Path Integral Monte Carlo and Hydrogen bond

Theo Saporiti CID: 01830497

May 12, 2020

Supervisor: Dr. Jarvist Moore Frost

Assessor: Prof. Jenny Nelson

Word count: 5994 words (6000 allowed)

Abstract

In this project a path integral Monte Carlo computer program based on the Metropolis-Hastings algorithm is written to numerically simulate quantum toy models, such as the simple quantum harmonic oscillator, and the hydrogen bond between water molecules. Toy models are compared with their expected theoretical results and it is attempted to observe nuclear quantum effects in the hydrogen bond. In the first part of the project, which is theoretical, Feynman's path integral formulation of quantum mechanics is introduced and used to reformulate the analytical problem of computing the zero-point energy of a system into a probabilistic one, hence justifying the use of Monte Carlo methods. A formal mathematical application of a Wick rotation is presented. The Metropolis-Hastings algorithm is explained and a modelization of the hydrogen bond is presented. In the second part of the project, which is computational, various physical quantities are measured in both toy models and the hydrogen bond model: for toy models the zero-point energy, the position probability density function in function of temperature and the size of particles are observed while for the hydrogen bond the bond length, the bond energy and tunnelling are observed. It is found that the execution of the Metropolis-Hastings algorithm is influenced by the simulated system. Simulated toy models behave as theoretically predicted: correct zero-point energies are measured, position probability density functions correctly vary with temperature, particles delocalize correctly in time and tunnelling is observed. Simulations regarding the hydrogen bond don't show any nuclear quantum effects and no secondary geometric isotope effect is observed.

Contents

1 Introductory chapter **4**
- 1.1 Motivation . . . 4
- 1.2 Aims and objectives . . . 4

2 Discussions of the methods employed **4**
- 2.1 Feynman's path integral formalism . . . 4
  - 2.1.1 Justification . . . 4
  - 2.1.2 Derivation of the path integral . . . 5
- 2.2 Wick rotation and Euclidean action . . . 5
  - 2.2.1 Justification . . . 5
  - 2.2.2 Derivation of the Euclidean action . . . 5
- 2.3 Measurement of the zero-point energy . . . 7
  - 2.3.1 Justification . . . 7
  - 2.3.2 Derivation of the zero-point energy . . . 7
- 2.4 Numerical implementation . . . 8
  - 2.4.1 Justification of the Metropolis-Hastings algorithm . . . 8
  - 2.4.2 Steps of the Metropolis-Hastings algorithm . . . 9
  - 2.4.3 Possible propositions of moves and optimisations . . . 9
  - 2.4.4 Units in the programme . . . 10
  - 2.4.5 Parameters required by the programme . . . 10
- 2.5 The hydrogen bond . . . 11
  - 2.5.1 Brief description . . . 11
  - 2.5.2 Theoretical model . . . 11
  - 2.5.3 Numerical implementation . . . 12

3 Results **13**
- 3.1 Verification of the Metropolis-Hastings algorithm . . . 13
- 3.2 Particle size . . . 13
  - 3.2.1 Theoretical predictions . . . 13
  - 3.2.2 Numerical results and discussion . . . 14
- 3.3 Application of the Metropolis-Hastings algorithm on toy models . . . 15
  - 3.3.1 Effect of temperature on the position PDF . . . 15
  - 3.3.2 Analytical computations, numerical modelling and results . . . 16
  - 3.3.3 Discussion of results . . . 19
- 3.4 Hydrogen bond . . . 20
  - 3.4.1 Fixed-R model of the H-bond . . . 20
  - 3.4.2 Free-R model of the H-bond . . . 20
  - 3.4.3 Measuring the bond length . . . 21
  - 3.4.4 Numerical results of both models and discussion . . . 21

4 Conclusions **23**

5 Acknowledgements **24**

A Conventions of notation **27**

B Complete derivation of the path integral **27**

C Proof of properties of $\tilde{f}$ **28**

D Explicit calculation for the zero-point energy **28**

E Proof of the Virial theorem **29**

F Convergence of the Metropolis-Hastings algorithm **29**

G Numerical implementation of the Cauchy distribution **30**

H Justification of unit prefixes **30**

I Source code **31**

Declaration of work undertaken

The work was distributed thematically: I focused on the theoretical aspects while my project partner focused on the writing of the computer program. The formal application of a Wick rotation is of my invention. The writing of the `C++` program was mainly done by my project partner and partially by me. Debugging was done by both. All figures and all appendices are of my own production. All results were produced by me, expect those of the free-R model of the hydrogen bond, produced by my project partner. No summer placements or previous works have been conducted with the supervisor and this project is not an extension of any previous work.

1 Introductory chapter

1.1 Motivation

Most of atomistic numerical simulations of chemical, biological and material systems treat nuclei as classical particles and only electrons as quantum mechanical entities. This approximation, dictated by computational limitations, neglects nuclear quantum effects (NQE) and hence it makes impossible to study fundamental properties dependent of them [1]. Feynman's path integral formalism can be used to to reduce the computational cost generated when accounting for these NQE. Its direct numerical evaluation is computationally too expensive and perturbative expansions fail for some systems, therefore a Monte Carlo approach is used [2].

1.2 Aims and objectives

The first aim of this project is to understand the path integral formulation of quantum mechanics and apply it to write a quantum Monte Carlo computer program. The objective is to get a full picture of the path integral formalism, maintain theoretical clarity in derivations, justify every mathematical step, give great importance to mathematical details and their physical implications, and to use all the grasped knowledge to write a `C++` programme. Specifically, an objective is to formulate and apply a Wick rotation formally.

The second aim of this project is to apply this computer program to simulate quantum toy models, to compare them with theoretical predictions, and to simulate the hydrogen bond interaction to numerically investigate NQE and the secondary geometric isotope effect. For toy models, the objective is to simulate a particle in an infinite well, double well and in an harmonic potential, and measure its zero-point energy (ZPE), size and its position probability density function with varying temperature. For the hydrogen bond the objective is to measure ZPEs and bond lengths under different conditions.

2 Discussions of the methods employed

Along this report we use the conventions stated in app. (A).

2.1 Feynman's path integral formalism

2.1.1 Justification

Consider a particle of mass $m\,[\mathrm{kg}]$ trapped in a potential $V(x)\,[\mathrm{J}]$ independent of time in a one-dimensional Universe. Given an Hamiltonian operator $\hat{H}$, the evolution in time $t\,[\mathrm{s}]$ of the quantum state $|\psi(t)\rangle$ is the solution of the Schrödinger equation

$$i\hbar\frac{\mathrm{d}}{\mathrm{d}t}|\psi(t)\rangle = \hat{H}\,|\psi(t)\rangle\,, \tag{1}$$

where $\hbar = 1.055\cdot 10^{-34}\,\mathrm{Js}$ [3] is the reduced Planck constant. If at $t = 0\,\mathrm{s}$ the state is $|\psi(0)\rangle$ then the solution of eq. (1) is $|\psi(t)\rangle = \hat{U}(t,0)\,|\psi(0)\rangle$, where

$$\hat{U}(t_f,t_i) := \exp\left(-\frac{i}{\hbar}(t_f-t_i)\hat{H}\right) \tag{2}$$

is the time evolution operator that evolves quantum states from time $t_i\,[\mathrm{s}]$ to $t_f\,[\mathrm{s}]$ [4, p.10]. An interesting mathematical observation is

$$|\psi(t_f)\rangle = \hat{U}(t_f,t_i)\,|\psi(t_i)\rangle = \hat{U}(t_f,t_i)I_{x_i}\,|\psi(t_i)\rangle = \int_{\mathbb{R}}\hat{U}(t_f,t_i)\,|x_i\rangle\,\langle x_i|\psi(t_i)\rangle\,\mathrm{d}x_i, \tag{3}$$

hence multiplying on the left by the position eigenstate $\langle x| = \langle x_f|$, $x\,[\mathrm{m}]$, we get [2]

$$\psi(x_f, t_f) = \int_{\mathbb{R}} K(x_f, t_f; x_i, t_i)\psi(x_i, t_i)\mathrm{d}x_i, \tag{4}$$

where

$$K(x_f, t_f; x_i, t_i) = K(f, i) := \left\langle x_f \middle| \hat{U}(t_f, t_i) \middle| x_i \right\rangle \tag{5}$$

is called the propagator [5]. Notationally $i \equiv (x_i, t_i)$ and $f \equiv (x_f, t_f)$ in $K(f, i)$.

2.1.2 Derivation of the path integral

Feynman's path integral arises as a consequence of the explicit calculation of $K(f, i)$, given in app. (B). It is found that [5]

$$K(f, i) = \lim_{N\to\infty} \int_{\mathbb{R}} \mathrm{d}x_1 \cdots \int_{\mathbb{R}} \mathrm{d}x_{N-1} A_\varepsilon^N \exp\left(\varepsilon \frac{i}{\hbar} \sum_{k=1}^{N} \left[\frac{m}{2}\left(\frac{x_k - x_{k-1}}{\varepsilon}\right)^2 - V(x_{k-1})\right]\right), \tag{6}$$

where $\varepsilon = (t_f - t_i)/N$, $x_0 = x_i$, $x_N = x_f$ and $A_\varepsilon := \sqrt{m/(2\pi i\hbar\varepsilon)}$. The physical interpretation of $K(f, i)$ is deduced by recognising the discrete approximation of the action $S[x(t)]\,[\mathrm{Js}]$ of a path $x(t) : [t_i; t_f] \to \mathbb{R}$. If we interpret $\varepsilon = (t_f - t_i)/N$ as a time step and we identify $x_k = x(t_i + k\varepsilon) = x(t_k)$ at time slice t_k then [4, p.12]

$$\varepsilon \sum_{k=1}^{N} \left[\frac{m}{2}\left(\frac{x_k - x_{k-1}}{\varepsilon}\right)^2 - V(x_{k-1})\right] \approx \int_{t_i}^{t_f} \left[\frac{1}{2}m\dot{x}^2 - V(x)\right] \mathrm{d}t = S[x(t)]. \tag{7}$$

For each infinitesimal volume element $\mathrm{d}V = \mathrm{d}x_1 \cdots \mathrm{d}x_{N-1}$ of $\mathbb{R}^{N-1}$ the $A_\varepsilon^N \exp(iS[x(t)]/\hbar)\mathrm{d}V$ contribution is added to the overall infinite sum. The contribution only depends on $(x_1, \ldots, x_{N-1})$ and the $(N-1)$-integral varies each of these coordinates: the physical interpretation is that each contribution comes from one specific path, represented by its discretization $(x_1, \ldots, x_{N-1})$, of all the infinite possible ones from x_0 to x_N [4, p.12]. This concept of summation over all possible paths defines the path integral [5]

$$\int \exp\left(\frac{i}{\hbar} S[x(t)]\right) \mathrm{D}x(t) := K(f, i). \tag{8}$$

2.2 Wick rotation and Euclidean action

2.2.1 Justification

In general $K(f, i) \in \mathbb{C}$: in sight of numerical treatment we want to only consider real-valued quantities [2]. $K(f, i)$ can become real-valued under three conditions: if by analytic continuation paths $x(t) : \mathbb{R} \to \mathbb{R}$ are allowed to be such that $x(t) : \mathbb{C} \to \mathbb{C}$, if we assume that the physical model evolves in time along the negative imaginary axis $t \in i\mathbb{R}_-$, and if a $\pi/2$-rotation on the complex plane of times, called a Wick rotation, is performed to rotate imaginary-valued times back into real-valued times [4, p.56].

2.2.2 Derivation of the Euclidean action

It is standard practise in the literature (see [2, 4, 6, 7]) to perform the Wick rotation in eq. (6) as a mere substitution $t \to ti =: \tau\,[\mathrm{s}]$, where $\tau \in \mathbb{R}$ is called the imaginary time and $t \in \mathbb{R}_-$ [2]. In this work we apply the Wick rotation more formally with mathematical arguments of our own invention: this allows a deeper understanding of its implications.

Consider a general continuous function $f : \mathbb{C} \to \mathbb{C}$. Define $\widetilde{f} : \mathbb{C} \to \mathbb{C}$ to be such that $\widetilde{f}(zi) = f(z)$ for all $z \in \mathbb{C}$. In app. (C) we prove useful properties of $\widetilde{f}$. Let $a, b \in \mathbb{R}$ such that $a < b$ and consider the integral

$$\int_{[-ai,-bi]} f(z)\mathrm{d}z = \int_a^b f(z(\lambda))\dot{z}(\lambda)\mathrm{d}\lambda = -i\int_a^b f(-\lambda i)\mathrm{d}\lambda, \tag{9}$$

where the line $[-ai, -bi]$ has been parametrized by $z(\lambda) = -\lambda i$ for $\lambda \in [a; b]$. Then using the same parametrization

$$\int_{[-ai,-bi]} f(z)\mathrm{d}z = \int_{[-ai,-bi]} \tilde{f}(zi)\mathrm{d}z = -i\int_a^b \tilde{f}(\lambda)\mathrm{d}\lambda. \tag{10}$$

Now Wick rotate the line $[-ai, -bi] \to [a, b]$ and compute the integral of $\widetilde{f}$ along it:

$$\int_{[a,b]} \widetilde{f}(z)\mathrm{d}z = \int_a^b \widetilde{f}(\lambda)\mathrm{d}\lambda, \tag{11}$$

where this time we used the parametrization $z(\lambda) = \lambda$. This proves that

$$\int_{[-ai,-bi]} f(z)\mathrm{d}z = \frac{1}{i}\int_{[a,b]} \widetilde{f}(z)\mathrm{d}z. \tag{12}$$

The above result is now applied to the path integral: recall that $t_i, t_f \in i\mathbb{R}_-$. Using properties in eq. (69) we see that the action becomes

$$S[x(t)] = \int_{[t_i,t_f]} \left[\frac{m}{2}\left(\frac{\mathrm{d}x}{\mathrm{d}t}(t)\right)^2 - V(x(t))\right]\mathrm{d}t = -\frac{1}{i}\int_{[it_i,it_f]} \left[\frac{m}{2}\left(\frac{\mathrm{d}\tilde{x}}{\mathrm{d}\tau}(\tau)\right)^2 + V(\tilde{x}(\tau))\right]\mathrm{d}\tau, \tag{13}$$

where we also changed the name of the integration variable to τ. By definition $\widetilde{x}(\tau) = \widetilde{x}(ti) = x(t)$, hence $\tilde{x}(\tau) \in \mathbb{R}$. This motivates us to define the Euclidean action $S_E[\tilde{x}(\tau)]$ [Js] as [2]

$$S_E[\tilde{x}(\tau)] := \int_{[\tau_i,\tau_f]} \left[\frac{m}{2}\left(\frac{\mathrm{d}\tilde{x}}{\mathrm{d}\tau}(\tau)\right)^2 + V(\tilde{x}(\tau))\right]\mathrm{d}\tau = \int_{\tau_i}^{\tau_f} H\left(\tilde{x}(\tau), \frac{\mathrm{d}\tilde{x}}{\mathrm{d}\tau}(\tau)\right)\mathrm{d}\tau, \tag{14}$$

where $\tau_i = it_i$, $\tau_f = it_f$ and clearly $S_E[\tilde{x}(\tau)] \in \mathbb{R}$. We recognise the presence of the time independent classical Hamiltonian $H(x, \dot{x})$ [J]. The relationship between the two actions is

$$S[x(t)] = iS_E[\tilde{x}(\tau)], \tag{15}$$

therefore the path integral becomes a real quantity [2]

$$\begin{aligned}\int \exp\left(\frac{i}{\hbar}S[x(t)]\right)\mathrm{D}x(t) &= \int \exp\left(-\frac{S_E[\tilde{x}(\tau)]}{\hbar}\right)\mathrm{D}\tilde{x}(\tau) \\ &= \lim_{N\to\infty}\int_{\mathbb{R}}\mathrm{d}\tilde{x}_1\cdots\int_{\mathbb{R}}\mathrm{d}\tilde{x}_{N-1}A_{\delta\tau}^N \exp\left(-\frac{\delta\tau}{\hbar}\sum_{k=1}^{N}\left[\frac{m}{2}\left(\frac{\tilde{x}_k - \tilde{x}_{k-1}}{\delta\tau}\right)^2 + V(\tilde{x}_{k-1})\right]\right),\end{aligned} \tag{16}$$

where $\delta\tau = i\varepsilon$ [s] and we defined the constant $A_{\delta\tau} := A_{\varepsilon=-i\delta\tau} = \sqrt{m/(2\pi\hbar\delta\tau)}$. Notice that $\tilde{x}_k = \tilde{x}(\tau_k) = x(t_k) = x_k$, therefore the tilde symbol can be dropped $\tilde{x}_k = x_k$.

2.3 Measurement of the zero-point energy

2.3.1 Justification

Denote $\{|n\rangle\}_{n\in\mathbb{N}}$ the energy eigenstates of the QM system, $\{E_n\}_{n\in\mathbb{N}}$ [J] their eigenenergies and suppose E_0 is not degenerate. We want to compute the ZPE $E_0 = \langle 0|\hat{H}|0\rangle$. Consider the system under the statistical mechanics (SM) formalism at thermal equilibrium where the possible microstates are $\{|n\rangle\}_{n\in\mathbb{N}}$ and they follow the Boltzmann distribution [2]. Microstates probabilities are denoted $p_n \in [0;1]$. We want to find a link between the partition function of the system

$$Z(\beta) := \sum_{n=0}^{\infty} e^{-\beta E_n/\hbar}, \qquad \text{with} \quad \beta := \frac{\hbar}{k_B T_K}, \tag{17}$$

and the real-valued path integral in eq. (16). β [s] is the thermodynamic temperature, T_K [K] is the temperature of the system and $k_B = 1.381 \cdot 10^{-23}$ J/K [3] the Boltzmann constant. Clearly $p_n = \exp(-\beta E_n/\hbar)/Z(\beta)$.

A connection between $Z(\beta)$ and eq. (16) exists under the condition that periodic boundary conditions (PBCs) are forced to all paths, meaning that imaginary time is cyclic.

2.3.2 Derivation of the zero-point energy

The following derivation has been modified, clarified and expanded from reference [4, p.56-57].

E_0 is clearly the SM average energy of the system in the limit of close to the absolute zero temperatures $\beta \to \infty$:

$$\lim_{\beta\to\infty} \langle E\rangle_{\text{SM}} = \lim_{\beta\to\infty} \frac{\sum_{n=0}^{\infty} E_n e^{-\beta E_n/\hbar}}{\sum_{n=0}^{\infty} e^{-\beta E_n/\hbar}} = \lim_{\beta\to\infty} \frac{e^{-\beta E_0/\hbar}(E_0 + E_1 e^{-\beta(E_1-E_0)/\hbar} + \cdots)}{e^{-\beta E_0/\hbar}(1 + e^{-\beta(E_1-E_0)/\hbar} + \cdots)} = E_0. \tag{18}$$

Indeed, as E_0 is not degenerated, $E_k - E_0 > 0$ for $k > 0$. In terms of $Z(\beta)$:

$$E_0 = \lim_{\beta\to\infty} \frac{\sum_{n=0}^{\infty} E_n e^{-\beta E_n/\hbar}}{\sum_{n=0}^{\infty} e^{-\beta E_n/\hbar}} = \lim_{\beta\to\infty} -\hbar\frac{\partial}{\partial\beta} \ln Z(\beta). \tag{19}$$

$Z(\beta)$ can be written in the bra-ket notation:

$$\begin{aligned} Z(\beta) &= \sum_{n=0}^{\infty} \left\langle n \left| I_{x'} e^{-\beta\hat{H}/\hbar} \right| n \right\rangle = \sum_{n=0}^{\infty} \int_{\mathbb{R}} \mathrm{d}x' \, \langle n \,|\, x'\rangle \left\langle x' \left| e^{-\beta\hat{H}/\hbar} \right| n \right\rangle \\ &= \int_{\mathbb{R}} \mathrm{d}x' \sum_{n=0}^{\infty} \left\langle x' \left| e^{-\beta\hat{H}/\hbar} \right| n \right\rangle \langle n \,|\, x'\rangle = \int_{\mathbb{R}} \mathrm{d}x' \left\langle x' \left| e^{-\beta\hat{H}/\hbar} I_n \right| x' \right\rangle. \end{aligned} \tag{20}$$

This is equivalent to eq. (5) if one sets $x_i = x_f = x'$ and $t_i = 0$, $t_f = -\beta i$: the position conditions are the PBCs while the time ones identify the thermodynamic temperature with the imaginary time $Ti = \beta$, where we denote $T = t_f$ the cyclic imaginary-valued time. $\hbar$ in eq. (17) forces β to have time units. Knowing $\beta = iT$:

$$\begin{aligned} Z(\beta) &= \int_{\mathbb{R}} \mathrm{d}x' \left\langle x' \left| e^{-iT\hat{H}/\hbar} \right| x' \right\rangle = \int_{\mathbb{R}} \mathrm{d}x' \left\langle x' \left| \hat{U}(T,0) \right| x' \right\rangle \\ &= \int_{\mathbb{R}} \mathrm{d}x' \int \exp\left(\frac{i}{\hbar} S[x(t)]\right) \mathrm{D}x(t) = \int_{\mathbb{R}} \mathrm{d}x' \int \exp\left(-\frac{S_E[\tilde{x}(\tau)]}{\hbar}\right) \mathrm{D}\tilde{x}(\tau). \end{aligned} \tag{21}$$

Paths in the path integral are such that $x(0) = x(T) = x' \Leftrightarrow \tilde{x}(0) = \tilde{x}(\beta) = x'$. The external integral over x' takes care that all possible PBCs are considered.

