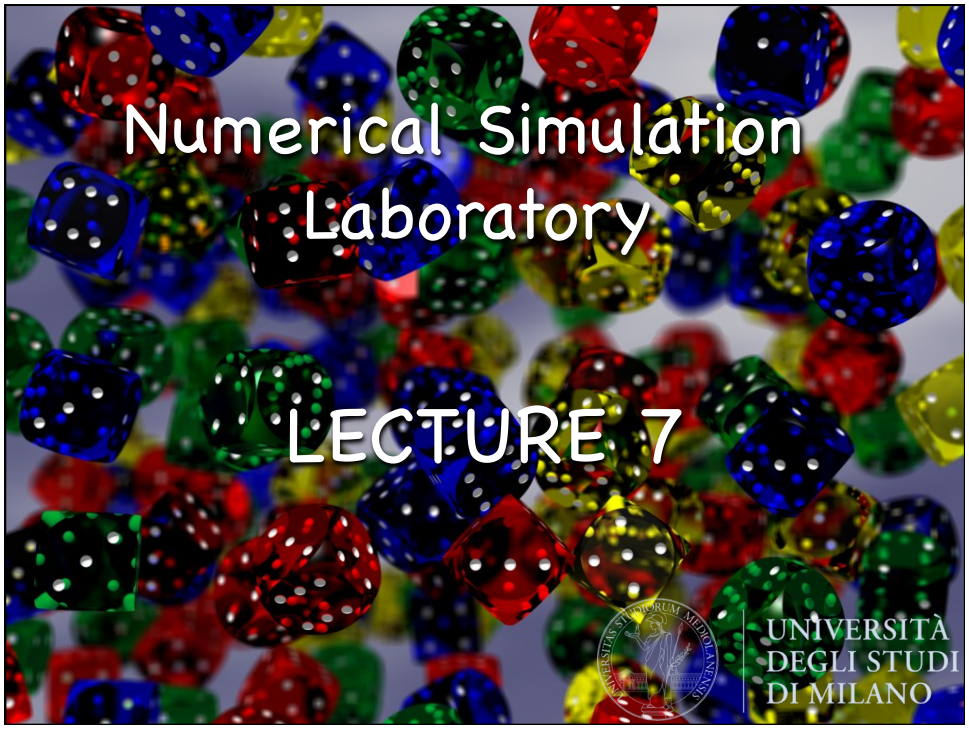
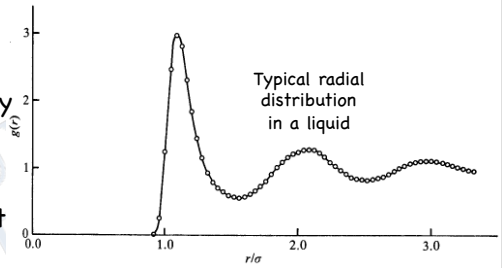


Lecture 7 - outline

- Structural quantities: Density and distribution functions
- Master equation
- Fokker-Planck equation
- Smart Monte Carlo
- Reweighting
- NVT Canonical Ensamble LJ code

Structural quantities

- The structure of simple monatomic fluids is characterized by a set of distribution functions for the atomic positions, the simplest of which is the pair (or radial) distribution function $g^{(2)}(\mathbf{r}_i,\mathbf{r}_j)$ or $g^{(2)}(r_{ij})$ or simply g(r).
- This function gives the probability of finding a pair of atoms a distance r apart, relative to the probability expected for a completely random distribution at the same density.



- To define g(r), we start from the n-particles densities which are obtained by integrating the configurational distribution function over the positions of all atoms except n, incorporating the appropriate normalization factors. In the canonical ensemble we have

$$\rho_N^{(n)}(\vec{r}_1,\cdots,\vec{r}_n)=\frac{N!}{(N-n)!}\int d\vec{r}_{n+1}\cdots d\vec{r}_N\,\frac{e^{-\beta V(\vec{r}_1,\cdots,\vec{r}_N)}}{Z}$$

but of course this definition can be generalized to any configurational probability by substituting any desired $p(r_1, \dots, r_N)$ to $\exp(-\beta V)/Z$

- The normalization implies that

$$\int d\vec{r}_1\cdots d\vec{r}_n\,\rho_N^{(n)}(\vec{r}_1,\cdots,\vec{r}_n)=\frac{N!}{(N-n)!}\int d\vec{r}_1\cdots d\vec{r}_N\,\frac{e^{-\beta V(\vec{r}_1,\cdots,\vec{r}_N)}}{Z}\to 1=\frac{N!}{(N-n)!}$$

so that $\int d\vec{r}_1\,\rho_N^{(1)}(\vec{r}_1)=N$ and for an homogeneous system $\rho_N^{(1)}(\vec{r}_1)=\frac{N}{V}$

- For an ideal gas we have

$$V(\vec{r}_1,\cdots,\vec{r}_N)=0 \quad and \quad Z_N=V^N \quad \Rightarrow \rho_N^{(n)}(\vec{r}_1,\cdots,\vec{r}_n)=\frac{N!\rho^n}{N^n(N-n)!}$$

it follows that the probability to find a particle in a certain position having fixed the position of another particle in a system with N particles is

$$\rho_N^{(2)}(ideal)=\rho^2\frac{N(N-1)}{N^2}=\rho\frac{N-1}{V}$$

which is proportional to (N-1)/V as it should

- The n-particles distribution function is defined as

$$g_N^{(n)}(\vec{r}_1,\cdots,\vec{r}_n)=\frac{\rho_N^{(n)}(\vec{r}_1,\cdots,\vec{r}_n)}{\prod_{i=1}^n\rho_N^{(1)}(\vec{r}_i)}$$

Radial distribution function

- The pair distribution function is useful, not only because it provides insight into the liquid structure, but also because the ensemble average of any pair function may be computed directly by knowing it
- Consider for example a pair additive potential energy:

$$V(\vec{r}_1,\cdots,\vec{r}_N)=\frac{1}{2}\sum_{\substack{i=1\\(i\neq j)}}^{N}\sum_{j=1}^{N}v\left(\left|\vec{r}_i-\vec{r}_j\right|\right)$$

- The expectation value of the potential energy is

$$\langle V\rangle=\int d\vec{r}_1\cdots d\vec{r}_N\,\frac{e^{-\beta V(\vec{r}_1,\cdots,\vec{r}_N)}}{Z_N}V(\vec{r}_1,\cdots,\vec{r}_N)=\text{..for homogeneous fluids...}=$$

$$=\frac{1}{2}N(N-1)\int d\vec{r}_1d\vec{r}_2\,v\left(\left|\vec{r}_1-\vec{r}_2\right|\right)\int d\vec{r}_3\cdots d\vec{r}_N\,\frac{e^{-\beta V(\vec{r}_1,\cdots,\vec{r}_N)}}{Z_N}=$$

$$=\frac{1}{2}\int d\vec{r}_1d\vec{r}_2\,v\left(\left|\vec{r}_1-\vec{r}_2\right|\right)\rho_N^{(2)}(\vec{r}_1,\vec{r}_2)=\frac{\rho^2}{2}\int d\vec{r}_1d\vec{r}_2\,v\left(\left|\vec{r}_1-\vec{r}_2\right|\right)g^{(2)}(\vec{r}_1,\vec{r}_2)=$$

$$=\frac{N^2}{2V^2}\int d\vec{r}_1d\vec{r}_2\,v\left(\left|\vec{r}_1-\vec{r}_2\right|\right)g\left(\left|\vec{r}_1-\vec{r}_2\right|\right)=\frac{N^2}{2V}\int d\vec{r}\,v(r)g(r)=$$

$$=2\pi\rho N\int_0^{+\infty}dr\,r^2v(r)g(r)$$

Tail corrections

- Thus the knowledge of the radial distribution function g(r) in principle give access to the expectation value of any pair function of the coordinates. See also the relevant connection with a scattering experiment in the supplementary material: g(r) can be indirectly measured via such kind of experiments
- Computer simulations frequently use a potential with a spherical cutoff at a distance r_c. It becomes useful to correct the results of simulations to compensate for the missing long-range part of the potential
- Contributions to the energy, pressure etc. for $r>r_c$ are frequently estimated by assuming that g(r) ≈ 1 in this region:

$$\langle V\rangle_{tail}=2\pi\rho N\int_{r_c}^{+\infty}dr\,r^2v(r)g(r)\cong 2\pi\rho N\int_{r_c}^{+\infty}dr\,r^2v(r)$$

