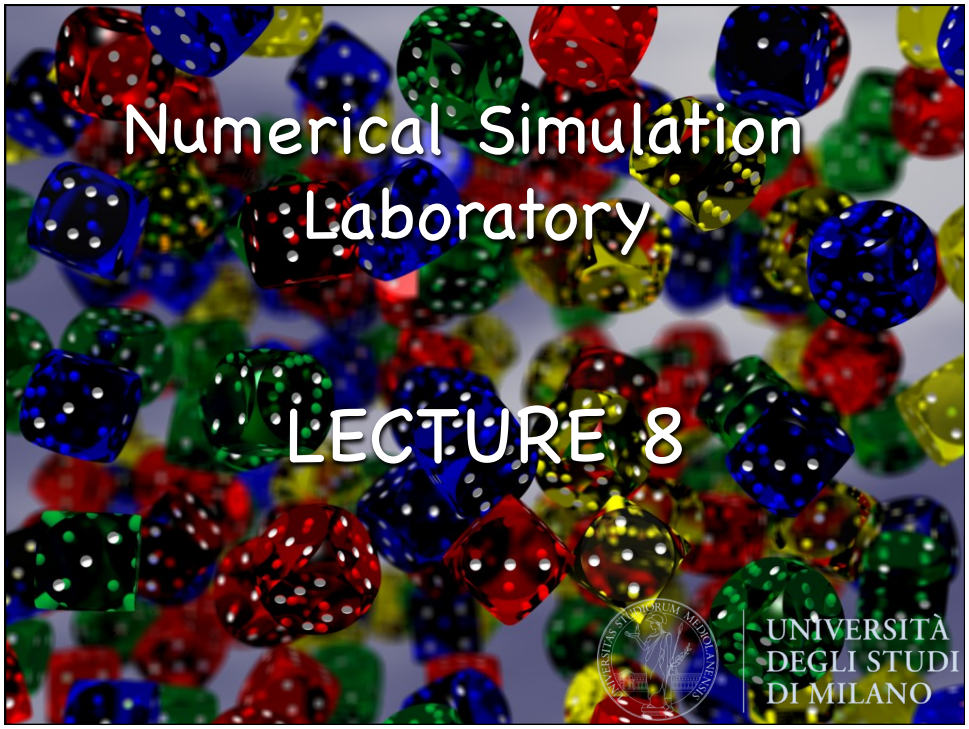
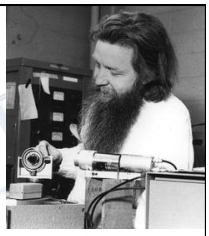


Lecture 8 - outline

- Variational Monte Carlo
- Quantum statistical mechanics
- Density matrices and density operator
- Quantum statistical ensembles
- Density matrix in coordinate representation
- Finite temperature Path Integral Monte Carlo (PIMC)
- Imaginary time evolution and projective Path Integral methods

Quantum Monte Carlo: Variational MC



- Monte Carlo simulation of quantum many-body systems dates from McMillan's observation (1965) that the integrals required to evaluate expectations, including the total energy, can be carried out by using the Metropolis algorithm
- The variational method has proved to be a very useful way of computing (approximate) ground-state properties of many-body systems.
- Conceptually it is quite simple; the variational principle tells us that for any function ψ_T , the variational energy E_T , defined as

$$E_T = \frac{\langle \psi_T | \hat{H} | \psi_T \rangle}{\langle \psi_T | \psi_T \rangle} = \frac{\int d\vec{r}_1 \cdots d\vec{r}_N \psi_T^*(\vec{r}_1 \cdots \vec{r}_N) \hat{H} \psi_T(\vec{r}_1 \cdots \vec{r}_N)}{\int d\vec{r}_1 \cdots d\vec{r}_N |\psi_T(\vec{r}_1 \cdots \vec{r}_N)|^2} \geq E_0 = \frac{\langle \psi_0 | \hat{H} | \psi_0 \rangle}{\langle \psi_0 | \psi_0 \rangle}$$

  will be a minimum when ψ_T is the exact ground-state solution (ψ_0) of the Schrödinger equation, being H the Hamiltonian of the system
- The method then consists in constructing a family of functions $\psi_T(\mathbf{a})$ and optimizing the vector parameter ($\mathbf{a}$) so that the previous energy functional is minimized for $\mathbf{a}=\mathbf{a}^*$

3

- The variational energy is a rigorous upper bound to the ground state energy and if the family of functions is well chosen, $\psi_T(\mathbf{a}^*)$ will be a good approximation to ψ_0 . Thus one can use $\psi_T(\mathbf{a}^*)$ to study the ground state properties of the system
- The task now is to evaluate the multidimensional integrals to get expectation values, in particular to calculate the variational energy E_T which should be minimized
- When the Metropolis Monte Carlo method is used to compute E_T and other expectation values, the method is called Variational Monte Carlo (VMC):

$$E_T = \frac{\int d\vec{r}_1 \cdots d\vec{r}_N \psi_T^*(\vec{r}_1 \cdots \vec{r}_N) \hat{H} \psi_T(\vec{r}_1 \cdots \vec{r}_N)}{\int d\vec{r}_1 \cdots d\vec{r}_N |\psi_T(\vec{r}_1 \cdots \vec{r}_N)|^2} = \frac{\int d\vec{r}_1 \cdots d\vec{r}_N \psi_T^*(\vec{r}_1 \cdots \vec{r}_N) \psi_T(\vec{r}_1 \cdots \vec{r}_N) \frac{\hat{H}\psi_T(\vec{r}_1 \cdots \vec{r}_N)}{\psi_T(\vec{r}_1 \cdots \vec{r}_N)}}{\int d\vec{r}_1 \cdots d\vec{r}_N |\psi_T(\vec{r}_1 \cdots \vec{r}_N)|^2} =$$

$$= \int d\vec{r}_1 \cdots d\vec{r}_N \frac{|\psi_T(\vec{r}_1 \cdots \vec{r}_N)|^2}{\int d\vec{r}_1 \cdots d\vec{r}_N |\psi_T(\vec{r}_1 \cdots \vec{r}_N)|^2} \frac{\hat{H}\psi_T(\vec{r}_1 \cdots \vec{r}_N)}{\psi_T(\vec{r}_1 \cdots \vec{r}_N)} = \int d\vec{r}_1 \cdots d\vec{r}_N P(\vec{r}_1 \cdots \vec{r}_N) E_{loc}(\vec{r}_1 \cdots \vec{r}_N)$$

- Given a trial wave-function ψ_T (and given a model Hamiltonian) the evaluation of E_T is "exact", i.e., it gives the value of E_T within the statistical uncertainty of the Monte Carlo calculation

4

Quantum statistical averages

- Alternatively, consider the situation of quantum statistical mechanics, where the observer has an incomplete knowledge about the preparation of a quantum system, one is no longer able to tell in which specific microstate $|\psi_i\rangle$ the system is prepared and can give only a probability ρ_i, of finding the system prepared in a particular microstate $|\psi_i\rangle$.
- We note that the states $|\psi_i\rangle$ are not necessarily orthogonal.
- Thus another average adds to the usual intrinsic quantum mechanical average.
- If one now performs a measurement of the observable A, one can measure only the average of the quantum mechanical expectation values, weighted by the probabilities ρ_i

$$\left\langle \hat{A} \right\rangle = \sum_i \rho_i \left\langle \psi_i \left| \hat{A} \right| \psi_i \right\rangle$$

The density matrix: $\rho_{kk'}$

- However, the previous expression is not the most general expression for a quantum statistical average.
- In fact, expanding the states $|\psi_i\rangle$ with respect to a complete basis set $|\phi_k\rangle$ we obtain

$$\left| \psi_i \right\rangle = \sum_k a_k^i \left| \phi_k \right\rangle$$

$$\left\langle \hat{A} \right\rangle = \sum_i \rho_i \sum_{k,k'} a_k^{i*} a_{k'}^i \left\langle \phi_k \left| \hat{A} \right| \phi_{k'} \right\rangle = \sum_{k,k'} \left(\sum_i \rho_i a_k^{i*} a_{k'}^i \right) \left\langle \phi_k \left| \hat{A} \right| \phi_{k'} \right\rangle$$

$$\Rightarrow \quad \left\langle \hat{A} \right\rangle = \sum_{k,k'} \rho_{kk'} \left\langle \phi_k \left| \hat{A} \right| \phi_{k'} \right\rangle$$

- The most general expression involves the so called density matrix: $\rho_{kk'}$

The density operator

- One can now interpret the numbers $\rho_{kk'} := \sum_i \rho_i a_k^{i*} a_{k'}^i$

  as the matrix elements of an operator, $\hat{\rho}$, in the basis of the $|\phi_k\rangle$,

$$\left\langle \hat{A} \right\rangle = \sum_{k,k'} \rho_{kk'} \langle \phi_k | \hat{A} | \phi_{k'} \rangle = \sum_{k,k'} \langle \phi_{k'} | \hat{\rho} | \phi_k \rangle \langle \phi_k | \hat{A} | \phi_{k'} \rangle = \sum_{k'} \langle \phi_{k'} | \hat{\rho} \hat{A} | \phi_{k'} \rangle = \mathrm{Tr}\left(\hat{\rho} \hat{A} \right)$$

- The trace, Tr(•) , is just the sum of the expectation values of the operator (diagonal matrix elements) in an arbitrary basis:

$$\mathrm{Tr}\left(\hat{O} \right) = \sum_{k'} \langle \phi_{k'} | \hat{O} | \phi_{k'} \rangle$$

- The value of the trace is independent of the basis chosen, since given $|\phi_k\rangle = \sum_i S_{ki} |\varphi_i\rangle$ (Unitarity ($S^*S = SS^* = I$) of the basis transformation S is required) it follows that

$$\mathrm{Tr}(\hat{O}) = \sum_k \langle \phi_k | \hat{O} | \phi_k \rangle = \sum_{k,j,j} S_{ki}^* S_{kj} \langle \varphi_i | \hat{O} | \varphi_j \rangle = \sum_{i,j} \delta_{ij} \langle \varphi_i | \hat{O} | \varphi_j \rangle = \sum_i \langle \varphi_i | \hat{O} | \varphi_i \rangle$$

Statistical and quantum averages

- Assume the state vector $|\Psi\rangle$ of the system exactly known, and represented by the following linear combination $|\Psi\rangle = \sum_i a_i |\psi_i\rangle$. a_i are complex numbers.