Eq. (21) gives the link between $Z(\beta)$ and the path integral. In app. (D) eq. (21) is injected into eq. (19) using its limit definition in eq. (16) that implies $\beta = N\delta\tau$ and $x_0 = x_N$: the complete computation is performed there, here we highlight two conceptually important steps of that calculation. The general result of app. (D), which can be guessed from eq. (19), is that E_0 can be seen as

$$E_0 = \lim_{\substack{\beta\to\infty \\ N\to\infty}} \langle F(\tilde{\mathbf{x}})\rangle \,, \tag{22}$$

where $F : \mathbb{R}^N \to \mathbb{R}$ is seen as a function of N random variables $\tilde{\mathbf{x}} = (\tilde{x}_0, \dots, \tilde{x}_{N-1})$ that are distributed according to the multi-variable probability density function (PDF)

$$\omega(\tilde{\mathbf{x}}) := \frac{e^{-S_E[\tilde{\mathbf{x}}]/\hbar}}{\int_{\mathbb{R}} \mathrm{d}\tilde{x}_0 \int_{\mathbb{R}} \mathrm{d}\tilde{x}_1 \cdots \int_{\mathbb{R}} \mathrm{d}\tilde{x}_{N-1} e^{-S_E[\tilde{\mathbf{x}}]/\hbar}}, \tag{23}$$

whose multi-variable integral is 1 and $\omega(\tilde{\mathbf{x}}) > 0$. $S_E[\tilde{\mathbf{x}}]$ is the discretized Euclidean action found in eq. (16). In app. (D) we compute two different forms of F.

The first expression of F is obtained by introducing at one line of the derivation in app. (D) an arithmetic average over all N time slices t_k of the Hamiltonian, associated to a path $\tilde{x}(\beta)$, at time t_k with $0 \le k \le N-1$. This can be done because time is cyclic and therefore any time $\tau \in [0; \beta]$ can be chosen as the end time of one cycle of time [6]. This is also done because numerically the average reduces statistical errors [6]. By doing so it is obtained [4, p.57]

$$F_K(\tilde{\mathbf{x}}) = \frac{1}{N}\sum_{k=1}^{N}\left[\frac{m}{2}\left(\frac{\tilde{x}_k - \tilde{x}_{k-1}}{\delta\tau}\right)^2 + V(\tilde{x}_k)\right]. \tag{24}$$

Using F_K is not suitable for numerical implementations because of $1/\delta\tau$ terms [4, p.57].

The second expression of F removes $1/\delta\tau$ terms by applying the Virial theorem to eq. (24). In app. (E) we show $\langle 0|\hat{T}|0\rangle = \langle 0|xV'(x)/2|0\rangle$, which justifies the substitution of the kinetic terms in eq. (24) with (first derivative) potential-like terms [4, p.57]:

$$F_V(\tilde{\mathbf{x}}) = \frac{1}{N}\sum_{k=1}^{N}\left[V(\tilde{x}_k) + \frac{\tilde{x}_k}{2}V'(\tilde{x}_k)\right]. \tag{25}$$

2.4 Numerical implementation

2.4.1 Justification of the Metropolis-Hastings algorithm

Given the PDF in eq. (23), a numerical evaluation of eq. (22) requires N integrals, each discretized into N integration points, hence the time complexity of the computation grows as $O(N^N)$. This asymptotic behaviour, called 'the curse of dimensions' [4, p.58], makes a direct numerical evaluation of eq. (22) practically impossible. In order to avoid inordinately long computational times, E_0 is estimated as [4, p.58]

$$E_0 = \lim_{\substack{\beta\to\infty \\ N\to\infty}} \langle F(\tilde{\mathbf{x}})\rangle \approx \frac{1}{M}\sum_{l=1}^{M} F(\tilde{\mathbf{x}}^l) =: E_{\mathrm{M}}, \tag{26}$$

where $\tilde{\mathbf{x}}^l = (\tilde{x}_0^l, \dots, \tilde{x}_{N-1}^l)$ is one of the M realisations of $\tilde{\mathbf{x}}$ $(1 \le l \le M)$ and M, N, β are assumed large. The error on E_{M} is [4, p.58]

$$\Delta E_{\mathrm{M}} = \frac{s_{F(\tilde{\mathbf{x}})}}{\sqrt{M}} = \sqrt{\frac{1}{M}\left[\langle F(\tilde{\mathbf{x}})^2\rangle - \langle F(\tilde{\mathbf{x}})\rangle^2\right]} = \sqrt{\frac{1}{M}\left[\frac{1}{M}\sum_{l=1}^{M} F(\tilde{\mathbf{x}}^l)^2 - \left(\frac{1}{M}\sum_{l=1}^{M} F(\tilde{\mathbf{x}}^l)\right)^2\right]}, \tag{27}$$

where $s_{F(\tilde{\mathbf{x}})}$ is the standard deviation of $F(\tilde{\mathbf{x}})$. Numerical results are presented as $E_{\mathrm{M}} \pm \Delta E_{\mathrm{M}}$. The realisations $\tilde{\mathbf{x}}^l$ are numerically generated through the Metropolis-Hastings algorithm (MHA) which randomly samples the distribution in eq. (23) with an importance sampling criteria, hence this approach to compute E_0 is a Monte Carlo method [7]. The MHA produces a Markov chain $\tilde{\mathbf{x}}^0 \to \tilde{\mathbf{x}}^1 \to \cdots \to \tilde{\mathbf{x}}^I$ (a random walk in the state space of discretized paths $\tilde{\mathbf{x}}$) where each path $\tilde{\mathbf{x}}^j$ (except $\tilde{\mathbf{x}}^0$) only depends on the previous one $\tilde{\mathbf{x}}^{j-1}$ and it is selected more likely in the highest probable regions of $\mathbb{R}^N$ given by eq. (23) [7]. Each generation of a path is called a sweep, hence in the previous chain I sweeps have been performed: I is called the Monte Carlo time [2].

2.4.2 Steps of the Metropolis-Hastings algorithm

Starting from the initial condition $\tilde{\mathbf{x}}^0$, the MHA runs as follows [2]:

1. Given the current j-th path, propose a possible $(j+1)$-th path $\tilde{\mathbf{x}}'$ by applying one of the available moves stated in sec. (2.4.3). The move is randomly chosen according to a specified moves distribution P_M (see sec. (2.4.5));

2. Compute $r = \omega(\tilde{\mathbf{x}}')/\omega(\tilde{\mathbf{x}}^j) = \exp(-\Delta S_E/\hbar)$ (the denominator of eq. (23) is not computed), with $\Delta S_E := S_E[\tilde{\mathbf{x}}'] - S_E[\tilde{\mathbf{x}}^j]$ being the change in Euclidean action between $\tilde{\mathbf{x}}'$ and $\tilde{\mathbf{x}}^j$, and accept the move with probability $p = \min(1, r)$: if accepted, set $\tilde{\mathbf{x}}^{j+1} = \tilde{\mathbf{x}}'$, otherwise set $\tilde{\mathbf{x}}^{j+1} = \tilde{\mathbf{x}}^j$;

3. If $j + 1 < I$ set $j \to j + 1$ and go back to step 1, otherwise stop here.

The MHA always ($p = 1$) accepts moves such that $\Delta S_E \leq 0$ and sometimes ($p = r < 1$) it accepts moves such that $\Delta S_E > 0$ [2].

In app. (F) we prove that the more I is large, the more the chain approaches the stationary distribution equilibrium given by eq. (23) [4, p.59]. This approaching process is called thermalization [7]: we numerically confirm thermalization when $\langle x^2 \rangle^j := [(x_0^j)^2 + \cdots + (x_{N-1}^j)^2]/N\,[\mathrm{m}^2]$ stabilises along sweeps after a certain measured R-th sweep [2], called the thermalisation time. No measurement $F(\tilde{\mathbf{x}}^l)$ is performed before the R-th sweep, thus the choice of $\tilde{\mathbf{x}}^0$ is arbitrary [4, p.62]. A cold start is defined as $\tilde{\mathbf{x}}^0 = \mathbf{1}\xi = (\xi, \ldots, \xi)$ for $\xi \in \mathbb{R}$, while an hot start is defined as $\tilde{\mathbf{x}}^0$ having random components [4, p.62].

2.4.3 Possible propositions of moves and optimisations

Given the current path $\tilde{\mathbf{x}}^j$, the proposed path $\tilde{\mathbf{x}}' = (\tilde{x}'_0, \ldots, \tilde{x}'_{N-1})$ is generated as follows [2, 8]:

1. **Local move**: initially set $\tilde{\mathbf{x}}' = \tilde{\mathbf{x}}^j$, chose a random site x_k^j, a random displacement $u\,[\mathrm{m}]$ either uniformly in the interval $[-h_j; h_j]$, $h_j\,[\mathrm{m}]$, or according to a Cauchy distribution of parameters $(x_0, \gamma) = (0, h_j)$ (see app. (G)) and modify $x'_k = x_k^j + u$. Update $h^j \to h_{j+1} = h_j \cdot r_A/r_I$: this is done so that the (empirical) acceptance rate per sweep of this move $r_A \in [0; 1]$ converges towards a desired fixed ideal rate $r_I \in [0; 1]$;

2. **Global displacement**: exactly like a local move, but the move $x'_k = x_k^j + u$ is performed at all sites $0 \leq k \leq N - 1$ and it is set $h_j = h_0$ always;

3. **Bisection**: select a random site x_k^j, then apply a global displacement but only for the subset of positions $\{x_m^j : m \in [k; (k + \mathrm{floor}((N-1)s_B) \bmod N)]\}$ for $s_B \in [0; 1]$;

4. **Mirror**: perform the inversion $\tilde{\mathbf{x}}' = -\tilde{\mathbf{x}}^j$;

5. **Center of mass displacement**: calculate the average position/'center of mass' $\langle x \rangle^j := (x_0^j + \cdots + x_{N-1}^j)/N$ [m] and perform a global displacement with $u = \langle x \rangle^j$;

6. **Swap**: select 2 random particles, select a random initial and final sites x_n^j, x_m^j such that $|x_n^j - x_m^j| > 1$ and swap all the sites $x_n^j < x_k^j < x_m^j$ among the two particles. This move is activated only when there are more than 2 particles.

When a new path $\tilde{\mathbf{x}}'$ is proposed, some of its components may be identical to those in $\tilde{\mathbf{x}}^j$, thus when calculating ΔS_E terms dependent of those components would cancel each other. In order to optimise the programme, the computation of ΔS_E is specifically implemented for each move so that only the positions $x'_k \neq x_k^j$ are taken into account [2]. For example, for a local move, only the moved site (potential energy variation) and its two nearest neighbours (kinetic energy variation) are considered.

The MHA is implemented as a `C++` programme: source code is available in app. (I).

2.4.4 Units in the programme

Standard SI units are used. Products of quantities that span different order of magnitudes are common, hence it is appropriate to introduce custom prefixes so that a physical quantity q can be expressed as $q = q_N \cdot 10^Q\,[q]$, where q_N (unitless) is close to unity and Q in the 10^Q prefix is the typical order of magnitude of q. In app. (H) we justify the following values of Q for all the relevant quantities in this work:

$$\hbar = \hbar_N \cdot 10^{-34}\,\text{Js}, \qquad m = m_N \cdot 10^{-30}\,\text{kg}, \qquad x = x_N \cdot 10^{-10}\,\text{m}, \qquad \delta\tau = \delta\tau_N \cdot 10^{-15}\,\text{s},$$
$$k_B = k_{B,N} \cdot 10^{-23}\,\text{J/K}, \qquad V = V_N \cdot 10^{-20}\,\text{J}, \qquad \omega = \omega_N \cdot 10^{15}\,\text{Hz}, \qquad \beta = \beta_N \cdot 10^{-15}\,\text{s}.$$

ω [Hz] is the angular frequency of an oscillator. This way maximal machine precision is guaranteed as all sums have the same prefix and $\Delta S_E/\hbar$ in r is close to unity. In the `C++` programme only q_N quantities are considered and only at the end of simulations q quantities are recovered by multiplying q_N with the correct final prefix, determined analytically.

2.4.5 Parameters required by the programme

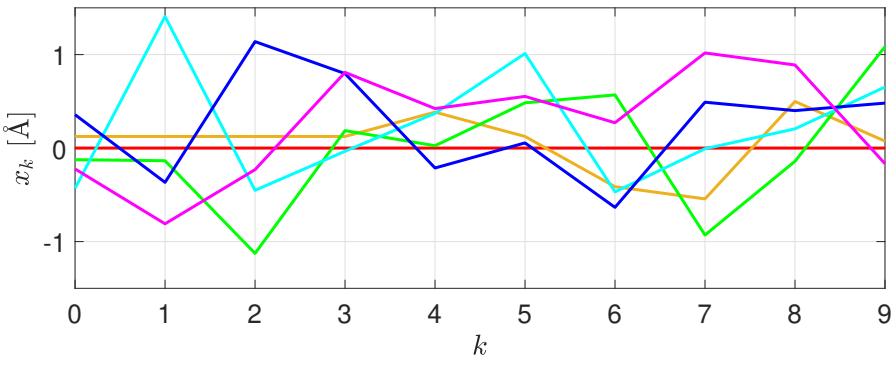

Figure 1: A set of $N = 10$ paths generated by the MHA, starting from a cold start and running $I = 100$, of a single particle $m_N = 1$. Between each represented path there are $J = 20$ sweeps. The potential is harmonic with $\omega_N = 10$.

To be ran, the programme needs the following inputs: a choice of potential $V_N(x_N) = V(x_N; \{\mu_i\})$ among those implemented with the value of the parameters $\{\mu_i\}$ defining it (for example ω_N), the mass m_N of a single particle or a list of masses $\{(m_N)_i\}$ where each mass corresponds to an independent particle, the number of sites N and the simulation time β_N

so that $\delta\tau_N = \beta_N/N$, the Monte Carlo time I, the thermalization time R, the stride jump J, the moves distribution $P_M = (P_1, \dots, P_6)$ where $P_i \in [0;1]$ and indices refer to the list in sec. (2.4.3), the local move initial h_0 and ideal acceptance ratio r_I, the bisection move relative size $s_B \in [0;1]$. J is the number of sweeps that have to be discarded between each $F(\tilde{\mathbf{x}}^l)$ measurement, hence $I - R = MJ$: M is determined by I, R, J. The MHA applies independently to all particles $(m_N)_i \in \{(m_N)_i\}$, hence they don't interact and ΔS_E is individually computed for each of them. Sec. (3.4) uses a modified version of the programme so that ΔS_E is the sum of all the individual Euclidean actions of the simulated particles. Fig. (1) shows the typical appearance of paths generated by the MHA.

2.5 The hydrogen bond

2.5.1 Brief description

An hydrogen bond (H-bond), denoted as the dotted line in $\mathrm{X} - \mathrm{H} \cdots \mathrm{Y}$, is an electrostatic interaction that can occur between molecules (intermolecular) or inside a molecule (intramolecular). The bond arises as a consequence of a polar covalent bond (the solid line in $\mathrm{X} - \mathrm{H} \cdots \mathrm{Y}$) between an hydrogen atom H and an highly electronegative atom X called the donor: because of that the shared valence electrons between X and H are more attracted to X and this induces a positive charge on the H atom (it becomes a proton), which is attracted to another highly electronegative atom Y called the acceptor. [9]

2.5.2 Theoretical model

We follow the same treatment as in [10]. We still denote by H the proton in the H-bond.

Consider a two states quantum system described by the reduced Hilbert space spanned by $|\mathrm{X\text{-}H, Y}\rangle$ and $|\mathrm{X, H\text{-}Y}\rangle$, differing only by the bond between H and either X or Y. Denote $d(\mathrm{A, B})\,[\mathrm{m}]$ the distance between two generic particles A and B. Define $R := d(\mathrm{X, Y})$, $r := d(\mathrm{X, H})$ and $r^* := d(\mathrm{H, Y})$. The triangle of vertices given by the positions of the particles X, H, Y, generates the angles $\phi\,[\mathrm{rad}]$, at the X vertex, and $\theta\,[\mathrm{rad}]$, at the Y vertex. By the law of cosines

$$r^* = \sqrt{R^2 + r^2 - 2rR\cos(\phi)}, \tag{28}$$

therefore if $\phi = 0$ then $\theta = 0$ and $r + r^* = R$ as in fig. (2). Given a potential V_z specific to particle $z \in \{\mathrm{X, Y}\}$, the effective Hamiltonian is

$$\hat{H} = \begin{pmatrix} V_X & \Delta_{XY} \\ \Delta_{XY} & V_Y \end{pmatrix}, \tag{29}$$

where $V_X = V_X(r)$, $V_Y = V_Y(r^*)$ and

$$\Delta_{XY} = \Delta_{XY}(R, \phi, \theta) = \Delta_1 \cos(\phi)\cos(\theta) e^{-b(R - R_1)}, \tag{30}$$

where $\Delta_1\,[\mathrm{J}], b\,[1/\mathrm{m}], R_1\,[\mathrm{m}]$ are constants. $\Delta_{XY} \neq 0\,\mathrm{J}$ is an interacting term that couples the two original states so that they aren't eigenstates. We use the Morse potential

$$V_z(r) = D_z\left(e^{-2a_z(r - r_{0,z})} - 2e^{-a_z(r - r_{0,z})}\right), \tag{31}$$

where $D_z\,[\mathrm{J}], a_z\,[1/\mathrm{m}], r_{0,z} > 0\,\mathrm{m}$ are constants specific to the z-particle. We only consider systems where the proton affinity of X and Y are the same: hence the z-subscript is dropped and it is $R_1 = 2r_0 + 1/a$. We assume $\phi = 0$, hence $\theta = 0$ and therefore Δ_{XY} is purely exponential and it only depends on R. Fig. (2) summarise the whole model and shows the

graph of $V_z(r)$: the bonding energy is D. To find the eigenvalues $\varepsilon_\pm$ [J] of $\hat{H}$ compute the characteristic polynomial

$$\chi(\varepsilon_\pm) = \det(\hat{H} - \varepsilon_\pm \hat{1}) = (V_X - \varepsilon_\pm)(V_Y - \varepsilon_\pm) - \Delta^2 = \varepsilon_\pm^2 - \varepsilon_\pm(V_Y + V_Y) + (V_X V_Y - \Delta^2) \quad (32)$$

and find its roots. Using the quadratic formula for $\chi(\varepsilon_\pm) = 0$:

$$\begin{aligned} \varepsilon_\pm &= \frac{1}{2}\left(V_X + V_Y \pm \sqrt{(V_X + V_Y)^2 - 4(V_X V_Y - \Delta^2)}\right) \\ &= \frac{1}{2}\left(V_X + V_Y \pm \sqrt{(V_X - V_Y)^2 + (2\Delta)^2}\right). \end{aligned} \quad (33)$$

$\varepsilon_+ \geq \varepsilon_-$, hence ε_+ describes an electronic exited state potential energy while ε_- describes the electronic ground state potential energy [10]. A Born-Oppenheimer approximation is applied with the potential of the ground state $\varepsilon_-(r)$: we only consider the QM motion of H confined in $\varepsilon_-(r)$ and we assume X and Y are classically localised and still in space and not influenced by H [10]. The approximation is justified because the masses of X and Y are bigger than the mass of H, hence the motion of H happens on a different time scale as the one of X and Y. Fig. (3) shows the plot of the normalized and shifted ε_- potential for three different R. The strength of the H-bond can be characterised by the value of R: if $R > 2.6\,\text{Å}$ the H-bond is said to be weak (there is a potential barrier around $r = 0\,\text{Å}$), if $2.3\,\text{Å} < R < 2.6\,\text{Å}$ it is moderate (the potential barrier $\varepsilon_-(R/2) \approx -D$) and if $R < 2.3\,\text{Å}$ it is strong (there is no potential barrier) [10].