- For example, the Lennard-Jones potential gives

$$\frac{\langle V_{LJ}\rangle_{tail}}{N}\cong 2\pi\rho\int_{r_c}^{+\infty}dr\,r^2\,4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12}-\left(\frac{\sigma}{r}\right)^{6}\right]=8\pi\rho\varepsilon\left[-\frac{\sigma^{12}}{9r^9}+\frac{\sigma^6}{3r^3}\right]_{r_c}^{+\infty}=$$

$$=8\pi\left(\frac{\rho}{\sigma^3}\right)\varepsilon\left[-\frac{1}{9}\left(\frac{\sigma}{r}\right)^9+\frac{1}{3}\left(\frac{\sigma}{r}\right)^3\right]_{r_c}^{+\infty}=8\pi\rho_{LJ}\varepsilon\left[\frac{1}{9}\left(\frac{\sigma}{r_c}\right)^9-\frac{1}{3}\left(\frac{\sigma}{r_c}\right)^3\right]=\frac{8\pi\rho_{LJ}\varepsilon}{9r_{c(LJ)}^9}-\frac{8\pi\rho_{LJ}\varepsilon}{3r_{c(LJ)}^3}$$

6

Tail corrections for pressure

- We thus can compute the tail correction to the pressure:

$$\frac{\langle w_{LJ}\rangle_{tail}}{3N\varepsilon} \cong \frac{2}{3}\pi\rho\int_{r_c}^{+\infty} dr\, r^2 48\left[\left(\frac{\sigma}{r}\right)^{12} - \frac{1}{2}\left(\frac{\sigma}{r}\right)^{6}\right] = 32\pi\rho\left[-\frac{\sigma^{12}}{9r^9} + \frac{\sigma^6}{6r^3}\right]_{r_c}^{+\infty} =$$

$$= 32\pi\left(\frac{\rho}{\sigma^3}\right)\left[-\frac{1}{9}\left(\frac{\sigma}{r}\right)^9 + \frac{1}{6}\left(\frac{\sigma}{r}\right)^3\right]_{r_c}^{+\infty} = 32\pi\rho_{LJ}\left[\frac{1}{9}\left(\frac{\sigma}{r_c}\right)^9 - \frac{1}{6}\left(\frac{\sigma}{r_c}\right)^3\right] =$$

$$= 32\pi\rho_{LJ}\left[\frac{1}{9r_{c(LJ)}^9} - \frac{1}{6r_{c(LJ)}^3}\right]$$

g(r) : the algorithm

- We have to deduce an algorithm for the calculation of the radial distribution function g(r). We observe that

$$\left\langle \delta(\vec{r}-\vec{r}_1)\right\rangle = \int \delta(\vec{r}-\vec{r}_1)\frac{e^{-\beta V(\vec{r}_1\cdots\vec{r}_N)}}{Z_N} d\vec{r}_1\cdots d\vec{r}_N = \int \frac{e^{-\beta V(\vec{r},\vec{r}_2,\cdots\vec{r}_N)}}{Z_N} d\vec{r}_2\cdots d\vec{r}_N$$

$$\Rightarrow \quad \rho_N^{(1)}(\vec{r}) = N\int \frac{e^{-\beta V(\vec{r},\vec{r}_2,\cdots\vec{r}_N)}}{Z_N} d\vec{r}_2\cdots d\vec{r}_N = \left\langle \sum_{i=1}^{N}\delta(\vec{r}-\vec{r}_i)\right\rangle$$

this is an algorithm for the one-particle density.
For the two-particles density we see that

$$\left\langle \delta(\vec{r}-\vec{r}_1)\delta(\vec{r}'-\vec{r}_2)\right\rangle = \int \frac{e^{-\beta V(\vec{r},\vec{r}',\vec{r}_3,\cdots\vec{r}_N)}}{Z_N} d\vec{r}_3\cdots d\vec{r}_N$$

$$\Rightarrow \rho_N^{(2)}(\vec{r},\vec{r}') = N(N-1)\int \frac{e^{-\beta V(\vec{r},\vec{r}',\vec{r}_3,\cdots\vec{r}_N)}}{Z_N} d\vec{r}_3\cdots d\vec{r}_N = \left\langle \sum\sum_{i\neq j=1}^{N}\delta(\vec{r}-\vec{r}_i)\delta(\vec{r}'-\vec{r}_j)\right\rangle$$

- In a homogeneous liquid where $g^{(2)}(\mathbf{r}_1,\mathbf{r}_2)$ is simply a function of the modulus of the difference between the two vectors $\mathbf{r}_1$ and $\mathbf{r}_2$: $g^{(2)}(\mathbf{r}_1,\mathbf{r}_2) = g^{(2)}(|\mathbf{r}_1-\mathbf{r}_2|) = g(r)$

it is not the case to compute directly the one-particle and the two-particles densities to compute g(r) as their ratio, instead we observe

$$\left\langle \frac{1}{N}\sum\sum_{i\neq j=1}^{N}\delta(\vec{r}+\vec{r}_j-\vec{r}_i)\right\rangle = \left\langle \frac{1}{N}\int\sum\sum_{i\neq j=1}^{N}\delta(\vec{r}+\vec{r}'-\vec{r}_i)\delta(\vec{r}'-\vec{r}_j)\,d\vec{r}'\right\rangle =$$

$$= \frac{1}{N}\int \rho_N^{(2)}(\vec{r}+\vec{r}',\vec{r}')\,d\vec{r}' = \frac{1}{N}\int \rho_N^{(2)}\left(\left|\vec{r}+\vec{r}'-\vec{r}'\right|\right)d\vec{r}' = \frac{1}{\rho}\rho_N^{(2)}\left(\left|\vec{r}\right|\right) = \rho g(r)$$

$$\Rightarrow \quad g(\vec{r}) = \frac{1}{\rho N}\left\langle \sum\sum_{i\neq j=1}^{N}\delta\left[\vec{r}-(\vec{r}_i-\vec{r}_j)\right]\right\rangle$$

$$\Rightarrow \quad g(r) = \int_0^{\pi}d\theta\int_0^{2\pi}d\varphi\int_r^{r+dr}dr'g(\vec{r}') = \frac{1}{\rho N\Delta V(r)}\left\langle \sum\sum_{i\neq j=1}^{N}\delta\left[\left|\vec{r}\right|-\left|(\vec{r}_i-\vec{r}_j)\right|\right]\right\rangle$$

with $\Delta V(r) = \frac{4\pi}{3}\left[(r+dr)^3 - r^3\right]$

- In practice, during the simulation, one should "fill" an histogram by increasing the bin by 2 at r whenever he finds two particles at distance between r and r+dr. At the end of the simulation he should normalize the histogram with the quantity $\rho N\Delta V(r)$.

9

Derivation of the Master equation

10

- Let's go back to a Markov process which, given two positive normalized functions p_1 and $p_{1|1}$, is fully characterized by
  - The Chapman-Kolmogorov equation

$$p_{1|1}(x_3,t_3|x_1,t_1) = \int p_{1|1}(x_3,t_3|x_2,t_2)p_{1|1}(x_2,t_2|x_1,t_1)\,dx_2 \quad for \quad t_3 \geq t_2 \geq t_1$$

  - The equation:

$$p_1(x_2,t_2) = \int p_{1|1}(x_2,t_2|x_1,t_1)p_1(x_1,t_1)\,dx_1$$

- According to these equations, the conditional probability is the crucial quantity determining in a Markov process the evolution with time of the probability distribution $p_1(x,t)$
- Let us consider the last equation for times t and t+τ, where τ is an infinitesimally short time interval. Proceeding in this way, we obtain the short-time evolution equation

$$p_1(x,t+\tau) = \int dy\, p_{1|1}(x,t+\tau|y,t)p_1(y,t)$$

- Next we can construct time derivatives of p_1

$$\frac{\partial p_1(x,t)}{\partial t} = \lim_{\tau\to 0}\frac{p_1(x,t+\tau)-p_1(x,t)}{\tau} =$$

$$= \lim_{\tau\to 0}\frac{1}{\tau}\left[\int p_{1|1}(x,t+\tau\,|\,y,t)\,p_1(y,t)\,dy - p_1(x,t)\right] = \cdots$$

- Since we will take the limit $\tau\to 0$, we can expand the transition probability $p_{1|1}$ in a power series in τ and keep only the lowest-order term. In order to conserve probability at all times, its most general form is:

$$p_{1|1}(x,t+\tau\,|\,y,t) \underset{\tau\to 0}{\approx} \delta(y-x)\left[1-\tau\int \left.\frac{\partial p_{1|1}(z,t'\,|\,y,t)}{\partial t'}\right|_{t'=t} dz\right] + \tau\left.\frac{\partial p_{1|1}(x,t'\,|\,y,t)}{\partial t'}\right|_{t'=t}$$

probability that no transition occurs during the time interval $t\to t+\tau$

probability of a transition from y to x during the time interval $t\to t+\tau$

$$p_{1|1}(x,t+\tau\,|\,y,t) \underset{\tau\to 0}{\approx} \delta(y-x)\left[1-\tau\int dz\, R_t\left(z\,|\,x\right)\right] + \tau R_t\left(x\,|\,y\right)$$

transition rates

- If we now substitute back $p_{1|1}$ we obtain

$$\frac{\partial p_1(x,t)}{\partial t} = \lim_{\tau\to 0}\frac{1}{\tau}\left\{\int\left[\delta(y-x)\left[1-\tau\int dz\, R_t\left(z\,|\,x\right)\right]+\tau R_t\left(x\,|\,y\right)\right]p_1(y,t)\,dy - p_1(x,t)\right\} =$$

$$= \int dy\, p_1(y,t)\left[R_t\left(x\,|\,y\right)-\delta(y-x)\int dz\, R_t\left(z\,|\,x\right)\right] = \int dy\, R_t\left(x\,|\,y\right)p_1(y,t) - \int dz\, R_t\left(z\,|\,x\right)p_1(x,t)$$

