Physics 514 – Classical N-Body Problems

Emanuel Gull

October 13, 2016

Chapter 1

Classical N-Body Problems

1.1 Introduction

In this chapter we will discuss algorithms for classical N-body problems. Problems of this type occur on length scales spanning many orders of magnitude:

- the universe ($\approx 10^{26}m$)
- galaxy clusters ($\approx 10^{24}m$)
- galaxies ($\approx 10^{21}m$)
- clusters of stars ($\approx 10^{18}m$)
- solar systems ($\approx 10^{13}m$)
- stellar dynamics ($\approx 10^{9}m$)
- climate modeling ($\approx 10^{6}m$)
- gases, liquids and plasmas in technical applications ($\approx 10^{-3} \cdots 10^{2}m$)

On smaller length scales quantum effects become important, see later part of the course.

The classical N-body problem is defined by the following system of ordinary differential equations:

$$m_i \frac{d\mathbf{v}_i}{dt} = \mathbf{F}_i = -\nabla_i V(\mathbf{x}_1, ..., \mathbf{x}_N) \tag{1.1}$$

$$\frac{d\mathbf{x}_i}{dt} = \mathbf{v}_i \tag{1.2}$$

The potential V is often the sum of an external potential and a two-body interaction term:

$$V(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \sum_i V_{\text{ext}}(\mathbf{x}_i) + \sum_{i<j} U_{ij}(|\mathbf{x}_i - \mathbf{x}_j|) \tag{1.3}$$

The two-body potential is only dependent on the relative distance between the two particles and not on their absolute position.

1.2 Examples

There are many different forms of the two-body potential U:

1. In astrophysical problems gravity is usually sufficient, except in dense plasmas, interiors of stars and close to black holes:

$$U_{ij}^{(\text{gravity})}(r) = -G\frac{m_i m_j}{r} \tag{1.4}$$

2. The simples model for non-ideal gases are hard spheres with radius a_i:

$$U_{ij}^{(\text{hard sphere})}(r) = \begin{cases} 0 & \text{for } r \geq a_i + a_j \\ \infty & \text{for } r < a_i + a_j \end{cases} \tag{1.5}$$

3. Some crystals and liquids can be modeled by the Lennard Jones potential

$$U_{ij}^{(\text{LJ})}(r) = 4\epsilon_{ij}\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]. \tag{1.6}$$

   The r^{-6} term describes the correct asymptotic behavior of the van der Vaals forces. The r^{-12} term models the hard core repulsion between atoms. The special form r^{-12} is chosen to allow for a fast and efficient calculation as the square of the r^6 term.

4. In ionic crystals and molten salts the electrostatic forces are dominant:

$$U_{ij}^{(\text{ionic})}(r) = b_{ij}r^{-n} + e^2\frac{Z_i Z_j}{r} \tag{1.7}$$

   where the Z_i and Z_j are the formal charges of the ions.

5. The simulation of large biomolecules such as proteins or even DNA is a big challenge. For non-bonded atoms aften the $1-6-12$ potential, a combination of Lennard-Jones and electrostatic potential is used:

$$U_{ij}^{(\text{1-6-12})}(r) = e^2\frac{Z_i Z_j}{r} + 4\epsilon_{ij}\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]. \tag{1.8}$$

   For bonded atoms there are two ways to model the bonding: either the distances between two atoms can be fixed, or the bonding can be described by a harmonic oscillator:

$$U_{ij}^{(\text{bond})}(r) = \frac{1}{2}K_{ij}(r - b_{ij})^2. \tag{1.9}$$

   The modeling of bond angles is a slightly more complex problem. Again, either the angle can be fixed, or modeled by a harmonic oscillator in the angle θ. Note that the angle is determined by the location of three atoms: this is a *three-body* interaction!