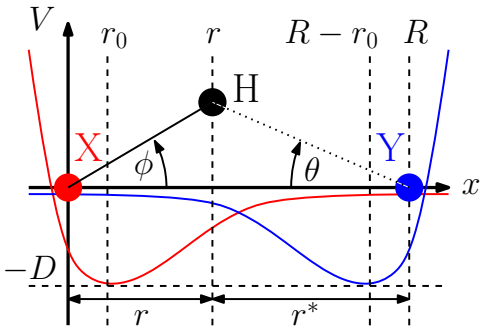

Figure 2: Visualisation and defined quantities of the H-bond model, and the shape of Morse's potential. It is assumed $\phi = \theta = 0$.

2.5.3 Numerical implementation

Reference [10] gives the values: $D = 120\,\text{kcal/mol}$ hence $D = 83.402 \cdot 10^{-20}\,\text{J}$, $a = 2.2\,\text{Å}^{-1}$, $r_0 = 0.96\,\text{Å}$, $\Delta_1 = 0.4D = 2.082\,\text{eV}$ hence $\Delta_1 = 33.361 \cdot 10^{-20}\,\text{J}$, $b = 2.2\,\text{Å}^{-1}$ and $\theta = \phi = 0$. The values of r_0 and a give $R_1 = 2.375\,\text{Å}$.

Symmetry moves (sec. (2.4.3)) assume the potential is symmetric around $x = 0\,\text{Å}$, therefore a shift of $-R/2$ is applied to all positions to get better numerical results because then the axis of symmetry of $\varepsilon_-(r)$ is shifted from $x = R/2$ to $x = 0\,\text{Å}$.

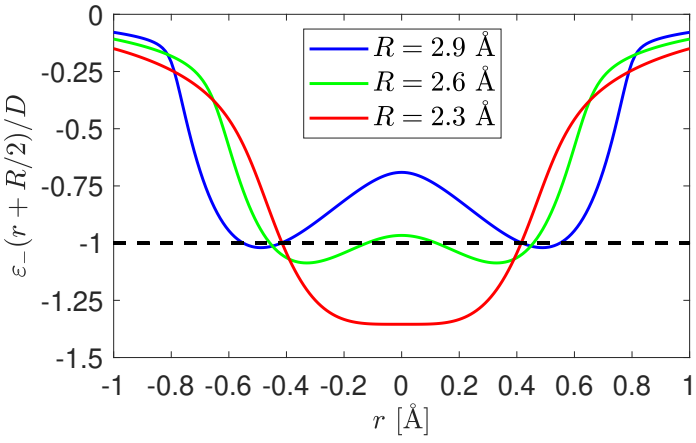

Figure 3: Plot of the electronic energy surface ε_- with varying R showing the three bond strength regimes.

Analytically $\varepsilon_-(r) \to 0\,\text{J}$ as $r \to \pm\infty$, but numerically $\varepsilon_-(r)$ is a difference of quantities that tend to infinity when $r \to \pm\infty$, hence numerically this difference involves big numbers. It is found empirically that for $8\,\text{Å} < |r| < 10\,\text{Å}$ the programme returns $\varepsilon_-(r) =$ `+inf` while for $10\,\text{Å} < |r|$ it returns $\varepsilon_-(r) =$ `NaN`, hence to avoid this an `if` statement is implemented to force $\varepsilon_-(r) = 0$ for all $|r| > 8\,\text{Å}$.

3 Results

3.1 Verification of the Metropolis-Hastings algorithm

Denote ΔS_E^j the j-th variation of the j-th sweep, with $j > 0$: if the move is accepted set $\Delta S_E^j = \Delta S_E$, otherwise $\Delta S_E^j = 0\,\text{Js}$. Then

$$S_E[\tilde{\mathbf{x}}^j] = S_E[\tilde{\mathbf{x}}^0] + \sum_{k=1}^{j} \Delta S_E^k. \tag{34}$$

We verify at each sweep j that the value of eq. (34) is the same as the one computed with the discretized formulation of $S_E[\tilde{\mathbf{x}}^j]$ in eq. (16). If all moves are correctly implemented eq. (34) is verified for each sweep, hence if at some sweep eq. (34) is not verified then at least one move is not correctly implemented. Setting $P_i = \delta_{ik}$, the single k-th move can be verified. This was done for all moves and for any sweep eq. (34) was found to be satisfied up to a maximal $10^{-8} \cdot 10^{-20}\,\text{Js}$ error.

Local moves should be attempted on average once every sweep at every position [2]. This was verified in all the simulations of this work: average attempts per site per sweep were always 1 with a maximal 10^{-3} error. It has also be verified in all simulations that r_A converges and then oscillates around r_I with a maximal 10^{-2} error, hence the adaptive interval size h_j is correctly implemented.

3.2 Particle size

3.2.1 Theoretical predictions

Consider a particle in a null potential fully localised at position $x = 0\,\text{Å}$ at time $t = 0\,\text{s}$, $\psi(x,0) = \delta(x)$. $\psi(x,t)$ for $t > 0\,\text{s}$ is the solution to Schrödinger equation (1) in position representation

$$i\hbar \frac{\partial \psi}{\partial t}(x,t) = -\frac{\hbar^2}{2m}\frac{\partial^2 \psi}{\partial^2 x}(x,t) \quad \Leftrightarrow \quad \frac{\partial \psi}{\partial t}(x,t) = \lambda \frac{\partial^2 \psi}{\partial x^2}(x,t), \tag{35}$$

where $\lambda = \hbar i/(2m)$. The solution of this PDE is [11]

$$\psi(x,t) = \frac{1}{\sqrt{4\pi\lambda t}} \exp\left(-\frac{x^2}{4\lambda t}\right). \tag{36}$$

This is a normalized Gaussian curve of variance $\sigma(t)^2 = 2\lambda t$. Because $ti = \tau$, the standard deviation becomes $\sigma(\tau) = \sqrt{\tau\hbar/m}$. We interpret $\sigma(\beta)$ as the size $s\,[\text{m}]$ of the particle [12]. As imaginary time τ increments the particle delocalizes linearly because $\psi(x,t)$ propagates through space [13]. Numerically s is calculated as the average of the estimated σ over all I sweeps:

$$s = \frac{1}{I}\sum_{j=1}^{I} \sqrt{\frac{1}{N}\sum_{k=0}^{N-1}(x_k^j)^2 - \left(\frac{1}{N}\sum_{k=0}^{N-1} x_k^j\right)^2}. \tag{37}$$

Reference [14] shows $s \leq d := \sqrt{2\hbar/(mT_K k_B)}\,[\text{m}]$: if this inequality is not verified then the MHA isn't correctly implemented.

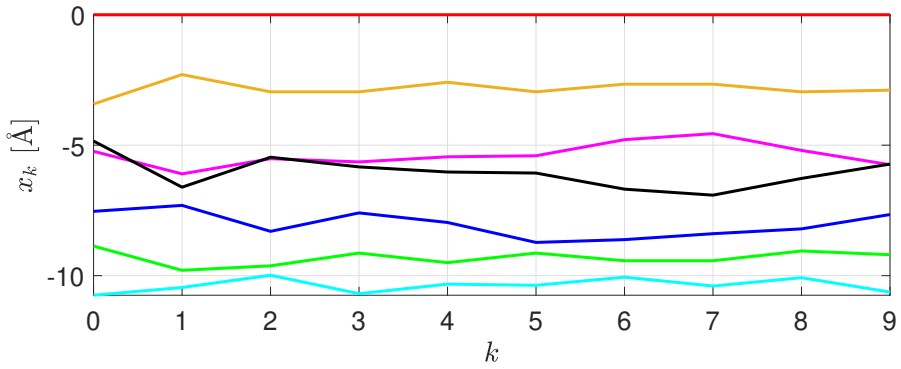

Figure 4: A set of paths generated by the MHA, starting from a cold start. Paths are approximately straight lines with small deviations.

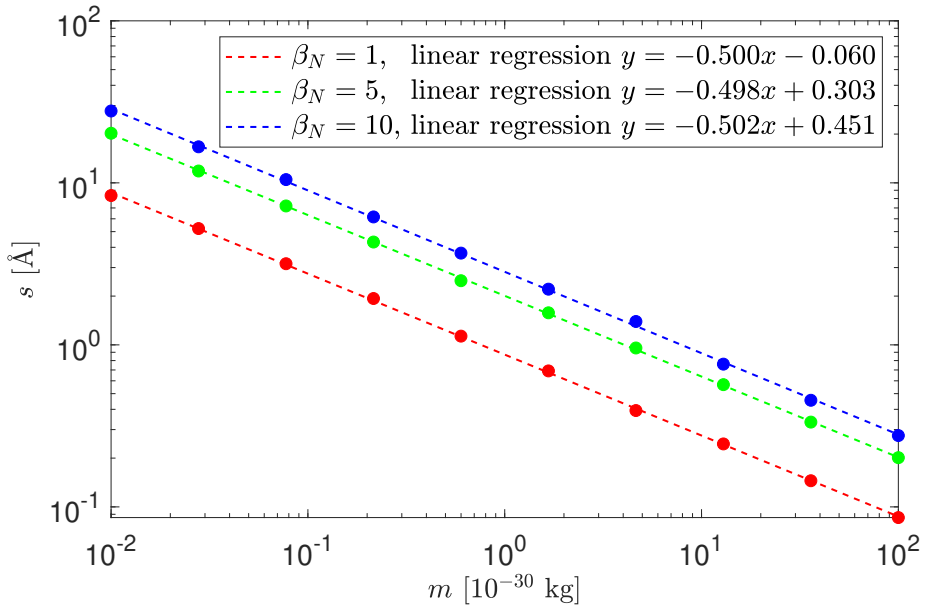

Figure 5: Size of particle against its mass m. Linear regressions (dashed lines) are performed on their respective measured points: their slope is approximately identical.

3.2.2 Numerical results and discussion

All simulations use $N = 100$, $I = 10^5$, $R = 1000$, $J = 500$, $h_0 = 1$, $r_I = 0.5$, $s_B = 0.3$, $P_i = \delta_{i,1}$, $V_N(x_N) \equiv 0$. m_N, β vary but $\delta\tau_N = 0.1$. $\tilde{\mathbf{x}}^0$ is a cold start.

Fig. (4) shows some paths $\tilde{\mathbf{x}}^j$ generated by the MHA. After the cold start $\tilde{\mathbf{x}}^0$ where $\sigma = 0$, $\sigma \neq 0$ and it oscillates around s. Trajectories are approximately straight lines with small deviations from $\langle x \rangle^j$ of the order of 1 Å because sawtooth-like trajectories implies a kinetic price in ΔS_E and the MHA accepts more willingly moves with low kinetic price.

Fig. (5) shows in a log-log plot the particle size s as a function of m for multiples β. For all considered β we observe a linear relationship between $\log(s) = x$ and $\log(m) = y$, hence s scales as $s \sim m^\alpha$ where $\alpha \in \mathbb{R}$. α is determined by linear regression with `MATLAB`'s function `polyfit()` using all (x, y) points of same β. For all three β, estimated α are close to $-1/2$, verifying in the $m_N \in [10^{-2}; 10^2]$ interval the expected $s \sim 1/\sqrt{m}$ relationship.

Intrigued by the spacing between the three interpolation lines in fig. (5), we investigate the behaviour of s with varying β. Fig. (6) shows s as a function of β in a log-log plot. Using the same argument as before, there is a relationship $s \sim \beta^\alpha$. The linear regression gives $\alpha = 0.511 \approx 1/2$, verifying in the $\beta_N \in [1; 20]$ interval the expected $s \sim \sqrt{\beta}$ relationship.

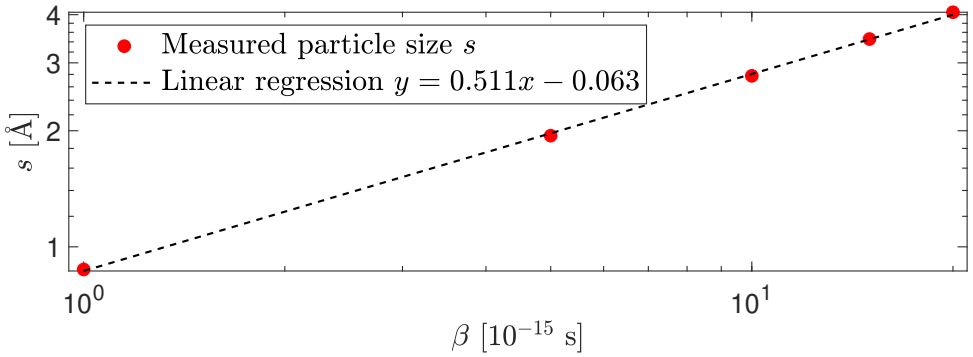

Figure 6: Size of a particle with mass $m_N = 1$ against time β. A linear regression (dashed line) is performed on the measured points.

Particle	m_N	T_K [K]	β_N	s [Å]	d [Å]
H	1673	1	7609	0.195	9
O	26567	1	7609	0.049	2.5
O	26567	100	76	0.044	0.3

Table 1: Numerical results to confirm the $s \leq d$ inequality.

Tbl. (1) tests and confirms the inequality $s \leq d$ with two particles: the hydrogen atom H and the oxygen atom O respectively of masses $m_N = 1673$ and $m_N = 26567$ [3].

3.3 Application of the Metropolis-Hastings algorithm on toy models

Consider three systems: a particle in either an infinite well, an harmonic or a double well potential. We observe the numerical approximation of the position PDF $|\psi(x)|^2 = |\langle x|\psi\rangle|^2$, denoted $|\phi(x)|^2$, along the variation of specific parameters, such as β, m or parameters that define the potential. When $|\phi(x)|^2$ is compared with $|\psi(x)|^2$ and E_M is compared with E_0, analytical expressions are used when possible, otherwise numerical approximations are computed in another way than the MHA. $|\phi(x)|^2$ is obtained by plotting a normalized histogram of positions generated considering all x_k^j of all thermalized paths $\tilde{\mathbf{x}}^j$, where $0 \leq k \leq N-1$ and $R \leq j \leq I$ [2].

3.3.1 Effect of temperature on the position PDF

We saw in eq. (18) $\langle E\rangle_\mathrm{SM} \to E_0$ in the limit $T_K \ll 1$: close to the absolute zero the system has a full QM ground state behaviour, hence $p_n \to \delta_{n0}$ and $\psi(x) = \psi_0(x)$, hence the PDF of the particle position is $|\psi_0(x)|^2$.

Above the absolute zero $p_{n>0} \neq 0$. The higher the temperature, the wider the microstates distribution spreads across the energy spectrum $\{E_n\}_{n\in\mathbb{N}}$: the available microstates population grows in number (more and more $n > 0$ states become reasonably probable $p_{n>0} \neq 0$) along with its upper energy bound. In the limit $T_K \gg 1$ the system can be observed under the classical SM eye where each microstate is differentiated by the classical position of the particle, considered without kinetic energy [15]. At position x the particle has energy $V(x)$, hence the classical SM partition function is

$$Z_C(\beta) := \int_{\mathbb{R}} e^{-\beta V(x)/\hbar} \mathrm{d}x. \tag{38}$$

Integration is necessary as $x \in \mathbb{R}$ is continuous hence there are infinite microstates. The analogy with eq. (17) is clear: E_n is substituted with $V(x)$ and the sum over n is substituted with an

integral over x. From $Z_C(\beta)$ we infer the PDF of the particle position

$$p_C(x) = \frac{e^{-\beta V(x)/\hbar}}{Z_C(\beta)}. \tag{39}$$

We can verify whenever the MHA is correctly implemented if we find in the limit $T_K \ll 1$ that $|\phi(x)|^2 \approx |\psi_0(x)|^2$ and $E_{\mathrm{M}} \approx E_0$, while in the limit $T_K \gg 1$ that $|\phi(x)|^2 \approx p_C(x)$. The temperature limits are numerically implemented by fixing $\delta\tau_N$ and varying β_N or N.

3.3.2 Analytical computations, numerical modelling and results

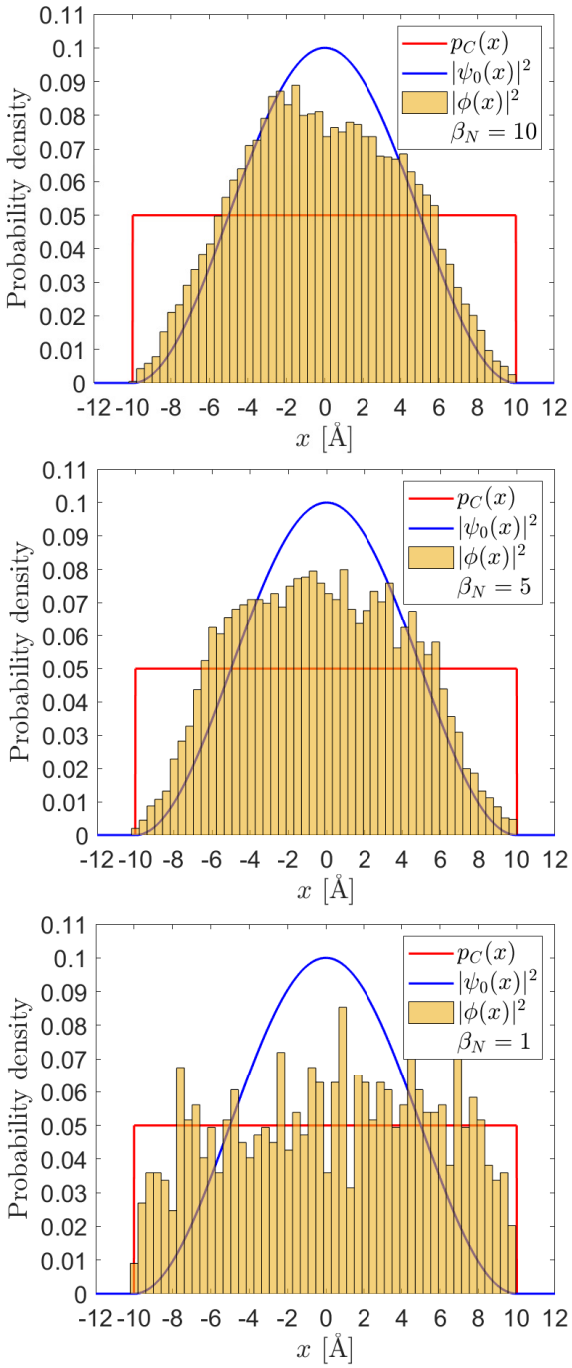

Figure 7: Variation of $|\phi(x)|^2$ along temperature, compared with $p_C(x)$ and $|\psi_0(x)|^2$, in the infinite well system.

