_cms
  integer , parameter :: n_phase =  100
  integer :: i_phase
  real(kind=dp) , dimension(0:n_phase) :: phase_dist
  integer :: n_top
  real(kind=dp), dimension(0:n_trot_max,0:n_part_max-1) :: u_k_corr
  integer :: i_g_r
  integer , parameter :: n_g_r = 2000, n_part_g_r = n_part_max
  integer , dimension(0:n_g_r,1:n_part_g_r) :: g_r
  real(kind=dp) :: gyration_radius, gyration_radius2
  real(kind=dp) :: E_pr_st_tp, E_pr_mdi_tp
  integer , parameter :: n_omega = 2000
  integer :: i_omega , n_cent_obs, n_cent_cor
  real(kind=dp) :: delta, d_omega
  real(kind=dp) , dimension(0:n_part_max-1,0:n_omega) :: s_r , s_i, s_cent
  real(kind=dp) , dimension(0:n_part_max-1, 1:2) :: r2_corr
  real(kind=dp) , dimension( 0:n_part_max-1) :: displacement
  real(kind=dp) , dimension( 0:n_part_max-1) :: part_gyration_radius
end module observation_data
!==============================================================================
module md_data
  use constants, only : dp
  use part_coordinates, only : n_part_max, n_order_max, n_trot_max
  implicit none
  save

  real(kind=dp), dimension(0:n_order_max-1,0:n_order_max) :: predict_coef
  real(kind=dp), dimension(0:n_order_max) :: correct_coef
  real(kind=dp), dimension(0:n_trot_max-1,0:n_part_max-1) :: f_mass, k_ges
  real(kind=dp), dimension(0:n_trot_max-1, 0:n_part_max-1) :: rand_force

end module md_data
!==============================================================================
module functions
  use constants, only : dp


contains
  !----------------------------------------------------------------------------
  real(kind=dp) function x_phys(a_trot,k_part)
    use part_coordinates
    use input_parameters
    use constants
    implicit none
    integer :: k_part, a_trot
    integer :: i_part_loc , j_part_loc
    real(kind=dp) :: l_loc

    x_phys = x(a_trot,k_part)
```

```
  ! higher order correction
  if ( f_hoa .eq. 1  )  then
     x_phys = x_phys - c_hoa * cl_force(a_trot,k_part)
  else if (f_hoa .eq.2 ) then
     if ( k_part .eq. 0 ) then
        i_part_loc = n_part-1
        j_part_loc = 1
        l_loc = - n_winding * d
     else if  ( k_part .eq. n_part-1 ) then
        i_part_loc = n_part-2
        j_part_loc = 0
        l_loc =  n_winding * d
     else
        i_part_loc = k_part -1
        j_part_loc = k_part +1
        l_loc = 0.
     end if
     x_phys = x_phys - c_hoa * (  cl_force(a_trot,k_part) - &
          & k * ( 2 *  x(a_trot,k_part) -  x(a_trot,i_part_loc) &
          &                              - x(a_trot,j_part_loc)- l_loc ) )
  end if

end function x_phys
!------------------------------------------------------------------------------
real(kind=dp) function u_phys(a_trot,k_part)
  use part_coordinates
  use input_parameters
  use constants
  implicit none
  integer :: k_part, a_trot

  u_phys = u(a_trot,k_part,0)
  ! higher order correction
  if ( f_hoa .eq. 1  )  then
     u_phys = u_phys - c_hoa * cl_rez_force(a_trot,k_part)
  else if (f_hoa .eq.2 ) then
      u_phys = u_phys - c_hoa * (  cl_force(a_trot,k_part) - &
          & 4 * k * sin(pi/n_part*k_part )**2 * u(a_trot,k_part,0) )
  end if

end function u_phys
!------------------------------------------------------------------------------
real(kind=dp) function x_expect(k_part)
  use part_coordinates
  use input_parameters
  implicit none
  integer :: k_part
  x_expect = 0.
  do i_trot = 0 , n_trot-1
     x_expect = x_expect +  x_phys(i_trot,k_part)
  end do
  x_expect = x_expect / n_trot