6. the Car-Parinello method combines a classical simulation of molecular dynamics of the motion of nuclei with a quantum chemical ab-initio calculation of the forces. This gives a much more accurate description of the motion of particles than the Lennard-Jones potential. However, many simulations on this length scale require an honest treatment of quantum mechanical effects. More about this in the second part of the course.

1.3 Solving the many-body problem

We will use the same methods to solve the many-body problems that we used to solve the few-body problem, but we will encounter several additional difficulties:

- the question of boundary conditions
- measuring thermodynamic quantities such as pressure
- performing simulations at constant temperature or pressure, rather than at constant energy or volume
- reducing the scaling of the force calculation for long range forces from $O(N^2)$ to $O(NlogN)$.
- overcoming critical slowing down of simulations at the phase transitions.

1.4 Boundary conditions

Open boundary conditions are natural for simulations of solar systems or for collisions of galaxies, molecules, or atomic nuclei, which naturally form open systems. For simulations of crystals, liquids, or gases, where the simulation volume represents a small subset of the total volume of the system, boundary effects from natural boundary conditions are not desired, unless e.g. surface effects need to be investigated. For these systems, *periodic boundary conditions* are usually better.

in the calculation of forces between two particles, all periodic images of the simulation volume have to be taken into account. For short range forces, like a Lennard-Jones force, the approximation of only taking the closest element (*minimum image method*) is usually good enough.

For long range forces on the other hand (forces that are as slow as r^{-d} or slower) the minimum image method is not a good approximation because of large finite size effects. Then the forces created by all periodic images of the second particle have to be summed over. The electrostatic potential Φ_p acting on a particle caused by other particles with charge q_i at sites $\mathbf{r}_i$ is

$$\Phi_p = \sum_{\mathbf{n},i} \frac{q_i}{|\mathbf{r_n} - \mathbf{r}_i|} \tag{1.10}$$

(the vectors $\mathbf{n}$ label the unit cells, $\mathbf{r_n}$ the position of the particle in the corresponding image of the root cell, while the indices i list the other particles.)

This direct summation converges very slowly. Its calculation can be accelerated by the Ewald summation technique, which replaces the sum by two faster converging sums. In general, the potential is split into a short ranged part which converges quickly in real space, and a long-ranged part which converges quickly in reciprocal space. For the Coulomb example:

$$\Phi_p = \sum_{\mathbf{n},i} q_i \frac{\mathrm{erfc}(\alpha|\mathbf{r_n} - \mathbf{r}_i|)}{|\mathbf{r_n} - \mathbf{r}_i|} + \frac{1}{\pi L} \sum_i \sum_{\mathbf{h}\neq 0} q_i \exp\left(\frac{-\pi|h|^2}{\alpha L^2}\right) \cos\left(\frac{2\pi}{L}\mathbf{h}\cdot(\mathbf{r}_o - \mathbf{r}_i)\right) \tag{1.11}$$

In this sum the $\mathbf{h}$ are integer reciprocal lattice vectors. The parameter α is arbitrary and can be chosen to optimize convergence.

This summation is still time-consuming. Typically one tabulates the differences between the Ewald sums and minimum image values on a grid laid over the simulation cell and interpolates for distances between grid points.

1.5 Molecular dynamics simulations of gases, liquids, and crystals

1.5.1 Ergodicity, initial conditions and equilibration

In many situations we are interested in following the time-evolution of initial conditions which are an integral part of the problem setup. The simulation of scattering problems or cosmological evolutions are such examples. In molecular dynamics simulations on the other hand one is interested in thermodynamic averages $\langle A\rangle$. In an *ergodic* system the phase space average is equivalent to the time average:

$$\langle A\rangle = \frac{\int A(\Gamma)P(\Gamma)d\Gamma}{\int P(\Gamma)d\Gamma} = \lim_{\tau\to\infty}\frac{1}{\tau}\int_0^\tau A(t)dt \tag{1.12}$$