We consider the infinite well potential

$$V(x) = \begin{cases} V_r & \text{if} \quad |x - x_r| < L/2, \\ 0 & \text{if} \quad |x - x_r| \geq L/2, \end{cases} \tag{40}$$

with $x_r = 0\,\text{Å}$, $V_r \to -\infty$. $L > 0\,\text{Å}$ is the length of the well. Numerically this is implemented as

$$V_N(x_N) = \begin{cases} V_{r,N} & \text{if} \quad |x_N - x_{r,N}| < L_N/2, \\ 0 & \text{if} \quad |x_N - x_{r,N}| \geq L_N/2, \end{cases} \tag{41}$$

where $x_{r,N} = 0$, $L_N \cdot 10^{-10} = L$ and $V_{r,N}$ is taken negatively very large. The ground state wave function is [13]

$$\psi_0(x) = \sqrt{\frac{2}{L}} \cos\left(\frac{\pi x}{L}\right), \tag{42}$$

hence the QM position PDF is

$$|\psi_0(x)|^2 \mathrm{d}x = \frac{2}{L_N} \cos\left(\frac{\pi x_N}{L_N}\right)^2 \mathrm{d}x_N. \tag{43}$$

The introduction of $\mathrm{d}x_N$ instead of $\mathrm{d}x$ is necessary because in our scaled plots we consider x_N and not x. The SM partition function is

$$Z_C(\beta) = \int_{-L/2}^{L/2} 1 \mathrm{d}x = L, \tag{44}$$

therefore the SM position PDF is

$$p_C(x)\mathrm{d}x = \frac{e^{-\beta V(x)/\hbar}\mathrm{d}x}{L} = \begin{cases} \mathrm{d}x/L & \text{if} \quad |x| < L/2, \\ 0\mathrm{d}x & \text{if} \quad |x| \geq L/2. \end{cases} \tag{45}$$

Notice how $p_C(x)$ is independent of temperature.

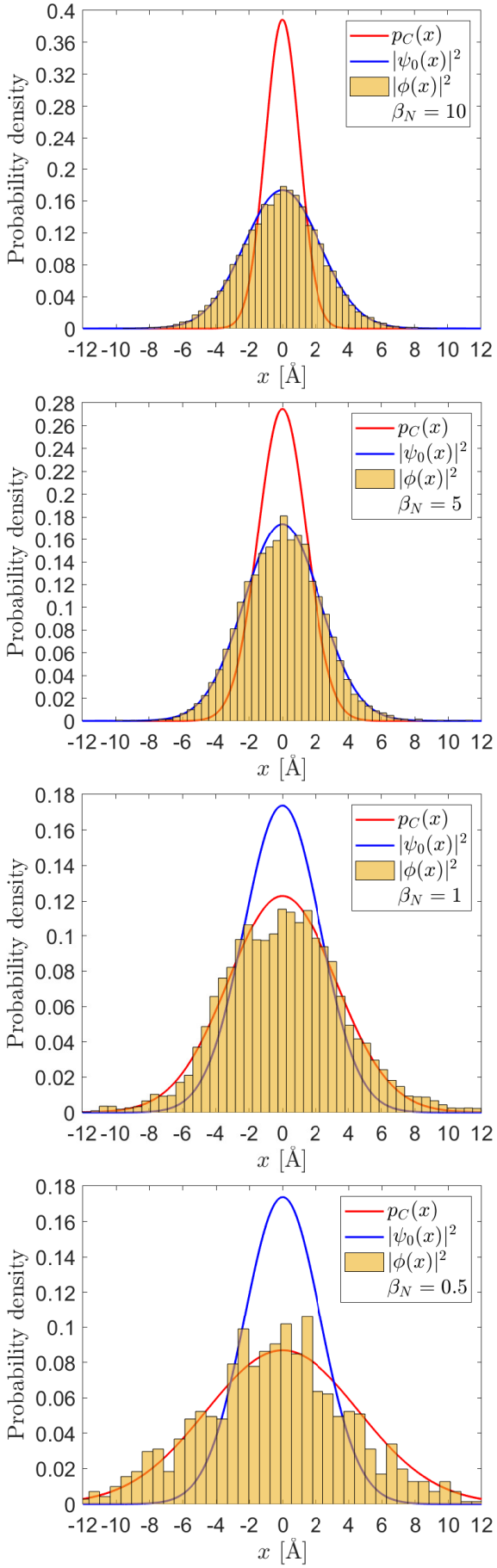

Figure 8: Variation of $|\phi(x)|^2$ along temperature, compared with $p_C(x)$ and $|\psi_0(x)|^2$, in the harmonic system.

We consider the harmonic potential

$$V(x) = \frac{1}{2} m\omega^2 x^2, \tag{46}$$

with m the mass of the particle and ω the angular frequency of the oscillator that characterises the width of the harmonic well. Numerically this is implemented as

$$V_N(x_N) = \frac{1}{2} m_N \omega_N^2 x_N^2. \tag{47}$$

The ground state wave function is [13]

$$\psi_0(x) = \left(\frac{m\omega}{\pi\hbar}\right)^{\frac{1}{4}} \exp\left(-\frac{m\omega x^2}{2\hbar}\right), \tag{48}$$

hence the QM position PDF is

$$|\psi_0(x)|^2 \mathrm{d}x = \sqrt{\frac{m_N\omega_N}{10\pi\hbar_N}} \exp\left(-\frac{m_N\omega_N x_N^2}{10\hbar_N}\right) \mathrm{d}x_N. \tag{49}$$

The ZPE is [13]

$$E_0 = E_{n=0} = \hbar\omega \left(\frac{1}{2} + n\right)\Big|_{n=0} = \frac{\hbar_N\omega_N}{2} \cdot 10^{-19}\,\mathrm{J}. \tag{50}$$

The form of E_n is simple enough to let us calculate analytically the partition function

$$Z(\beta) = \sum_{n=0}^{\infty} e^{-\beta E_n/\hbar} = e^{-\beta\omega/2} \sum_{n=0}^{\infty} e^{-\beta\omega n} = \frac{e^{-\beta\omega/2}}{1 - e^{-\beta\omega}}, \tag{51}$$

hence the microstates probabilities are

$$p_n(\beta) = \frac{e^{-\beta E_n/\hbar}}{Z(\beta)} = e^{-\beta\omega n}\left(1 - e^{-\beta\omega}\right). \tag{52}$$

We see for $\beta \to \infty$ we have $p_n \to \delta_{n0}$, as predicted. The SM partition function is evaluated using the Gauss integral [16]

$$Z_C(\beta) = \int_{\mathbb{R}} \exp\left(-\frac{m\omega^2 x^2 \beta}{2\hbar}\right) \mathrm{d}x = \sqrt{\frac{2\pi\hbar}{m\omega^2\beta}}, \tag{53}$$

hence the SM position PDF is

$$\begin{aligned} p_C(x)\mathrm{d}x &= \sqrt{\frac{m\omega^2\beta}{2\pi\hbar}} \exp\left(-\frac{m\omega^2 x^2\beta}{2\hbar}\right) \mathrm{d}x \\ &= \sqrt{\frac{m_N\omega_N^2\beta_N}{20\pi\hbar_N}} \exp\left(-\frac{m_N\omega_N^2 x_N^2\beta_N}{20\hbar_N}\right) \mathrm{d}x_N. \end{aligned} \tag{54}$$

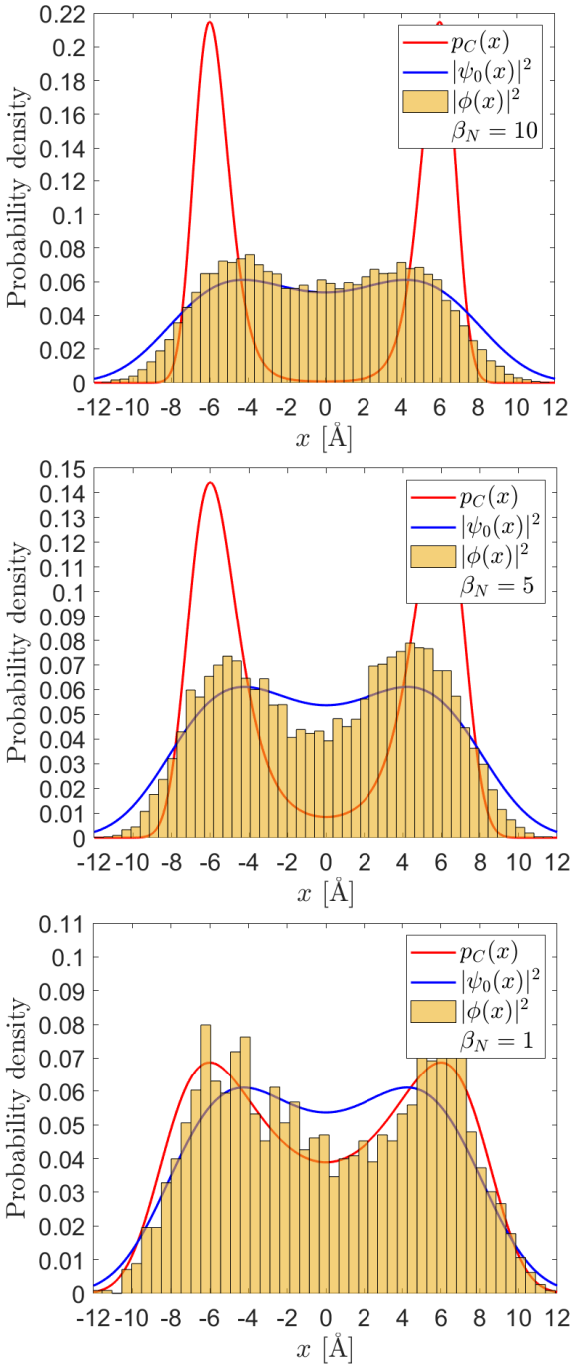

Figure 9: Variation of $|\phi(x)|^2$ along temperature, compared with $p_C(x)$ and $|\psi_0(x)|^2$, in the double well system.

We consider the double well potential

$$V(x) = V_r \left[1 - \left(\frac{x}{x_r}\right)^2\right]^2, \tag{55}$$

with $V_r, x_r > 0$ constants: V_r is the height of the potential barrier between the two wells and x_r is the distance of their minima from $x = 0$ Å. Numerically this is implemented as

$$V_N(x_N) = V_{r,N} \left[1 - \left(\frac{x}{x_{r,N}}\right)^2\right]^2 \tag{56}$$

where $V_r = V_{r,N} \cdot 10^{-20}$ and $x_r = x_{r,N} \cdot 10^{-10}$. A numerical approximation of $|\psi_0(x)|^2$ and E_0 are calculated using the `MATLAB` package `Chebfun` [17]. The SM partition function is numerically approximated using the `integral()` function in `MATLAB`. All those numerical approximations are denoted as they were analytical.

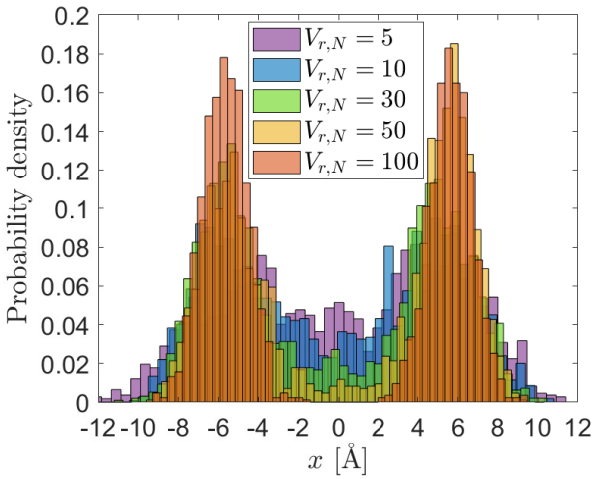

Figure 10: Variation of $|\phi(x)|^2$ along V_r in the double well potential. Tunnelling reduces and the two peaks of $|\phi(x)|^2$ increase.

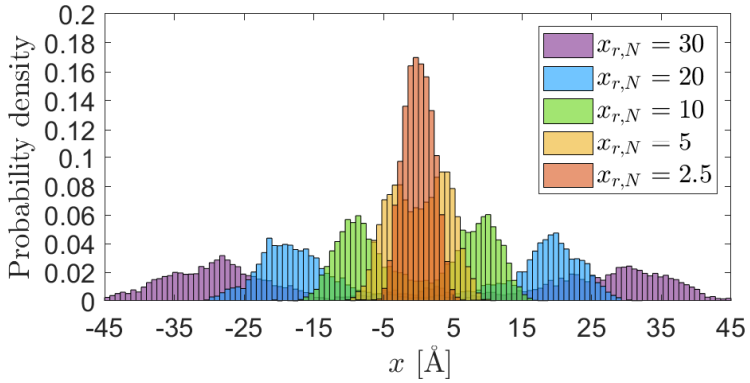

Figure 11: Variation of $|\phi(x)|^2$ along x_r in the double well potential. Tunnelling reduces and the two peaks of $|\phi(x)|^2$ separate.

ω_N	$E_{\mathrm{M}}\,[10^{-20}\mathrm{J}]$	$E_0\,[10^{-20}\mathrm{J}]$
1	5.371 ± 0.180	5.273
2	10.346 ± 0.251	10.546
3	16.055 ± 0.339	15.819
4	20.861 ± 0.326	21.091
5	26.257 ± 0.412	26.364

Table 2: Measured E_{M} of a $m_N = 1$ particle in an harmonic potential for different ω_N.

m_N	$E_{\mathrm{M}}\,[10^{-20}\mathrm{J}]$	m_N	$E_{\mathrm{M}}\,[10^{-20}\mathrm{J}]$
10^{-6}	5.259 ± 0.169	10^{1}	5.351 ± 0.171
10^{-5}	5.273 ± 0.163	10^{2}	5.470 ± 0.252
10^{-4}	5.273 ± 0.147	10^{3}	5.280 ± 0.149
10^{-3}	5.234 ± 0.169	10^{4}	5.221 ± 0.170
10^{-2}	5.434 ± 0.176	10^{5}	5.288 ± 0.185
10^{-1}	5.301 ± 0.179	10^{6}	5.274 ± 0.174

Table 3: Measured E_{M} of a particle in a $\omega_N = 1$ harmonic potential for different masses m_N.

In all simulations $m_N = 1$, $I = 10^5$, $R = 1000$, $J = 500$, $P_M = (1, 1, 1, 1, 1, 1)$, $h_0 = 5$, $r_I = 0.5$, $s_B = 0.3$. $\delta\tau = 0.1$ is fixed and β_N, N varied. $\tilde{\mathbf{x}}^0$ is an hot start with random positions between $[-10; 10]$ Å. When β is varied, for the infinite well it was taken $L_N = 20$, $V_{r,N} = -10^4$, for the harmonic potential it was taken $\omega_N = 10$ and for the double well it was taken $V_{r,N} = 6$ and $x_{r,N} = 6$. When m is varied in the harmonic potential, it was taken $\omega_N = 1$, $\beta_N = 10$ and it was used $F = F_V$. When V_r is varied in the double well system, it was taken $x_{r,N} = 6$. When x_r is varied, it was taken $V_{r,N} = 5$. On both last cases $\beta_N = 5$.

3.3.3 Discussion of results

General observations for all systems Fig. (7), (8) and (9) show $|\phi(x)|^2$ with varying temperature β_N, compared with $p_C(x)$ and $|\psi_0(x)|^2$. When β_N is the highest (low temperatures) $|\phi(x)|^2$ follows $|\psi_0(x)|^2$, when β_N is the lowest (high temperatures) $|\phi(x)|^2$ follows $p_C(x)$. This proves $|\phi(x)|^2$ is correctly normalized, as $p_C(x), |\psi_0(x)|^2$ are. When temperature is not in a limiting case (β_N is between these two bounds), $|\phi(x)|^2$ is an intermediate PDF between the previous two. In the $\beta \ll 1$ limit, $|\phi(x)|^2$ follows $p_C(x)$ but also shows fluctuations around it. As β gets lower, because $\beta = N\delta\tau$ and $\delta\tau$ is fixed, then also the number of sites $N \sim \beta$ gets lower: less sites compose the paths $\tilde{\mathbf{x}}^j$, hence statistics (like histograms) based on x_k^j are less accurate. More sweeps I or more sites N (a lower $\delta\tau$) would improve the statistics hence $|\phi(x)|^2$. All $|\psi_0(x)|^2$, $p_C(x)$ and $|\phi(x)|^2$ are symmetrical around the $x = 0$ axis: this was expected as all potentials are also symmetric.

Infinite well We discuss fig. (7). For all β_N cases the $|x| \geq L/2$ region is never explored: a proposed move outside the well region makes ΔS_E extremely high (due to $V_{r,N}$), hence such a move is practically impossible. In the $\beta_N = 10$ case the peak of $|\psi_0(x)|^2$ is not completely reached by $|\phi(x)|^2$: we couldn't find an explanation to this anomaly. In the $\beta_N = 1$ case, fluctuations of $|\phi(x)|^2$ around $p_C(x)$ are distributed in a comb-like pattern. This could be explained recalling fig. (4): inside the well the particle behaves as in a null potential, hence paths $\tilde{\mathbf{x}}^j$ are approximately straight lines, hence all x_k^j tend to accumulate around certain points, the comb-like peaks. In the $\beta_N = 1$ case the two tails of $|\phi(x)|^2$ are not discontinuous at $|x| = L/2$ but they rather go to zero in a $|\psi_0(x)|^2$ way. This might happen because moves are always limited by an interval $[-h_j; h_j]$, hence the boundaries of the potential $|x| \approx L/2$ are never fully explored. An estimate E_{M} couldn't be obtained because F_K, F_V depend on $V_{r,N}$, F_K gives inconsistent results (it is a bad estimator) and F_V considers $V'(x)$, which is null for $|x| < L/2$.

Harmonic potential We discuss fig. (8). For $\beta_N \in \{5, 10\}$, $|\phi(x)|^2$ follows almost perfectly $|\psi(x)|^2$: there is a peak at $x = 0$ Å and the PDF exponentially decreases at $x \to \pm\infty$. As temperature gets higher, the peak of $p_C(x)$ gets lower and wider until $p_C(x)$ is followed by $|\phi(x)|^2$: the microstates population is growing so that it includes states where the particle can

be found even further away from $x = 0\,\text{Å}$. In tbl. (2) E_0 is contained within the error bars of E_M for all ω_N, hence E_M correctly estimates E_0 regardless of the shape of the harmonic potential. In tbl. (3) $E_0 = 5.273 \cdot 10^{-20}\,\text{J}$ is contained within the error bars of E_M for all m_N: E_0 is independent of m, and we see the same applies to E_M. m_N in tbl. (3) aren't close to unity, but nevertheless the `C++` programme produces accurate and precise results.

Double well We discuss fig. (9). For $\beta_N \in \{5, 10\}$, $|\phi(x)|^2$ broadly follows $|\psi(x)|$ but $|\phi(x)|^2$ is higher around the peaks and lower around the tails. This could be explained by the asymptotic behaviour of $V(x)$ that goes as x^4: the potential may be too sharp for the moves, meaning a too high ΔS_E at the tails. As the histogram is normalized, if there are less visited sites at the tails, there will be more at the peaks. $|\psi_0(x)|^2$ shows two dunes, reflecting the presence of two wells. When the temperature is low, $p_C(x)$ displays two distinct sharp peaks. As the temperature gets higher, the two peaks get lower and they start to merge: around $x = 0\,\text{Å}$ a tunnelling of the potential barrier V_r analogous to $|\psi_0(x)|^2$ emerges. Fig. (10) displays this tunnelling effect. As V_r is increased, $|\phi(x)|^2$ reduces around $x = 0\,\text{Å}$ and increases around $|x| = x_r$: the cost ΔS_E of a move exploring the $x = 0\,\text{Å}$ region increases, hence the MHA explores more often the ΔS_E-favourable $|x| = x_r$ regions. The peaks separation in fig. (11) can be explained with a similar argument: as x_r gets larger, the kinetic part of ΔS_E gets bigger and so ΔS_E.

3.4 Hydrogen bond

We simulate an H-bond using two different models: the fixed-R model [10], where X and Y are still at a fixed R, and the free-R model, where X and Y are relatively free to move so R is free to change. H can be substituted with an ionised deuterium atom denoted D [10].