- Which lead us to the Master equation:

$$\frac{\partial p(x,t)}{\partial t} = \int dy\left[R_t\left(x\,|\,y\right)p(y,t) - R_t\left(y\,|\,x\right)p(x,t)\right]$$

- The quantities $R_t(x|y)$ are transition probabilities per time unit or transition rates. The Master equation can be read as a balance equation for probability flows: the change per unit of time of the probability of state x is the sum of two terms with opposite effects. Firstly, there is a probability flux from all other states y into the state x. Secondly, there is a probability flux out of state x into all other states y. The change per time unit of the probability p is caused by the difference of those probability fluxes.
- Discrete Master equation:

$$\frac{\partial p_n(t)}{\partial t} = \sum_m\left[R_t^{nm}p_m(t) - R_t^{mn}p_n(t)\right]$$

- The master equation is thus an equivalent form of the Chapman-Kolmogorov equation for Markov processes, but it is easier to handle and more directly related to physical concepts.
- In the master equation the quantities representing the knowledge about the system are the transition rates. One considers them as given by the specific system, and then has an equation for the probabilities which determine the statistical evolution of that system under conditions of uncertainty or restricted information.
- In social sciences, the transition rates can often be inferred from phenomenological considerations based on plausibility arguments.
- In natural sciences, fundamental considerations may be the starting point in the construction of the transition rates or the transition rates may be derived from first principles. Given the transition rates, the master equation is an instrument to model equilibrium and out of equilibrium statistical systems
- Let's make a step further and derive another relevant equation connected with a Markov process …

Kramers-Moyal expansion

Let us now set the stage for a further approximation of the master equation

$$\frac{\partial p(x,t)}{\partial t} = \int dy \left[R_t\left(x|y\right) p(y,t) - R_t\left(y|x\right) p(x,t) \right]$$

and make the following assumptions:

- First express the transition probability R_t as a function of the size of the jumps and of the starting point:
  we write $R_t(x|y) = R_t(x|x-r) =: R_t(x-r; r)$, with the 'jump distance' $r:=x-y$. Similarly, $R_t(y|x) = R_t(x-r|x) =: R_t(x; -r)$.
- There are only small jumps, i.e., $R_t(x-r; r)$ as a function of r is a sharply peaked function around r=0 :

$$\exists \delta > 0 : R_t\left(x - r; r\right) \approx 0, \quad |r| > \delta$$

- $R_t(x-r; r)$ is a slowly varying function of its first argument

$$\exists \delta' > 0 : R_t\left(x - r; r\right) \approx R_t\left(x; r\right), \quad |r| < \delta'$$

- Moreover R_t and p are sufficiently smooth functions of both arguments.
- We rewrite the Master equation as

$$\frac{\partial p(x,t)}{\partial t} = \int dr\, R_t\left(x-r;r\right)p(x-r,t) - p(x,t)\int dr\, R_t\left(x;-r\right)$$

- We can now perform a Taylor expansion in x - r around r = 0 in the first integral on the right-hand side:

$$\frac{\partial p(x,t)}{\partial t} = \int dr\left\{R_t\left(x\middle|r\right)p(x,t) - r\frac{\partial}{\partial x}\left[R_t\left(x\middle|r\right)p(x,t)\right] + \tfrac{1}{2}r^2\frac{\partial^2}{\partial x^2}\left[R_t\left(x\middle|r\right)p(x,t)\right] + \cdots\right\} +$$

- Yelding

$$\frac{\partial p(x,t)}{\partial t} = \sum_{n=1}^{\infty}\frac{(-1)^n}{n!}\frac{\partial^n}{\partial x^n}\left[a_n(x)p(x,t)\right]$$

$$-p(x,t)\int dr\, R_t\left(x;-r\right)$$

$$a_n(x) = \int_{-\infty}^{+\infty} dr\, r^n R_t(x;r)$$

- Which is the Kramers-Moyal expansion of the master equation. As such it is only a rewriting of the master equation, changing it from an integro-differential equation into a partial differential equation of infinite order.

Fokker-Planck Equation

- In deriving the Kramers-Moyal expansion, we have not yet made use of our assumption that the transition rates $R_t(x,r)$ are supposed to be slowly varying functions of their first argument. For these slowly varying functions (which will give rise to slowly varying probability densities p(x,t)) we can make the assumption that we can truncate the Kramers-Moyal expansion at a certain order of derivatives.
- A truncation after the second order leads to the so-called Fokker-Planck equation:

$$\frac{\partial p(x,t)}{\partial t} = -\frac{\partial}{\partial x}\left[p(x,t)\int dr\, r\, R_t\left(x;r\right)\right] + \tfrac{1}{2}\frac{\partial^2}{\partial x^2}\left[p(x,t)\int dr\, r^2\, R_t\left(x;r\right)\right]$$

$$\frac{\partial p(x,t)}{\partial t} = -\frac{\partial}{\partial x}\left[a_1(x)p(x,t)\right] + \frac{1}{2}\frac{\partial^2}{\partial x^2}\left[a_2(x)p(x,t)\right]$$

a_1 is called the drift coefficient and a_2 the diffusion coefficient.

- Note that $[a_n(x)\Delta t]$ are the n^{th} moment of the "distribution of step size in time Δt" (which depends on the starting point x)

$$a_n(x)\Delta t = \int_{-\infty}^{+\infty} dr\, r^n R_t(x;r)\Delta t$$

Brownian motion: again Diffusion

- In fact, let's go back to the Brownian motion (without drift) introduced in Lecture 3. Its coordinate x may be treated on a coarse time scale as a Markov process:

$$x(t+\Delta t) = x(t) + \sigma Z\sqrt{\Delta t} \qquad \text{being } Z\sim\mathcal{N}(0;1)$$

- Accordingly we have the picture of a particle that makes random jumps back and forth over the x-axis. The jumps may have any length, but the probability for large jumps falls off rapidly. Moreover the probability is symmetrical and independent of the starting point. Hence

$$a_1 = \int_{-\infty}^{+\infty} dr\, r\, R_t(x;r) = 0 \qquad a_2\Delta t = \int_{-\infty}^{+\infty} dr\, r^2 R_t(x;r)\Delta t = \sigma^2\Delta t$$

- The Fokker-Plank equation for p(x,t) is therefore the diffusion equation:

$$\frac{\partial p(x,t)}{\partial t} = \frac{a_2}{2}\frac{\partial^2}{\partial x^2}p(x,t) \quad \Rightarrow \quad a_2 = \sigma^2 = 2D$$

Consequently $a_2/2$ correspond to the diffusion constant D

Drift force & drift velocity

- We see that respect to the simpler Diffusion equation, the Fokker-Planck equation contains the possible presence of an external drift force.
- We now want to make a more general contact between the Fokker-Planck equation for the probability density of a Markov process and the equation governing the time development of the sample path x(t) of this process considering the addition of a drift velocity term to the Wiener process.
- A feasible form for this new term can be derived assuming that (think about the Brownian motion) in presence of an external drift force F and a finite viscosity, η, of the dense fluid the the Brownian particle will rapidly acquire a drift velocity:

$$m\ddot{x} = F - k\eta v \qquad \ddot{x} = 0 \Rightarrow \qquad v_\infty = \frac{1}{k\eta}F = -\frac{1}{k\eta}\frac{dU}{dx} = \mu$$

- By adding a drift velocity term to the Brownian motion we obtain

$$x(t+\Delta t) = x(t) + \mu\Delta t + Z\sigma\sqrt{\Delta t} \qquad \text{being } Z\sim\mathcal{N}(0;1)$$

as already discussed in lecture 3, and $a_1\Delta t = \mu\Delta t$.