end function x_expect
!------------------------------------------------------------------------------
real(kind=dp) function mod_2pi(in)
  use constants
  implicit none
  real(kind=dp) in

  mod_2pi = in
  mod_2pi = mod_2pi / (2 * pi ) - int( mod_2pi / (2 * pi ) )
  if ( mod_2pi .lt. 0 ) then
     mod_2pi = mod_2pi + 1.d0
  end if

  mod_2pi = mod_2pi * 2 * pi

end function mod_2pi
!------------------------------------------------------------------------------
real(kind=dp) function mod_pi(in)
  use constants
  implicit none
  real(kind=dp) in

  mod_pi = in
  mod_pi = mod_pi / ( pi ) - int( mod_pi / ( pi ) )
  if ( mod_pi .lt. 0 ) then
     mod_pi = mod_pi + 1.d0
  end if

  mod_pi = mod_pi * pi

end function mod_pi
!------------------------------------------------------------------------------
integer function int_pi(in)
  use constants
  use input_parameters
  implicit none
  real(kind=dp) in


  int_pi = mod( nint( (in - mod_pi(in) )/ pi ) , 2 * n_vol )
  if ( int_pi .lt. 0 ) int_pi = int_pi+ n_vol *2

end function int_pi
!------------------------------------------------------------------------------
!integer function int_2pi(in)
!  use constants
!  use input_parameters
!  real(kind=dp) in
!  real(kind=dp) mod_pi


!  int_2pi = mod( nint( (in - mod_2pi(in) )/ (2. * pi) )  , n_vol )

!end function int_2pi

!------------------------------------------------------------------------------
real(kind=dp) function pbc(x_loc)
  use part_coordinates
  implicit none

  real(kind=dp) x_loc

  pbc =  x_loc
  do while ( pbc >= d )
     pbc = pbc - d
  end do

  do while ( pbc < 0)
     pbc = pbc + d
  end do


end function pbc
!------------------------------------------------------------------------------
real(kind=dp) function potential(x_loc,i_deriv,m_part)
  use constants
  use input_parameters
  use part_coordinates
  implicit none

  real(kind=dp) :: x_loc, const_loc, x_loc_temp
  integer :: i_deriv, i_pot, sign, m_part
  integer :: int_pi_loc

  select case (f_potential)
  case (0)
     !   no potential
     potential = 0.d0
  case (1)
     !   Harmonic potiential (k = 1 )
     if (i_deriv == 0 ) then
        potential = .5d0 * x_loc * x_loc
     else if (i_deriv == 1 ) then
        potential =  x_loc
     else if (i_deriv == 2 ) then
        potential =  1.d0
     end if
  case (2)
     ! Frenkel-Kontotova ( -  pot_strength * cos( x ) )
     if (i_deriv == 0 ) then
        potential =  - cos( x_loc )
     else if (i_deriv == 1 ) then
        potential =  sin( x_loc )
     else if (i_deriv == 2 ) then
        potential =  cos( x_loc )
     end if
  case (3)
     ! Doppelmulden Potential ( pot_strength * (x^2-1)^2 )
     if (i_deriv == 0 ) then
        potential =  ( x_loc**2 -1 )**2
     else if (i_deriv == 1 ) then
        potential =   4 * (x_loc**2 -1) * x_loc
     else if (i_deriv == 2 ) then
        potential =   4 * ( (x_loc  +2 )* x_loc -1 )
     end if
  case (4)
     ! Zuffallspotential
     if (i_deriv == 0 ) then

        const_loc = 0.d0
        sign = 1
        int_pi_loc = int_pi(x_loc)
        do i_pot = 0 , int_pi_loc-1
           const_loc = const_loc + rand_pot_data(i_pot)* sign
           sign = - sign
        end do
        const_loc = 2 * const_loc + rand_pot_data( int_pi_loc ) * sign

        potential =  - cos(x_loc) * rand_pot_data(int_pi_loc) &
             & +  const_loc