3.4.1 Fixed-R model of the H-bond

Only H in the ε_- potential is simulated [10]. Numerical parameters are $m_N = 1673$ for H and $m_N = 3345$ for D [3], $N = 100$, $\beta_N = 10$, $I = 10^5$, $R = 1000$, $J = 500$, $P_M = (1, 1, 1, 1, 1, 1)$, $h_0 = 0.5$, $r_I = 0.8$, $s_B = 0.3$.

3.4.2 Free-R model of the H-bond

X,Y and H are simulated simultaneously. The H-bond is either intermolecular or intramolecular, hence X and Y are chemically bonded to other parts of their respective molecule [9]: X and Y can't freely move because of this interaction, but they oscillate around their equilibrium point. We modelize this limitation with an arbitrary trapping potential (given analytically and numerically)

$$V_T(x) = V_r \left(\frac{x - x_r}{L/2} \right)^{20} \quad \Leftrightarrow \quad V_{T,N}(x_N) = V_{r,N} \left(\frac{x_N - x_{r,N}}{L_N/2} \right)^{20}, \tag{57}$$

where $V_r = V_T(x_r \pm L/2)$, x_r is the equilibrium point and L the size of the equilibrium region. Numerically, for both X and Y, $L_N = 0.3$, $V_{r,N} = 1$ and $\pm x_{r,N}$ is varied (+ for Y, − for X). X and Y don't interact: the interaction between H and both X,Y is given by $\varepsilon_-(r + R')$, which is calculated for every k-site, with $R = (x_k^j)_\text{Y} - (x_k^j)_\text{X}$ (we set the position of X $(x_k^j)_\text{X} < 0$ and Y $(x_k^j)_\text{Y} > 0$) and it is centred around $r = [(x_k^j)_\text{Y} + (x_k^j)_\text{X}]/2$, hence $R' = R/2 - [(x_k^j)_\text{Y} + (x_k^j)_\text{X}]/2$. Crucially each proposed move for any particle considers the whole Euclidean action $\Delta S_E = (\Delta S_E)_\text{X} + (\Delta S_E)_\text{H} + (\Delta S_E)_\text{Y}$ so that the system is bound as a whole. The global R of a whole

simulation is calculated as the average

$$R = \frac{1}{NM} \sum_{l=1}^{M} \sum_{k=0}^{N-1} \left[(x_k^l)_{\mathrm{Y}} - (x_k^l)_{\mathrm{X}} \right] . \tag{58}$$

The error ΔR on R, computed analogously as ΔE_{M}, was found to be $\Delta R < 10^{-2}$ Å hence it was not plotted in figures. Numerical parameters are $\{(m_N)_i\} = \{m_N, 26567, 26567\}$ where $m_N = 1673$ for H and $m_N = 3345$ for D (last two particles are X = Y = O an oxygen atom) [3], $N = 100$, $\beta_N = 10$, $I = 10^5$, $R = 1000$, $J = 500$, $P_M = (1, 0.4, 0.75, 0.4, 0.4, 0)$ (no swaps), $h_0 = 1$, $r_I = 0.8$, $s_B = 0.3$.

3.4.3 Measuring the bond length

The length of the H-bond $d(\mathrm{X}, \mathrm{H})$ has two definitions [10]:

1. $r_C(R)$ [Å]: the classical distance between X and the first encountered minimum of ε_-;
2. r_m [Å]: the distance between X and the first encountered peak of $|\psi(x)|^2$.

$r_C(R)$ is implemented by numerically finding the minimum of ε_- through `MATLAB`'s `fminbnd()` function, while $r_m \pm \Delta r_m$ is estimated as the center of the first encountered maximal bar of the $|\phi(x)|^2$ histogram (100 bins are used), with half the width of the bar as the error Δr_m on r_m.

3.4.4 Numerical results of both models and discussion

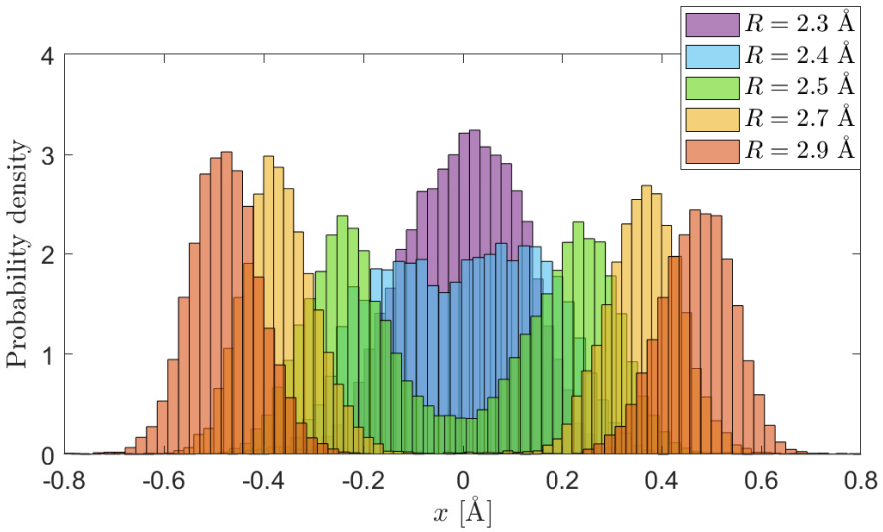

Figure 12: Variation of $|\phi|^2$ along R. Tunnelling reduces and the two peaks of $|\phi|^2$ separate.

Fig. (12) shows $|\phi(x)|^2$ in the fixed-R model with varying R. $|\phi(x)|^2$ for $R \geq 2.7$ Å isn't symmetric around $x = 0$ Å. This might be an effect of the MHA on ε_-. $R \geq 2.7$ Å is in the weak bond regime, hence ε_- starts to describe two distant wells separated by a increasingly larger potential barrier (see fig. (3)): tunnelling decreases as R increases ($|\phi(x)|^2 \to 0$ around $x_N = 0$ [Å] and the two peaks of $|\phi(x)|^2$ increase and separate) and for $R \geq 2.7$ Å it is almost non-existent. The only way the MHA can visit these wells is by the means of moves such as global displacement, mirror or center of mass displacement, which occur with a fewer frequency than all the other moves. The MHA doesn't switch between the wells often, hence $\tilde{\mathbf{x}}^j$ accumulates around one of them.

Fig. (13) shows the velocity distribution of an H particle in the fixed-R model for $R = 2.3\,\text{Å}$. The j-speeds vector $\tilde{\mathbf{v}}^j$ $(R \leq j \leq I)$ has components $v_k^j := (x_{k+1}^j - x_k^j)/\delta\tau$ $(0 \leq k < N-1)$, and an histogram is produced over all v_k^j [m/s]. The distribution has the shape of a Gaussian curve centred at $v = 0\,\text{m/s}$. Consider the symmetry of $\varepsilon_-(r + R/2)$ around $r = 0\,\text{Å}$: the H particle is equally likely to be in one of the two wells, so it switches between them along sweeps. Each migration contributes to one side ($v > 0\,\text{m/s}$ or $v < 0\,\text{m/s}$) of the distribution: overall contributions are symmetric around $v = 0\,\text{m/s}$. The distribution decays exponentially as $|v|$ gets larger. Large $|v|$ have an high kinetic cost in ΔS_E, hence the decay is a consequence of the MHA: the H particle is more likely to have close to zero kinetic energy (low kinetic cost). The H particle never reaches an speed higher than $10^5\,\text{m/s}$, 0.03% the speed of light: this justifies the use of the path integral formalism as relativistic effects are negligible.

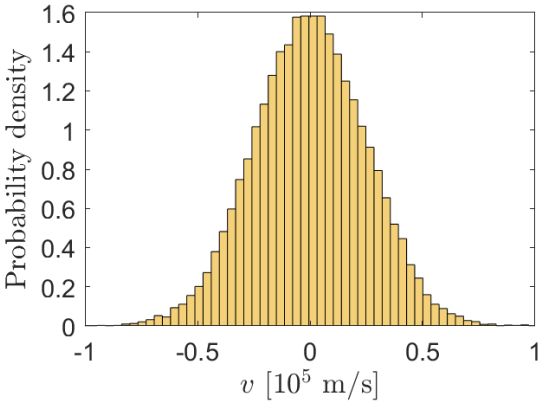

Figure 13: Velocity distribution of all thermalized sweeps.

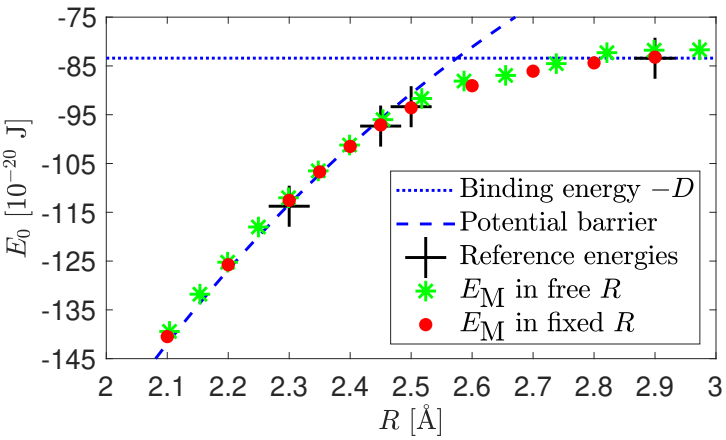

Figure 14: Measured ZPE against R using a proton.

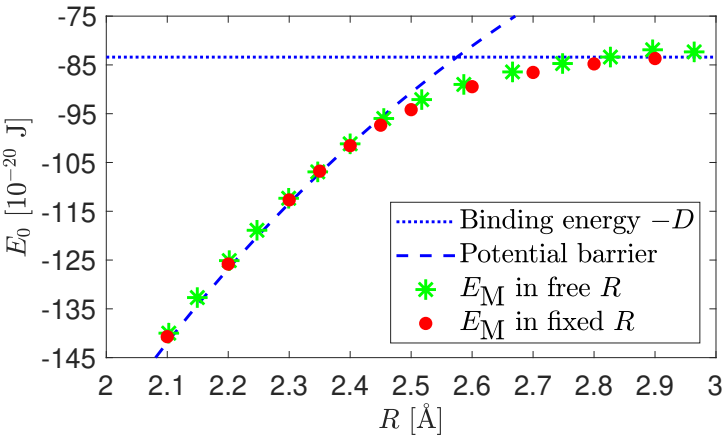

Figure 15: Measured ZPE against R using deuterium.

Figs. (14) and (15) show E_M against R for the two models compared with the potential barrier $\varepsilon_-(R/2)$ with either H or D. Reference energies for H come from [10]: fig. (14) agrees with them. Qualitatively figs. (14) and (15) are identical, which is expected, and quantitatively they are almost identical, which is not. For a given model, the difference of E_M when H $\to$ D is $\approx 1 \cdot 10^{-20}\,\text{J}$, hence negligible. The choice of particle is irrelevant: no secondary geometric isotope effect is observed. For a given particle, the difference of E_M in the two models is negligible: intrinsic properties of the system don't depend on the model. In the strong bond regime $E_\text{M} > \varepsilon_-(R/2)$, at around $R = 2.4\,\text{Å}$ they are equivalent and after that point $E_\text{M} < \varepsilon_-(R/2)$. Bond nomenclature is verified: strong bonds overcome the potential barrier, weak ones don't and moderate ones are of the same magnitude. If the system is bound then $E_\text{M} < -D$: it is always the case except for the last 3 points in both figures in the free-R model. This means after $R > 2.8\,\text{Å}$ the H-bond breaks.

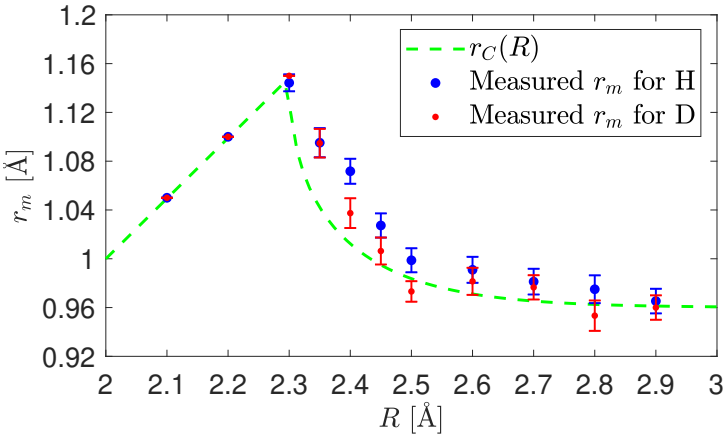

Figure 16: Measured bond length against R using the fixed-R model.

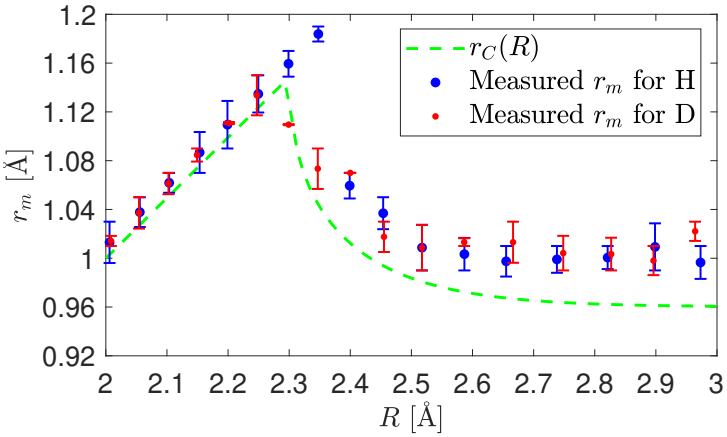

Figure 17: Measured bond length against R using the free-R model.

Figs. (16) and (17) show $r_m \pm \Delta r_m$ against R for H and D compared with $r_C(R)$ in the two models. Values of R in fig. (17) couldn't be controlled as in fig. (16) because R is not known beforehand. In figs. (16) and (17) $r_C(R)$ is the same because $r_C(R)$ is an analytical property of ε_-. In the strong bond regime $r_C(R) = R/2$, as in that regime there is only a single minimum centred between X and Y (see fig. (3)), and empirical bond lengths align well with $r_C(R)$, as $|\phi(x)|^2$ is symmetrical (see fig. (12) for $R = 2.3\,\text{Å}$). Then $r_C(x)$ decays as R gets larger in the moderate and weak bond regimes, as in these regimes there are two minima (see fig. (3)) that drive apart as R increases. Denote $(r_m)_\text{P}, (\Delta r_m)_\text{P}$ the $r_m, \Delta r_m$ of particle $\text{P} \in \{H, D\}$. In the moderate bond regime $r_C(R) \leq (r_m)_\text{D} < (r_m)_\text{H}$ within errors $(\Delta r_m)_\text{H}, (\Delta r_m)_\text{D}$ for both models. In this regime the substitution $\text{H} \to \text{D}$ decreases r_m, in agreement with reference [10]. At the beginning of the moderate bond regime, only in fig. (17) and only for $\text{P} = \text{H}$ an increase in r_m with respect to $r_C(R)$ is observed, while [10] observes this phenomena for both $\text{P} \in \{\text{H}, \text{D}\}$ in the fixed-R model. This phenomena is expected in all models and also for $\text{P} = \text{D}$ because close to $R = 2.3\,\text{Å}$ it is $E_0 \approx -D$ (see figs. (14) and (15)). In the weak bond regime no inequality between $(r_m)_\text{H}, (r_m)_\text{D}$ can be clearly determined in both models. For both $\text{P} \in \{\text{H}, \text{D}\}$, in fig. (16) we see $(r_m)_\text{P} \approx r_C(R)$ while in fig. (17) we see $(r_m)_\text{P} \geq r_C(R)$. It was expected that figs. (16) and (17) would be qualitatively close to being identical, but instead they differ. A possible error in the MHA implementation could be the cause.

4 Conclusions

Feynman's path integral formulation of quantum mechanics has been understood, justified and presented with theoretical clarity and mathematical rigour. A formal derivation of the Euclidean action by the means of a Wick rotation has been invented and the implications of the latter have been highlighted. A complete quantum Monte Carlo `C++` programme has been written and successfully applied to toy models and the H-bond. The MHA was found to be influenced in its execution by simulated system. Computational results regarding properties of the three toy models, namely ZPEs, position PDFs in function of β, and particle sizes, agree with theoretical predictions in the limit of numerical errors. Computational results regarding the H-bond do not agree with expectations: neither NQE (regarding measured ZPEs) nor secondary geometric isotope effect (regarding measured bond lengths) are observed. Tunnelling was observed in toy models and the H-bond.

Further expansion of this work could modify the `C++` programme to simulate 3D systems: this way phase transitions, usually arising in 2D systems [19], could be observed. More focus could be put in numerically implementing analytical methods to extract higher eigenenergies $E_{n>0}$, such as in [4, p.57]. A more formal study of the execution time of the MHA could give important informations on the optimal values of P_M and r_I: we hypothesise these values are largely potential-dependant. The choice of k-indexes in $S[\tilde{\mathbf{x}}]$, for example approximating the particle's speed with a backward Euler method instead of a forward one, could be altered and differences in the execution of the MHA could be studied.

5 Acknowledgements

I would like to thank our supervisor, Dr. Jarvist Moore Frost, firstly for proposing this interesting project to us and secondly for his guidance through the two terms of work and his willingness to answer all our questions and doubts about the research subject. I would also like to thank my project partner for being able to put up with my strong work passion for this project.

References

[1] Markland T, Ceriotti M. Nuclear quantum effects enter the mainstream. *Nature Reviews Chemistry*. 2018;2(3): 1-2. Available from: https://doi.org/10.1038/s41570-017-0109.

[2] Westbroek MJE, King PR, Vvedensky DD, Dürr S. User's guide to Monte Carlo methods for evaluating path integrals. *American Journal of Physics*. 2018;86(4): 1-10. Available from: https://doi.org/10.1119/1.5024926.

[3] IUPAC. *Compendium of Chemical Terminology, 2nd ed. (the "Gold Book")*. Available from: https://goldbook.iupac.org/ [Accessed 3rd April 2020].

[4] Rosenfelder R. Path Integrals in Quantum Physics. *Arxiv*. [Preprint] 2017. Available from: https://arxiv.org/abs/1209.1315v4.

[5] Feynman RP, Hibbs AR. *Quantum Mechanics and Path Integrals. Emended Edition*. New York: Dover Publications Inc.; 2005.

[6] Thijssen JM. *Computational Physics. Second edition*. New York: Cambridge University Press; 2007.

[7] Mittal S, Westbroek MJE, King PR, Vvedensky DD. Path integral Monte Carlo method for the quantum anharmonic oscillator. *Arxiv*. [Preprint] 2018. Available from: https://arxiv.org/abs/1811.04669.

[8] Del Maestro A. *Path Integral Monte Carlo and the Worm Algorithm in the Spatial Continuum*. [Presentation] Université de Sherbrooke. 3rd June 2014.

[9] Wikipedia. *Hydrogen bond*. Available from: https://en.wikipedia.org/wiki/Hydrogen_bond [Accessed 6th May 2020].

[10] McKenzie RH, Bekker C, Athokpam B, Ramesh SG. Effect of quantum nuclear motion on hydrogen bonding. *The Journal of Chemical Physics*. 2014;140(17): 1-6. Available from: https://doi.org/10.1063/1.4873352.

[11] Wikipedia. *Heat equation*. Available from: https://en.wikipedia.org/wiki/Heat_equation [Accessed 6th May 2020].

[12] Schultz TD. Slow Electrons in Polar Crystals: Self-Energy, Mass, and Mobility. *Physical Review*. 1959;116(3): 528. Available from: https://doi.org/10.1103/PhysRev.116.526.

[13] Pritchard J. *Second Year Quantum Mechanics*. [Lecture] Imperial College. 7th October 2019.

[14] Feynman RP. *Statistical Mechanics. A set of lectures*. Reading: W. A. Benjamin Inc.; 1972.

[15] Frost JM. Royal Society University Research Fellow. Personal communication. 6th December 2019.

[16] Weisstein EW. *Gaussian Integral*. Available from: http://mathworld.wolfram.com/GaussianIntegral.html [Accessed 3rd April 2020].