Here Γ denotes a point in phase space (e.g. position and velocity of all particles), $P(\Gamma)$ is an (unnormalized) probability distribution over phase space, e.g. $\exp[-\frac{E(\Gamma)}{k_B T}]$, and the denominator is the normalization of the probability distribution (the partition function). If one is interested in thermal averages rather than the time evolution from an initial state, the precise form of the initial conditions should not matter. A typical simple choice consists of atoms arranged on a regular grid and velocities chosen randomly for each component, according to a Maxwell distribution

$$P[v_\alpha] \propto \exp\left(\frac{-mv_\alpha^2}{2k_B T}\right). \tag{1.13}$$

The proportionality constant of the velocity is chosen such that the total energy is the one desired for the physical problem. Time evolution of the system will then destroy the initial regular arrangement of atoms and randomize the configuration.

An important issue is that the system has to be equilibrated (or *thermalized*) for some time before a distribution of position and velocities representative of the equilibrium is reached and measurements can be started. This thermalization time is best determined by observing time series of physical observables, such as the kinetic energy (temperature). Careful: equilibration times depend on the quantities and that the kinetic energy has reached its equilibrium value does not imply that other observables (e.g. the pair-correlation function) have also already reached their equilibrium value!

1.5.2 Measurements

A simple observable is the self-diffusion constant D. In a liquid or geaseous system it can be determined from the time dependence of the positions:

$$\Delta^2(t) = \frac{1}{N}\sum_{i=1}^{N}[r_i(t) - r_i(0)]^2 = 2dDt + \Delta_0^2. \tag{1.14}$$

In a crystal the atoms remain at the same location in the lattice and thus $D = 0$.

A measurement of D is one way to observe melting of a crystal. Another quantity that is easy to measure is the mean kinetic energy

$$\langle E_k \rangle = \frac{1}{2}\langle \sum_{i=1}^{N} m_i v_i^2 \rangle \tag{1.15}$$

The kinetic energy is proportional to the mean temperature

$$\langle E_k \rangle = \frac{G}{2} k_B T, \tag{1.16}$$

where $G = d(N-1) \approx dN$ is the number of degrees of freedom.

In a system with fixed boundaries the particles are reflected at the boundaries. The pressure P is just the force per area acting on the boundary walls of the system. In the case of periodic boundary conditions there are no walls. The pressure P can then be measured using the following equation from the virial theorem:

$$P = \frac{N k_B T}{V} + \frac{1}{dV}\sum_{i<j} r_{ij} \cdot F_{ij}(t). \tag{1.17}$$

Here F_{ij} denotes the force between particles i and j and r_{ij} is their distance. The first term of this equation is the pressure due to the kinetic energy of the particles, the so-called kinetic pressure. This term alone gives the ideal gas law. The second term is the pressure (force per area) due to the interaction forces.

The *pair correlation function*

$$g(\mathbf{r}) = \frac{1}{\rho(N-1)} \left\langle \sum_{i \neq j} \delta(\mathbf{r} + \mathbf{r_i} - \mathbf{r_j}) \right\rangle \tag{1.18}$$

gives information about the distribution of particles with respect to each other. Its Fourier transform is the (static) structure factor $S(\mathbf{k})$

$$g(\mathbf{r}) - 1 = \frac{1}{(2\pi)^d \rho} \int [S(\mathbf{k}) - 1] \exp(i\mathbf{k} \cdot \mathbf{r}) d\mathbf{k} \tag{1.19}$$

$$S(\mathbf{k}) - 1 = \rho \int [g(\mathbf{r}) - 1] \exp(i\mathbf{k} \cdot \mathbf{r}) d\mathbf{r} \tag{1.20}$$

If the angular dependence is of no interest, a radial pair correlation function

$$g(r) = \frac{1}{4\pi} \int g(\mathbf{r}) \sin\theta d\theta d\phi \tag{1.21}$$

and its Fourier transform can be used instead.