     else if (i_deriv == 1 ) then
        potential =   sin(x_loc) * rand_pot_data(int_pi(x_loc))
     else if (i_deriv == 2 ) then
        potential =   cos(x_loc) * rand_pot_data(int_pi(x_loc))
     end if
  case(5)
     ! Harmonic Oscillators ( x- m_part 2 pi ) ** 2
     x_loc_temp = x_loc - m_part * 2 * pi
     if (i_deriv == 0 ) then
        potential = .5d0 * x_loc_temp * x_loc_temp
     else if (i_deriv == 1 ) then
        potential =  x_loc_temp
     else if (i_deriv == 2 ) then
        potential =  1.d0
     end if

  case default
     stop 'f_potential makes no sense'
  end select


  potential = potential * pot_strength

end function potential



!==============================================================================

end module functions
```

G.6 Normal mode transformation

```
!==============================================================================
!Routines for the normal mode Transformation
!==============================================================================
subroutine init_normal_mode
  ! This suburoutine needs the correct value of d and n_winding !
  use normal_mode_data
  use input_parameters
  use constants
  use part_coordinates
  use r250_data
  implicit none

  real ran_loc

  sqrt2 = 2**(.5d0)
  insqrt2 = 2**(-.5d0)
  in_sqrt_N = (n_part)**(-.5d0)
  in_sqrt_P = (n_trot)**(-.5d0)
  in_sqrt_PN = in_sqrt_N * in_sqrt_P


  ! calculate Transaltion Vector----
  do i_part = 0 , n_part-1
     translation(i_part) = n_winding * d / (1.d0 * n_part )  * ( 1.d0 * i_part )
  end do

  ! disorderd phase
  if (f_rand_phase .eq. 1) then
     do i_part = 0 , n_part-1
        call r250(mz,ran_loc,1,kptr)
        translation(i_part) = translation(i_part) + 2 * pi * (ran_loc-0.5)
     end do
  end if

  !-calculate constant c---------------------
  c_const = n_winding**2 /(1.d0 * n_part )  !
  !-----------------------------------------

end subroutine init_normal_mode

!==============================================================================
subroutine trans_recip
 use part_coordinates
 use normal_mode_data
 implicit none
 real(kind=dp), dimension( : ), allocatable :: four_dummy_trot
 real(kind=dp), dimension( : ), allocatable :: four_dummy_part
 allocate( four_dummy_trot( 1:2*n_trot ) )
 allocate( four_dummy_part( 1:2*n_part ) )

 !-- Translation
 do i_part = 0 , n_part -1
    do i_trot =  0 , n_trot -1
       x(i_trot,i_part) =   x(i_trot,i_part) -  translation(i_part)
    end do
 end do

 !----------------------------------------------------------------------------
 ! decouple different oscillators
 do i_trot = 0 , n_trot-1

  do i_part = 0 , n_part-1
   four_dummy_part(2*i_part+1) = x(i_trot,i_part) ! re part
   four_dummy_part(2*i_part+2) = 0.               ! im part
  end do

  call four1(four_dummy_part,n_part,1,n_part)

  u(i_trot,0,0) = four_dummy_part(1)
  if ( n_part .gt. 1 ) then
   u(i_trot,n_part/2,0) = four_dummy_part(n_part+1 )
  end if

  do i_part = 1 , n_part/2 - 1
   u(i_trot,i_part,0) = four_dummy_part(2*i_part+1)  * sqrt2
   u(i_trot,i_part+n_part/2,0 ) = four_dummy_part(n_part + 2*i_part+2) * sqrt2
  end do

 end do ! i_trot
 !----------------------------------------------------------------------------