- The quantum mechanical expectation value of an observable now reads

$$\langle \Psi | \hat{A} | \Psi \rangle = \sum_{i,j} a_i^* a_j \langle \psi_i | \hat{A} | \psi_j \rangle$$

- However, if one forms the purely statistical average in a mixed state, where the $|\psi_i\rangle$ just appear with the probabilities ρ_i,

$$\left\langle \hat{A} \right\rangle = \sum_i \rho_i \langle \psi_i | \hat{A} | \psi_i \rangle$$

- Mixed terms are not present in the latter case. Even if one prepares a system in such a way that in the mixed state the ρ_i correspond exactly to the $|a_i|^2$, nevertheless in general one does not obtain the same average, since information about the phases of a_i is not contained in the statistical average. The statistical mixture of the states $|\psi_i\rangle$, with weights ρ_i , should not be confused with the quantum linear superposition

Pure and mixed states

- If a quantum mechanical system is in a certain microstate, described by a state vector $|\psi_i\rangle$ we say it is in a pure state.
- If on the other hand the system has been prepared as a statistical mixture of microstates $|\psi_i\rangle$ with probabilities ρ_i we are dealing with a mixed state
- The mixed, as well as pure, states can be completely described by the matrix elements of a density operator; i.e., all quantum mechanical and statistical quantum averages of arbitrary observables can be calculated if the density matrix is known
- To this end, we first denote the density operator in an arbitrary basis $|\phi_k\rangle$ in Dirac's notation:

$$\rho_{kk'} := \langle \phi_{k'} | \hat{\rho} | \phi_k \rangle \quad \Rightarrow \quad \hat{\rho} = \sum_{k,k'} |\phi_{k'}\rangle \rho_{kk'} \langle \phi_k |$$

- If we prepare our system in a pure state, i.e. in such a way that it may be in only one of the states $|\psi_i\rangle$, for example i=l, with the probabilities $\rho_{ll}=1$, $\rho_{ii}=0$ $\forall i \neq l$ and $\rho_{ik}= 0$ for $i \neq k$, the density operator is

$$\hat{\rho} = |\psi_l\rangle\langle\psi_l|$$

- For mixed states, several terms contribute to the previous sum; i.e., one cannot exactly determine the state that the system assumes.
- If the system is in a pure state then statistical quantum averages correspond to ordinary quantum averages:

$$\langle \hat{A} \rangle_{stat} = \mathrm{Tr}\left(\hat{\rho}^{pure} \hat{A}\right) = \sum_k \langle \phi_k | \psi_l \rangle \langle \psi_l | \hat{A} | \phi_k \rangle =$$

$$= \sum_k \langle \psi_l | \hat{A} | \phi_k \rangle \langle \phi_k | \psi_l \rangle = \langle \psi_l | \hat{A} | \psi_l \rangle = \langle \hat{A} \rangle_{quantum}$$

Quantum statistics: canonical ensemble

- Consider the case of N particles in a box of volume V; one can imagine to perform the preparation of this system fixing the temperature $T=1/\beta$ by allowing the exchange of energy with an heat-bath. In this case we know that the probability to observe a state with energy E_n is given by the Boltzmann weight:

$$\rho_n = e^{-\beta E_n} \Big/ \sum_n e^{-\beta E_n}$$

- The system will have a probability ρ_n to be found in the microstate which corresponds to the eigenstate $|\phi_n\rangle$ of the Hamiltonian, H, and the measurement of an observable A will gives

$$\left\langle \hat{A} \right\rangle = \sum_n \rho_n \left\langle \phi_n \left| \hat{A} \right| \phi_n \right\rangle = \sum_n \frac{e^{-\beta E_n}}{\sum_m e^{-\beta E_m}} \left\langle \phi_n \left| \hat{A} \right| \phi_n \right\rangle = \cdots$$

$$\cdots = \sum_n \left\langle \phi_n \left| \frac{e^{-\beta \hat{H}}}{\sum_m \left\langle \phi_m \left| e^{-\beta \hat{H}} \right| \phi_m \right\rangle} \hat{A} \right| \phi_n \right\rangle = \sum_n \left\langle \phi_n \left| \frac{e^{-\beta \hat{H}}}{Tr(e^{-\beta \hat{H}})} \hat{A} \right| \phi_n \right\rangle = Tr\left(\hat{\rho} \hat{A}\right)$$

- $\hat{\rho}$ is the canonical density operator; the statistical mixture which corresponds to this preparation is called canonical ensemble

$$\hat{\rho} = \frac{e^{-\beta \hat{H}}}{Z} \quad ; \quad Z = Tr\left(e^{-\beta \hat{H}}\right)$$

Density matrix in the coordinate representation

- As via any other basis, it is possible to describe a statistical system in the coordinate representation; we can write the density matrix as

$$\rho(x',x) \overset{def.}{=} \langle x'|\hat{\rho}|x\rangle = \sum_i p_i \langle x'|\phi_i\rangle\langle\phi_i|x\rangle = \sum_i p_i \phi_i^*(x)\phi_i(x')$$

Wave functions of the states $|\phi_i\rangle$

which for a pure state $|\phi_i\rangle$ becomes

$$\rho(x',x) = \phi_i^*(x)\phi_i(x')$$

- In the coordinate representation we write

$$\left\langle \hat{A} \right\rangle = \mathrm{Tr}(\hat{\rho}\hat{A}) = \int dx \left\langle x \left| \hat{\rho}\hat{A} \right| x \right\rangle = \int dx' \left| x' \right\rangle \left\langle x' \right| = 1$$

$$= \int dx\, dx' \langle x|\hat{\rho}|x'\rangle\langle x'|\hat{A}|x\rangle = \int dx\, dx' \rho(x,x')\langle x'|\hat{A}|x\rangle$$

- In case the observable is diagonal in the coord. representation: $\hat{A} = f(\hat{x})$

$$\left\langle f(\hat{x}) \right\rangle = \int dx\, dx' \rho(x,x') \left\langle x' \left| f(\hat{x}) \right| x \right\rangle = \int dx\, dx' \rho(x,x') f(x) \left\langle x' \left| x \right\rangle \right. =$$

$$= \int dx\, dx' \rho(x,x') f(x) \delta(x-x') = \int dx\, f(x)\, \rho(x,x)$$

Canonical density matrix for one free particle

- We look for the canonical density matrix in the momentum representation for a free particle in a box of volume V=L^3 and periodic boundary conditions. The Hamiltonian of a free particle is H = p^2/2m, and the energy eigenfunctions are plane waves

$$\hat{H}|\phi_k\rangle = E_k|\phi_k\rangle \qquad E_k = \frac{\hbar^2\vec{k}^2}{2m} \qquad \phi_{\vec{k}}(\vec{r}) = \frac{1}{\sqrt{V}}e^{i\vec{k}\cdot\vec{r}} \qquad \vec{k} = \frac{2\pi}{L}(n_x, n_y, n_z)$$
$$n_i = 0, \pm 1, \pm 2, \cdots$$

- The energy eigenvalues are thus discrete, but their mutual separation for macroscopic volumes is so small that one may again return to continuous momenta and energies. The canonical partition function becomes

$$Q_1(T,V) = \mathrm{Tr}(e^{-\beta\hat{H}}) = \sum_{\vec{k}}\langle\phi_{\vec{k}}|e^{-\beta\hat{H}}|\phi_{\vec{k}}\rangle = \sum_{\vec{k}} e^{-\frac{\beta\hbar^2}{2m}\vec{k}^2}$$

- Since, as already mentioned, the eigenvalues k lie very close together in a large volume, the calculation of Q can be performed without large error by replacing the sum by an integral (see supplementary material).

$$Q_1(T,V) = \frac{V}{(2\pi)^3}\int d^3\vec{k}\, e^{-\frac{\beta\hbar^2}{2m}\vec{k}^2} = \frac{V}{(2\pi)^3}\left(\frac{2m\pi}{\beta\hbar^2}\right)^{\frac{3}{2}} = \frac{V}{\lambda_T^3}$$

13

- The matrix elements of the density operator thus become 14

$$\langle\phi_{\vec{k}'}|\hat{\rho}|\phi_{\vec{k}}\rangle = \frac{\lambda_T^3}{V}e^{-\frac{\beta\hbar^2}{2m}\vec{k}^2}\delta_{\vec{k}'\vec{k}}$$

Thus, the density matrix is diagonal, and the matrix elements have the same form as the classical momentum distribution

- We look for the canonical density matrix in the coordinate representation for a free particle in a box of volume V = L^3 and periodic boundary conditions

$$\langle\vec{r}'|\hat{\rho}|\vec{r}\rangle = \sum_{\vec{k}'\vec{k}}\langle\vec{r}'|\phi_{\vec{k}'}\rangle\langle\phi_{\vec{k}'}|\hat{\rho}|\phi_{\vec{k}}\rangle\langle\phi_{\vec{k}}|\vec{r}\rangle = \sum_{\vec{k}'\vec{k}}\phi_{\vec{k}'}(\vec{r}')\left(\frac{\lambda_T^3}{V}e^{-\frac{\beta\hbar^2}{2m}\vec{k}^2}\delta_{\vec{k}'\vec{k}}\right)\phi_{\vec{k}}^*(\vec{r}) =$$

$$= \frac{\lambda_T^3}{V^2}\frac{V}{(2\pi)^3}\int d^3\vec{k}\, e^{-\frac{\beta\hbar^2}{2m}\vec{k}^2 + i\vec{k}\cdot(\vec{r}'-\vec{r})} = \frac{\lambda_T^3}{V(2\pi)^3}\left(\frac{2m\pi}{\beta\hbar^2}\right)^{\frac{3}{2}} e^{-\frac{m}{2\beta\hbar^2}(\vec{r}'-\vec{r})^2} = \frac{1}{V}e^{-\frac{\pi}{\lambda_T^2}(\vec{r}'-\vec{r})^2}$$

- Hence, in the coordinate representation the density matrix is no longer a diagonal matrix, but a Gaussian function.
- The diagonal elements can be interpreted as the density distribution in coordinate space. The non-diagonal elements can be interpreted as the transition probability to move from a position r to a new one r'. Note that a free particle, statistically, performs a Brownian motion!