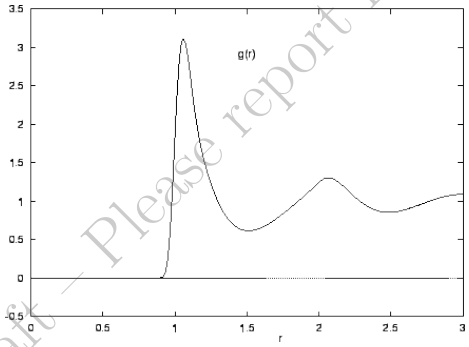

Figure 1.1: Angle integrated 'radial' pair correlation function $g(r)$ for a fluid with strong repulsion at small distances

This structure factor can be measured in X-ray diffraction or neutron scattering experiments. In a perfect crystal the structure factor shows sharp δ-function like Bragg peaks and a periodic long range structure in $g(\mathbf{r})$. Liquids still show broad maxima at distances of nearest neighbors, second nearest neighbors, etc, but these features decay rapidly with distance.

The specific heat at constant volume c_V can in principle be calculated as a temperature derivative of the internal energy. Since such numerical derivatives are rather noisy the preferred method is a calculation from the energy fluctuations

$$c_v = \frac{\langle E^2 \rangle - \langle E \rangle^2}{k_B T^2}. \tag{1.22}$$

1.5.3 Simulations at constant energy

The equations of motion of a dissipationless system conserve the total energy and the simulation is thus done in the microcanonical ensemble. Discretization of the time evolution however introduces errors in the energy conservation, and as a consequence the total energy will slowly change over time. To remain in the microcanonical ensemble, energy corrections are necessary from time to time. These are best done by a rescaling of all the velocities with a constant factor.

1.5.4 Simulations at constant temperature

The canonical ensemble (constant T) is usually of greater relevance than the microcanonical enemble (constant energy). The crudest, ad-hoc method for constant temperature is a rescaling of the velocities (as in the constant energy case), such that the kinetic energy (and, as a consequence of Eq. 1.16, the temperature) stays constant, rather than the total energy. This changes the dynamics of the system abruptly in each step and is therefore not done in practice.

A better method is the Nosé-Hoover thermostat. In this algorithm the system is coupled reversibly to a heat bath by a friction term η:

$$m_i \frac{\mathbf{v}_i}{dt} = \mathbf{F}_i - \eta \mathbf{v}_i \tag{1.23}$$

$$\frac{d\mathbf{r}_i}{dt} = \mathbf{v_i} \tag{1.24}$$

The friction term η is chosen such that the system relaxes to a given temperature T. If T is too low we want to heat up the system, if T is too high we need to cool it down. This can be achieved by setting

$$\frac{d\eta}{dt} = \frac{1}{m_s}\Big(E_k - \frac{1}{2} G k_B T\Big) \tag{1.25}$$

(compare the right hand side to Eq. 1.16), where m_s is the coupling constant to the heat bath.

1.5.5 Simulations at constant pressure

Until now we have always worked at a fixed volume. To perform simulations at constant pressure we need to allow the volume to change. This can be done by rescaling the coordinates with the linear size L of the system

$$\mathbf{r} = L\mathbf{x}. \tag{1.26}$$

Similarly to abruptly rescaling the velocity to reach the desired temperature. Alternatively, we can introduce additional degrees of freedom to the equations of motion: The volume of the system is denoted by $\Omega = L^D$. We extend the

Lagrangian by including an external pressure P_0 and an inertia M for pressure changes (e.g. the mass of a piston):

$$\mathcal{L} = \sum_{i=1}^{N} \frac{m_i}{2} L^2 \left(\frac{d\mathbf{x_i}}{dt}\right)^2 - \sum_{i<j} V(L(\mathbf{x_i} - \mathbf{x_j})) + \frac{M}{2}\left(\frac{d\Omega}{dt}\right)^2 + P_0\Omega \qquad (1.27)$$

The Euler-Lagrange equations $\partial_x \mathcal{L} - d_t \partial_v \mathcal{L}$ applied to this Lagrangian give the equations of motion

$$\frac{d^2\mathbf{x_i}}{dt^2} = \frac{1}{m_i L}\mathbf{F_i} - \frac{2}{D\Omega}\frac{d\Omega}{dt}\frac{d\mathbf{x_i}}{dt} \qquad (1.28)$$

$$\frac{d^2\Omega}{dt^2} = \frac{P - P_0}{M} \qquad (1.29)$$

P is the pressure as defined in Eq. 1.17. These equations of motion are then integrated using generalizations of the Verlet algorithm.