 !----------------------------------------------------------------------------
 ! diagonolize the trotter polymers
 do i_part = 0 , n_part-1

  do i_trot = 0 , n_trot-1
   four_dummy_trot(2*i_trot+1) = u(i_trot,i_part,0) ! re part
   four_dummy_trot(2*i_trot+2) = 0.                 ! im part
  end do


  call four1(four_dummy_trot,n_trot,1,n_trot)

  u(0,i_part,0) = four_dummy_trot(1)
  if (n_trot .gt. 1) then
   u(n_trot/2,i_part,0) = four_dummy_trot(n_trot+1 )
  end if

  do i_trot = 1 , n_trot/2 - 1
   u(i_trot,i_part,0) = four_dummy_trot(2*i_trot+1) * sqrt2
   u(i_trot+n_trot/2,i_part,0 ) = four_dummy_trot(n_trot + 2*i_trot+2) * sqrt2
  end do

 end do ! i_part
 !----------------------------------------------------------------------------

 i_order = 0
 do i_part = 0 , n_part-1
    do i_trot = 0 , n_trot-1
       u(i_trot,i_part,i_order) = u(i_trot,i_part,i_order) * in_sqrt_PN
    end do
 end do
end subroutine trans_recip
!==============================================================================
subroutine init_derivatives
 use part_coordinates
 implicit none

 !-------set derivatives to 0--------------
 do i_order = 1 , n_order                 !
    do i_part = 0, n_part-1               !
       do i_trot = 0, n_trot-1            !
          u(i_trot,i_part,i_order) = 0.   !
       end do                             !
    end do                                !
 end do                                   !
 !----------------------------------------
end subroutine init_derivatives
!==============================================================================
subroutine trans_real
 use part_coordinates
 use normal_mode_data
 implicit none

 real(kind=dp), dimension(1:2*n_trot_max) :: four_dummy_trot
 real(kind=dp), dimension(1:2*n_part_max) :: four_dummy_part


!-- trotter index fourie-transform -----------------------------
 !$OMP PARALLEL DO  PRIVATE(i_part,i_trot,four_dummy_trot) &
 !$ &               SHARED(u,x,insqrt2,n_part,n_trot)
 do i_part = 0 , n_part-1
```

```
  four_dummy_trot(1) = u(0,i_part,0)
  four_dummy_trot(2) = 0.
  if (n_trot .gt. 1) then
   four_dummy_trot(n_trot+1 ) = u(n_trot/2,i_part,0)
   four_dummy_trot(n_trot+2 ) = 0.
  end if

  do i_trot = 1 , n_trot/2 - 1
     four_dummy_trot(2*i_trot+1) = u(i_trot,i_part,0) * insqrt2 ! Re_part
     four_dummy_trot(2*i_trot+1+n_trot) = u(-i_trot+n_trot/2,i_part,0) &
          &                                    * insqrt2 ! Re_part
     four_dummy_trot( 2*i_trot+2) = - u(-i_trot+n_trot,i_part,0) &
          &                                    * insqrt2 !Im_part
     four_dummy_trot(2*i_trot+2+n_trot) = u(i_trot+n_trot/2,i_part,0)  &
          &                                    * insqrt2  !Im_part
  end do

  call four1(four_dummy_trot,n_trot,-1,n_trot_max)


  do i_trot = 0 , n_trot-1
     x(i_trot,i_part) = four_dummy_trot(2*i_trot+1) ! * in_sqrt_p
  end do

 end do ! i_part
 !$OMP END PARALLEL DO



!-- part index fourie transformation---------------------------------
 !$OMP PARALLEL DO  PRIVATE(i_part,i_trot,four_dummy_part) &
 !$ &               SHARED(x,insqrt2,n_part,n_trot,in_sqrt_pn,translation)
 do i_trot = 0 , n_trot-1

  four_dummy_part(1) = x(i_trot,0)
  four_dummy_part(2) = 0.
  if (n_part .gt. 1) then
   four_dummy_part(n_part+1 ) = x(i_trot,n_part/2)
   four_dummy_part(n_part+2 ) = 0.
  end if

  do i_part = 1 , n_part/2 - 1
   four_dummy_part(2*i_part+1) = x(i_trot,i_part) * insqrt2 ! Re_part
   four_dummy_part(2*i_part+1+n_part) = x(i_trot,-i_part+n_part/2)  &
        &                              * insqrt2 ! Re_part
   four_dummy_part( 2*i_part+2) = - x(i_trot,-i_part+n_part) &
        &                              * insqrt2 !Im_part
   four_dummy_part(2*i_part+2+n_part) = x(i_trot,i_part+n_part/2)   &
        &                              * insqrt2 !Im_part
  end do