The Path integral Monte Carlo method

- One of the Feynman's early successes (1953): mapping with path integrals of a quantum system onto a classical model of special interacting ring "polymers"
- Beside providing a useful physical picture, the path integral approach translates directly into a powerful computational tool (Ceperley-Pollock, 1984/87)
- Consider now the calculation of quantum averages at finite temperature; such (mixed) expectation values (here expressed in the canonical ensemble) are obtained via the density matrix

$$\left\langle \hat{O} \right\rangle = \mathrm{Tr}(\hat{\rho}\hat{O}) \quad \text{with} \quad \hat{\rho} = e^{-\beta \hat{H}} / \mathrm{Tr}(e^{-\beta \hat{H}})$$

- The traces can be carried out in any complete basis, we will use the coordinate representation. Moreover, we will discuss the method for the simpler case of one single quantum particle in an external potential.

- For a diagonal operator in the coordinate representation we have:

$$\left\langle \hat{O} \right\rangle = \mathrm{Tr}(\hat{\rho}\hat{O}) = \mathrm{Tr}(\hat{\rho})^{-1} \int d\vec{r}\, \rho(\vec{r},\vec{r},\beta) O(\vec{r})$$

with $\rho(\vec{r},\vec{r},\beta) = \left\langle \vec{r} \left| e^{-\beta \hat{H}} \right| \vec{r} \right\rangle$

- We can exploit the exact Trotter decomposition of the density matrix:

$$e^{-\tau \hat{H}} = \left(e^{-\frac{\tau}{M}\hat{H}} \right)^M = \lim_{M\to\infty} \left(e^{-\frac{\tau}{M}\hat{T}} e^{-\frac{\tau}{M}\hat{V}} \right)^M$$

$$\sum_n \left| \phi_n \right\rangle \left\langle \phi_n \right| = 1 \qquad \int d\vec{r}\, \left| \vec{r} \right\rangle \left\langle \vec{r} \right| = 1$$

⇒ Exact Path Integral representation:

M–1 times

$$\rho(\vec{r},\vec{r},\beta) = \left\langle \vec{r} \left| e^{-\frac{\beta}{M}\hat{H}} \times e^{-\frac{\beta}{M}\hat{H}} \times \cdots \times e^{-\frac{\beta}{M}\hat{H}} \right| \vec{r} \right\rangle = \cdots$$

$$= \int d\vec{r}_2 \cdots \int d\vec{r}_M \left\langle \vec{r} \left| e^{-\frac{\beta}{M}\hat{H}} \right| \vec{r}_2 \right\rangle \left\langle \vec{r}_2 \left| e^{-\frac{\beta}{M}\hat{H}} \right| \vec{r}_3 \right\rangle \cdots \left\langle \vec{r}_M \left| e^{-\frac{\beta}{M}\hat{H}} \right| \vec{r} \right\rangle$$

$\frac{\beta}{M} = \frac{1}{k_B (MT)}$

High temperature

17

- Let's focus on one of the M matrix elements: $\left\langle \vec{r} \left| e^{-\frac{\beta}{M}\hat{H}} \right| \vec{r}' \right\rangle$

- Approximation: the exact density matrix is not known, but we can resort to some high temperature approximation with $\beta/M \ll 1$; for example the "primitive" approximation:

$$\left\langle \vec{r} \left| e^{-\frac{\beta}{M}(\hat{T}+\hat{V})} \right| \vec{r}' \right\rangle \cong \left\langle \vec{r} \left| e^{-\frac{\beta \hat{V}}{M\,2}} e^{-\frac{\beta}{M}\hat{T}} e^{-\frac{\beta \hat{V}}{M\,2}} \right| \vec{r}' \right\rangle = \cdots$$

$$= e^{-\frac{\beta}{M}\frac{V(\vec{r})}{2}} \left\langle \vec{r} \left| e^{-\frac{\beta}{M}\hat{T}} \right| \vec{r}' \right\rangle e^{-\frac{\beta}{M}\frac{V(\vec{r}')}{2}} =$$

Free particle density matrix

Note that the kinetic part of the density matrix can be read as a Boltzmann weight with harmonic interaction (spring) among r and r'

$$\left\langle \vec{r} \left| e^{-\frac{\beta}{M}(\hat{T}+\hat{V})} \right| \vec{r}' \right\rangle \cong e^{-\frac{\beta}{M}\frac{V(\vec{r})}{2}} \frac{\exp\left(-\left|\vec{r}-\vec{r}'\right|^2 M / 4\lambda\beta\right)}{\left(4\pi\lambda\,\beta/M\right)^{\frac{3}{2}}} e^{-\frac{\beta}{M}\frac{V(\vec{r}')}{2}}$$

"primitive" approximation

$$\lambda = \frac{\hbar^2}{2m}$$

Accurate up to terms of order $\delta\tau^2$: one assumes $\frac{\delta\tau^2}{2}\left[\hat{T},\hat{V}\right] = 0$

18

- Let me call $\rho_p(r,r',\beta/M)$ the approximate ("primitive") density matrix:

$$\rho_p(\vec{r},\vec{r}',\beta/M) = e^{-\frac{\beta}{M}\frac{V(\vec{r})}{2}} \frac{\exp\left(-\left|\vec{r}-\vec{r}'\right|^2 M / 4\lambda\beta\right)}{\left(4\pi\lambda\,\beta/M\right)^{\frac{3}{2}}} e^{-\frac{\beta}{M}\frac{V(\vec{r}')}{2}}$$

- The approximated full density matrix is given by the following path integral representation:

$$\rho(\vec{r}_1,\vec{r}_1,\beta) \cong \int d\vec{r}_2 \cdots \int d\vec{r}_M \; \rho_p(\vec{r}_1,\vec{r}_2,\beta/M)\rho_p(\vec{r}_2,\vec{r}_3,\beta/M)\cdots\rho_p(\vec{r}_M,\vec{r}_1,\beta/M)$$

and the thermal expectation value can be expressed by

$$\left\langle \hat{O} \right\rangle = \mathrm{Tr}(\hat{\rho}\hat{O}) = \mathrm{Tr}(\hat{\rho})^{-1} \int d\vec{r}_1 \, \rho(\vec{r}_1,\vec{r}_1,\beta) O(\vec{r}_1) =$$

$$\cong \int d\vec{r}_1 \int d\vec{r}_2 \cdots \int d\vec{r}_M \frac{\rho_p(\vec{r}_1,\vec{r}_2,\beta/M)\rho_p(\vec{r}_2,\vec{r}_3,\beta/M)\cdots\rho_p(\vec{r}_M,\vec{r}_1,\beta/M)}{\mathrm{Tr}(\hat{\rho})} O(\vec{r}_1) =$$

$$= \int d\vec{r}_1 \int d\vec{r}_2 \cdots \int d\vec{r}_M \; p(\vec{r}_1,\vec{r}_2,\cdots,\vec{r}_M)\, O(\vec{r}_1)$$

with $$p(\vec{r}_1,\vec{r}_2,\cdots,\vec{r}_M) = \frac{\rho_p(\vec{r}_1,\vec{r}_2,\beta/M)\rho_p(\vec{r}_2,\vec{r}_3,\beta/M)\cdots\rho_p(\vec{r}_M,\vec{r}_1,\beta/M)}{\mathrm{Tr}(\hat{\rho})}$$

as probability density. We can compute $\langle O \rangle$ with Monte Carlo integration

- Two very important observations:

1. One can choose M>>1 such that the bias induced by the ("primitive") approximation gives deviation from the exact results which are below the statistical uncertainties of the Monte Carlo integration: the method is "exact"!
2. Interpreting $p(R_1,R_2,\ldots,R_M)$ as a classical Boltzmann weight, it turns out that the quantum system is isomorphic to a classical system of special interacting ring polymers, one for each quantum particle:

$$p(R_1,R_2,\cdots,R_M)=\frac{\rho_p(R_1,R_2,\beta/M)\rho_p(R_2,R_3,\beta/M)\cdots\rho_p(R_M,R_1,\beta/M)}{Tr(\hat{\rho})}$$

in fact for N quantum particles it easy to show that:

$$\rho_p(R,R',\beta/M)=e^{-\frac{\beta}{M}\frac{V(R)}{2}}\prod_{i=1}^{N}\frac{e^{-|\vec{r}_i-\vec{r}_i'|^2M/4\lambda\beta}}{(4\pi\lambda\beta/M)^{\frac{3}{2}}}e^{-\frac{\beta}{M}\frac{V(R')}{2}}$$

$$R_i=\left\{\vec{r}_1^{\,i},\cdots,\vec{r}_N^{\,i}\right\}$$

Quantum System of N interacting particles

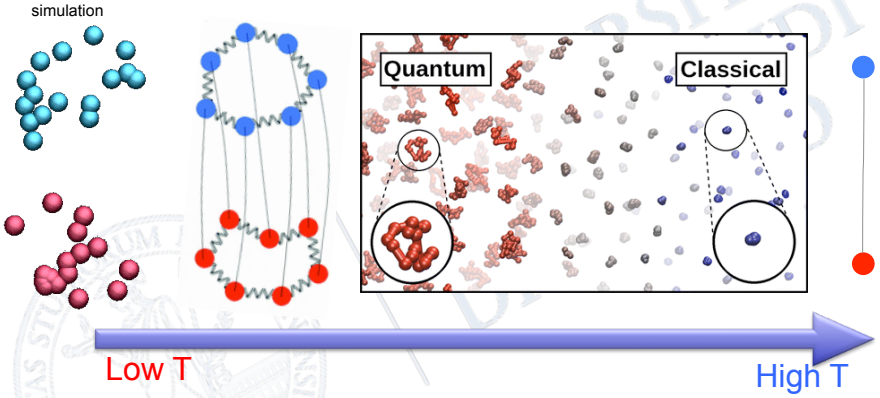
From 1 to M

←Isomorphic→

Classical System of N special interacting polimers

Path Integral representation of quantum particles

Path Integral Molecular Dynamics simulation



- Typically, quantum effects disappears at high temperatures where the kinetic spring-like quantum correlations become strong and the polymers coalesce into points:

$$\rho_p(R,R',\beta/M)=e^{-\frac{\beta}{M}\frac{V(R)}{2}}\prod_{i=1}^{N}\frac{e^{-|\vec{r}_i-\vec{r}_i'|^2M/4\lambda\beta}}{(4\pi\lambda\beta/M)^{\frac{3}{2}}}e^{-\frac{\beta}{M}\frac{V(R')}{2}}$$