Importance sampling with M(RT)^2

- Sometimes, the greatest shortcoming of the Metropolis algorithm is that the usual random displacement sampled from T(x|y) does not contain any information on p(x), the probability density we need to sample. This leads to a higher rejection rate than necessary
- Suppose now that p(x) can be factorized: $p(x)=d_1(x)\cdot d_2(x)$, and that we are already able to sample $d_1(x)$ directly, i.e., without the metropolis algorithm.
- We can try to use $d_1(x)$ in the construction of the random walk: the trial transition probability will consist in keeping $T(x|y)=d_1(x)$, what is the Metropolis acceptance of such a move?

$$A(x|y)=\min\left[1,\frac{T(y|x)\times p(x)}{T(x|y)\times p(y)}\right]=\min\left[1,\frac{d_1(y)\times d_1(x)d_2(x)}{d_1(x)\times d_1(y)d_2(y)}\right]=\min\left[1,\frac{d_2(x)}{d_2(y)}\right]$$

  Thus in the "limit" $p(x)=d_1(x)$ and $d_2(x)=1$, we realize the "perfect" sampling: A(x|y)=1 always!
- More realistically, when $d_2(x)\neq 1$ we have reduced the computational effort in the calculation of A(x|y) and being T(x|y) more "similar" to p(x), we have improved the efficiency in the algorithm, i.e., the acceptance A(x|y) on average will grow

- Such a factorization is therefore something that one should always try to realize.
- However there are cases where this is not possible and where the efficiency of the sampling obtained by proposing usual random moves is too much inefficient. Have we other possibilities in these cases? Yes, via what is called "smart Monte Carlo" or "Langevin Monte Carlo".
- This different approach includes importance sampling based on the Fokker-Planck equation: Consider a diffusion process characterized by a time-dependent probability density p(x,t).
- It is known that the probability density of a diffusion processes with drift obeys the following multivariate Fokker-Planck equation:

$$\frac{\partial p(\vec{x},t)}{\partial t}=\sum_i \frac{\partial}{\partial x_i}\left[D\frac{\partial}{\partial x_i}p(\vec{x},t)-\gamma F_i(\vec{x})p(\vec{x},t)\right]$$

  where D is the diffusion constant and F_i is the i-th component of a drift force caused by an external potential.
- As usual, we wish to converge to the stationary probability density p(x)...

21

- An unchanging probability, for which $\partial p/\partial t=0$, may be obtained by setting the left-hand side of the previous equation to zero, namely

$$\sum_i D\left\{\frac{\partial^2 p(\vec{x})}{\partial x_i^2} - \frac{\gamma}{D}\frac{\partial}{\partial x_i}\left[F_i(\vec{x})p(\vec{x})\right]\right\} = 0$$

- This equation can most readily be satisfied if each term of the previous sum vanishes, yielding

$$\frac{\partial^2 p(\vec{x})}{\partial x_i^2} = \frac{\gamma}{D}\left[p(\vec{x})\frac{\partial}{\partial x_i}F_i(\vec{x}) + F_i(\vec{x})\frac{\partial}{\partial x_i}p(\vec{x})\right]$$

- The drift velocity F therefore must be of the form $F_i(\vec{x}) = \frac{D}{\gamma}g(p)\frac{\partial p(\vec{x})}{\partial x_i}$ in order to obtain a second derivative of p on the right-hand side. If we substitute this form of F into the previous equation we find

$$\frac{\partial^2 p}{\partial x_i^2} = p\frac{\partial g}{\partial p}\left(\frac{\partial p}{\partial x_i}\right)^2 + pg\frac{\partial^2 p}{\partial x_i^2} + g\left(\frac{\partial p}{\partial x_i}\right)^2$$

- Cancellation of the second derivative terms requires that $g=1/p$. This choice also leads to cancellation of the first derivative terms.

- Therefore the stationary probability density p(x) results from choosing the drift vector to be:

$$\vec{F}(\vec{x}) = \frac{D}{\gamma}g(p)\vec{\nabla}p(\vec{x}) = \frac{D}{\gamma}\frac{1}{p(\vec{x})}\vec{\nabla}p(\vec{x}) = \boxed{\frac{D}{\gamma}\vec{\nabla}\ln p(\vec{x})}$$

Clearly this drift causes the move to be "biased" by p(x). This biased diffusion process incorporates importance sampling.

- We now have a diffusion equation which gives the desired probability distribution, but how do we implement it using Monte Carlo?
- We have seen that Fokker-Planck trajectories are generated by means of Brownian motion with drift:

$$\vec{x}(t+\Delta t) = \vec{x}(t) + D\,\Delta t\,\vec{\nabla}\ln p(\vec{x}) + \vec{\chi}$$

here χ is a randomly fluctuating displacement which is distributed according to a multi-dimensional Gaussian with a mean of zero and a variance of $2D\Delta t$, i.e., proportional to what is called a multivariate Gaussian white noise.

- By using a discretized finite time step we have introduced a bias into the stochastic dynamics for any $\Delta t > 0$ due to the approximations introduced to derive the Fokker-Plank equation.

22

Smart Monte Carlo method

- Do we have a method to correct this bias and obtain the correct sampling of p(x)? Yes, again the Metropolis!
- In fact, we have obtained a dynamical equation for generating guided trajectories containing a drift force: $F(x)=(D/\gamma)\,\nabla \ln p(x)$. This dynamical equation directs "walkers" toward regions of large p(x), but suffers by being exact only at $\Delta t=0$. We will now show that by combining this equation with a Metropolis algorithm the time step bias can be eliminated.
- From the generalized Metropolis method recall that the complete transition probability K(x|y) is composed by a trial transition probability T(x|y) times an acceptance probability A(x|y) of the proposed move y→x. We can use this approach, and choose the T(x|y) which corresponds to the previously discretized Langevin equation.
- In order to be able to do this, we have to find the functional form of T(x|y) which corresponds to the previous Brownian motion difference equation with the added condition that $T(x|y, \Delta t=0)=\delta(x-y)$

23

- We realize that if χ is a multivariate Gaussian random variable with zero mean and variance $2D\Delta t$, also

$$\vec{x} - \vec{y} - (D\,\Delta t)\,\vec{\nabla} \ln p(\vec{y}) = \vec{\chi}$$

is such a Gaussian random variable; therefore we can choose

$$T(\vec{x}\,|\,\vec{y}) \propto e^{-\frac{\left\{\vec{x}-\vec{y}-(D\Delta t)\,\vec{\nabla}\ln p(\vec{y})\right\}^2}{4D\Delta t}}$$

Thus x is a Gaussian random variable with mean $y+(D\Delta t)\nabla \ln[p(y)]$ and variance $2D\Delta t$; we have already learned how to sample such Gaussian random variables via the Box-Muller method.

- Note that in general , being T(x|y) different from T(y|x), for the Metropolis acceptance probability we need the calculation of :

$$\frac{T(\vec{y}\,|\,\vec{x})}{T(\vec{x}\,|\,\vec{y})} = \frac{e^{-\frac{\left\{\vec{y}-\vec{x}-D\Delta t\,\vec{\nabla}\ln p(\vec{x})\right\}^2}{4D\Delta t}}}{e^{-\frac{\left\{\vec{x}-\vec{y}-D\Delta t\,\vec{\nabla}\ln p(\vec{y})\right\}^2}{4D\Delta t}}}$$

- $(D\Delta t)$ turns out to be an adjustable parameter to force $A(x|y)\approx 50\%$

24

Re-weighting

- Suppose we have to compute the following series of integrals

$$I_a = \int_\Omega d\vec{x}\, g(\vec{x}) \frac{p_a(\vec{x})}{\int_\Omega d\vec{x}\, p_a(\vec{x})} \quad with \quad p_a(\vec{x}) \geq 0 \quad \forall \vec{x} \in \Omega$$

  where the probability density p_a depends on the parameter a.

- When we have computed the previous integral for a particular value of a ($a=\alpha$) via Monte Carlo by using points in $\mathbb{R}^n$ sampled form the probability distribution $p_{a=\alpha}$, the re-weighting technique enables us to compute, in principle, the whole series of integrals I_a by using the same single set of points sampled from $p_{a=\alpha}$.

- In principle the technique is very powerful, however it has some limitations due to the variance of the calculations of the integrals I_a , for $a \neq \alpha$, which can grow in an uncontrolled way. This happens when p_a becomes too different from p_α for $a \neq \alpha$; in fact in this case we are implementing the "importance sampling" in a bad way thus inducing an increment of the variance in our calculation of I_a.

- One could have problems with the re-weighting technique also when p_a is "pathologic" (e.g. non continuous) as a function of the parameter a.

- To implement the re-weighting technique, we need to know what are the new functions we have to evaluate, on the points sampled with p_α , in place of the function g. Such functions, obviously, must depend on the parameter a:

$$I_{a(\neq\alpha)} = \int_\Omega d\vec{x}\, g(\vec{x}) \frac{p_a(\vec{x})}{\int_\Omega d\vec{x}\, p_a(\vec{x})} = \frac{\int_\Omega d\vec{x}\, g(\vec{x}) p_a(\vec{x})}{\int_\Omega d\vec{x}\, p_a(\vec{x})} =$$

$$= \frac{\int_\Omega d\vec{x}\, g(\vec{x}) \frac{p_a(\vec{x})}{p_\alpha(\vec{x})} p_\alpha(\vec{x})}{\int_\Omega d\vec{x}\, \frac{p_a(\vec{x})}{p_\alpha(\vec{x})} p_\alpha(\vec{x})} = \frac{\int_\Omega d\vec{x}\, \left[g(\vec{x}) w_a(\vec{x})\right] p_\alpha(\vec{x})}{\int_\Omega d\vec{x}\, w_a(\vec{x}) p_\alpha(\vec{x})} \Bigg|_{w_a(\vec{x}) = \frac{p_a(\vec{x})}{p_\alpha(\vec{x})}} =$$

$$= \frac{\int_\Omega d\vec{x}\, \left[g(\vec{x}) w_a(\vec{x})\right] \frac{p_\alpha(\vec{x})}{\int_\Omega d\vec{x}\, p_\alpha(\vec{x})}}{\int_\Omega d\vec{x}\, w_a(\vec{x}) \frac{p_\alpha(\vec{x})}{\int_\Omega d\vec{x}\, p_\alpha(\vec{x})}} \approx \frac{\frac{1}{N}\sum_{i=1}^N g(\vec{x}_i) w_a(\vec{x}_i)}{\frac{1}{N}\sum_{i=1}^N w_a(\vec{x}_i)} \Bigg|_{\vec{x}_i \text{ sampled with } p_\alpha(\vec{x})}$$