  call four1(four_dummy_part,n_part,-1,n_part_max)

  do i_part = 0 , n_part-1
   x(i_trot,i_part) = four_dummy_part(2*i_part+1) * in_sqrt_pn
   x(i_trot,i_part) =  x(i_trot,i_part) + translation(i_part)
  end do
 end do ! i_trot
 !$OMP END PARALLEL DO


end subroutine trans_real
!================================================================================
subroutine Transform_force
 use part_coordinates
 use normal_mode_data
 implicit none
! real(kind=dp), dimension( : ), allocatable :: four_dummy_trot
! real(kind=dp), dimension( : ), allocatable :: four_dummy_part
! allocate( four_dummy_trot( 1:2*n_trot ) )
! allocate( four_dummy_part( 1:2*n_part ) )

 real(kind=dp), dimension(1:2*n_trot_max) :: four_dummy_trot
 real(kind=dp), dimension(1:2*n_part_max) :: four_dummy_part


!--decouple different oscillators
 !$OMP PARALLEL DO  PRIVATE(i_part,i_trot,four_dummy_part)  &
 !$ &               SHARED(force,sqrt2,n_part,n_trot)
 do i_trot = 0 , n_trot-1

  do i_part = 0 , n_part-1
   four_dummy_part(2*i_part+1) = force(i_trot,i_part) ! re part
   four_dummy_part(2*i_part+2) = 0.                ! im part
  end do

  call four1(four_dummy_part,n_part,1,n_part,n_part_max)

  force(i_trot,0) = four_dummy_part(1)
  if ( n_part .gt. 1 ) then
   force(i_trot,n_part/2) = four_dummy_part(n_part+1 )
  end if

  do i_part = 1 , n_part/2 - 1
   force(i_trot,i_part) = four_dummy_part(2*i_part+1)  * sqrt2
   force(i_trot,i_part+n_part/2 ) = four_dummy_part(n_part + 2*i_part+2)  &
        & * sqrt2
  end do

 end do ! i_trot
 !$OMP END PARALLEL DO

!--diagonolize the trotter polymers----------
 !$OMP PARALLEL DO  PRIVATE(i_part,i_trot,four_dummy_trot)  &
 !$ &               SHARED(force,sqrt2,n_part,n_trot)
 do i_part = 0 , n_part-1

  do i_trot = 0 , n_trot-1
   four_dummy_trot(2*i_trot+1) = force(i_trot,i_part) ! re part
   four_dummy_trot(2*i_trot+2) = 0.                 ! im part
  end do


  call four1(four_dummy_trot,n_trot,1,n_trot,n_trot_max)

  force(0,i_part) = four_dummy_trot(1)
  if (n_trot .gt. 1) then
   force(n_trot/2,i_part) = four_dummy_trot(n_trot+1 )
  end if

  do i_trot = 1 , n_trot/2 - 1
   force(i_trot,i_part) = four_dummy_trot(2*i_trot+1) * sqrt2
   force(i_trot+n_trot/2,i_part) = four_dummy_trot(n_trot + 2*i_trot+2) * sqrt2
  end do


  do i_trot = 0 , n_trot-1
     force(i_trot,i_part) = force(i_trot,i_part) * in_sqrt_pn
  end do

 end do ! i_part
 !$OMP END PARALLEL DO

end subroutine Transform_force
!================================================================================
!================================================================================
subroutine Transform_cl_force
 use part_coordinates
 use normal_mode_data
 implicit none
 real(kind=dp), dimension( : ), allocatable :: four_dummy_trot
 real(kind=dp), dimension( : ), allocatable :: four_dummy_part
 allocate( four_dummy_trot( 1:2*n_trot ) )
 allocate( four_dummy_part( 1:2*n_part ) )