"Exact" Quantum Monte Carlo methods: T=0 K

- Let's try to find a way to use Monte Carlo in order to compute the exact expectation value of an operator on the unknown ground state ψ_0

$$\langle \hat{O} \rangle = \frac{\langle \psi_0 | \hat{O} | \psi_0 \rangle}{\langle \psi_0 | \psi_0 \rangle} = \frac{\int d\vec{r}\ \psi_0^*(\vec{r}) \hat{O} \psi_0(\vec{r})}{\int d\vec{r} \left| \psi_0(\vec{r}) \right|^2}$$

- It is evidently not possible to obtain the exact ψ_0 ; however, we can reach a extremely accurate approximation, such that the calculation of the previous expectation value with this state, using Monte Carlo methods, differ from the (unknown) exact value by a quantity which is smaller than the statistical error of the calculation; the method is thus essentially "exact"
- All the "zero temperature" (i.e. related to the ground state) "exact" Quantum Monte Carlo (QMC) methods obtain this result by taking advantage of the imaginary time evolution of quantum states
- The evolution in (real) time of a state which solves the Schrödinger eq.

$$i\hbar \frac{\partial}{\partial t} |\psi(t)\rangle = \hat{H} |\psi(t)\rangle \quad \text{with initial condition} \quad |\psi(t=0)\rangle = |\psi(0)\rangle$$

is controlled by the evolution operator: $\hat{U}(t) = e^{-\frac{i}{\hbar}\hat{H}t} \quad \Rightarrow \quad |\psi(t)\rangle = e^{-\frac{i}{\hbar}\hat{H}t} |\psi(0)\rangle$

Imaginary-time evolution

- We can use the time evolution of a quantum state to obtain the Schrödinger equation in integral form:

$$|\psi(t)\rangle = e^{-\frac{i}{\hbar}\hat{H}t} |\psi(0)\rangle \quad \Rightarrow \quad \langle \vec{r} | \psi(t) \rangle = \langle \vec{r} | e^{-\frac{i}{\hbar}\hat{H}t} | \psi(0) \rangle$$

$$\Rightarrow \quad \langle \vec{r} | \psi(t) \rangle = \psi(\vec{r}, t) = \int d\vec{r}' \langle \vec{r} | e^{-\frac{i}{\hbar}t\hat{H}} | \vec{r}' \rangle \langle \vec{r}' | \psi(0) \rangle = \int d\vec{r}'\ G(\vec{r}, \vec{r}', t)\ \psi(\vec{r}', 0)$$

G(R,R',t) is the so called propagator (or Green's function)

- We can take advantage of a Wick's rotation ($t \to it/\hbar = \tau$) to write the imaginary time evolution of the quantum state:

$$G(\vec{r}, \vec{r}', t) = \langle \vec{r} | e^{-\frac{i}{\hbar}t\hat{H}} | \vec{r}' \rangle \xrightarrow[\frac{i}{\hbar}t \to \tau]{} G(\vec{r}, \vec{r}', \tau) = \langle \vec{r} | e^{-\tau \hat{H}} | \vec{r}' \rangle$$

$$\psi(\vec{r}, \tau) = \int d\vec{r}' \langle \vec{r} | e^{-\tau \hat{H}} | \vec{r}' \rangle \psi(\vec{r}', 0)$$

if the probability amplitude $\psi(r,0)$ and $\psi(r,\tau)$, and the propagator $G(r,r',\tau)$ are real and positive quantities, such that can be interpreted respectively as probability densities and transition probabilities, one can imagine to built a Markov process with a Metropolis algorithm that converge to the sampling of $\psi(r,\tau)$

- Before we do this, it is instructive to understand the effect of the imaginary time evolution; by using eigenstates ($|\psi_n\rangle$ with eigenvalues E_n) of the Hamiltonian

$$\left|\psi(\tau)\right\rangle = e^{-\tau\hat{H}}\left|\psi(0)\right\rangle = e^{-\tau\hat{H}}\sum_n c_n\left|\psi_n\right\rangle = \sum_n c_n e^{-\tau E_n}\left|\psi_n\right\rangle \underset{\tau>>1}{\approx} c_0 e^{-\tau E_0}\left|\psi_0\right\rangle$$

thus for large imaginary time evolutions the quantum state $\psi(R,\tau)$ becomes essentially proportional to the exact ground state (exponential convergence)

- This is the idea which is at the basis of the projector quantum Monte Carlo Methods: Ground state is seen as the imaginary time evolution of a trial variational state. The first two methods which used a random walk to sample $\psi(R,\tau)$ were the Green Function (GFMC, Kalos 1970) and Diffusion Monte Carlo (DMC)
- Note however that the imaginary time propagator

$$G(\vec{r},\vec{r}',\tau) = \left\langle \vec{r}\left|e^{-\tau\hat{H}}\right|\vec{r}'\right\rangle$$

is nothing more than a density matrix (not normalized)

... this has suggested, more recently, the T=0 K Path Integral methods or the Path Integral projector methods

Path integral projector methods:

- Path Integral Ground State (PIGS, Schmidt 2001):
  Trotter decomposition of the imaginary time evolution operator:
  $e^{-\tau\hat{H}} = \left(e^{-\frac{\tau}{M}\hat{H}}\right)^M = \lim_{M\to\infty}\left(e^{-\frac{\tau}{M}\hat{T}}e^{-\frac{\tau}{M}\hat{V}}\right)^M$ $\Rightarrow$ Path Integral representation:

$$\Psi_0(\vec{r}) = \lim_{\tau\to\infty}\int d\vec{r}_1\cdots d\vec{r}_M \left\langle \vec{r}\left|e^{-\frac{\tau}{M}\hat{H}}\right|\vec{r}_1\right\rangle\times\cdots\times\left\langle \vec{r}_{M-1}\left|e^{-\frac{\tau}{M}\hat{H}}\right|\vec{r}_M\right\rangle \Psi_T(\vec{r}_M) \quad \text{if } \langle\Psi_T|\Psi_0\rangle\neq 0$$

First approximation: finite imaginary time propagation: up to τ

$$\Psi_0(\vec{r}) \cong \Psi_\tau(\vec{r}) = \int d\vec{r}_1\cdots d\vec{r}_N \left\langle \vec{r}\left|e^{-\frac{\tau}{M}\hat{H}}\right|\vec{r}_1\right\rangle\times\cdots\times\left\langle \vec{r}_{M-1}\left|e^{-\frac{\tau}{M}\hat{H}}\right|\vec{r}_M\right\rangle \Psi_T(\vec{r}_M)$$

Second approximation: the exact propagator is not known, we use some short time approximation with $\delta\tau=\tau/M<<1$; for example the **primitive approximation**:

$$\left\langle \vec{r}\left|e^{-\delta\tau(\hat{T}+\hat{V})}\right|\vec{r}'\right\rangle \cong \left\langle \vec{r}\left|e^{-\frac{\delta\tau}{2}\hat{V}}e^{-\delta\tau\hat{T}}e^{-\frac{\delta\tau}{2}\hat{V}}\right|\vec{r}'\right\rangle =$$

$$= e^{-\frac{\delta\tau}{2}V(\vec{r})}\left\langle \vec{r}\left|e^{-\delta\tau\hat{T}}\right|\vec{r}'\right\rangle e^{-\frac{\delta\tau}{2}V(\vec{r}')} = e^{-\frac{\delta\tau}{2}V(\vec{r})}\frac{\exp\left(-\left|\vec{r}_i-\vec{r}_i'\right|^2/4\gamma\delta\tau\right)}{\left(4\pi\gamma\,\delta\tau\right)^{\frac{3}{2}}}e^{-\frac{\delta\tau}{2}V(\vec{r}')} \qquad \text{With } \gamma=\frac{\hbar^2}{2m}$$

Accurate up to terms of order $\delta\tau^2$: one assumes $\frac{\delta\tau^2}{2}\left[\hat{T},\hat{V}\right]=0$

- "Ground State" wave function:
  series of convolutions = projection steps in imaginary time

$$\psi_0(\vec{r}) \cong \psi_\tau(\vec{r}) = \int d\vec{r}_1 \cdots d\vec{r}_M \; G(\vec{r},\vec{r}_1,\tfrac{\tau}{M}) \times \cdots \times G(\vec{r}_{M-1},\vec{r}_M,\tfrac{\tau}{M}) \; \psi_T(\vec{r}_M)$$

- For example, using ψ_τ the expectation value of a diagonal operator turns out

$$\left\langle \psi_0 \middle| \hat{O} \middle| \psi_0 \right\rangle \Big/ \left\langle \psi_0 \middle| \psi_0 \right\rangle = \left\langle \psi_0 \middle| \psi_0 \right\rangle^{-1} \int d\vec{r} \; \psi_0^*(\vec{r}) \hat{O}(\vec{r}) \psi_0(\vec{r}) \cong$$

$$\cong \left\langle \psi_\tau \middle| \psi_\tau \right\rangle^{-1} \int d\vec{r} \int \prod_{i=1}^{M} d\vec{r}_i d\vec{r}_i' \left[\psi_T(\vec{r}_M) \prod_{i=M}^{1} G(\vec{r}_i,\vec{r}_{i-1},\tfrac{\tau}{M}) \right] \hat{O}(\vec{r}) \left[\prod_{j=0}^{M-1} G(\vec{r}_j',\vec{r}_{j+1}',\tfrac{\tau}{M}) \psi_T(\vec{r}_M') \right] =$$

$$= \int \prod_{i=1}^{2M+1} d\vec{r}_i \; \hat{O}(\vec{r}_{M+1}) \frac{\psi_T(\vec{r}_1) \prod_{j=1}^{2M} G(\vec{r}_j,\vec{r}_{j+1},\tfrac{\tau}{M}) \psi_T(\vec{r}_{2M+1})}{\left\langle \psi_\tau \middle| \psi_\tau \right\rangle} \qquad \rightarrow p(\vec{r}_1,\cdots,\vec{r}_{2M+1})$$