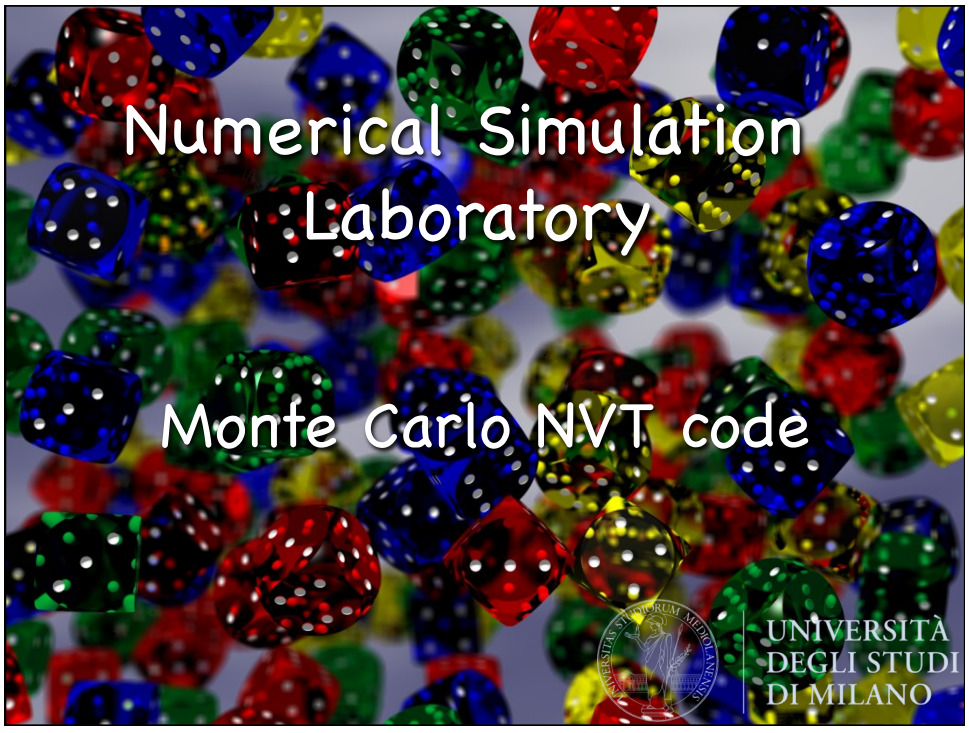
Ratio of two Monte Carlo estimations

- Importance sampling $\Rightarrow w_a \approx 1$

Lecture 7: Suggested books

- M.P. Allen and D.J. Tildesley, *Computer Simulation of Liquids* - Clarendon Press
- D. Frenkel and B. Schmidt, *Understanding Molecular Simulation, From Algorithms to Applications* – Academic Press
- L. Hammond, W. A. Lester, Jr., P. J. Reynolds, *Monte Carlo Methods in Ab Initio Quantum Chemistry* - World Scientific (1994)
- W. Paul, J. Baschnagel, *Stochastic Processes. From Physics to Finance*, Springer (2000)
- N.G. Van Kampen, *Stochastic Processes in Physics and Chemistry*, Elsevier (2007)

27



Monte Carlo NVT code: main

```
#include<iostream>
#include<fstream>
#include<ostream>
#include<cmath>
#include<iomanip>
#include"Monte_Carlo_NVT.h"
#include"random.h"
using namespace std;

int main(){
  Input(); //Inizialization
  int nconf = 1;
  for(int iblk=1; iblk <= nblk; ++iblk) //Simulation
  {
    Reset(iblk);   //Reset block averages
    for(int istep=1; istep <= nstep; ++istep)
    {
      Move();
      Measure();
      Accumulate(); //Update block averages
      if(istep%10 == 0){
        ConfXYZ(nconf);//Write actual configuration in XYZ format
        nconf += 1;
      }
    }
    Averages(iblk);   //Print results for current block
  }
  ConfFinal(); //Write final configuration
  return 0;
}
```

Monte Carlo NVT code: Input

```
#include<iostream>
#include<fstream>
#include<ostream>
#include<cmath>
#include<iomanip>
#include"Monte_Carlo_NVT.h"
#include"random.h"
using namespace std;

int main(){
  Input(); //Inizialization
  int nconf = 1;
  for(int iblk=1; iblk <= nblk; ++iblk) //Simulation
  {
    Reset(iblk);   //Reset block averages
    for(int istep=1; istep <= nstep; ++istep)
    {
      Move();
      Measure();
      Accumulate(); //Update block averages
      if(istep%10 == 0){
        ConfXYZ(nconf);//Write actual configuration in XYZ format
        nconf += 1;
      }
    }
    Averages(iblk);   //Print results for current block
  }
  ConfFinal(); //Write final configuration
  return 0;
}
```

Monte Carlo NVT code: Input .1

```
void Input(void)
{
  int seed[3];
  ifstream ReadInput,ReadConf;
  cout << "Classic Lennard-Jones fluid        " << endl;
  cout << "Monte Carlo simulation             " << endl << endl;
  cout << "Interatomic potential v(r) = 4 * [(1/r)^12 - (1/r)^6]" << endl << endl;
  cout << "Boltzmann weight exp(- beta * sum_{i<j} v(r_ij) ), beta = 1/T " << endl << endl;
  cout << "The program uses Lennard-Jones units " << endl;

//Read seed for random numbers
  int p1, p2;
  ifstream Primes("Primes");
  Primes >> p1 >> p2 ;
  Primes.close();
  ifstream input("seed.in");
  input >> seed[0] >> seed[1] >> seed[2] >> seed[3];
  rnd.SetRandom(seed,p1,p2);
  input.close();

//Read input informations
  ReadInput.open("input.dat");
  ReadInput >> temp;
  beta = 1.0/temp;
  cout << "Temperature = " << temp << endl;

  ReadInput >> npart;
  cout << "Number of particles = " << npart << endl;
  ReadInput >> rho;
  cout << "Density of particles = " << rho << endl;
  vol = (double)npart/rho;
  box = pow(vol,1.0/3.0);
  cout << "Volume of the simulation box = " << vol << endl;
  cout << "Edge of the simulation box = " << box << endl;
```

Input.dat

```
1.1
108
0.8
2.5
0.26
30
1000
```

```
  ReadInput >> temp;
  ReadInput >> npart;
  ReadInput >> rho;
  ReadInput >> rcut;
  ReadInput >> delta;
  ReadInput >> nblk;
  ReadInput >> nstep;
```

.2

```
  ReadInput >> rcut;
  cout << "Cutoff of the interatomic potential = " << rcut << endl << endl;

  //Tail corrections for potential enery and pressure
  vtail = (8.0*pi*rho)/(9.0*pow(rcut,9)) - (8.0*pi*rho)/(3.0*pow(rcut,3));
  ptail = (32.0*pi*rho)/(9.0*pow(rcut,9)) - (16.0*pi*rho)/(3.0*pow(rcut,3));
  cout << "Tail correction for the potential energy = " << vtail << endl;
  cout << "Tail correction for the virial           = " << ptail << endl;

  ReadInput >> delta;

  ReadInput >> nblk;

  ReadInput >> nstep;

  cout << "The program perform Metropolis moves with uniform translations" << endl;
  cout << "Moves parameter = " << delta << endl;
  cout << "Number of blocks = " << nblk << endl;
  cout << "Number of steps in one block = " << nstep << endl << endl;
  ReadInput.close();


//Prepare arrays for measurements
  n_obs = 2; //Number of properties of the "walker"

  iv = 0; //Potential energy
  iw = 1; //Virial

  n_props = 2; //Number of observables

//measurement of g(r)
  igofr = 2;
  nbins = 100;
  n_props = n_props + nbins;
  bin_size = (box/2.0)/(double)nbins;
```

.3

```
//Read initial configuration
  cout << "Read initial configuration from file config.0 " << endl << endl;
  ReadConf.open("config.0");
  for (int i=0; i<npart; ++i)
  {
    ReadConf >> x[i] >> y[i] >> z[i];
    x[i] = Pbc( x[i] * box );
    y[i] = Pbc( y[i] * box );
    z[i] = Pbc( z[i] * box );
  }
  ReadConf.close();

//Evaluate potential energy and virial of the initial configuration
  Measure();

//Print initial values for the potential energy and virial
  cout << "Initial potential energy (with tail corrections) = " << walker[iv]/(double)npart +
vtail << endl;
  cout << "Virial                   (with tail corrections) = " << walker[iw]/(double)npart +
ptail << endl;
  cout << "Pressure                 (with tail corrections) = " << rho * temp + (walker[iw] +
(double)npart * ptail) / vol << endl << endl;
}
```