[17] Trefethen N. *Double-well Schroedinger eigenstates*. Available from: https://www.chebfun.org/examples/ode-eig/DoubleWell.html [Accessed 6th May 2020].

[18] Weisstein EW. *Trotter Product Formula*. Available from: http://mathworld.wolfram.com/TrotterProductFormula.html [Accessed 3rd April 2020].

[19] Christensen K, Moloney NR. *Statistical Mechanics 2019-2020*. [Lecture] Imperial College. 4th November 2019.

[20] Wikipedia. *Cauchy distribution*. Available from: https://en.wikipedia.org/wiki/Cauchy_distribution [Accessed 6th May 2020].

[21] Taylor EF, Wheeler JA. *Spacetime Physics: Introduction to Special Relativity. Second Edition*. New York: W.H. Freeman; 1992.

A Conventions of notation

We use the convention $\mathbb{R}_+ := [0;+\infty[$ and $\mathbb{R}_- :=]-\infty;0]$. Given $a \in \mathbb{C}$ and $B \subset \mathbb{C}$, we denote $aB \subset \mathbb{C}$ the set composed of all elements of B all multiplied by a. For example the negative imaginary axis (including 0) is $i\mathbb{R}_-$.

The identity operator in terms of position eigenstates and energy eigenstates is given respectively by

$$I_x := \int_{\mathbb{R}} \mathrm{d}x \, |x\rangle \langle x| \qquad \text{and} \qquad I_n := \sum_{n=0}^{\infty} |n\rangle \langle n| \,, \tag{59}$$

where $x \in \mathbb{R}$ and $n \in \mathbb{N}$. We denote with $\hat{1}$ the 2×2 identity matrix.

The notation $[a, b]$, usually written under the integral symbol when calculating complex integrals, signifies a straight line contour oriented from point a to point b, where $a, b \in \mathbb{C}$. For example if $a = 0$ and $b = i$ then $[a, b] = i[0; 1]$ oriented from 0 to i.

B Complete derivation of the path integral

This derivation is a combination of those found in [4, p.56-57] and [2]. The Hamiltonian $\hat{H} = \hat{T} + \hat{V}$ is separated into its kinetic energy part $\hat{T}$ and its potential energy part $\hat{V}$. Discretize the time interval $t_f - t_i$ in a lattice composed of $N \in \mathbb{N}$ time steps of duration $\varepsilon = (t_f - t_i)/N$. Let $\hat{A}, \hat{B}$ be two arbitrary operators: Trotter's formula [18] states that

$$e^{\hat{A}+\hat{B}} = \lim_{n\to\infty} \prod_{k=1}^{n} e^{\hat{A}/n} e^{\hat{B}/n}. \tag{60}$$

By assuming $N \gg 1$ we can apply Trotter's formula to get

$$K(f,i) = \left\langle x_f \left| e^{-i\varepsilon N(\hat{T}+\hat{V})/\hbar} \right| x_i \right\rangle \approx \left\langle x_f \left| \prod_{k=1}^{N} e^{-i\varepsilon\hat{T}/\hbar} e^{-i\varepsilon\hat{V}/\hbar} \right| x_i \right\rangle. \tag{61}$$

If we insert $N-1$ identity operators in position space between each product then we get

$$K(f,i) \approx \int_{\mathbb{R}} \cdots \int_{\mathbb{R}} \left\langle x_N \left| e^{-i\varepsilon\hat{T}/\hbar} e^{-i\varepsilon\hat{V}/\hbar} \right| x_{N-1} \right\rangle \left\langle x_{N-1} \left| e^{-i\varepsilon\hat{T}/\hbar} e^{-i\varepsilon\hat{V}/\hbar} \right| x_{N-2} \right\rangle \tag{62}$$

$$\cdots \left\langle x_2 \left| e^{-i\varepsilon\hat{T}/\hbar} e^{-i\varepsilon\hat{V}/\hbar} \right| x_1 \right\rangle \left\langle x_1 \left| e^{-i\varepsilon\hat{T}/\hbar} e^{-i\varepsilon\hat{V}/\hbar} \right| x_0 \right\rangle \mathrm{d}x_1 \cdots \mathrm{d}x_{N-1}, \tag{63}$$

where it is denoted $x_i = x_0$, $x_f = x_N$. It is clear that $\hat{V}|x\rangle = V(x)|x\rangle$ hence $\exp(-i\varepsilon\hat{V}/\hbar)|x\rangle = \exp(-i\varepsilon V(x)/\hbar)$. In the momentum representation

$$\hat{T} = \frac{p^2}{2m} \qquad \text{and} \qquad |x\rangle = \frac{1}{\sqrt{2\pi\hbar}} \exp\left(-\frac{ipx}{\hbar}\right), \tag{64}$$

where $p\,[\mathrm{kg}\cdot\mathrm{m/s}]$ and, using the Gauss integral [16] (let $a, b \in \mathbb{C} \setminus \{0\}$ with $\mathrm{Re}(a) > 0$)

$$\int_{\mathbb{R}} e^{-ax^2+bx} \mathrm{d}x = \sqrt{\frac{\pi}{a}} \exp\left(\frac{b^2}{4a}\right), \tag{65}$$

it is possible to compute

$$\begin{aligned} \left\langle x_k \left| e^{-i\varepsilon\hat{T}/\hbar} \right| x_{k-1} \right\rangle &= \frac{1}{2\pi\hbar} \int_{\mathbb{R}} \exp\left(\frac{ipx_k}{\hbar}\right) \exp\left(-\frac{i\varepsilon p^2}{2m\hbar}\right) \exp\left(-\frac{ipx_{k-1}}{\hbar}\right) \mathrm{d}p \\ &= \frac{1}{2\pi\hbar} \int_{\mathbb{R}} \exp\left(-\frac{i\varepsilon p^2}{2m} + \frac{ip(x_k - x_{k-1})}{\hbar}\right) \mathrm{d}p \\ &= \sqrt{\frac{m}{2\pi i\hbar\varepsilon}} \exp\left(\frac{im}{2\hbar\varepsilon}(x_k - x_{k-1})^2\right) = A_\varepsilon \exp\left(\frac{im}{2\hbar\varepsilon}(x_k - x_{k-1})^2\right) \end{aligned} \tag{66}$$

for all $k = 1, \dots, N$, where the constant A_ε is defined as

$$A_\varepsilon := \sqrt{\frac{m}{2\pi i\hbar\varepsilon}}, \tag{67}$$

and hence it can be written

$$\begin{aligned} K(f,i) &\approx A_\varepsilon^N \int_{\mathbb{R}} \mathrm{d}x_1 \cdots \int_{\mathbb{R}} \mathrm{d}x_{N-1} \exp\left(\sum_{k=1}^{N}\left[\frac{im}{2\hbar\varepsilon}(x_k - x_{k-1})^2 - i\frac{\varepsilon}{\hbar}V(x_{k-1})\right]\right) \\ &= A_\varepsilon^N \int_{\mathbb{R}} \mathrm{d}x_1 \cdots \int_{\mathbb{R}} \mathrm{d}x_{N-1} \exp\left(\varepsilon\frac{i}{\hbar}\sum_{k=1}^{N}\left[\frac{m}{2}\left(\frac{x_k - x_{k-1}}{\varepsilon}\right)^2 - V(x_{k-1})\right]\right). \end{aligned} \tag{68}$$

C Proof of properties of $\widetilde{f}$

Let $f, g : \mathbb{C} \to \mathbb{C}$ be two continuous functions and define $\widetilde{f} : \mathbb{C} \to \mathbb{C}$ to be such that $\widetilde{f}(zi) = f(z)$. We assert the following properties:

$$\begin{aligned} \widetilde{f+g} &= \widetilde{f} + \widetilde{g}, \\ \widetilde{f \circ g} &= f \circ \widetilde{g}, \\ \widetilde{\frac{\mathrm{d}f}{\mathrm{d}z}} &= i\frac{\mathrm{d}\widetilde{f}}{\mathrm{d}z}. \end{aligned} \tag{69}$$

The first property follows from the fact that if $\widetilde{f}(zi) = f(z)$ and $\widetilde{g}(zi) = g(z)$ then $\widetilde{f}(zi) + \widetilde{g}(zi) = (f+g)(z)$ and this proves that $\widetilde{f+g}(zi) = \widetilde{f}(zi) + \widetilde{g}(zi)$. For the second property consider $\widetilde{g}(zi) = g(z)$, then $(f \circ \widetilde{g})(zi) = (f \circ g)(z)$ so that $f \circ \widetilde{g} = \widetilde{f \circ g}$. The third property can be seen by differentiating with respect of z on both sides $\widetilde{f}(zi) = f(z)$. This gives

$$\frac{\mathrm{d}}{\mathrm{d}z}\widetilde{f}(zi) = i\frac{\mathrm{d}\widetilde{f}}{\mathrm{d}z}(zi) = \frac{\mathrm{d}f}{\mathrm{d}z}(z) \tag{70}$$

and therefore $\widetilde{\mathrm{d}f/\mathrm{d}z} = i\mathrm{d}\widetilde{f}/\mathrm{d}z$.

D Explicit calculation for the zero-point energy

Noting $x' = \tilde{x}_0 = \tilde{x}_N$ we can compute

$$\begin{aligned} E_0 &= \lim_{\beta\to\infty} -\hbar\frac{\partial}{\partial\beta}\ln Z(\beta) \\ &= \lim_{\beta\to\infty} \frac{-\hbar}{Z(\beta)} \int_{\mathbb{R}} \mathrm{d}x' \int \frac{\partial}{\partial\beta} \exp\left(-\frac{S_E[\tilde{x}(\tau)]}{\hbar}\right) \mathrm{D}\tilde{x}(\tau) \\ &= \lim_{\beta\to\infty} \frac{1}{Z(\beta)} \int_{\mathbb{R}} \mathrm{d}x' \int e^{-S_E[\tilde{x}(\tau)]/\hbar} H\left(\tilde{x}(\beta), \frac{\mathrm{d}\tilde{x}}{\mathrm{d}\tau}(\beta)\right) \mathrm{D}\tilde{x}(\tau), \end{aligned} \tag{71}$$

where $H(\tilde{x}(\beta), (\mathrm{d}\tilde{x}/\mathrm{d}\tau)(\beta))$ is the Hamiltonian of the system. Because time is cyclic any time $\tau \in [0;\beta]$ could have been selected as the end time without changing the physics. Hence, in sight of a numerical implementation, we can take the average over all time slices [6]

$$H\left(\tilde{x}(\beta), \frac{\mathrm{d}\tilde{x}}{\mathrm{d}\tau}(\beta)\right) \approx \frac{1}{N}\sum_{k=1}^{N} H\left(\tilde{x}(k\delta\tau), \frac{\mathrm{d}\tilde{x}}{\mathrm{d}\tau}(k\delta\tau)\right) = \frac{1}{N}\sum_{k=1}^{N}\left[\frac{m}{2}\left(\frac{\tilde{x}_k - \tilde{x}_{k-1}}{\delta\tau}\right)^2 + V(\tilde{x}_{k-1})\right], \tag{72}$$

where the velocity is approximated with a forward Euler scheme. Hence we can continue

$$\begin{aligned} E_0 &= \lim_{\beta\to\infty} \frac{\int_{\mathbb{R}} \mathrm{d}\tilde{x}_0 \cdots \int_{\mathbb{R}} \mathrm{d}\tilde{x}_{N-1} \frac{1}{N} \sum_{k=1}^{N} \left[\frac{m}{2} \left(\frac{\tilde{x}_k - \tilde{x}_{k-1}}{\delta\tau} \right)^2 + V(\tilde{x}_{k-1}) \right] e^{-S_E[\tilde{x}(\tau)]/\hbar}}{\int_{\mathbb{R}} \mathrm{d}\tilde{x}_0 \int_{\mathbb{R}} \mathrm{d}\tilde{x}_1 \cdots \int_{\mathbb{R}} \mathrm{d}\tilde{x}_{N-1} e^{-S_E[\tilde{x}(\tau)]/\hbar}} \\ &= \lim_{\beta\to\infty} \left\langle \frac{1}{N} \sum_{k=0}^{N-1} \left[\frac{m}{2} \left(\frac{\tilde{x}_k - \tilde{x}_{k-1}}{\delta\tau} \right)^2 + V(\tilde{x}_{k-1}) \right] \right\rangle . \end{aligned} \tag{73}$$

E Proof of the Virial theorem

We follow [4, p.57]. Consider the ground state expectation value of the operator $\left[\hat{x}\hat{p}, \hat{H}\right]$:

$$\left\langle 0 \left| \left[\hat{x}\hat{p}, \hat{H}\right] \right| 0 \right\rangle = \langle 0| \hat{x}\hat{p}(\hat{H}|0\rangle) - (\langle 0| \hat{H})\hat{x}\hat{p}|0\rangle = 0, \tag{74}$$

where in the last equality we used the fact that $\hat{H} = \hat{T} + \hat{V}$ is hermitian. The commutator can be explicitly computed as

$$\begin{aligned} \left[\hat{x}\hat{p}, \frac{\hat{p}^2}{2m} + \hat{V}\right] &= -xi\hbar \frac{\partial}{\partial x} \left(V(x) - \frac{\hbar^2}{2m} \frac{\partial^2}{\partial x^2} \right) + i\hbar \left(V(x) - \frac{\hbar^2}{2m} \frac{\partial^2}{\partial x^2} \right) \left(x \frac{\partial}{\partial x} \right) \\ &= i\hbar \Bigg[\left(x \frac{\hbar^2}{2m} \frac{\partial^3}{\partial x^3} - xV'(x) - xV(x) \frac{\partial}{\partial x} \right) \\ &+ \left(xV(x) \frac{\partial}{\partial x} - \frac{\hbar^2}{2m} \frac{\partial}{\partial x} \left(\frac{\partial}{\partial x} + x \frac{\partial^2}{\partial x^2} \right) \right) \Bigg] \\ &= i\hbar \Bigg(x \frac{\hbar^2}{2m} \frac{\partial^3}{\partial x^3} - xV'(x) - xV(x) \frac{\partial}{\partial x} + xV(x) \frac{\partial}{\partial x} - \frac{\hbar^2}{2m} \frac{\partial^2}{\partial x^2} \\ &- \frac{\hbar^2}{2m} \frac{\partial^2}{\partial x^2} - x \frac{\hbar^2}{2m} \frac{\partial^3}{\partial x^3} \Bigg) = i\hbar \left(-xV'(x) - \frac{\hbar^2}{m} \frac{\partial^2}{\partial x^2} \right) = i\hbar \left(2\hat{T} - xV'(x) \right) \end{aligned}$$

because in position representation

$$\hat{p} = -i\hbar \frac{\partial}{\partial x} \qquad \text{and} \qquad \hat{p}^2 = -\hbar^2 \frac{\partial^2}{\partial x^2},$$

hence it is found that

$$\left\langle 0 \left| \hat{T} \right| 0 \right\rangle = \left\langle 0 \left| \frac{x}{2} V'(x) \right| 0 \right\rangle . \tag{75}$$

F Convergence of the Metropolis-Hastings algorithm

This proof is a mix of those in [19] and [4, p.59]. Consider a large number of identical Markov chains. The probability the move $\mathbf{x} \to \mathbf{y}$ is accepted is given by

$$P(\mathbf{x} \to \mathbf{y}) = T(\mathbf{x} \to \mathbf{y}) \cdot A(\mathbf{x} \to \mathbf{y}), \tag{76}$$

where T is the transfer probability and A the acceptance probability of the move. Assume that for the transfer probability the property of 'Detailed balance'

$$T(\mathbf{x} \to \mathbf{y}) = T(\mathbf{y} \to \mathbf{x}) \tag{77}$$

is verified and consider an acceptance probability of the form

$$A(\mathbf{x} \to \mathbf{y}) = \min(1, r) = \min\left(1, \frac{\omega(\mathbf{y})}{\omega(\mathbf{x})}\right). \tag{78}$$

Then, the variation of the number of chains that evolve as $\mathbf{x} \to \mathbf{y}$ is

$$\Delta N(\mathbf{x} \to \mathbf{y}) = N(\mathbf{x})P(\mathbf{x} \to \mathbf{y}) - N(\mathbf{y})P(\mathbf{y} \to \mathbf{x}). \tag{79}$$

At equilibrium ([19] shows that equilibrium is always reached) $\Delta N(\mathbf{x} \to \mathbf{y}) = 0$ for all possible $\mathbf{x}, \mathbf{y}$, hence

$$\frac{N(\mathbf{y})}{N(\mathbf{x})} = \frac{P(\mathbf{x} \to \mathbf{y})}{P(\mathbf{y} \to \mathbf{x})} = \frac{A(\mathbf{x} \to \mathbf{y})}{A(\mathbf{y} \to \mathbf{x})}. \tag{80}$$

If $\omega(\mathbf{y}) > \omega(\mathbf{x})$ then $r > 1$ so that $A(\mathbf{x} \to \mathbf{y}) = 1$ and $A(\mathbf{y} \to \mathbf{x}) = 1/r$. If $\omega(\mathbf{x}) > \omega(\mathbf{y})$ then $r < 1$ so that $A(\mathbf{x} \to \mathbf{y}) = r$ and $A(\mathbf{y} \to \mathbf{x}) = 1$. In both cases

$$\frac{N(\mathbf{y})}{N(\mathbf{x})} = r = \frac{\omega(\mathbf{y})}{\omega(\mathbf{x})}, \tag{81}$$

hence at equilibrium $N(\mathbf{x}) \sim \omega(\mathbf{x})$, which is the assertion.

G Numerical implementation of the Cauchy distribution

The Cumulative distribution function (CDF) of a random variable distributed according to the Cauchy distribution, of parameters $(x_0, \gamma) = (0, h_j)$, is [20]

$$F(x) = \frac{1}{\pi} \arctan\left(\frac{x}{h_j}\right) + \frac{1}{2}. \tag{82}$$

Its inverse is

$$F^{-1}(y) = h_j \tan\left[\pi\left(y - \frac{1}{2}\right)\right]. \tag{83}$$

If Y is a random variable uniformly distributed in the interval $[-1/2, 1/2]$ then

$$X = F^{-1}(Y) = h_j \tan(\pi Y) \tag{84}$$

is distributed according to the Cauchy distribution.

H Justification of unit prefixes

We justify the choice of the powers Q. For k_B and $\hbar$ the choice is dictated by their experimental value. For the mass m we are interested to modelize subatomic particles such as electrons or protons, which are of order $Q = -30$. For the position x we consider atomic distances, which are of the order of 1 Å hence of order $Q = -10$. For the potential V experimental data show that atomic potentials are of order $Q = -20$ [10]. For the angular frequency ω of an harmonic oscillator, we see that

$$V(x) = \frac{1}{2}m\omega^2 x^2 = \frac{1}{2}m_N \omega_N^2 x_N^2 \cdot 10^{2Q-50}\,\mathrm{J} \tag{85}$$

and therefore in order to have a potential of order -20 it must be $Q = 15$. For $\delta\tau$ we consider the approximation of the kinetic energy

$$K = \frac{1}{2}mv^2 = \frac{1}{2}m_N v^2 \cdot 10^{-30}\,\mathrm{kg} \approx \frac{1}{2}m_N \left(\frac{\delta x_N}{\delta \tau_N}\right)^2 \cdot 10^{-2Q-50}\,\mathrm{J}. \tag{86}$$

The Lagrangian is $K - V$, therefore having K of the same order as V would be advantaging. Special relativity imposes the range of values $[0; c]$ for the velocities v of bradyons, where

$c = 299792458\,\mathrm{m/s}$ is the speed of light in a vacuum, and relativistic effects have to be taken into account when $v > 0.1c$ [21]. The theoretical framework is not relativistic, therefore we won't account for those effects and we will assume $v < 0.1c$ in our simulations. At this point the most sensible guess of the order of v would be $(8-1)/2 = 3.5$, but instead we chose to assign the order 5 so that for $\delta\tau$ $Q = -15$: in fact this ensures that the order of K is -20 as wished. Finally for the simulation time β recall that $\beta = N\delta\tau$ and therefore the order of β can be the same as $\delta\tau$.