"canonical" probability distribution; can be sampled with the Metropolis algorithm

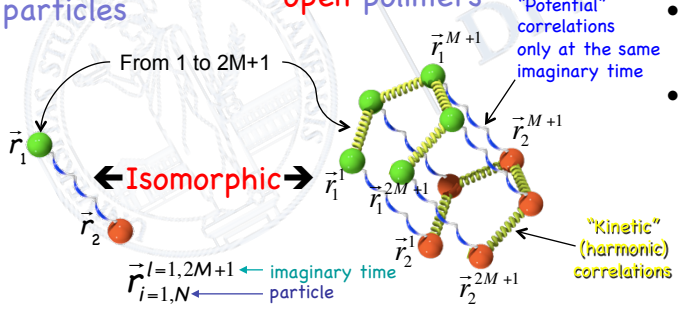
- "Exact" ground state averages are equivalent to canonical averages of a classical system of special interacting linear polymers:

$$\frac{\left\langle \psi_0 \middle| \hat{O} \middle| \psi_0 \right\rangle}{\left\langle \psi_0 \middle| \psi_0 \right\rangle} \cong \frac{\left\langle \psi_\tau \middle| \hat{O} \middle| \psi_\tau \right\rangle}{\left\langle \psi_\tau \middle| \psi_\tau \right\rangle} = \int \prod_{i=1}^{2M+1} d\vec{r}_i \; \hat{O}(\vec{r}_{M+1}) \frac{\psi_T(\vec{r}_1) \prod_{j=1}^{2M} G(\vec{r}_j,\vec{r}_{j+1},\tfrac{\tau}{M}) \psi_T(\vec{r}_{2M+1})}{\left\langle \psi_\tau \middle| \psi_\tau \right\rangle}$$

$p(\vec{r}_1,\cdots,\vec{r}_{2M+1})$ "canonical" probability distribution; can be sampled with the Metropolis algorithm

Quantum system of N interacting particles

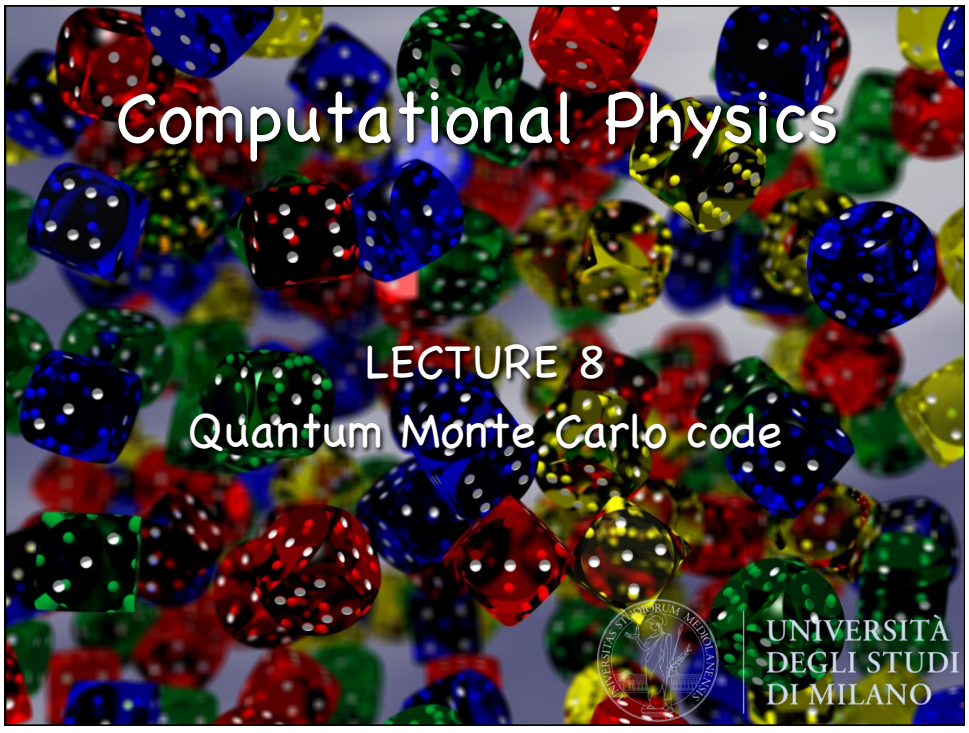
Classical system of N special interacting linear open polimers



- M projections: linear polymers with 2M+1 atoms
- Monte Carlo sampling (Metropolis) of 3N·(2M+1) degree of freedom

Lecture 8: Suggested books

- Kalos, Whitlock *Monte Carlo Methods* - Wiley 1986
- W. Krauth, *Statistical Mechanics: Algorithms and Computations* – Oxford 2006
- Hammond, Lester, Reynolds, *Monte Carlo methods in ab initio quantum chemistry* – World Scientific 1994
- Landau, Binder, *Monte Carlo simulations in statistical physics* – Cambridge 2000



Quantum Monte Carlo code: main 1


```
#include <iostream>
#include <fstream>
#include <cmath>
#include <TRandom3.h>
#include "constants.h"
#include "functions.h"

#define LEFT 0
#define RIGHT 1

TRandom3* generator;

using namespace std;

int main()
{
        readInput();
        initialize();
/* at this time, every variable you see, such for instance "equilibration",
has been either acquired from "input.dat" by the readInput() function or
opportunely initialized by the initialize() function. */
        for(int i=0;i<equilibration;i++)
        {
                if(PIGS)   // only a PIGS polymer has a start and an end.
                {
                        brownianMotion(LEFT);
                        brownianMotion(RIGHT);
                }

                translation();

                for(int j=0;j<brownianBridgeAttempts;j++)
                        brownianBridge();
        }
```

```
void readInput()
{
        ifstream input_file("input.dat");
        char* string_away = new char[60];

        input_file >> string_away >> timeslices;
        input_file >> string_away >> temperature;
        input_file >> string_away >> imaginaryTimePropagation;
        input_file >> string_away >> brownianMotionReconstructions;
        input_file >> string_away >> delta_translation;
        input_file >> string_away >> brownianBridgeReconstructions;
        input_file >> string_away >> brownianBridgeAttempts;
        input_file >> string_away >> MCSTEPS;
        input_file >> string_away >> equilibration;
        input_file >> string_away >> blocks;
        input_file >> string_away >> histogram_bins;
        input_file >> string_away >> histogram_start;
        input_file >> string_away >> histogram_end;
        input_file >> string_away >> timeslices_averages_start>>timeslices_averages_end;
        input_file.close();
        delete string_away;
}
```

Quantum Monte Carlo code: main 2


```
        for(int b=0;b<blocks;b++)
        {
                for(int i=0;i<MCSTEPS;i++)
                {
                        if(PIGS)
                        {
                                brownianMotion(LEFT);
                                brownianMotion(RIGHT);
                        }
                        translation();

                        for(int j=0;j<brownianBridgeAttempts;j++)
                                brownianBridge();

                        upgradeAverages();
                }
                cout<<"Completed block: "<<b+1<<"/"<<blocks<<endl;
                endBlock();
        }

        consoleOutput();
        finalizePotentialEstimator();
        finalizeKineticEstimator();
        finalizeHistogram();

        deleteMemory();  // de-allocate dynamic variables.
        return 0;
}
```

Quantum Monte Carlo code: main 1


```
#include <iostream>
#include <fstream>
#include <cmath>
#include <TRandom3.h>
#include "constants.h"
#include "functions.h"

#define LEFT 0
#define RIGHT 1

TRandom3* generator;

using namespace std;

int main()
{
          readInput();
          initialize();
/* at this time, every variable you see, such for instance "equilibration",
has been either acquired from "input.dat" by the readInput() function or
opportunely initialized by the initialize() function. */
          for(int i=0;i<equilibration;i++)
          {
                    if(PIGS)   // only a PIGS polymer has a start and an end.
                    {
                              brownianMotion(LEFT);
                              brownianMotion(RIGHT);
                    }

                    translation();

                    for(int j=0;j<brownianBridgeAttempts;j++)
                              brownianBridge();
          }
```

Quantum Monte Carlo code: initialize()


```
// Initialization of the variables and allocation of the memory.
void initialize()
{
lambda = hbar*hbar/(2*mass);
if(temperature==0)
          PIGS=1;
else
          PIGS=0;

if(PIGS)
          dtau = imaginaryTimePropagation/(timeslices-1);
else
          dtau = hbar/(boltzmann*temperature*timeslices);

acceptedTranslations=0;
acceptedVariational=0;
acceptedBB=0;
acceptedBM=0;
totalTranslations=0;
totalVariational=0;
totalBB=0;
totalBM=0;

generator = new TRandom3();

positions=new double[timeslices];
potential_energy=new double[timeslices];
potential_energy_accumulator=new double[timeslices];
potential_energy_square_accumulator=new double[timeslices];

kinetic_energy=new double[timeslices];
kinetic_energy_accumulator=new double[timeslices];
kinetic_energy_square_accumulator=new double[timeslices];

positions_histogram=new double[histogram_bins];
positions_histogram_accumulator=new double[histogram_bins];
positions_histogram_square_accumulator=new double[histogram_bins];

...
```

...

```
for(int i=0;i<timeslices;i++)
          positions[i]=0.0;

for(int i=0;i<timeslices;i++)
{
          potential_energy[i]=0;
          potential_energy_accumulator[i]=0;
          potential_energy_square_accumulator[i]=0;

          kinetic_energy[i]=0;
          kinetic_energy_accumulator[i]=0;
          kinetic_energy_square_accumulator[i]=0;
}

for(int i=0;i<histogram_bins;i++)
{
          positions_histogram[i]=0;
          positions_histogram_accumulator[i]=0;
          positions_histogram_square_accumulator[i]=0;
}
alpha=0;
}
```