Monte Carlo NVT code: Pbc

```
double Pbc(double r)
//Algorithm for periodic boundary conditions with side L=box
{
    return r - box * rint(r/box);
}
```

Monte Carlo NVT code: Measure

```
#include<iostream>
#include<fstream>
#include<ostream>
#include<cmath>
#include<iomanip>
#include"Monte_Carlo_NVT.h"
#include"random.h"
using namespace std;

int main(){
  Input(); //Inizialization
  int nconf = 1;
  for(int iblk=1; iblk <= nblk; ++iblk) //Simulation
  {
    Reset(iblk);   //Reset block averages
    for(int istep=1; istep <= nstep; ++istep)
    {
      Move();
      Measure();
      Accumulate(); //Update block averages
      if(istep%10 == 0){
        ConfXYZ(nconf);//Write actual configuration in XYZ format
        nconf += 1;
      }
    }
    Averages(iblk);   //Print results for current block
  }
  ConfFinal(); //Write final configuration
  return 0;
}
```

Monte Carlo NVT code: Measure

```
void Measure()
{
  int bin;
  double v = 0.0, w = 0.0;
  double vij, wij;
  double dx, dy, dz, dr;
//reset the hystogram of g(r)
  for (int k=igofr; k<igofr+nbins; ++k) walker[k]=0.0;
//cycle over pairs of particles
  for (int i=0; i<npart-1; ++i)
  {
    for (int j=i+1; j<npart; ++j)
    {
// distance i-j in pbc
     dx = Pbc(x[i] - x[j]);
     dy = Pbc(y[i] - y[j]);
     dz = Pbc(z[i] - z[j]);
     dr = dx*dx + dy*dy + dz*dz;
     dr = sqrt(dr);

//update of the histogram of g(r)

     if(dr < rcut) // contribution to energy and virial
     {
       vij = 1.0/pow(dr,12) - 1.0/pow(dr,6);
       wij = 1.0/pow(dr,12) - 0.5/pow(dr,6);
       v += vij;
       w += wij;
     }
    }
  }

  walker[iv] = 4.0 * v;
  walker[iw] = 48.0 * w / 3.0;
}
```

insert your code here

Monte Carlo NVT code: Reset

```
#include<iostream>
#include<fstream>
#include<ostream>
#include<cmath>
#include<iomanip>
#include"Monte_Carlo_NVT.h"
#include"random.h"
using namespace std;

int main(){
  Input(); //Inizialization
  int nconf = 1;
  for(int iblk=1; iblk <= nblk; ++iblk) //S
  {
    Reset(iblk);   //Reset block averages
    for(int istep=1; istep <= nstep; ++iste
    {
      Move();
      Measure();
      Accumulate(); //Update block averages
      if(istep%10 == 0){
        ConfXYZ(nconf);//Write actual configuration in XYZ format
        nconf += 1;
      }
    }
    Averages(iblk);   //Print results for current block
  }
  ConfFinal(); //Write final configuration
  return 0;
}
```

```
void Reset(int iblk) //Reset block averages
{

   if(iblk == 1)
   {
       for(int i=0; i<n_props; ++i)
       {
           glob_av[i] = 0;
           glob_av2[i] = 0;
       }
   }

   for(int i=0; i<n_props; ++i)
   {
     blk_av[i] = 0;
   }
   blk_norm = 0;
   attempted = 0;
   accepted = 0;
}
```

Monte Carlo NVT code: Move

```
#include<iostream>
#include<fstream>
#include<ostream>
#include<cmath>
#include<iomanip>
#include"Monte_Carlo_NVT.h"
#include"random.h"
using namespace std;

int main(){
  Input(); //Inizialization
  int nconf = 1;
  for(int iblk=1; iblk <= nblk; ++iblk) //Simulation
  {
    Reset(iblk);   //Reset block averages
    for(int istep=1; istep <= nstep; ++istep)
    {
      Move();
      Measure();
      Accumulate(); //Update block averages
      if(istep%10 == 0){
        ConfXYZ(nconf);//Write actual configuration in XYZ format
        nconf += 1;
      }
    }
    Averages(iblk);   //Print results for current block
  }
  ConfFinal(); //Write final configuration
  return 0;
}
```

Monte Carlo NVT code: Move

```
void Move(void)
{
  int o;
  double p, energy_old, energy_new;
  double xold, yold, zold, xnew, ynew, znew;

  for(int i=0; i<npart; ++i)
  {
  //Select randomly a particle (for C++ syntax, 0 <= o <= npart-1)
    o = (int)(rnd.Rannyu()*npart);

  //Old
    xold = x[o];
    yold = y[o];
    zold = z[o];
    energy_old = Boltzmann(xold,yold,zold,o);

  //New
    xnew = Pbc( x[o] + delta*(rnd.Rannyu() - 0.5));
    ynew = Pbc( y[o] + delta*(rnd.Rannyu() - 0.5));
    znew = Pbc( z[o] + delta*(rnd.Rannyu() - 0.5));
    energy_new = Boltzmann(xnew,ynew,znew,o);

  //Metropolis test
    p = exp(beta*(energy_old-energy_new));
    if(p >= rnd.Rannyu())
    {
    //Update
       x[o] = xnew;
       y[o] = ynew;
       z[o] = znew;
       accepted = accepted + 1.0;
    }
    attempted = attempted + 1.0;
  }
}
```

```
double Boltzmann(double xx, double yy,
double zz, int ip)
{
  double ene=0.0;
  double dx, dy, dz, dr;

  for (int i=0; i<npart; ++i)
  {
    if(i != ip)
    {
// distance ip-i in pbc
      dx = Pbc(xx - x[i]);
      dy = Pbc(yy - y[i]);
      dz = Pbc(zz - z[i]);

      dr = dx*dx + dy*dy + dz*dz;
      dr = sqrt(dr);

      if(dr < rcut)
      {
        ene += 1.0/pow(dr,12)-1.0/pow(dr,6);
      }
    }
  }

  return 4.0*ene;
}
```

MC NVT code: Accumulate

```
#include<iostream>
#include<fstream>
#include<ostream>
#include<cmath>
#include<iomanip>
#include"Monte_Carlo_NVT.h"
#include"random.h"
using namespace std;

int main(){
  Input(); //Inizialization
  int nconf = 1;
  for(int iblk=1; iblk <= nblk; ++iblk) //Simulation
  {
    Reset(iblk);   //Reset block averages
    for(int istep=1; istep <= nstep; ++istep)
    {
      Move();
      Measure();
      Accumulate(); //Update block averages
      if(istep%10 == 0){
        ConfXYZ(nconf);//Write actual configuration in XYZ format
        nconf += 1;
      }
    }
    Averages(iblk);   //Print results for current block
  }
  ConfFinal(); //Write final configuration
  return 0;
}
```

```
void Accumulate(void) //Update block averages
{

   for(int i=0; i<n_props; ++i)
   {
     blk_av[i] = blk_av[i] + walker[i];
   }
   blk_norm = blk_norm + 1.0;
}
```

Monte Carlo NVT code: Averages

```
#include<iostream>
#include<fstream>
#include<ostream>
#include<cmath>
#include<iomanip>
#include"Monte_Carlo_NVT.h"
#include"random.h"
using namespace std;

int main(){
  Input(); //Inizialization
  int nconf = 1;
  for(int iblk=1; iblk <= nblk; ++iblk) //Simulation
  {
    Reset(iblk);   //Reset block averages
    for(int istep=1; istep <= nstep; ++istep)
    {
      Move();
      Measure();
      Accumulate(); //Update block averages
      if(istep%10 == 0){
        ConfXYZ(nconf);//Write actual configuration in XYZ format
        nconf += 1;
      }
    }
    Averages(iblk);   //Print results for current block
  }
  ConfFinal(); //Write final configuration
  return 0;
}
```