It is also possible to combine the constant pressure and constant temperature formalisms, leading to a constant-pressure-constant-temperature (NPT) ensemble simulation.

1.6 Scaling with system size

The time-intensive part of a classical N-body simulation is the calculation of the forces. The updating of positions and velocities according to the forces is rather fast: it scales *linearly* with the number of particles N.

For short range forces the number of particles within the interaction range is limited, and the calculation of the forces scales with $O(N)$ and thus poses no big problem.

Rapidly decaying potentials, like the Lennard-Jones potential, can be cut off at a distance r_c. The error thus introduced into the estimates for quantities like the pressure can be estimated from Eq. 1.17 and Eq. 1.18 as

$$\Delta P = -\frac{2\pi\rho^2}{3}\int_{r_c}^{\infty} \frac{\partial V}{\partial r} g(r) r^3 dr, \qquad (1.30)$$

assuming that the pair-potential $V(r)$ is the same for all particle pairs. If $V(r)$ decays faster than r^{-3} (in general: faster than r^{-d} where d is the dimensionality) this correction becomes small as r_c is increased.

Long range forces, like Coulomb forces or gravity, on the other hand, pose a big problem. No finite cut-off may be introduced without incurring substantial errors. Each particle asserts a force onto every other particle, thus requiring $(N-1) * N/2 \sim O(N^2)$ force evaluations. This is prohibitive for large scale simulations. There are however approximate ways to deal with this problem. Due to the time step discretization Δt we cannot avoid making errors in the time integration. Thus we can tolerate a small error in the force calculations and use one of a number of algorithms that, while introducing small controllable errors in the forces, need only $O(N \log N)$ computations.

1.6.1 The Particle-Mesh (PM) algorithm

The particle-mesh (PM) algorithm maps the force calculation to the solution of a Poisson equation which can be done in a time proportional to $O(N \log N)$. It works as follows:

1. A regular mesh with $M \sim N$ mesh poimnts is introduced in the simulation volume.
2. The masses of the particles are assigned – in a clever way – to nearby mesh points.
3. The potential equation (often a Poisson equation) is solved on the mesh using a fast solver in $O(M \log M) \sim O(N \log N)$ steps.
4. The potential at the position of each particle is interpolated – again in a clever way – from the potential at the nearby mesh points and the force upon the particle calculated from the gradient of the potential.

The charge assignment and the potential interpolation are the tricky parts. They should be done such that errors are minimized. Further information can be found in textbooks[1].

We want to fulfill at least the following conditions:

1. At large particle separations the errors should become negligible
2. The charge assigned to the mesh points and the forces interpolated from mesh points should vary smoothly as the particle position changes
3. Total momentum should be conserved, *i.e.* the force $\mathbf{F}_{ij}$ acting on a particle I from the particle j should fulfill $\mathbf{F}_{ij} = \mathbf{F}_{ji}$.

The simplest scheme is the *nearest grid point* (NGP) scheme, where the full particle mass is assigned to the nearest grid point and the force is also evaluated at the nearest grid point. More elaborate schemes like the *cloud in cell* scheme assign the charge to the 2^d nearest grid points and also interpolate the forces from these grid points. The algorithm becomes more accurate but also more complex as more points are used.

In periodic systems and for forces which have a Green's function $g(\mathbf{r})$ (e.g. solutions of a Poisson equation) one of the best methods is the fast Fourier method.