!--decouple different oscillators
 do i_trot = 0 , n_trot-1

  do i_part = 0 , n_part-1
   four_dummy_part(2*i_part+1) = cl_force(i_trot,i_part) ! re part
   four_dummy_part(2*i_part+2) = 0.              ! im part
  end do

  call four1(four_dummy_part,n_part,1,n_part)

  cl_rez_force(i_trot,0) = four_dummy_part(1)
  if ( n_part .gt. 1 ) then
  cl_rez_force(i_trot,n_part/2) = four_dummy_part(n_part+1 )
  end if

  do i_part = 1 , n_part/2 - 1
   cl_rez_force(i_trot,i_part) = four_dummy_part(2*i_part+1)  * sqrt2
   cl_rez_force(i_trot,i_part+n_part/2 ) = four_dummy_part(n_part + 2*i_part+2)&
        &                                                    * sqrt2
  end do

 end do ! i_trot


!--diagonolize the trotter polymers----------
 do i_part = 0 , n_part-1

  do i_trot = 0 , n_trot-1
   four_dummy_trot(2*i_trot+1) = cl_rez_force(i_trot,i_part) ! re part
   four_dummy_trot(2*i_trot+2) = 0.                          ! im part
  end do


  call four1(four_dummy_trot,n_trot,1,n_trot)

  cl_rez_force(0,i_part) = four_dummy_trot(1)
  if (n_trot .gt. 1) then
   cl_rez_force(n_trot/2,i_part) = four_dummy_trot(n_trot+1 )
  end if

  do i_trot = 1 , n_trot/2 - 1
   cl_rez_force(i_trot,i_part) = four_dummy_trot(2*i_trot+1) * sqrt2
   cl_rez_force(i_trot+n_trot/2,i_part) = four_dummy_trot(n_trot + 2*i_trot+2) &
        &                                          * sqrt2
  end do

 end do ! i_part


 do i_part =  0 , n_part-1
    do i_trot = 0 , n_trot-1
       cl_rez_force(i_trot,i_part) = cl_rez_force(i_trot,i_part) * in_sqrt_pn
    end do
 end do



end subroutine Transform_cl_force
!================================================================================
!     nn is an integer 2**n
SUBROUTINE four1(data,nn,isign,n_array)
!SUBROUTINE four1(data,nn,isign)
  use constants
  INTEGER isign,nn, n_array
!  REAL(kind=dp) data(2*nn)
  REAL(kind=dp) data(2*n_array)
  INTEGER i,istep,j,m,mmax,n
  REAL(kind=dp) tempi,tempr
```

```
      REAL(kind=dp) theta,wi,wpi,wpr,wr,wtemp
      n = 2*nn
      j = 1
      do 11 i= 1, n,2
         if (j.gt.i) then
            tempr = data(j)
            tempi = data(j+1)
            data(j) = data(i)
            data(j+1) = data(i+1)
            data(i) = tempr
            data(i+1) = tempi
         end if
         m = n/2
1        if ((m.ge.2).and.(j.gt.m)) then
            j = j-m
            m = m/2
            goto 1
         end if
         j = j + m
11       continue
         mmax=2
2        if (n.gt.mmax) then
            istep = 2*mmax
            theta = 6.28318530717959d0 / (isign*mmax)
            wpr=-2.d0*sin(0.5d0*theta)**2
            wpi = sin(theta)
            wr = 1.d0
            wi = 0.d0
            do 13 m=1, mmax, 2
               do 12 i = m, n, istep
                  j = i + mmax
!                 tempr = sngl(wr)*data(j)-sngl(wi)*data(j+1)
                  tempr =      wr *data(j)-     wi *data(j+1)
!                 tempi = sngl(wr)*data(j+1)+sngl(wi)*data(j)
                  tempi =      wr *data(j+1)+     wi *data(j)
                  data(j) = data(i) - tempr
                  data(j+1) = data(i+1) - tempi
                  data(i) = data(i) + tempr
                  data(i+1) = data(i+1) + tempi
12                continue
                  wtemp=wr
                  wr=wr*wpr-wi*wpi+wr
                  wi=wi*wpr+wtemp*wpi+wi
13                continue
                  mmax = istep
                  goto 2
               end if
            END subroutine four1
            !  (C) Copr. 1986-92 Numerical Recipes Software +1].
```

G.7 Random number generator