I Source code

Downloadable source code is available at https://github.com/MightyBee/PIMC-Hbonds.

```
#define _USE_MATH_DEFINES
#include <vector>
#include <array>
#include <iostream>
#include <fstream>
#include <string>
#include <random>
#include <memory>
#include <ctime>
#include <cmath>
#include "ConfigFile.tcc" //Villard L., Computational Physics I-II, EPFL,
    2018-2019.

using namespace std;

double hbar(10.54571628); // IUPAC

std::mt19937 rng(time(0));


/*################################## NOTES ###################################/
  - theoretically x_0, x_1, ... , x_(N_slices)       , (N_slices+1) points
  - we consider boundary conditions, x_0 = x_(N_slices)
  - hence we only consider x_0, x_1, ... , x_(N_slices-1) , N_slices points
  - to get the "full picture" simply add one more point, equal to x_0
*/

/****** PLAN OF THE CODE #####//
  - Part A : headears
      - A.1 : function headers
      - A.2 : definitions of class "Potential_ext" and inherited classes
      - A.3 : definitions of class "Potential_ext" and inherited classes
      - A.4 : definitions of class "System"

  - Part B : main
      - B.1 : parameters acquisition and system initialization
      - B.2 : metropolis algorithm
      - B.3 : statistics writing

  - Part A : definitions
      - C.1 : definitions of the methods of class "Potential_ext" and
              inherited classes
      - C.2 : definitions of the methods of class "Potential_ext" and
              inherited classes
      - C.3 : definitions of the methods of class "System"
      - C.4 : function definitions
```

```
46
47
48 */
49
50 //#########################################################################//
51 //#########################################################################//
52 //########################                                 ########################//
53 //########################      PART A : HEADERS           ########################//
54 //########################                                 ########################//
55 //#########################################################################//
56 //#########################################################################//
57
58
59 //#########################################################################//
60 //########################## A.1 : FUNCTION HEADERS ########################//
61 //#########################################################################//
62
63 // Generate a random (uniform) double between 'min' and 'max'
64 double randomDouble(const double& min=0.0, const double& max=1.0,
65                     const bool& closed=true);
66
67 // Generate a random double from a normal Cauchy distribution
68 double CauchyDistribution();
69
70 // Generate a random double from one of the implemeneted distributions
71 double GenerateDist(const double& h);
72
73
74
75 //#########################################################################//
76 //####################### A.2 : CLASSES Potential_ext #####################//
77 //#########################################################################//
78
79 // Abstract class for external potential
80 class Potential_ext {
81 public:
82   // pure virtual method => abstract class
83   // return V at point x
84   virtual double operator()(const double& x) const = 0;
85   double e0_estimator(const double& x) const{return 0;};
86 };
87
88
89 // Class for a null potential
90 class PotExt_null: public Potential_ext {
91 public:
92   double operator()(const double& x) const {return 0.0;}
93 };
94
95
96 // Class for a harmonic potential
97 class PotExt_harm: public Potential_ext {
98 public:
99   PotExt_harm(const ConfigFile& configFile);
100  double operator()(const double& x) const;
101 private:
102  // mass and squared frequency
103  double m, omega2;
104 };
105
```

```
106
107 // Class for a double well potential
108 class PotExt_double: public Potential_ext {
109 public:
110   PotExt_double(const ConfigFile& configFile);
111   double operator()(const double& x) const;
112 private:
113   // barrier height and position of the wells
114   double V0, x0;
115 };
116
117
118 // Class for a square potential (barrier for V0>0 and well for V0<0)
119 class PotExt_square: public Potential_ext {
120 public:
121   PotExt_square(const ConfigFile& configFile);
122   double operator()(const double& x) const;
123 private:
124   // potential height, position of the centre and width of the square well
125   double V0, x0, L;
126 };
127
128
129 // Class for a sinusoidal potential
130 class PotExt_sin: public Potential_ext {
131 public:
132   PotExt_sin(const ConfigFile& configFile);
133   double operator()(const double& x) const;
134 private:
135   // potential height and period
136   double V0, L;
137 };
138
139
140 // Class for a Lennard-Jones potential
141 class PotExt_LJ: public Potential_ext {
142 public:
143   PotExt_LJ(const ConfigFile& configFile);
144   double operator()(const double& x) const;
145 private:
146   double V0, x0;
147 };
148
149 // Class for a H-bond potential
150 class PotExt_OHbonds: public Potential_ext {
151 public:
152   PotExt_OHbonds(const ConfigFile& configFile);
153   // Morse potential
154   double Vmorse(const double& x) const;
155   // First derivative of Morse potential
156   double dVmorse(const double& x) const;
157   // Estimator of zero-point energy
158   double e0_estimator(const double& x) const;
159   // Return the value of the potential
160   double operator()(const double& x) const;
161 private:
162   // Characteristic constants of the potential
163   double D, a, r0, delta1, b, R1;
164   // R : distance between the donor and accpetor
165   double R, DELTA;
```

```
};



//#########################################################################//
//###################### A.2 : CLASSES Potential_int ######################//
//#########################################################################//

// Abstract class for internal potential
class Potential_int {
public:
  // pure virtual method => abstract class
  // return V for particle 1,2 at position x1,x2 resp.
  virtual double operator()(const double& x1, const double& x2) const = 0;
};


// Class for a null internal potential (no interactions between particles)
class PotInt_null: public Potential_int {
public:
  double operator()(const double& x1, const double& x2) const {return 0.0;}
};


// Class for a harmonic potential between two particles
class PotInt_harm: public Potential_int {
public:
  PotInt_harm(const ConfigFile& configFile);
  double operator()(const double& x1, const double& x2) const;
private:
  // stifness and rest length
  double k, l0;
};


// Class for a Lennard-Jones potential between two particles
class PotInt_LJ: public Potential_int {
public:
  PotInt_LJ(const ConfigFile& configFile);
  // standard Lennard-Jones potential
  double LJ(const double& r) const;
  // Lennard-Jones with parameters defined below
  double operator()(const double& x1, const double& x2) const;
private:
  double V0, x0, G;
};



//#########################################################################//
//########################## A.3 : CLASSES System #########################//
//#########################################################################//


// Class System : - contains all the physical properties of the system
//                  considered, as well as the number of slices
//                - it can generate moves but it's not this class that
//                  will porpose it
class System {
public:
```

```
  // constructor
  System(const ConfigFile& configFile);
  // initlaize the system : hot start
  void initialize(const double& pos_min, const double& pos_max);
  // write in an output file the external potential used
  void write_potExt(const string& output);
  // return the number of particles in the system
  size_t nb_part() const {return N_part;}
  // return the number of slices in the system
  size_t nb_slices() const {return N_slices;}
  // return the number of time each site is visited by MH algorithm
  vector<vector<int>> get_visits() const {return verif;}
  // write the poisitions of all particles
  ostream& write(ostream& output) const;
  // return the kinetic term of a particle between a bead and a neighbour
  double kinetic(const int& particle, const int& bead, const int& bead_pm,
                 const double& displacement=0.0) const;
  // compute the whole euclidean action directly
  double energy();
  // return the euclidean action measured along the simulation's progress
  double get_H(){return H;}

  // returns if a move is accepted or not
  bool metropolisAcceptance();

  // different moves possible
  bool localMove(const double& h);
  bool globalDisplacement(const double& h);
  bool bisection(const double& h, const double& sRel);
  bool swap();
  bool inverse();
  bool symmetryCM();

  // energy measures
  void measure_energy(double, double);
  void average_energy();


private:
  ////////  physical parameters  ////////
  unsigned int N_part;
  unsigned int N_slices;
  double beta;
  double d_tau;
  int q;
  vector<double> mass;
  double omega;
  unique_ptr<Potential_ext> ptr_Vext;
  unique_ptr<Potential_int> ptr_Vint;
  // table of positions : first index indicates the particle
  //                      second index indicates the slice
  vector<vector<double>> table;

  ////////  utilitary variables  ////////
  // mm : time slice randomly selected during each iteration,
  // mm_plu=mm+1, mm_min=mm-1 with boundary conditions
  unsigned int mm, mm_plu, mm_min;
  // particle randomly selected during each iteration
  unsigned int nn;
  // displacement proposed
```

```
286   double dis;
287   // part of the action that is changed by moves
288   double s_old, s_new;
289   // euclidean action
290   double H;
291 
292   vector<double> energies_psi;
293   vector<double> energies_h;
294   vector<vector<int>> verif;
295 };
296 
297 ostream& operator<<(ostream& output, const System& s);
298 
299 
300 
301 
302 
303 
304 
305 
306 //////////////////////////////////////////////////////////////////////////////
307 //////////////////////////////////////////////////////////////////////////////
308 //////////////////////                                  //////////////////////
309 //////////////////////       PART B : MAIN              //////////////////////
310 //////////////////////                                  //////////////////////
311 //////////////////////////////////////////////////////////////////////////////
312 //////////////////////////////////////////////////////////////////////////////
313 
314 
315 int main(int argc, char* argv[]){
316 
317 
318   //////////////////////////////////////////////////////////////////////////
319   //////////////////// B.1 : PARAMETERS ACQUISITION ////////////////////////
320   //////////////////////////////////////////////////////////////////////////
321 
322   ////////////////////////// VILLARD LIBRARY ///////////////////////////////
323 
324   // Default input configuration file
325   string inputPath("configuration.in");
326   if(argc>1) // specified input file specified by user
327     inputPath = argv[1];
328 
329   // Parameters are read et stocked in a "map" of strings
330   ConfigFile configFile(inputPath);
331 
332   for(int i(2); i<argc; ++i) // complementary inputs
333     configFile.process(argv[i]);
334 
335 
336   ////////////////////////// READ PARAMETERS ///////////////////////////////
337 
338   // number of Monte Carlo iterations (aka sweeps)
339   unsigned int N_sweeps(configFile.get<unsigned int>("N_sweeps"));
340   // number of thermalisation sweeps
341   unsigned int N_thermalisation(configFile.get<unsigned int>("N_thermal"));
342   // initial minimum position
343   double pos_min(configFile.get<double>("pos_min"));
344   // initial maximal position
345   double pos_max(configFile.get<double>("pos_max"));
```

```
346   // displacement parameter of a point in the path
347   vector<double> h(3,configFile.get<double>("h"));
348   // proportion of each move
349   double p_loc(configFile.get<double>("p_local"));
350   double p_dsp(configFile.get<double>("p_displacement"));
351   double p_swap(configFile.get<double>("p_swap"));
352   double p_inv(configFile.get<double>("p_inverse"));
353   double p_sym(configFile.get<double>("p_symmetryCM"));
354   double p_bis(configFile.get<double>("p_bisection"));
355   // relative size (0 to 1) of bisection move
356   double s_bis(configFile.get<double>("s_bisection"));
357   // numbers of tries for each moves
358   vector<unsigned int> NbTries(6,0);
359   // acceptance rates for the three moves
360   vector<double>       accrate(6,0.0);
361   // "instantaneous" acceptance rate for local moves
362   double tmp_accrate(0.0);
363   // ideal acceptance rate for local moves
364   double idrate(configFile.get<double>("idrate"));
365   // output is written every n_stride iterations
366   size_t n_stride(configFile.get<size_t>("n_stride"));
367
368
369   //Output files
370   string output(configFile.get<string>("output"));
371   string output_pos(output+"_pos.out");
372   ofstream fichier_output(output_pos.c_str());
373   fichier_output.precision(15);   // Precision
374
375   string output_energy(output+"_nrg.out");
376   ofstream fichier_energy(output_energy.c_str());
377   fichier_energy.precision(15);   // Precision
378
379   string output_rate(output+"_rate.out");
380   ofstream fichier_rate(output_rate.c_str());
381   fichier_rate.precision(15);
382
383   // initialization of the system
384   System s(configFile);
385   s.initialize(pos_min,pos_max);
386   s.write_potExt(output);
387   fichier_output << s << endl;
388
389   //UNCOMMENT IF YOU WANT TO COMPARE WITH OUTPUT2_NRG.OUT
390   //s.measure_energy();
391   //double V0(configFile.get<double>("R"));
392   //double V0(configFile.get<double>("V0"));
393   //double x0(configFile.get<double>("x0"));
394
395   double last_measured_time(time(0));
396
397
398   //////////////////////////////////////////////////////////////////////////////
399   //////////////////// B.2 : METROPOLIS ALGORITHM ////////////////////
400   //////////////////////////////////////////////////////////////////////////////
401
402   //For every sweep...
403   for(size_t i(0); i < N_sweeps; i++){
404
405     // show progress of the simulation
```

```
406     if(time(0) - last_measured_time >= 5){
407       last_measured_time = time(0);
408       cout << floor((double)i/N_sweeps*100) << " %" << endl;
409     }
410 
411     //For every particle...
412     /*(we try one average each move for each particle one time every sweep
413       if all moves proportions are set to 1) */
414     for(size_t j(0); j < s.nb_part(); j++){
415 
416       ///////////  local move  ///////////
417       /*if proportion of tries compared to the number of sweeps is
418         smaller than the target, try a move*/
419       if(NbTries[0]*1.0/((i*s.nb_part()+j+1)*s.nb_slices()) < p_loc){
420         tmp_accrate=0.0;
421         for(size_t k(0); k < s.nb_slices(); k++){
422           NbTries[0]++;
423           if(s.localMove(h[0])){
424             accrate[0]++;
425             tmp_accrate++;
426           }
427         }
428         tmp_accrate/=s.nb_slices();
429         // adjust the parametr h to reach the ideal accpetance rate
430         h[0]*=tmp_accrate/idrate;
431       }
432 
433       ///////////  global displacement  ///////////
434       /*if proportion of tries compared to the number of sweeps is
435         smaller than the target, try a move*/
436       if(NbTries[1]*1.0/(i*s.nb_part()+j+1) < p_dsp){
437         NbTries[1]++;
438         if(s.globalDisplacement(h[1])){
439           accrate[1]++;
440         }
441       }
442 
443       ///////////  bisection ///////////
444       if(NbTries[2]*1.0/(i*s.nb_part()+j+1) < p_bis){
445         NbTries[2]++;
446         if(s.bisection(h[2], s_bis)){
447           accrate[2]++;
448         }
449       }
450 
451       ///////////  swap  ///////////
452       if(NbTries[3]*1.0/(i*s.nb_part()+j+1) < p_swap){
453         NbTries[3]++;
454         if(s.swap()){
455           accrate[3]++;
456         }
457       }
458 
459       ///////////  inverse  ///////////
460       if(NbTries[4]*1.0/(i*s.nb_part()+j+1) < p_inv){
461         NbTries[4]++;
462         if(s.inverse()){
463           accrate[4]++;
464         }
465       }
```

```
466
467       //////////  symmetryCM  //////////
468       if(NbTries[5]*1.0/(i*s.nb_part()+j+1) < p_sym){
469         NbTries[5]++;
470         if(s.symmetryCM()){
471           accrate[5]++;
472         }
473       }
474     }
475
476
477     //######################## OUTPUT IN FILE ##########################//
478     if((i%n_stride) == 0){
479       fichier_output << s << endl;
480       fichier_energy << s.energy() << " " << s.get_H() << endl;
481       fichier_rate << tmp_accrate << endl;
482
483       //Energy measurement
484       if(i >= N_thermalisation){
485         //fichier_energy << s.energy() << " " << s.get_H() << endl;
486         //s.measure_energy(V0, x0);
487       }
488     }
489   }
490   fichier_output.close();
491   fichier_energy.close();
492   fichier_rate.close();
493
494
495   //#########################################################################//
496   //###################### B.3 : STATISTICS WRITING ######################//
497   //#########################################################################//
498
499   // statistics on the moves
500   // number of tries and acceptance's rates
501   string output_stat(output+"_stat.out");
502   fichier_output.open(output_stat.c_str());
503   fichier_output.precision(15);
504   for(size_t i(0); i<accrate.size(); i++){
505     accrate[i]/=NbTries[i];
506     fichier_output << NbTries[i] << " " << accrate[i] << endl;
507   }
508
509   // statistics on the visits of patricles and slices
510   for(const auto& part : s.get_visits()){
511     for(const auto& v : part){
512       fichier_output << v/(NbTries[0]*1.0/(s.nb_part()*s.nb_slices()))
513                      << " ";
514     }
515     fichier_output << endl;
516   }
517
518   fichier_output.close();
519
520   //Energy
521   s.average_energy();
522
523   //############################ END OF MAIN ############################//
524   return 0;
525 }
```

```
526
527
528
529
530
531
532
533
534
535 //#########################################################################//
536 //#########################################################################//
537 //########################                                 ########################//
538 //########################    PART C : DEFINITIONS         ########################//
539 //########################                                 ########################//
540 //#########################################################################//
541 //#########################################################################//
542
543
544
545 //#########################################################################//
546 //############ C.1 : CLASS 'Potential_ext' METHODS DEFINITIONS ############//
547 //#########################################################################//
548
549
550 //############################  PotExt_harm  ##############################//
551 ///// constructor /////
552 PotExt_harm::PotExt_harm(const ConfigFile& configFile) :
553   Potential_ext(),
554   m(configFile.get<double>("mass")),
555   omega2(pow(configFile.get<double>("omega"),2))
556   {}
557
558 ///// potential opertor /////
559 double PotExt_harm::operator()(const double& x) const {
560   return 0.5 * m * pow(x, 2) * omega2;
561 }
562
563
564 //############################  PotExt_double  ############################//