We start from a potential

$$\Phi(\mathbf{r}) = \int d^3\mathbf{r}' \rho(\mathbf{r}') g(\mathbf{r} - \mathbf{r}') \tag{1.31}$$

[1] R.W. Hockney and J.W. Eastwood, Computer Simulations using Particles, Taylor & Francis (1981, 1984, 1994), ISBN 978-0852743928

where $\rho(\mathbf{r})$ is the charge distribution and the Greens function is

$$g(\mathbf{r}) = \frac{G}{||\mathbf{r}||} \tag{1.32}$$

in three space dimensions. The convolution in Eq. 1.31 is best performed by Fourier transforming the equation, which reduces it to a multiplication:

$$\hat{\Phi}(\mathbf{k}) = \hat{\rho}(\mathbf{k})\hat{g}(\mathbf{k}). \tag{1.33}$$

On the finite mesh of the PM method, the discrete charge distribution is Fourier transformed in $O(M \log M)$ steps using the fast Fourier transform (FFT) algorithm. The Fourier transform of the Greens function in Eq. 1.32 is

$$\hat{g}(\mathbf{k}) = \frac{G}{||\mathbf{k}||^2} \tag{1.34}$$

Using this equation gives a 'poor man's' Poisson solver. The discretization errors can be reduced by using a modified Green's function

$$\hat{g}(\mathbf{k}) \propto \frac{1}{\sin^2(k_x L/2) + \sin^2(k_y L/2) + \sin^2(k_z L/2)} \tag{1.35}$$

where L is the linear dimension of the simulation volume. This form is also differentiable at the Brillouin zone boundary ($k_x = \pm\pi/2$, $k_y = \pm\pi/2$, or $k_z = \pm\pi/2$). Further information and help with some subtleties can be found in the textbooks.

The PM algorithm is very efficient but has problems with

- non-uniform particle distributions, such as clustering of stars and galaxies
- strong correlation effects between particles. Bound states, such as binary stars, will never be found in PM simulations of galaxies
- complex geometries

The first two of these problems can be solved by the P^3M and AP^3M methods.

1.6.2 The P^3M and AP^3M methods

The PM method is good for forces due to far away particles but bad for short ranged problems. The P^3M method solves this problem by splitting the force $\mathbf{F}$ into a long ranged part $\mathbf{F}_l$ and a short range force $\mathbf{F}_s$:

$$\mathbf{F} = \mathbf{F}_s + \mathbf{F}_l \tag{1.36}$$

The long range force $\mathbf{F}_l$ is chosen to be small and smoothly varying for short distances. It can be computed efficiently using the particle-mesh (PM) method. The short range force $\mathbf{F}_s$ has a finite interaction radius r and is calculated

exactly, summing up the particle-particle forces. Thus the name for the method is particle-particle / particle-mesh, or P^3M algorithm.

For nearly uniform particle distributions the number of particles within the range of $\mathbf{F}_s$ is small and independent of N. The P^3M algorithm then scales as $O(N) + O(M \log M)$ with $M \sim N$.

Attractive long range forces, like gravity, tend to clump particles and lead to extremely non-uniform particle distributions. A typical example are solar systems, start clusters, galaxies, and galaxy clusters. In this case, with a mesh of $M \sim N$ points, it will happen that almost all particles clump within the range r of the short-range force $\mathbf{F}_s$. Then the PP part scales like $O(N^2)$, and we're back to the original scaling. Alternatively we can increase the number of mesh points from M to $M \gg N$, which again is not optimal.

The solution to this problem is refining the mesh adaptively in the regions of space with a high particle density. In a simulation of a collision between two galaxies we will use a fine mesh at the location of the galaxies and a coarse mesh in the rest of the simulation space. The adaptive P^3M method, AP^3M, automatically refines the mesh in regions of space with high particle densities and is often used, besides tree codes, for cosmological simulations.