Quantum Monte Carlo code: main 1


```
#include <iostream>
#include <fstream>
#include <cmath>
#include <TRandom3.h>
#include "constants.h"
#include "functions.h"

#define LEFT 0
#define RIGHT 1

TRandom3* generator;

using namespace std;

int main()
{
        readInput();
        initialize();
/* at this time, every variable you see, such for instance "equilibration",
has been either acquired from "input.dat" by the readInput() function or
opportunely initialized by the initialize() function. */
        for(int i=0;i<equilibration;i++)
        {
                if(PIGS)   // only a PIGS polymer has a start and an end.
                {
                        brownianMotion(LEFT);
                        brownianMotion(RIGHT);
                }

                translation();

                for(int j=0;j<brownianBridgeAttempts;j++)
                        brownianBridge();
        }
```

Quantum Monte Carlo code: translation()


```
void translation()
{
totalTranslations++;
double delta = generator->Uniform(-delta_translation,delta_translation);
double acc_density_matrix_difference=0;
int last = timeslices;
if(PIGS)
        last=timeslices-1;

for(int i=0;i<last;i++)
{
        int inext = index_mask(i+1);
        double newcorr,oldcorr;
        newcorr = potential_density_matrix(positions[i]+delta,positions[inext]+delta);
        oldcorr = potential_density_matrix(positions[i],positions[inext]);
        acc_density_matrix_difference += oldcorr-newcorr;
}
// metropolis: PIGS contains also the statistical weight of the variational Wave Function.
double acceptance_probability = exp(-acc_density_matrix_difference);

if(PIGS)
        acceptance_probability *= variationalWaveFunction(positions[0]+delta)
                                *variationalWaveFunction(positions[timeslices-1]+delta)
                                /(variationalWaveFunction(positions[0])
                                *variationalWaveFunction(positions[timeslices-1]));

if(generator->Rndm()<acceptance_probability)
{
        for(int i=0;i<timeslices;i++)
                positions[i]+=delta;
        acceptedTranslations++;
}
}
```

```
// This is the primitive approximation without the kinetic correlation
double potential_density_matrix(double val, double val_next)
{
        double dens_left = -dtau*external_potential(val)/2;
        double dens_right = -dtau*external_potential(val_next)/2;

        return dens_left+dens_right;
}
```

```
double external_potential(double val)
{
        double k_elastic = 1;
        return k_elastic*val*val/2.0;
}
```

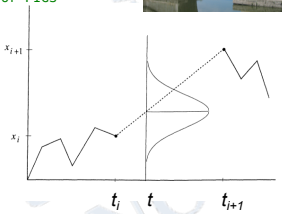
```
double variationalWaveFunction(double v)
{
        return 1.0;
        //return exp(-0.5*v*v);
}
```

Quantum Monte Carlo code: main 1


```
#include <iostream>
#include <fstream>
#include <cmath>
#include <TRandom3.h>
#include "constants.h"
#include "functions.h"

#define LEFT 0
#define RIGHT 1

TRandom3* generator;

using namespace std;

int main()
{
        readInput();
        initialize();
/* at this time, every variable you see, such for instance "equilibration",
has been either acquired from "input.dat" by the readInput() function or
opportunely initialized by the initialize() function. */
        for(int i=0;i<equilibration;i++)
        {
                if(PIGS)   // only a PIGS polymer has a start and an end.
                {
                        brownianMotion(LEFT);
                        brownianMotion(RIGHT);
                }

                translation();

                for(int j=0;j<brownianBridgeAttempts;j++)
                        brownianBridge();
        }
```


QMC code: brownianBridge()

```
void brownianBridge()
{
totalBB++;
int available_starting_points = timeslices-brownianBridgeReconstructions-1; // for PIGS
if(!PIGS) available_starting_points = timeslices-1;
int starting_point = (int)(generator->Rndm()*available_starting_points);
int endpoint = index_mask(starting_point + brownianBridgeReconstructions + 1);
double starting_coord = positions[starting_point];
double ending_coord = positions[endpoint];
double new_segment[brownianBridgeReconstructions+2];
new_segment[0]=starting_coord;
new_segment[brownianBridgeReconstructions+1]=ending_coord;
double previous_position = starting_coord;
for(int i=0;i<brownianBridgeReconstructions;i++)
{
        int left_reco = brownianBridgeReconstructions-i;
        // gaussian sampling of the free particle propagator
        double average_position = previous_position + (ending_coord-previous_position)/(left_reco+1);
        double variance = 2*lambda*dtau*left_reco/(left_reco+1);
        double newcoordinate = generator->Gaus(average_position,sqrt(variance));
        new_segment[i+1] = newcoordinate;
        previous_position=newcoordinate;
}
double acc_density_matrix_difference=0;
for(int i=0;i<brownianBridgeReconstructions+1;i++)
{
        int i_old = index_mask(starting_point+i);
        int i_next_old = index_mask(starting_point+i+1);
        double newcorr,oldcorr;
        newcorr = potential_density_matrix(new_segment[i],new_segment[i+1]);
        oldcorr = potential_density_matrix(positions[i_old],positions[i_next_old]);
        acc_density_matrix_difference += oldcorr-newcorr;
}
double acceptance_probability = exp(-acc_density_matrix_difference);
if(generator->Rndm()<acceptance_probability)
{
        for(int i=1;i<brownianBridgeReconstructions+1;i++)
        {
                int i_old = index_mask(starting_point+i);
                positions[i_old]=new_segment[i];
        }
        acceptedBB++;
}
}
```



$$\left(W(t)\middle|W(t_j)=x_j, j=1,\cdots,k\right)=\mathrm{N}\left(\frac{(t_{i+1}-t)x_i+(t-t_i)x_{i+1}}{(t_{i+1}-t_i)},\frac{(t_{i+1}-t)(t-t_i)}{(t_{i+1}-t_i)}\right)$$

$$=\mathrm{N}\left(\frac{(n-1)x_i+x_{i+1}}{n},\frac{(n-1)}{n}2\lambda d\tau\right)=\mathrm{N}\left(x_i+\frac{x_{i+1}-x_i}{n},\frac{(n-1)}{n}2\lambda d\tau\right)$$

```
int index_mask(int ind)
{
if(PIGS)
        return ind;  // no pbc
Else
{
        int new_ind=ind;
        while(new_ind>=timeslices)   // pbc
                new_ind-=timeslices;
        return new_ind;
}
}
```

Quantum Monte Carlo code: main 1


```
#include <iostream>
#include <fstream>
#include <cmath>
#include <TRandom3.h>
#include "constants.h"
#include "functions.h"

#define LEFT 0
#define RIGHT 1

TRandom3* generator;

using namespace std;

int main()
{
        readInput();
        initialize();
/* at this time, every variable you see, such for instance "equilibration",
has been either acquired from "input.dat" by the readInput() function or
opportunely initialized by the initialize() function. */
        for(int i=0;i<equilibration;i++)
        {
                if(PIGS)   // only a PIGS polymer has a start and an end.
                {
                        brownianMotion(LEFT);
                        brownianMotion(RIGHT);
                }

                translation();

                for(int j=0;j<brownianBridgeAttempts;j++)
                        brownianBridge();
        }
```

QMC code: brownianMotion(int) 1

```
void brownianMotion(int which) // BM is called only for PIGS simulations
{
int starting_point, endpoint, left_reco;
double starting_coord, ending_coord, average_position, variance, newposition, oldposition;
totalBM++;
if(which==LEFT)
{
        starting_point = 0;
        endpoint = brownianMotionReconstructions+1;
        ending_coord = positions[endpoint];
        average_position = ending_coord;
        variance = 2*lambda*dtau*(brownianMotionReconstructions+1);
        starting_coord = generator->Gaus(average_position,sqrt(variance));
        oldposition = positions[starting_point];
        newposition = starting_coord;
}
else
{
        starting_point = timeslices-2-brownianMotionReconstructions;
        endpoint = timeslices-1;
        starting_coord = positions[starting_point];
        average_position = starting_coord;
        variance = 2*lambda*dtau*(brownianMotionReconstructions+1);
        ending_coord = generator->Gaus(average_position,sqrt(variance));
        oldposition = positions[endpoint];
        newposition = ending_coord;
}
double new_segment[brownianMotionReconstructions+2];
new_segment[0]=starting_coord;
new_segment[brownianMotionReconstructions+1]=ending_coord;
double previous_position = starting_coord;
for(int i=0; i<brownianMotionReconstructions; i++)
{
        left_reco = brownianMotionReconstructions-I; // gaussian sampling of the free particle propagator
        average_position = previous_position + (ending_coord-previous_position)/(left_reco+1);
        variance = 2*lambda*dtau*left_reco/(left_reco+1);
        double newcoordinate = generator->Gaus(average_position,sqrt(variance));
        new_segment[i+1] = newcoordinate;
        previous_position=newcoordinate;
}
```

For $X(t)\sim BM(\mu,\sigma^2)$ with constant μ and $\sigma > 0$ and given $X(0)=X_0$ the path can be obtained as:

$$X(t_{i+1}) = X(t_i) + \mu(t_{i+1} - t_i) + \sigma Z_{i+1}\sqrt{t_{i+1} - t_i} \qquad i = 0, \cdots, n-1$$

$$\Rightarrow X[t + \delta t(n_{BM}+1)] = X(t) + Z_{i+1}\sqrt{2\lambda\delta t(n_{BM}+1)} \quad \text{for } \mu = 0$$

$$\left(W(t)\middle|W(t_j) = x_j, j = 1, \cdots, k\right) = \mathrm{N}\left(x_i + \frac{x_{i+1} - x_i}{n}, \frac{(n-1)}{n} 2\lambda d\tau\right)$$

QMC code: brownianMotion(int) 2

```
double acc_density_matrix_difference=0;
        for(int i=0;i<brownianMotionReconstructions+1;i++)
        {
                double newcorr,oldcorr;
                newcorr = potential_density_matrix(new_segment[i],new_segment[i+1]);
                oldcorr = potential_density_matrix(positions[starting_point+i],positions[starting_point+i+1]);
                acc_density_matrix_difference += oldcorr-newcorr;
        }

        double acceptance_probability = exp(-acc_density_matrix_difference)
                                *variationalWaveFunction(newposition)/variationalWaveFunction(oldposition);
        if(generator->Rndm()<acceptance_probability)
        {
                for(int i=0;i<brownianMotionReconstructions+2;i++)
                {
                        positions[starting_point+i]=new_segment[i];
                }
                acceptedBM++;
        }
}
```