Monte Carlo NVT code: Averages

```
void Averages(int iblk) //Print results for current block
{
   double err, r, gdir;
   ofstream Gofr, Gave, Epot, Pres;
   const int wd=12;

    cout << "Block number " << iblk << endl;
    cout << "Acceptance rate " << accepted/attempted << endl << endl;

    Epot.open("output.epot.0",ios::app); Pres.open("output.pres.0",ios::app);
    Gofr.open("output.gofr.0",ios::app); Gave.open("output.gave.0",ios::app);

    stima_pot = blk_av[iv]/blk_norm/(double)npart + vtail; //Potential energy
    glob_av[iv] += stima_pot;
    glob_av2[iv] += stima_pot*stima_pot;
    err_pot=Error(glob_av[iv],glob_av2[iv],iblk);

    stima_pres = rho * temp + (blk_av[iw]/blk_norm + ptail * (double)npart) / vol; //Pressure
    glob_av[iw] += stima_pres;
    glob_av2[iw] += stima_pres*stima_pres;
    err_press=Error(glob_av[iw],glob_av2[iw],iblk);

//Potential energy per particle
    Epot << setw(wd) << iblk <<  setw(wd) << stima_pot << setw(wd) << glob_av[iv]/(double)iblk
<< setw(wd) << err_pot << endl;
//Pressure
    Pres << setw(wd) << iblk <<  setw(wd) << stima_pres << setw(wd) << glob_av[iw]/
(double)iblk << setw(wd) << err_press << endl;

//g(r)

    cout << "----------------------------" << endl << endl;

    Epot.close();
    Pres.close();
    Gofr.close();
}
```

insert your code here

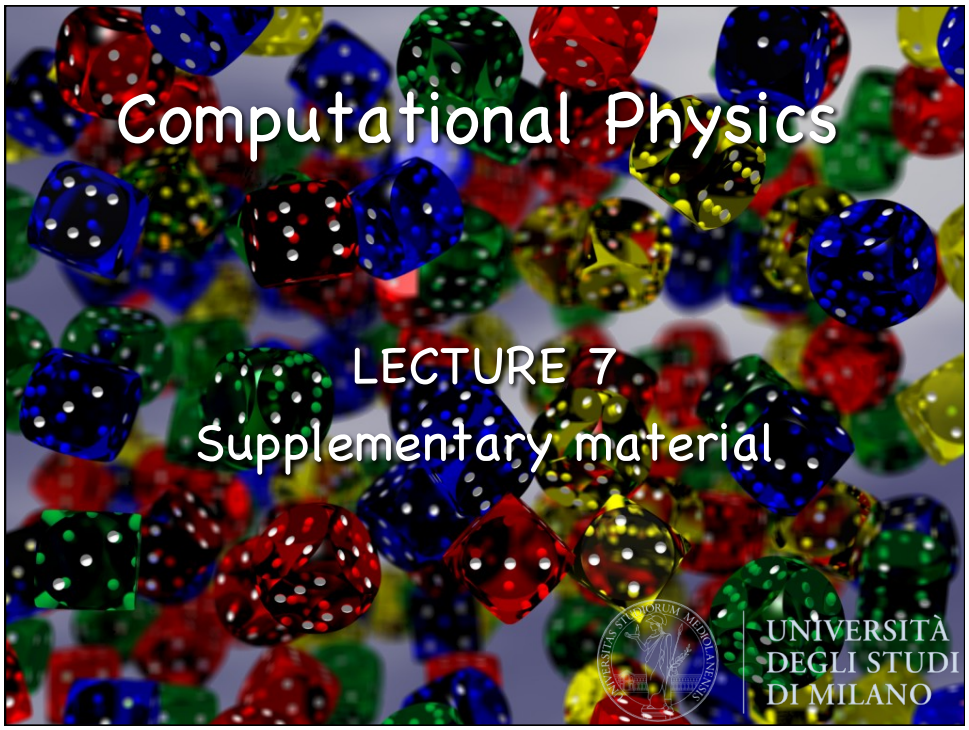
```
double Error(double sum, double sum2, int iblk){
    if(iblk == 1) return 0.0;
    else return sqrt((sum2/(double)iblk - pow(sum/(double)iblk,2))/(double)(iblk-1));
}
```

Monte Carlo NVT code: Conf*

```
#include<iostream>
#include<fstream>
#include<ostream>
#include<cmath>
#include<iomanip>
#include"Monte_Carlo_NVT.h"
#include"random.h"
using namespace std;

int main(){
  Input(); //Inizialization
  int nconf = 1;
  for(int iblk=1; iblk <= nblk; ++iblk) //Simulation
  {
    Reset(iblk);   //Reset block averages
    for(int istep=1; istep <= nstep; ++istep)
    {
      Move();
      Measure();
      Accumulate(); //Update block averages
      if(istep%10 == 0){
        ConfXYZ(nconf);//Write actual configuration in XYZ format
        nconf += 1;
      }
    }
    Averages(iblk);   //Print results for current block
  }
  ConfFinal(); //Write final configuration
  return 0;
}
```

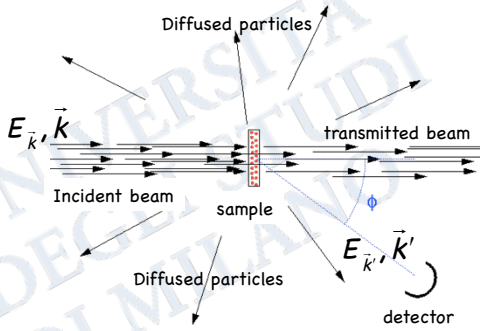
Monte Carlo NVT code: Conf*

```
void ConfXYZ(int nconf){ //Write configuration in .xyz format
  ofstream WriteXYZ;

  WriteXYZ.open("frames/config_" + to_string(nconf) + ".xyz");
  WriteXYZ << npart << endl;
  WriteXYZ << "This is only a comment!" << endl;
  for (int i=0; i<npart; ++i){
    WriteXYZ << "LJ  " << Pbc(x[i]) << "   " <<  Pbc(y[i]) << "   " << Pbc(z[i]) << endl;
  }
  WriteXYZ.close();
}

void ConfFinal(void)
{
  ofstream WriteConf;

  cout << "Print final configuration to file config.final " << endl << endl;
  WriteConf.open("config.final");
  for (int i=0; i<npart; ++i)
  {
    WriteConf << x[i]/box << "   " <<  y[i]/box << "   " << z[i]/box << endl;
  }
  WriteConf.close();

  rnd.SaveSeed();
}
```



Scattering experiments

- In a typical scattering (diffusion) experiment a beam of "particles" (photons, neutrons, electrons,...) with linear momentum $\hbar\mathbf{k}$ and energy E_k is radiated over a sample.



- The scattering experiments analyses how these "particles" are diffused, i.e. one counts how many "particles" of the incident beam come out with linear momentum $\hbar\mathbf{k}'$ and energy $E_{k'}$

Exchanged momentum: $\hbar\mathbf{q} = \hbar(\mathbf{k}-\mathbf{k}')$

Exchanged energy: $\hbar\omega = E_k - E_{k'}$

- Hypothesis: the "particles" of the incident beam interact weakly with the sample

45

1st Born approximation

- This means that most of the "probe-particles" do not exchange energy or momentum with the sample, thus they are found in the transmitted beam
- In such case we can treat the interaction term among "particles" and the sample at first order of perturbative theory, which is called 1st Born approximation
- The 1st Born approximation, in practice, assumes that the field inside the scattering volume is constant and equal to the incident field
- This is the case of thermal neutrons ($E_k \approx k_B T$) , and of photons (visible light and X-rays) scattered by a transparent sample, i.e. matter which is not reflecting like a metal for visible light and that does not present absorption for the considered photon frequency
- To be concrete we will discuss an incident beam of non-relativistic massive particles, i.e. the ("thermal") neutron scattering case

46

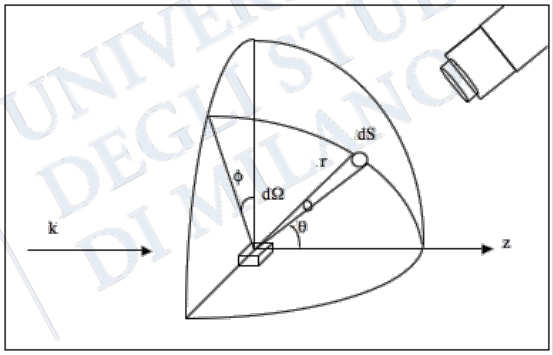
Neutron scattering

- Since neutrons are neutral, their interaction with the nuclei is short-ranged; in contrast to electrons, they penetrate deep into the solid.
- Furthermore, due to their magnetic moment and the associated dipole interaction with magnetic moments of the solid, neutrons can also be used to investigate magnetic properties like magnetic order in the system or dynamical properties: spin waves
- Therefore, the usefulness of neutron scattering originates from the comparatively simple and weak interaction of the neutrons with condensed matter and the well adapted "mechanical properties" (λ_{DB}≈inter-atomic distance for "thermal" energies) of the neutron beams to the microscopic and energetic structure of matter
- The weakness of the interaction renders the system in the ground state for elastic scattering or in a well defined excited state for inelastic scattering

47

Scattering cross sections

- Consider a beam of thermal (i.e. cooled down [moderated] to thermal energy 2-100 meV [30 <T< 1100 K]) neutrons, all with the same energy E (monochromatic beam) incident on a target
- Suppose we set up a neutron counter and measure the number of neutrons scattered in a given direction as function of their energy E'.