1.6.3 The Tree codes

Another approach to speeding up the force calculation is by collecting clusters of far away particles into effective pseudoparticles. The mass of these pseudoparticles is the total mass of all particles in the cluster they represent. To keep track of these clusters a tree is constructed. Details of this method are explained very well in the book 'Many-body tree methods in phsyics' by S. Pfalzner and P. Gibbon[2].

1.6.4 The multipole expansion method

In more or less homogeneous systems another algorithm can be used which at first sight scales like $O(N)$: The *fast multipole method* or FMM calculates a high order multipole expansion for the potential due to the particles and uses this potential to calculate the forces. The calculation of the high order multipole moments is a big programming task, but scales like $O(N)$.

There are two reasons why this method is not the optimal method: first, we need about $O(\log N)$ multipoles to get accurate results for N particles, meaning the total scaling is back to $O(N \log N)$. Secondly, the calculation of the multipole moments is a computationally intensive task and the prefactor of the $N \log N$ term is much larger than in the tree codes. The multipole method is still useful in combination with tree codes. Modern tree codes calculate not only the total mass of a cluster, but also higher order multipole moments, up to the hexadecapole.

[2]Many-body tree methods in physics, Susanne Pfalzner and Paul Gibbon, Cambridge University Press (1996), ISBN 978-0521495646

1.7 Phase transitions

Infinite systems may exhibit phase transitions. In simulations of a Lennard Jones fluid (see exercises) we will see a first example of a phase transition: the melting of a crystal. Structural phase transitions in continuous systems are usually first order, with second order transitions occurring only at special points (e.g. the critical end-point of a first order line).

In *first order* phase transitions both phases, *e.g.* ice and water, can coexist at the same time. There are several characteristic features of first order phase transitions that can be used to distinguish them from second order ones. One such feature is the latent heat for melting and evaporation. If the internal energy is increased, the temperature first increases until the phase transition. Then it stays constant as more and more of the crystal melts. The temperature will rise again only once enough energy is added to melt the whole crystal. Alternatively this can be seen as a jump at the transition temperature of the internal energy as a function of temperature. Similarly, at constant pressure a volume change can be observed at the melting transition. Another indication is a jump in the self diffusion constant at T_c.

A more direct observation is the measurement of a quantity like the structure factor in different regions of the simulation volume. At first order phase transitions, regions of both phases (*e.g.* crystal and liquid or liquid and gas) can be observed at the same time. In second order phase transitions (*e.g.* a system undergoing a magnetic phase transition as a function of temperature) a smooth change as a function of order parameter is observed, and the system is always either in one phase or in the other.

When simulating a first-order phase transition one encounters a problem: To trigger the phase transition a domain of the new phase has to be formed. As this formation of the domain can cost energy proportional to its boundary the formation of such new domains can be suppressed, resulting in undercooled or overheated liquids. The huge time scales for melting are a big problem for molecular dynamics simulations of first order phase transitions. We will discuss later how generalized ensemble algorithms and Monte Carlo simulations can be used to introduce a faster, 'artificial' dynamics, speeding up the simulation of phase transitions.

1.8 From fluid dynamics to molecular dynamics

Depending on the strength and type of interaction different algorithms are used for the simulation of classical systems.

- Ideal or nearly ideal gases with weak interactions can be modeled by the Navier-Stokes equations
- If the forces are stronger, the Navier-Stokes equations are no longer appropriate. In that case particle-in-cell (PIC) algorithms can be used:

  - Like in the finite element method the simulation volume is split into cells. Then the fluid volume in that cell is replaced by a pseudo-particle. This pseudo-particle carries the total mass, momentum, charge, etc of all the particles (often millions) contained in the fluid cell.
  - This pseudo-particle is then propagated using molecular dynamics
  - Finally the new mass, charge, and momentum densities on the mesh are interpolated from the new positions of the pseudoparticles

- If interactions or correlations are even stronger, each particle has to be simulated explicitly, using the methods discussed in this chapter.