565 ///// constructor /////
566 PotExt_double::PotExt_double(const ConfigFile& configFile) :
567   Potential_ext(),
568   V0(configFile.get<double>("V0")),
569   x0(configFile.get<double>("x0"))
570   {}
571
572 ///// potential opertor /////
573 double PotExt_double::operator()(const double& x) const {
574   return V0*pow(pow(x/x0,2)-1,2);
575 }
576
577
578 //############################  PotExt_square  ############################//
579 ///// constructor /////
580 PotExt_square::PotExt_square(const ConfigFile& configFile) :
581   Potential_ext(),
582   V0(configFile.get<double>("V0")),
583   x0(configFile.get<double>("x0")),
584   L(configFile.get<double>("L"))
585   {}
```

```
586
587 ///// potential opertor /////
588 double PotExt_square::operator()(const double& x) const {
589   if(abs(x - x0) < L/2){
590     return V0;
591   }else{
592     return 0;
593   }
594 }
595
596
597 //////////////////////////////  PotExt_sin  ////////////////////////////////
598 ///// constructor /////
599 PotExt_sin::PotExt_sin(const ConfigFile& configFile) :
600   Potential_ext(),
601   V0(configFile.get<double>("V0")),
602   L(configFile.get<double>("L"))
603   {}
604
605 ///// potential opertor /////
606 double PotExt_sin::operator()(const double& x) const {
607   return 0.5*V0*(1-cos(2*M_PI/L*x));
608 }
609
610
611 //////////////////////////////  PotExt_LJ  ////////////////////////////////
612 ///// constructor /////
613 PotExt_LJ::PotExt_LJ(const ConfigFile& configFile) :
614   Potential_ext(),
615   V0(configFile.get<double>("V0")),
616   x0(configFile.get<double>("x0"))
617   {}
618
619 ///// potential opertor /////
620 double PotExt_LJ::operator()(const double& x) const {
621   return 4 * V0 * (pow(x/x0,-12) - pow(x/x0,-6) );
622 }
623
624 //////////////////////////  PotExt_OHbonds  ////////////////////////////
625 ///// constructor /////
626 PotExt_OHbonds::PotExt_OHbonds(const ConfigFile& configFile) :
627   Potential_ext(),
628   D(83.402), a(2.2), r0(0.96), delta1(0.4*D), b(2.2), R1(2*r0+1/a),
629   R(configFile.get<double>("R")),
630   DELTA(delta1*exp(-b*(R-R1)))
631   {}
632
633 ///// Morse potential /////
634 double PotExt_OHbonds::Vmorse(const double& x) const{
635   return D*(exp(-2*a*(x-r0))-2*exp(-a*(x-r0)));
636 }
637
638 ///// first derivative of Morse potential /////
639 double PotExt_OHbonds::dVmorse(const double& x) const{
640   return 2*a*D*(exp(-a*(x-r0))-exp(-2*a*(x-r0)));
641 }
642
643 ///// potential opertor /////
644 double PotExt_OHbonds::operator()(const double& x) const{
645   if(abs(x) < 8){
```

```
    return 0.5*(Vmorse(R/2+x)+Vmorse(R/2-x) - sqrt(pow(Vmorse(R/2+x)-Vmorse(
    R/2-x),2)+4*DELTA*DELTA));
  }else{
    return 0.0;
  }
}

///// estimataor of zero-point energy /////
double PotExt_OHbonds::e0_estimator(const double& x) const{
  return x/4 * (dVmorse(R/2+x) - dVmorse(R/2-x) - (Vmorse(R/2+x) - Vmorse(R
    /2-x))*(dVmorse(R/2+x) + dVmorse(R/2-x))/sqrt(pow(Vmorse(R/2+x) - Vmorse(
    R/2-x), 2) + 4*DELTA*DELTA));
}



//#########################################################################//
//############ C.2 : CLASS 'Potential_int' METHODS DEFINITIONS ############//
//#########################################################################//

//############################  PotInt_harm  ############################//
///// constructor /////
PotInt_harm::PotInt_harm(const ConfigFile& configFile) :
  Potential_int(),
  k(configFile.get<double>("k")),
  l0(configFile.get<double>("l0"))
  {}

///// potential opertor /////
double PotInt_harm::operator()(const double& x1, const double& x2) const {
  return 0.5*k*pow(abs(x2-x1)-l0,2);
}


//############################  PotInt_LJ  ############################//
///// constructor /////
PotInt_LJ::PotInt_LJ(const ConfigFile& configFile) :
  Potential_int(),
  V0(configFile.get<double>("Vmin")),
  x0(configFile.get<double>("x0")),
  G(configFile.get<double>("G"))
  {}

///// standard Lennard-Jones potential /////
double PotInt_LJ::LJ(const double& r) const {
  if(r>0.35){
    return pow(r,-12)-2*pow(r,-6);
  }else{
    return pow(0.35,-12)-2*pow(0.35,-6);
  }
}

///// potential opertor /////
double PotInt_LJ::operator()(const double& x1, const double& x2) const {
  return V0*LJ(abs(x1-x2)/x0);
}
```

```
703 //////////////////////////////////////////////////////////////////////////////
704 //////////////// C.3 : CLASS 'System' METHODS DEFINITION ////////////////
705 //////////////////////////////////////////////////////////////////////////////
706
707
708 ////////////////////////////  constructor  ////////////////////////////
709 System::System(const ConfigFile& configFile) :
710   N_part(configFile.get<unsigned int>("N_part")),
711   N_slices(configFile.get<unsigned int>("N_slices")),
712   beta(configFile.get<double>("beta")),
713   d_tau(beta/N_slices),
714   mass(N_part,configFile.get<double>("mass")),
715   omega(configFile.get<double>("omega")),
716   table(N_part, vector<double>(N_slices, 0.0)),
717   mm(0), mm_plu(0), mm_min(0), nn(0),
718   dis(0.0), s_old(0.0), s_new(0.0), H(0.0),
719   verif(N_part, vector<int>(N_slices, 0))
720   {
721     for(unsigned int i(0); i<N_part; i++){
722       mass[i]=configFile.get<double>("m"+to_string(i+1));
723     }
724     // choosing the external potential
725     string V_ext(configFile.get<string>("V_ext"));
726     if(V_ext=="null") ptr_Vext = move(unique_ptr<Potential_ext>(new
    PotExt_null()));
727     else if(V_ext=="harmonic") ptr_Vext = move(unique_ptr<Potential_ext>(new
     PotExt_harm(configFile)));
728     else if(V_ext=="double") ptr_Vext = move(unique_ptr<Potential_ext>(new
    PotExt_double(configFile)));
729     else if(V_ext=="square") ptr_Vext = move(unique_ptr<Potential_ext>(new
    PotExt_square(configFile)));
730     else if(V_ext=="sin") ptr_Vext = move(unique_ptr<Potential_ext>(new
    PotExt_sin(configFile)));
731     else if(V_ext=="LJ") ptr_Vext = move(unique_ptr<Potential_ext>(new
    PotExt_LJ(configFile)));
732     else if(V_ext=="OHbonds") ptr_Vext = move(unique_ptr<Potential_ext>(new
    PotExt_OHbonds(configFile)));
733     else{
734       cerr << "Please choose a valid potential." << endl;
735     }
736     // choosing the internal potential
737     string V_int(configFile.get<string>("V_int"));
738     if(V_int=="null") ptr_Vint = move(unique_ptr<Potential_int>(new
    PotInt_null()));
739     else if(V_int=="harmonic") ptr_Vint = move(unique_ptr<Potential_int>(new
     PotInt_harm(configFile)));
740     else if(V_int=="LJ") ptr_Vint = move(unique_ptr<Potential_int>(new
    PotInt_LJ(configFile)));
741     else{
742       cerr << "Please choose a valid potential." << endl;
743     }
744   }
745
746
747 // initialize random paths for each particles
748 void System::initialize(const double& pos_min, const double& pos_max){
749   for(auto& particle : table){
750     for(auto& pos : particle){
751       pos = randomDouble(pos_min, pos_max);
752     }
```

```
753   }
754   H=energy();
755 }
756
757
758 void System::write_potExt(const string& output){
759   string output_pot(output+"_pot.out");
760   ofstream f_pot(output_pot.c_str());
761   f_pot.precision(15);
762   size_t N(10000);
763   double x(0.0);
764   for(size_t i(0); i<N; i++){
765     double xi(-50.0), xf(50.0);
766     x=xi+i*(xf-xi)/(N-1);
767     f_pot << x << " " << (*ptr_Vext)(x) << endl;
768   }
769   f_pot.close();
770 }
771
772
773 // write the paths of all particles in one line
774 ostream& System::write(ostream& output) const{
775   for(const auto& particle : table){
776     for(const auto& pos : particle){
777       output << pos << " ";
778     }
779   }
780   return output;
781 }
782
783
784 double System::kinetic(const int& particle, const int& bead, const int&
     bead_pm, const double& displacement) const{
785   return 0.5*mass[particle]*pow(((table[particle][bead]+displacement)-table[
     particle][bead_pm])/d_tau,2);
786 }
787
788 double System::energy(){
789   double E(0.0);
790   for(size_t part(0); part<N_part; part++){
791     for(size_t bead(0); bead<N_slices; bead++){
792       E+=kinetic(part,bead,(bead+1)%N_slices);
793       E+=(*ptr_Vext)(table[part][bead]);
794       for(size_t part2(part+1); part2<N_part; part2++){
795         E+=(*ptr_Vint)(table[part][bead],table[part2][bead]);
796       }
797     }
798   }
799   return E;
800 }
801
802
803
804 bool System::metropolisAcceptance(){
805   return ( randomDouble(0,1) <= exp(-(0.1*d_tau/hbar) * (s_new - s_old)) );
806 }
807
808
809
810 //################################  moves  ################################//
```

```
811
812
813 bool System::localMove(const double& h){
814   // random integer between 0 and N_slices-1
815   mm = rng()%N_slices;
816   // mm-1 with periodic boundary condition
817   mm_min = (mm + N_slices - 1)%N_slices;
818   // mm+1 with periodic boundary condition
819   mm_plu = (mm + 1)%N_slices;
820   // random integer between 0 and N_part-1
821   nn = rng()%N_part;
822
823   verif[nn][mm]++;
824
825   dis=GenerateDist(h);
826
827   /* as we take the difference of new and old action S_new-S_old, we can //
828   // consider only the part of the action that is affected by the        //
829   // proposed new position                                               */
830   s_old = kinetic(nn,mm,mm_plu) + kinetic(nn,mm,mm_min)
831       + (*ptr_Vext)(table[nn][mm]);
832   s_new = kinetic(nn,mm,mm_plu,dis) + kinetic(nn,mm,mm_min,dis)
833       + (*ptr_Vext)(table[nn][mm]+dis);
834
835   if(N_part>1){
836     for(size_t i(0); i<N_part; i++){
837       if(i!=nn){
838         s_old+=(*ptr_Vint)(table[i][mm],table[nn][mm]);
839         s_new+=(*ptr_Vint)(table[i][mm],table[nn][mm]+dis);
840       }
841     }
842   }
843
844   if(metropolisAcceptance()){ // metropolis acceptance
845     table[nn][mm] += dis;    // update position with new one
846     H += (s_new - s_old);    // update total action
847     return true;
848   }else{
849     return false;
850   }
851 }
852
853
854
855
856 bool System::globalDisplacement(const double& h){
857   // random integer between 0 and N_part-1
858   nn = rng()%N_part;
859
860   dis=GenerateDist(h);
861
862   /* no relative move between the time slices //
863   // --> only the potential action changes    */
864   s_old=0.0;
865   s_new=0.0;
866   for(size_t j(0); j<N_slices; j++){
867     s_old+=(*ptr_Vext)(table[nn][j]);
868     s_new+=(*ptr_Vext)(table[nn][j]+dis);
869     if(N_part>1){
870       for(size_t i(0); i<N_part; i++){
```

```
        if(i!=nn){
          s_old+=(*ptr_Vint)(table[i][j],table[nn][j]);
          s_new+=(*ptr_Vint)(table[i][j],table[nn][j]+dis);
        }
      }
    }
  }

  if(metropolisAcceptance()){ // metropolis acceptance
    for(auto& pos : table[nn]){
      pos+=dis;
    }
    H += (s_new - s_old);
    return true;
  }else{
    return false;
  }
}



bool System::bisection(const double& h, const double& sRel){
  // random integer between 0 and N_slices-1
  mm = rng()%N_slices;
  // mm-1 with periodic boundary condition
  mm_min = (mm + N_slices - 1)%N_slices;
  // random integer between 0 and N_part-1
  nn = rng()%N_part;

  dis=GenerateDist(h);
  size_t l(N_slices*sRel);

  s_old=0.0;
  s_new=0.0;
  int ind_j(0);
  for(size_t j(0); j<l; j++){
    ind_j=(mm+j)%N_slices;
    s_old+=(*ptr_Vext)(table[nn][ind_j]);
    s_new+=(*ptr_Vext)(table[nn][ind_j]+dis);
    if(N_part>1){
      for(size_t i(0); i<N_part; i++){
        if(i!=nn){
          s_old+=(*ptr_Vint)(table[i][ind_j],table[nn][ind_j]);
          s_new+=(*ptr_Vint)(table[i][ind_j],table[nn][ind_j]+dis);
        }
      }
    }
  }
  s_old += kinetic(nn,mm,mm_min)
         + kinetic(nn,(mm+l-1)%N_slices,(mm+l)%N_slices);
  s_new += kinetic(nn,mm,mm_min,dis)
         + kinetic(nn,(mm+l-1)%N_slices,(mm+l)%N_slices,dis);

  if(metropolisAcceptance()){ // metropolis acceptance
    for(size_t i(0); i<l; i++){
      table[nn][(mm+i)%N_slices]+=dis;
    }
    H += (s_new - s_old);
    return true;
```

```
931   }else{
932     return false;
933   }
934 }
935
936
937
938
939 bool System::swap(){
940   if (N_part>1){
941     // random integer between 0 and N_part-1
942     mm_min = rng()%N_part;
943     // another, but different, random integer between 0 and N_part-1
944     mm_plu = (mm_min+1+rng()%(N_part-1))%N_part;
945     // random integer between 0 and N_slices-1 (bead where the swap starts)
946     mm = rng()%N_slices;
947     //length of the swap (nb of slices swapped)
948     nn = rng()%(N_slices-1)+1;
949
950     s_old=0.0;
951     s_new=0.0;
952     int ind_j(0), ind_j_pm(0);
953
954     ind_j=mm;
955     ind_j_pm=(mm+N_slices-1)%N_slices;
956     s_old += kinetic(mm_min,ind_j,ind_j_pm)
957            + kinetic(mm_plu,ind_j,ind_j_pm);
958     s_new += kinetic(mm_min,ind_j,ind_j_pm,table[mm_plu][mm]-table[mm_min][
    mm])
959            + kinetic(mm_plu,ind_j,ind_j_pm,table[mm_min][mm]-table[mm_plu][
    mm]);
960
961     for(size_t j(0); j<nn; j++){
962       ind_j=(mm+j)%N_slices;
963       ind_j_pm=(ind_j+N_slices-1)%N_slices;
964       if(mass[mm_min]!=mass[mm_plu]){
965         for(size_t i(0); i<N_part; i++){
966           if(i!=mm_min and i!=mm_plu){
967             // change for particle(mm_min)
968             s_old+=(*ptr_Vint)(table[i][ind_j],table[mm_min][ind_j]);
969             s_new+=(*ptr_Vint)(table[i][ind_j],table[mm_plu][ind_j]);
970             // change for particle(mm_plu)
971             s_old+=(*ptr_Vint)(table[i][ind_j],table[mm_plu][ind_j]);
972             s_new+=(*ptr_Vint)(table[i][ind_j],table[mm_min][ind_j]);
973           }
974         }
975       }
976       /* in swapped part of paths :                          //
977       // K1_new = m1*(K2_old/m2), K2_new=m2/m1*K1_old        */
978       if(j){
979         s_old += kinetic(mm_min,ind_j,ind_j_pm)
980                + kinetic(mm_plu,ind_j,ind_j_pm);
981         s_new += mass[mm_min]/mass[mm_plu]*kinetic(mm_plu,ind_j,ind_j_pm)
982                + mass[mm_plu]/mass[mm_min]*kinetic(mm_min,ind_j,ind_j_pm);
983       }
984     }
985     ind_j=(mm+nn-1)%N_slices;
986     ind_j_pm=(mm+nn)%N_slices;
987     s_old += kinetic(mm_min,ind_j,ind_j_pm)
988            + kinetic(mm_plu,ind_j,ind_j_pm);
```

```
    s_new += kinetic(mm_min,ind_j,ind_j_pm,table[mm_plu][ind_j]-table[mm_min][ind_j])
           + kinetic(mm_plu,ind_j,ind_j_pm,table[mm_min][ind_j]-table[mm_plu][ind_j]);


    if(metropolisAcceptance()){ // metropolis acceptance
      double tmp(0.0);
      for(size_t j(0); j<nn; j++){
        ind_j=(mm+j)%N_slices;
        tmp=table[mm_min][ind_j];
        table[mm_min][ind_j]=table[mm_plu][ind_j];
        table[mm_plu][ind_j]=tmp;
      }
      H += (s_new - s_old);
      return true;
    }
  }
  return false;
}




bool System::inverse(){
  nn = rng()%N_part; // random integer between 0 and N_part-1

  // no relative move between the time slices
  // --> only the potential action changes
  s_old=0.0;
  s_new=0.0;
  for(size_t j(0); j<N_slices; j++){
    s_old+=(*ptr_Vext)( table[nn][j]);
    s_new+=(*ptr_Vext)(-table[nn][j]);
    if(N_part>1){
      for(size_t i(0); i<N_part; i++){
        if(i!=nn){
          s_old+=(*ptr_Vint)(table[i][j], table[nn][j]);
          s_new+=(*ptr_Vint)(table[i][j],-table[nn][j]);
        }
      }
    }
  }

  if(metropolisAcceptance()){ // metropolis acceptance
    for(auto& pos : table[nn]){
      pos*=-1;
    }
    H += (s_new - s_old);
    return true;
  }else{
    return false;
  }
}




bool System::symmetryCM(){
  nn = rng()%N_part; // random integer between 0 and N_part-1
```

```
  dis= 0;
  for(const auto& pos : table[nn]){
    dis+=pos;
  }
  dis*=-2.0/table[nn].size();

  // no relative move between the time slices
  // --> only the potential action changes
  s_old=0.0;
  s_new=0.0;
  for(size_t j(0); j<N_slices; j++){
    s_old+=(*ptr_Vext)(table[nn][j]);
    s_new+=(*ptr_Vext)(table[nn][j]+dis);
    if(N_part>1){
      for(size_t i(0); i<N_part; i++){
        if(i!=nn){
          s_old+=(*ptr_Vint)(table[i][j],table[nn][j]);
          s_new+=(*ptr_Vint)(table[i][j],table[nn][j]+dis);
        }
      }
    }
  }

  if(metropolisAcceptance()){ // metropolis acceptance
    for(auto& pos : table[nn]){
      pos+=dis;
    }
    H += (s_new - s_old);
    return true;
  }else{
    return false;
  }
}




void System::measure_energy(double V0, double x0){
  double temp_energy_H(0), temp_energy_ETH(0);

  double R(V0); //For H-bond, V0 is R
  double D(83.402), a(2.2), r0(0.96), delta1(0.4*D);
  double b(2.2), R1(2*r0+1/a), DELTA(delta1*exp(-b*(R-R1)));

  temp_energy_ETH += (*ptr_Vext)(table[0][0])
                   + (*ptr_Vext).e0_estimator(table[0][0]);

  for(size_t i(1); i < table[0].size(); i++){
    temp_energy_ETH += (*ptr_Vext)(table[0][i])
                     + (*ptr_Vext).e0_estimator(table[0][i]);
    temp_energy_H += mass[0]/2 * pow((table[0][i] - table[0][i-1])/d_tau, 2)
                   + (*ptr_Vext)(table[0][i]);
  }
  temp_energy_H += mass[0]/2 * pow((table[0][0] - table[0][N_slices-1])/
   d_tau, 2)
                 + (*ptr_Vext)(table[0][0]);

  energies_psi.push_back(temp_energy_ETH/N_slices);
  energies_h.push_back(temp_energy_H/N_slices);
}
```

```
1106
1107
1108
1109
1110 void System::average_energy(){
1111   ofstream fichier_output;
1112   fichier_output.open("e0.out");
1113   fichier_output.precision(15);
1114
1115   double temp_energy(0), temp_error(0);
1116
1117   cout << "Finally, with d_tau = " << d_tau << endl;
1118
1119   //PSI energies
1120   for(size_t i(0); i < energies_psi.size(); i++){
1121     temp_energy += energies_psi[i];
1122     temp_error += pow(energies_psi[i], 2);
1123   }
1124   temp_energy = temp_energy/energies_psi.size();
1125   temp_error = sqrt((temp_error/energies_psi.size() - pow(temp_energy, 2))
1126                     /energies_psi.size());
1127
1128   cout << "PSI: " << temp_energy << " +- " << temp_error << endl;
1129   fichier_output << "PSI: " << temp_energy << " +- " << temp_error << endl;
1130
1131   //H energies
1132   temp_energy = 0;
1133   temp_error = 0;
1134
1135   for(size_t i(0); i < energies_h.size(); i++){
1136     temp_energy += energies_h[i];
1137     temp_error += pow(energies_h[i], 2);
1138   }
1139   temp_energy = temp_energy/energies_h.size();
1140   temp_error = sqrt((temp_error/energies_h.size() - pow(temp_energy, 2))
1141                     /energies_h.size());
1142
1143   cout << "H:   " << temp_energy << " +- " << temp_error << endl;
1144   fichier_output << "H:   " << temp_energy << " +- " << temp_error << endl;
1145
1146   fichier_output.close();
1147 }
1148
1149
1150
1151
1152 ostream& operator<<(ostream& output, const System& s){
1153   return s.write(output);
1154 }
1155
1156
1157
1158 //////////////////////////////////////////////////////////////////////////////
1159 //////////////////// C.4 : FUNCTION DEFINITIONS ///////////////////////////
1160 //////////////////////////////////////////////////////////////////////////////
1161
1162 // Generate a random (uniform) double between 'min' and 'max'
1163 double randomDouble(const double& min, const double& max,
1164                     const bool& closed){
1165   if(closed) return (min + (max-min) * (double)rng()/rng.max());
```

```
  else return (min + (max-min) * ((double)rng()+0.5)/(rng.max()+1.0));
}

// Generate a random double from a normal Cauchy distribution
double CauchyDistribution(){
  return tan(M_PI*(randomDouble(-0.5,0.5,false)));
}

// Generate a random double from one of the implemeneted distributions
double GenerateDist(const double& h){
  if(rng()%2){
    return h * randomDouble(-1.0,1.0); // proposed displacement
  }else{
    return h * CauchyDistribution(); // proposed displacement
  }
}
```