- The distance of the counter from the target is assumed to be large compared with to the dimensions of the counter and the target, so that the small angle dΩ subtended by the counter at the target is well defined. Let the directions of the scattered neutron be θ and ϕ

48

- The partial differential cross section is defined as:

$$\frac{d^2\sigma}{d\Omega dE'} = \frac{\text{(number of neutrons scattered per second into a small solid angle } d\Omega \text{ in the direction } \theta \text{ and } \phi \text{ with final energy between E' and E'+dE')}}{(\ \Phi\ d\Omega\ dE'\ )}$$

where Φ is the flux of the incident neutrons, i.e. the number through unit area per second

- Suppose we do not analyse the energy of the scattered neutrons, but simply count all the neutrons scattered into a solid angle $d\Omega$ in the direction θ and ϕ . The cross section corresponding to these measurements, known as the differential cross section, is defined by

$$\frac{d\sigma}{d\Omega} = \frac{\text{(number of neutrons scattered per second into } d\Omega \text{ in the direction } \theta, \phi)}{(\ \Phi\ d\Omega\ )}$$

- The total scattering cross section is defined by the equation

$$\sigma_{tot} = \frac{\text{(total number of neutrons scattered per second)}}{(\ \Phi\ )}$$

49

- From their definitions the three cross-sections are related by the following equations

$$\frac{d\sigma}{d\Omega} = \int_0^\infty dE' \left(\frac{d^2\sigma}{d\Omega dE'}\right) \quad ; \quad \sigma_{tot} = \int d\Omega \left(\frac{d\sigma}{d\Omega}\right)$$

- The dimensions of the total cross-section is [area], and more precisely is an area rescaled with the ratio of two pure numbers: the number of scattered neutrons divided by the number of incoming neutrons, i.e. the percentage of the scattered neutrons; it is a sort of measure of the "surface" able to deviate the beam. The definition of cross-sections apply to any kind of scattering

- The nuclear forces which cause the scattering have a range of about 10^{-14} to 10^{-15} m; the wavelength of thermal neutrons is of the order of 10^{-10} m, and is thus much larger than this range. In these circumstances the scattering, analysed in terms of partial waves, comes entirely from s waves (l=0)

- In fact, if waves of any kind are scattered by an object small compared to the wavelength of the waves, then the scattered wave is spherically symmetric

50

The dynamic structure factor

- In the 1th Born approximation where the Fermi's golden rule applies one can easily obtain that:

$$\frac{d^2\sigma}{d\Omega dE'} = \frac{1}{N}\frac{k'}{k}\frac{a^2}{2\pi\hbar}\int_{-\infty}^{\infty} dt e^{i\omega t}\left\langle \sum_j e^{-i\vec{q}\cdot\vec{R}_j(t)} \sum_j e^{i\vec{q}\cdot\vec{R}_j} \right\rangle = \frac{k'}{k}\frac{a^2}{\hbar} S(\vec{q},\omega)$$

where we have introduced the dynamic structure factor defined as

$$S(\vec{q},\omega) = \frac{1}{2\pi N}\int_{-\infty}^{\infty} dt e^{i\omega t}\left\langle \sum_j e^{-i\vec{q}\cdot\vec{R}_j(t)} \sum_j e^{i\vec{q}\cdot\vec{R}_j} \right\rangle = \frac{1}{2\pi N}\int_{-\infty}^{\infty} dt e^{i\omega t}\left\langle \hat{\rho}_{\vec{q}}(t)\hat{\rho}^{+}_{\vec{q}}(0) \right\rangle$$

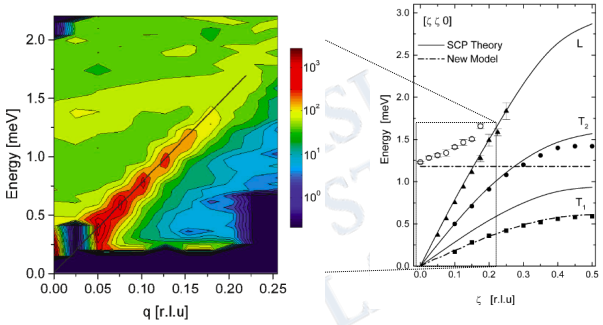
- It is very easy to see that ρ_q , which is generally called the density fluctuation operator, is the Fourier transform of the number density operator:

$$\hat{\rho}(\vec{r},t) = \sum_j \delta(\vec{r} - \vec{r}_j(t)) \quad \Rightarrow \quad \hat{\rho}_{\vec{q}}(t) = \int d\vec{r} e^{-i\vec{q}\cdot\vec{r}}\hat{\rho}(\vec{r},t) = \sum_j e^{-i\vec{q}\cdot\vec{r}_j(t)}$$

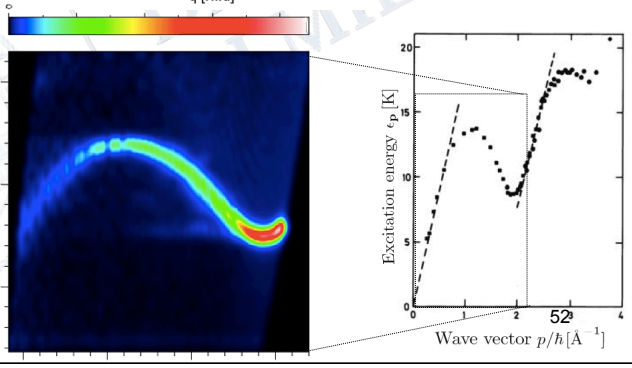
51

- We give her two examples:

Inelastic scattering: S(**q**,ω) in solids (phonons)



Inelastic scattering: S(**q**,ω) in He-II



52

The static structure factor

- The differential scattering cross-section is connected with a scattering experiment in which one registers all events related to a momentum transfer equal to $\hbar\mathbf{q}$ but independently on any particular energy transfer
- It is obtained by integrating with respect to the energy the partial differential scattering cross-section:

$$\frac{d\sigma}{d\Omega} = \int dE \frac{d^2\sigma}{d\Omega dE} = \hbar \int_{-\infty}^{\infty} d\omega \frac{k'}{k} \frac{a^2}{\hbar} S(\vec{q},\omega) = \frac{k'}{k} a^2 \int_{-\infty}^{\infty} d\omega\, S(\vec{q},\omega) = \frac{k'}{k} a^2 S(\vec{q})$$

it then proportional to the function S(q) which is called static structure factor

- This function is also easily obtained in a scattering experiment and it is very important because give information on the static structure of the system:

$$S(\vec{q}) = \int_{-\infty}^{\infty} d\omega\, S(\vec{q},\omega) = \int_{-\infty}^{\infty} d\omega \frac{1}{2\pi N} \int_{-\infty}^{\infty} dt e^{i\omega t} \left\langle \hat{\rho}_{\vec{q}}(t) \hat{\rho}_{\vec{q}}^{+}(0) \right\rangle = \ldots$$

53

- Thus

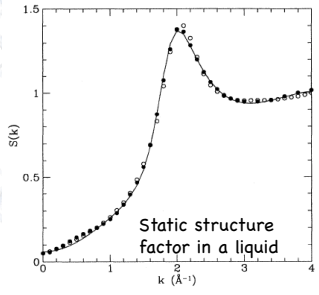
$$S(\vec{q}) = \frac{1}{N} \int_{-\infty}^{\infty} dt\, \delta(t) \left\langle \hat{\rho}_{\vec{q}}(t) \hat{\rho}_{\vec{q}}^{+}(0) \right\rangle = \frac{1}{N} \left\langle \hat{\rho}_{\vec{q}} \hat{\rho}_{\vec{q}}^{+} \right\rangle$$

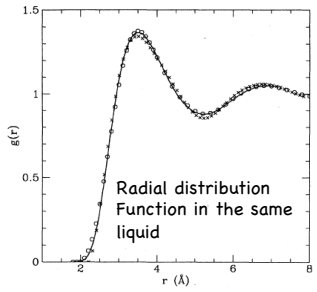
- In an isotropic and homogeneous system there is a direct relation between S(q) and the radial (2-particles) distribution function:

$$S(\vec{q}) = \frac{1}{N} \left\langle \sum_j e^{-i\vec{q}\cdot\vec{r}_j} \sum_l e^{i\vec{q}\cdot\vec{r}_l} \right\rangle = 1 + \frac{1}{N} \left\langle \sum_{j\neq l} e^{-i\vec{q}\cdot(\vec{r}_j - \vec{r}_l)} \right\rangle = \ldots$$

$$= 1 + \frac{1}{N} \left\langle \sum_{j\neq l} \int d\vec{r}\, d\vec{r}' e^{-i\vec{q}\cdot(\vec{r}-\vec{r}')} \delta(\vec{r} - \vec{r}_j)\, \delta(\vec{r}' - \vec{r}_l) \right\rangle = 1 + \frac{1}{N} \int d\vec{r}\, d\vec{r}' e^{-i\vec{q}\cdot(\vec{r}-\vec{r}')} \rho^{(2)}(\vec{r},\vec{r}') =$$

Uniform system $\longrightarrow = 1 + \rho \int d(\vec{r} - \vec{r}') e^{-i\vec{q}\cdot(\vec{r}-\vec{r}')} g^{(2)}(\vec{r} - \vec{r}') = 1 + \rho \int d\vec{r} e^{-i\vec{q}\cdot\vec{r}} g(\vec{r})$





54