- In astrophysical simulations there are huge differences in density and length scales. For these simulations *hybrid methods* are needed. Parts of the system are treated as fluids and simulated using fluid dynamics. Other parts (typically the denser and more correlated parts) are simulated as particles. The border line between fluid and particle treatment is fluid and determined only by the fact that currently no more than 10^8 particles can be treated.

1.9 Warning

Tree codes, multipole expansion methods, (A)P^{3}M, PM, etc methods accept a small error in exchange for a very large speedup. After some time these small errors will add up and may influence the physics. How? This is often open and you will need to check how reliable your results are.

1.10 The Fast Fourier Transform (FFT) algorithm

The Poisson solvers described in this section used a Fourier transform to transform real-space data into k-space, a multiplication in k-space (corresponding to a convolution in real space), and an additional Fourier transform back from k-space to real space. This discrete Fourier transform transforms (in one dimension) a vector y_n with N values into a vector c_k in Fourier space. The discrete Fourier transform (DFT) is given by

$$c_k = \sum_{n=0}^{N-1} y_n \exp\left(\frac{-2\pi ikn}{N}\right) \tag{1.37}$$

, and its inverse transform by

$$y_n = \frac{1}{N}\sum_{n=0}^{N-1} c_k \exp\left(\frac{2\pi ikn}{N}\right). \tag{1.38}$$

as the values of c_k (or y_n)have to be computed for all k, and as each value contains a sum over n, this operation requires $O(N^2)$ steps. The idea behind the Cooley Tuckey Fast Fourier algorithm is a recursive divide and conquer strategy: split this problem into smaller sub-problems, then solve those iteratively, at a total cost of $O(N \log N)$ rather than $O(N^2)$.

The algorithm in its current incarnation was developed by James Cooley and John Tukey in 1965, though a much earlier variant by Gauss exists and several mathematicians and engineers developed variants of it over the centuries.

To illustrate the algorithm we focus on the simple most case, $N = 2^m$. We start by splitting the terms in Eq. 1.37 into two groups: even terms (E_k) and odd terms ($O_k \exp^{-2\pi ik/N}$), so that

$$E_k = \sum_{r=0}^{N/2-1} y_{2r} \exp\left(\frac{-2\pi ik(2r)}{N}\right) = \sum_{k=0}^{N/2-1} y_{2r} \exp\left(-\frac{2\pi ikr}{N/2}\right) \tag{1.39}$$

$$\sum_{r=0}^{N/2-1} y_{2r+1} \exp\left(-\frac{2\pi ik(2r+1)}{N}\right) = \tag{1.40}$$

$$\sum_{r=0}^{N/2-1} y_{2r+1} \exp\left(-\frac{2\pi ikr}{N/2}\right) \exp(-2\pi ik/N) = e^{-2\pi ik/N} O_k.$$

Both of these terms are Fourier transforms themselves, each with $N/2$ terms. We can subdivide them further to obtain values for E_k and O_k. The advantage of this procedure is speed: in the first step we have N samples. In the next step we have two transforms with $N/2$ samples, then four transforms with $N/4$ samples, etc. At each level we compute N coefficients.

The total number of levels in $m = \log N$, so we have a total of $NlogN$ operations.

The inverse transform works just the same, with a '-' sign in the exponent.

In general, the number of points N is not 2^m with m integer. In this case, a the Cooley Tukey algorithm expresses N as $N_1 \times N_2$, where typically N_1, the 'radix', is small. The DFT then performs N_1 DFTs of size N_2, multiplies the result by twiddle factors, and follows it up by N_2 DFTs of size N_1 (both of these may again be subdivided).

Adaptation to real numbers, sine and cosine transforms, as well as fast Fourier transforms for non-equidistant grids exist.

In practice you should not implement FFTs yourself but instead use a library: FFTW (the fastest Fourier transform in the West) is a standard package that defines an interface, and several highly optimized vendor libraries (MKL, ACML, ESSL) may be available on your computer.