Quantum Monte Carlo code: main 2


```
        for(int b=0;b<blocks;b++)
        {
                for(int i=0;i<MCSTEPS;i++)
                {
                        if(PIGS)
                        {
                                brownianMotion(LEFT);
                                brownianMotion(RIGHT);
                        }
                        translation();

                        for(int j=0;j<brownianBridgeAttempts;j++)
                                brownianBridge();

                        upgradeAverages();
                }
                cout<<"Completed block: "<<b+1<<"/"<<blocks<<endl;
                endBlock();
        }

        consoleOutput();
        finalizePotentialEstimator();
        finalizeKineticEstimator();
        finalizeHistogram();

        deleteMemory();  // de-allocate dynamic variables.
        return 0;
}
```

QMC code: upgradeAverages()


```
/* This function accumulates the expectation values in their respective variables.
   At the end of the block, these variables are divided by the MCSTEPS value and
   the block average and its squared value are accumulated in apposite variables.
 */
void upgradeAverages()
{
for(int i=0;i<timeslices;i++) potential_energy[i]
                              += external_potential(positions[i]);
int flag=0;
if(PIGS) flag=1;

for(int i=flag;i<timeslices-flag;i++)
{
            int i_mod = index_mask(i);
            int i_mod_next = index_mask(i+1);
            kinetic_energy[i_mod]
            += kineticEstimator(positions[i_mod],positions[i_mod_next]);
}
if(flag)
{
            kinetic_energy[0]+=variationalLocalEnergy(positions[0]);
            kinetic_energy[timeslices-1]
            += variationalLocalEnergy(positions[timeslices-1]);
}
upgradeHistogram();
}
```

```
// (-hbar*hbar/2m)d^2/dx^2G(x,x',dtau)
double kineticEstimator(double value,double next_value)
{double kinetic_prime = (value-next_value)/(2*lambda*dtau);
double kinetic_second= 1./(2*lambda*dtau);
double term_1 = (dtau/2)*external_potential_prime(value)+kinetic_prime;
double term_2 = (dtau/2)*external_potential_second(value)+kinetic_second;
return -(hbar*hbar/(2*mass))*(term_1*term_1 - term_2);}
```

```
// (-hbar*hbar/2m)(d^2/dx^2G(x,x',dtau))/G(x,x',dtau)
double variationalLocalEnergy(double val)
{double psi = variationalWaveFunction(val);
double laplacian_psi = variationalWaveFunction_second(val);
return -(hbar*hbar/(2*mass))*laplacian_psi/psi;}
```

```
void upgradeHistogram()
{
double delta_pos = (histogram_end-histogram_start)/histogram_bins;
for(int i=timeslices_averages_start; i<=timeslices_averages_end; i+=1)
{
            int k=1;
            while((histogram_start + k*delta_pos)<positions[i]) k++;
            positions_histogram[k-1]+=1;
            }
}
```

Quantum Monte Carlo code: main 2


```
for(int b=0;b<blocks;b++)
{
          for(int i=0;i<MCSTEPS;i++)
          {
                    if(PIGS)
                    {
                              brownianMotion(LEFT);
                              brownianMotion(RIGHT);
                    }
                    translation();

                    for(int j=0;j<brownianBridgeAttempts;j++) brownianBridge();
                    upgradeAverages();
          }
          cout<<"Completed block: "<<b+1<<"/"<<blocks<<endl;
          endBlock();
}

consoleOutput();
finalizePotentialEstimator();
finalizeKineticEstimator();
finalizeHistogram();

// de-allocate dynamic variables.
deleteMemory();
return 0;
}
```

```
void endBlock()  // calculating and accumulating block averages
{
for(int i=0;i<timeslices;i++)
{
            potential_energy[i]/=MCSTEPS;
            potential_energy_accumulator[i]+=potential_energy[i];
            potential_energy_square_accumulator[i]
            +=potential_energy[i]*potential_energy[i];
            potential_energy[i]=0;
            kinetic_energy[i]/=MCSTEPS;
            kinetic_energy_accumulator[i]+=kinetic_energy[i];
            kinetic_energy_square_accumulator[i]
            +=kinetic_energy[i]*kinetic_energy[i];
            kinetic_energy[i]=0;
}
for(int i=0;i<histogram_bins;i++)
{
            positions_histogram[i]/=MCSTEPS;
            positions_histogram_accumulator[i]+=positions_histogram[i];
            positions_histogram_square_accumulator[i]
            +=positions_histogram[i]*positions_histogram[i];
            positions_histogram[i]=0;
}
}
```

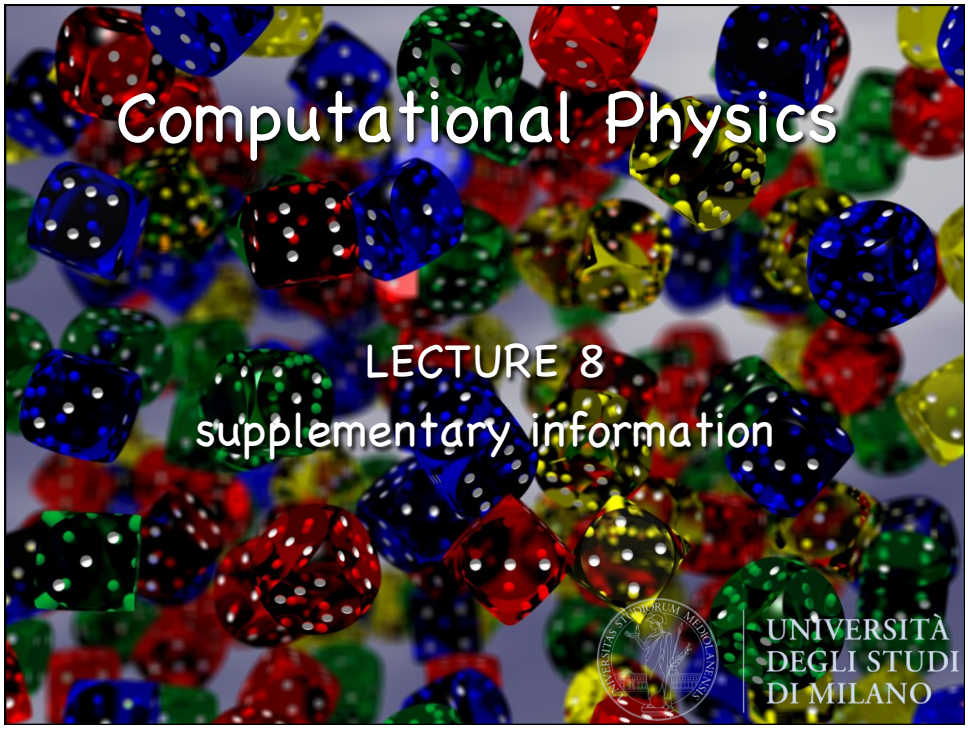
Quantum Monte Carlo code: main 2


```
for(int b=0;b<blocks;b++)
{
          for(int i=0;i<MCSTEPS;i++)
          {
                    if(PIGS)
                    {
                              brownianMotion(LEFT);
                              brownianMotion(RIGHT);
                    }
                    translation();

                    for(int j=0;j<brownianBridgeAttempts;j++) brownianBridge();
                    upgradeAverages();
          }
          cout<<"Completed block: "<<b+1<<"/"<<blocks<<endl;
          endBlock();
}

consoleOutput();
finalizePotentialEstimator();
finalizeKineticEstimator();
finalizeHistogram();

// de-allocate dynamic variables.
deleteMemory();
return 0;
}
```

```
void consoleOutput()
{
          cout<<"Acceptances:"<<endl;
          if(PIGS) cout<<"BM: "<<((double)acceptedBM)/totalBM<<endl;
          cout<<"Transl: "<<((double)acceptedTranslations)/totalTranslations<<endl;
          cout<<"BB: "<<((double)acceptedBB)/totalBB<<endl;
}
```



```
void finalizePotentialEstimator()
{
ofstream out("potential.dat");
for(int i=0;i<timeslices;i++)
{
          double potential_energy_average = potential_energy_accumulator[i]/blocks;
          double potential_energy_square_avg = potential_energy_square_accumulator[i]/blocks;
          double p_error =sqrt(abs(potential_energy_average*potential_energy_average-potential_energy_square_avg)/blocks);
          out<<i<<" "<<potential_energy_average<<" "<<p_error<<endl;
}
out.close();
}

void finalizeKineticEstimator()
{
ofstream out("kinetic.dat");
for(int i=0;i<timeslices;i++)
{
          double kinetic_energy_average = kinetic_energy_accumulator[i]/blocks;
          double kinetic_energy_square_avg = kinetic_energy_square_accumulator[i]/blocks;
          double k_error =sqrt(abs(kinetic_energy_average*kinetic_energy_average-kinetic_energy_square_avg)/blocks);
          out<<i<<" "<<kinetic_energy_average<<" "<<k_error<<endl;

}
out.close();
}

void finalizeHistogram()
{
ofstream out("probability.dat");
double current_position, hist_average, hist_square_avg, error;
double delta_pos = (histogram_end-histogram_start)/histogram_bins;
double norma = 0.0;
for(int i=0; i<histogram_bins; i++)
{
          norma += positions_histogram_accumulator[i]/blocks;
}
norma *= delta_pos;
for(int i=0; i<histogram_bins; i++)
{
          current_position = histogram_start + (i+0.5)*delta_pos;
          hist_average = positions_histogram_accumulator[i]/blocks;
          hist_square_avg = positions_histogram_square_accumulator[i]/blocks;
          error =sqrt(abs(hist_average*hist_average-hist_square_avg)/blocks);
          out << current_position << " " << hist_average/norma << " " << error/norma << endl;
}
out.close();
}
```



Canonical density matrix for N free particles

- Assume the many-particle wave function to be the product of the one-particle states

$$\Psi_{\vec{k}_1\cdots\vec{k}_N}(\vec{r}_1,\cdots,\vec{r}_N)=\left\langle \vec{r}_1,\cdots,\vec{r}_N \middle| \vec{k}_1\cdots\vec{k}_N \right\rangle=\prod_{i=1}^{N}\phi_{\vec{k}_i}(\vec{r}_i)$$