```
!cccccccccc

     SUBROUTINE INR250(MZ,ISEED,KPTR)
!C
!C INITIALIZATION ROUTINE FOR R250 RANDOM NUMBER GENERATOR
!C FOR 32 BIT COMPUTERS ONLY !
!C USES SUN FOR STARTUP
!C
     INTEGER MZ(250)
     INTEGER ISEED, KPTR
!C
     INTEGER IUF(31),IGF(31)
     INTEGER IADDR1(1000),IADDR2(1000)
     REAL RRANF(2000)
!C
!C
     KPTR = 1
```

```
!C
      DO 100 I = 1,30
        IGF(I) = 2 ** (I - 1)
        IUF(I) = 2 ** I - 1
 100  CONTINUE
!C
      IGF(31) = 2**30
      IUF(31) = 2**30 + (2**30 - 1)
!C
      CALL SUN(ISEED,RRANF,250)
!C
!C MAP THE REALS TO POSITIVE INTEGERS
!C
      DO 200 I = 1,250
        MZ(I) = IDNINT(DBLE(RRANF(I))*DBLE(FLOAT(IUF(31))))
 200  CONTINUE
!C
!C SPECIAL TREATMENT OF THE FIRST 31 NUMBERS
!C
      DO 300 I = 1,31
        MZ(I) = IOR(MZ(I),IGF(I))
 300  CONTINUE
!C
      DO 400 I = 1,31
        MZ(I) = IAND(MZ(I),IUF(I))
 400  CONTINUE
!C
!C MIXING
!C
      CALL SUN(ISEED,RRANF,2000)
!C
      DO 500 I = 1,1000
        IADDR1(I) = 1. + 250. * RRANF(I)
        IADDR2(I) = 1. + 250. * RRANF(I + 1000)
 500  CONTINUE
!C
      DO 600 I = 1,1000
        NTEMP          = MZ(IADDR1(I))
        MZ(IADDR1(I)) = MZ(IADDR2(I))
        MZ(IADDR2(I)) = NTEMP
 600  CONTINUE
!C
!C WARMING UP
!C
      DO 800 I = 1,8
!C
        DO 710  K = 1,147
          MZ(K) = IEOR(MZ(K),MZ(K + 103))
 710    CONTINUE
!C
        DO 720 K = 148,250
          MZ(K) = IEOR(MZ(K),MZ(K - 147))
 720    CONTINUE
!C
 800  CONTINUE
!C
      RETURN
      END
!C
!C ---------------------------------------------------------------------
!C
      SUBROUTINE SUN(ISEED,RANF,N)
!C
!C STORES N REAL RANDOM NUMBERS IN RANF
!C ISEED IS A STARTUP VALUE
!C
      IMPLICIT DOUBLE PRECISION (A-H, O-Z)
!C
      INTEGER ISEED
      REAL RANF(N)
!C
!C
!C
      DATA FACTOR /41475557.0D0/, TWO28 /268435456.0D0/
!C
!C
!C
      IF (ISEED .GE. 0) THEN
        R=DBLE(ISEED)/TWO28
        R=DMOD(R*FACTOR,1.0D0)
        ISEED=-1
      END IF
!C
       DO 100 I = 1,N
        R=DMOD(R*FACTOR,1.0D0)
        RANF(I) = SNGL(R)
 100  CONTINUE
!C
      RETURN
      END
!C
!C
!C ---------------------------------------------------------------------
!C
      SUBROUTINE R250(MZ,RAN,N,KPTR)
!C
!C STORES N REAL RANDOM NUMBERS IN RAN USING THE INTEGERS IN MZ
!C THE RANDOM NUMBERS HAVE UNIFORM DISTRIBUTION IN THE INTERVAL (0,1)
!C
!      PARAMETER(RINV=1./2147483647.)  ! ifc : warning precision to big
      PARAMETER(RINV=1./2.147483647D9)
      INTEGER MZ(250)
      INTEGER N,KPTR
      REAL RAN(N)
      INTEGER I,L
!C
!C
      I = 0
      L = N + KPTR - 1
!C
 10   CONTINUE
!C
      KMIN = KPTR
      KMAX = MIN(L,147)
      DO 100 K = KMIN,KMAX
        MZ(K)  = IEOR(MZ(K),MZ(K + 103))
        I      = I + 1
        RAN(I) = MZ(K) * RINV
 100  CONTINUE
!C
      KMIN = MAX(KMIN,KMAX+1)
      KMAX = MIN(L,250)
!*VOCL LOOP,NOVREC
      DO 200 K = KMIN,KMAX
        MZ(K)  = IEOR(MZ(K),MZ(K - 147))
        I      = I + 1
        RAN(I) = MZ(K) * RINV
 200  CONTINUE
!C
      IF  (KMAX .EQ. 250) THEN
        KPTR = 1
        L    = L - 250
        GOTO 10
      END IF
!C
      KPTR = KMAX + 1
!C
      RETURN
      END
!C
!CCCCCCCCCCCCCCCCCCCC
!C
```

G.8 Input/output parameter file