- This wave function is an eigenfunction of the Hamiltonian:

$$H\left|\vec{k}_1\cdots\vec{k}_N\right\rangle=E\left|\vec{k}_1\cdots\vec{k}_N\right\rangle \qquad \hat{H}=\sum_{i=1}^{N}\frac{\hat{\vec{p}}_i^{\,2}}{2m} \qquad E=\sum_{i=1}^{N}\frac{\hbar^2\vec{k}_i^{\,2}}{2m}$$

- It is easy to see that

$$Q_N(T,V)=\mathrm{Tr}(e^{-\beta\hat{H}})=\sum_{\vec{k}_1\cdots\vec{k}_N}\left\langle \vec{k}_1\cdots\vec{k}_N \middle| e^{-\beta\hat{H}} \middle| \vec{k}_1\cdots\vec{k}_N \right\rangle=\prod_{i=1}^{N}\sum_{\vec{k}_i}\left\langle \vec{k}_i \middle| e^{-\beta\frac{\hbar\vec{k}_i^{\,2}}{2m}} \middle| \vec{k}_i \right\rangle=\prod_{i=1}^{N}Q_1(T,V)=\left[Q_1(T,V)\right]^N=\frac{V^N}{\lambda^{3N}}$$

- The introduction of the density matrix is thus not sufficient to remove the problem that identical quantum mechanical particles are indistinguishable: we have obtained the classical result. The reason is that in non symmetrized wave function the particles are still considered distinguishable. The density matrix in the momentum and coordinate representations reads

$$\left\langle \vec{k}_1'\cdots\vec{k}_N' \middle| \hat{\rho} \middle| \vec{k}_1\cdots\vec{k}_N \right\rangle=\frac{\lambda^3}{V}e^{-\beta\sum_{i=1}^{N}\frac{\hbar^2\vec{k}_i^{\,2}}{2m}}\delta_{\vec{k}_i'\vec{k}_i}$$

$$\rho(\vec{r}_1\cdots\vec{r}_N;\vec{r}_1'\cdots\vec{r}_N')=\left\langle \vec{r}_1'\cdots\vec{r}_N' \middle| \hat{\rho} \middle| \vec{r}_1\cdots\vec{r}_N \right\rangle=\frac{1}{V}e^{-\frac{\pi}{\lambda^2}\sum_{i=1}^{N}(\vec{r}_i'-\vec{r}_i)^2}$$

Indistinguishable particles

- Note that we will obtain a consistent quantum statistical theory only if we regard the fact that identical particles are indistinguishable in the quantum mechanical states. Symbols: D=Distinguishable; S=Symmetric; A= Antisymmetric

- In general when particles are interacting with each other we can write

$$\rho_D(\vec{r}_1\cdots\vec{r}_N;\vec{r}_1'\cdots\vec{r}_N';\beta) = \sum_{\text{all states}} e^{-\beta E_i}\Psi_i(\vec{r}_1\cdots\vec{r}_N)\Psi_i^*(\vec{r}_1'\cdots\vec{r}_N')$$

- When the particles obey Bose statistics, then

$$\rho_S(\vec{r}_1\cdots\vec{r}_N;\vec{r}_1'\cdots\vec{r}_N';\beta) = \sum_{\substack{\text{Symmetric}\\ \text{states only}}} e^{-\beta E_i}\Psi_i(\vec{r}_1\cdots\vec{r}_N)\Psi_i^*(\vec{r}_1'\cdots\vec{r}_N') = \frac{1}{N!}\sum_P \rho_D(\vec{r}_1\cdots\vec{r}_N;P\vec{r}_1'\cdots P\vec{r}_N';\beta)$$

  where we have introduced the notation P to indicate the permutation of particles; here Σ_P means that we sum over all permutations
- For the antisymmetric case [$(-1)^P$= (-)1 for even (odd) permutation]

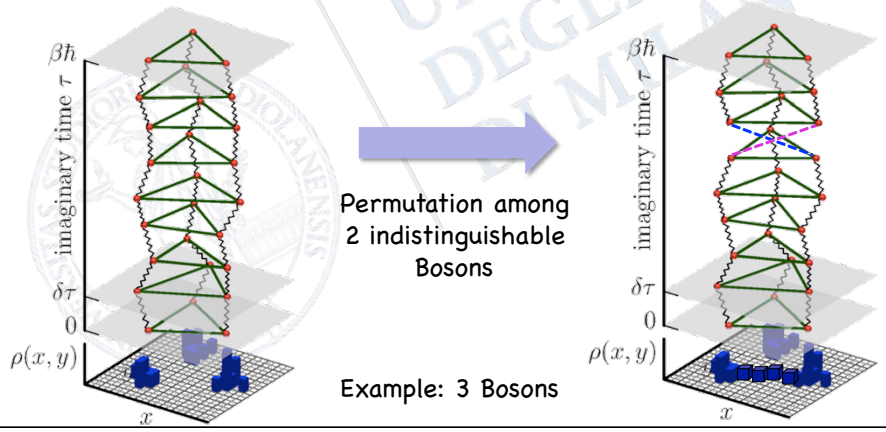
$$\rho_A(\vec{r}_1\cdots\vec{r}_N;\vec{r}_1'\cdots\vec{r}_N';\beta) = \frac{1}{N!}\sum_P (-1)^P \rho_D(\vec{r}_1\cdots\vec{r}_N;P\vec{r}_1'\cdots P\vec{r}_N';\beta)$$

Bose statistics: permutations sampling

- Now we should reintroduce the sum over the permutations in order obtain a symmetric density matrix … this wold be hard: #permutation too large!!!:

$$\rho(R,R,\beta) = \frac{1}{N!}\sum_P \zeta^P \left\langle \vec{r}_1,\cdots,\vec{r}_N \left| e^{-\beta\hat{H}} \right| P\vec{r}_1,\cdots,P\vec{r}_N \right\rangle$$

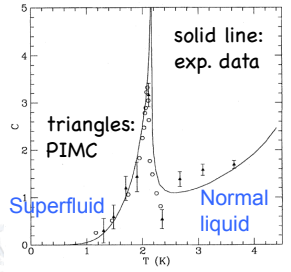
- Easier (only for Bosons!): sampling of permutations along the Monte Carlo simulation



PIMC results

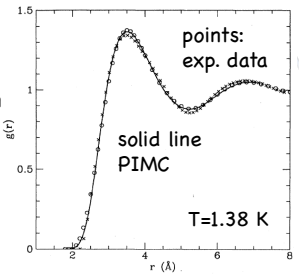
- Some examples of calculations obtained with the PIMC method for liquid Helium (^{4}He)

Specific heat: the lambda transition is reproduced

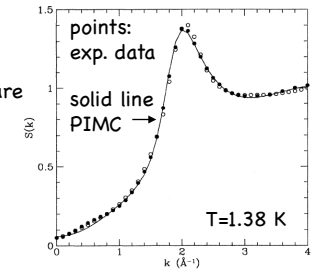


- The role of permutations is thus essential: configuration path like that present in this picture can be obtained only by sampling of permutation-exchanges between particles

Radial distribution function



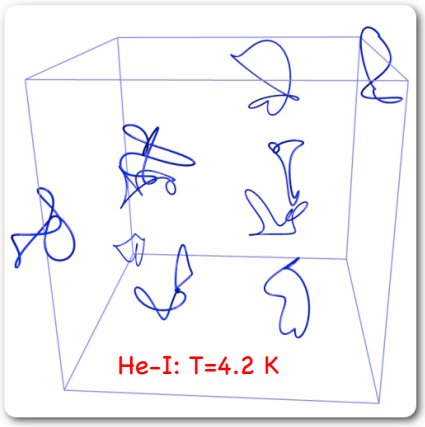
Static structure factor

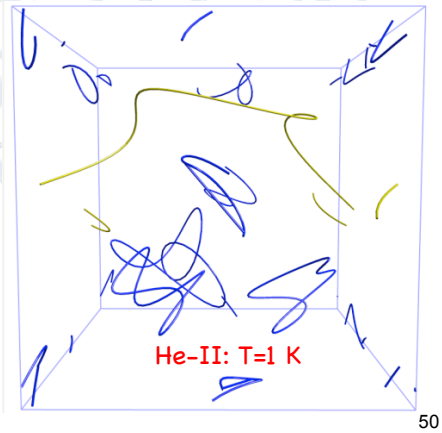


- When the size of the polymer equals the inter-polymer spacing, roughly $\rho^{-1/3}$, it is at least possible for the polymer to link up by exchanging end points. This defines the degeneracy temperature

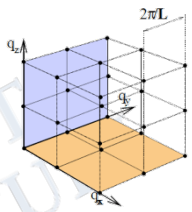
$$T_D = \frac{\rho^{2/3}\hbar^2}{mk_B}$$

- For temperatures $T>T_D$ (=2.32 T_c for ^{4}He) quantum statistics (either Bosonic or Fermionic) are not very important, but below T_D are essential!





The Thermodynamic Limit



- What happens to the various momentum summations in the thermodynamic limit, $V \to \infty$?
- When the allowed momenta become arbitrarily close together, the discrete summations over momentum must be replaced by continuous integrals. For each dimension, the increment in momentum appearing inside the discrete summations is: $\Delta q = 2\pi/L$ … so that $L\Delta q/2\pi = 1$.
- Thus in 1D, the summation over the discrete values of q can be formally rewritten as $\sum_{q_n} \{\cdots\} = L \sum_{q_n} \frac{\Delta q}{2\pi} \{\cdots\}$ where $q_n = 2\pi n/L$ and $n \in Z$
- When we take $L \to \infty$, q becomes a continuous variable, so that the summation can now be replaced by a continuous integral:

$$\sum_{q_n} \{\cdots\} \to L \int_{-\infty}^{\infty} \frac{dq}{2\pi} \{\cdots\}$$

- Similarly, in in d-dimensions: $\sum_{\vec{q}_n} \{\cdots\} \to L^d \int \frac{d\vec{q}}{(2\pi)^d} \{\cdots\} = \frac{V}{(2\pi)^d} \int d\vec{q} \{\cdots\}$