```
------Read DATA-------------------
          500000       Relaxation Steps
         1000000       Observation Steps
     0.64000000E-01       Temperature [K]
     0.10000000E+00       Mass              ( =       0.48496606E+01amu)
     0.10000000E+00       Spring Constant [k_B K/A^2]
                0       0 : default conf
                        1 : read conf.sta
--- Data for def configuration
               64       d = 2 pi n_vol, for def. conf.
                1       n_winding
                0       f_roughness  0: alpha=0; 1: alpha =.5
                0       f_rand_phase 0: det phase , 1:rand phase
                2       0 : no potential
                        1 : harm. osz.
                        2 : - cos( x )
                        3 : (x^2-1)^2
                        4 : random
                        5 : (x-i_part 2 pi)^2
     0.10000000E+01       Potential strength
------Algorithm------------------
                0       0 : Primitiv Algorithem
                        1 : use HOA for ex. pot
                        2 : use HOA for all
                0       0 : T V Decomposition
                        1 : Q V Decomposition
                0       0 : x_cms moves
                        1 : x_cms is fixed
     0.00000000E+00       x_cms
                1       0 : mass for coll. h.c.
                        1 : cent. dynamic. mass.
     0.10000000E+00       ren. gap for fic. masses
     0.10000000E+02       sampling frequency
Simulation Parameters:
     0.50000000E-02       Time Step          ( =       0.38193000E-13s)
     0.20000000E+00       Friction
          853901       issed (random gen. init)
                5       Observation Step
                1       Number of Threds
              256       Trotter Number          ( 2^x )
               64       Number of Particles     ( 2^x )
                2       Order of the Integrator ( <=5 )
------Calculated Results----------
     0.81802335E+01      0.89346836E-01      class. Kinetic Energy
     0.22125529E+00      0.97140554E-02      Pot. Energy in the springs
     0.19739209E+01      0.00000000E+00      minimal Pot. Energy in the springs
    -0.33932825E+00      0.13375693E-01      Pot. Energy with external potential

     0.53164508E+00      0.86101955E-01      Kinetic Energy (Primitive Estimator)
     0.53717840E+00      0.98870488E-02      Kinetic Energy (Virial Estimator)

     0.00000000E+00      0.00000000E+00      Correction to Pot(*2) and Kin(*1) Energy
     0.38222657E-02      prob. for a Particle to be on top of the pot
     0.31358549E+01      Fisher's disorder parameter
     0.51098544E+01      0.32818491E+00      Gyration radius
     0.84315996E+01      Time needed from i_md = 400 to i_md = 900
```

Erklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbständig verfaßt und keine anderen als die angegebenen Hilfsmittel verwendet habe.

Mainz, den 4. August 2003

(Florian Krajewski)