## 4.20 Common (and subtle) mistakes in update algorithms

- **Measure factors:** Note that the detailed balance condition is correctly written in terms of probabilities:

$$\frac{W_f(\phi \mapsto \phi')}{W_f(\phi' \mapsto \phi)} = \frac{p_{\text{eq.}}(\phi')}{p_{\text{eq.}}(\phi)} = e^{(H(\phi)-H(\phi'))/k_BT}$$

  The second equality is true only if $p(\phi) \propto e^{-H(\phi)/k_BT}$. We have implicitly assumed this for most of the discussion in this section, but this is not always the case!

  As a simple example consider variables $(x, y)$, with distribution $p(x, y) \propto e^{-H(x^2+y^2)/k_BT}$. Now, one might want to use polar coordinates $(r, \theta)$ instead. Now

$$p(x, y)\, dx\, dy = p(x, y)\, r\, dr\, d\theta \equiv p(r, \theta)\, dr\, d\theta \Rightarrow p(r, \theta) \propto re^{-H(r^2)/k_BT}$$

  The Jacobian factor $r$ follows the Boltzmann factor everywhere.

  Thus, for example, if we do restricted Metropolis update

$$r \to r' = r + S(X - 0.5)$$

  where $X$ is a uniform random number in interval [0,1], this is accepted with the probability

$$p_{\text{accept}}(r \mapsto r') = \min\left(1, \frac{r'e^{-H(r'^2)/k_BT}}{re^{-H(r^2)/k_BT}}\right)$$

  It is important to keep track of the correct measure (Jacobian) factors!

  Excercise: how would you update a 3-dim. vector $\vec{v}$ which is restricted to unit length (i.e. traces a surface of 2-sphere) and which has interaction energy $H = -\vec{v} \cdot \vec{c}$, with some constant vector $\vec{c}$? The measure is assumed homogeneous on the 2-sphere.

- **Adjustable Metropolis scale:** Metropolis updates have adjustable scale factor. This can be automatically tuned by the update algorithm to acceptance $\sim 60\%$, for example. However, this tuning must not happen during measurements, or it can ruin the detailed balance. Thus, automatic tuning should be done only in the "thermalisation" phase.

- **Other tunables in updates/measurements:** Like the Metropolis scale, adjusting some other tunables (e.g. the measurement interval) on the fly lead to incorrect sampling. These have to be done before the measurements are taken.

# 5 Particles in a potential

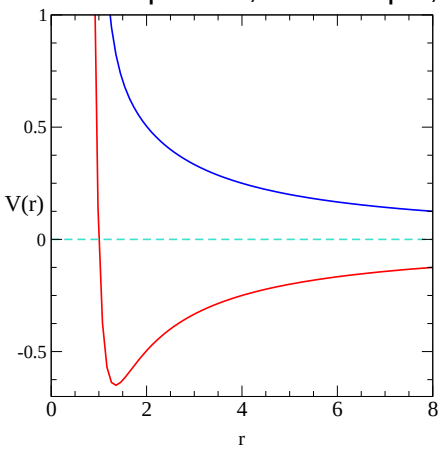
Let us consider a case where a set of particles interact with a 2-particle potential $V(\vec{r}_1 - \vec{r}_2)$. For concretness, let us assume 2 dimensions. We can take the potential to be a long-range Coulomb potential with a "hard core"-like repulsion, for example,

$$V_{ij} = \frac{q_i q_j}{|\vec{r}_i - \vec{r}_j|} + \frac{1}{|\vec{r}_i - \vec{r}_j|^8}$$

where $q_i \pm 1$ are the charges of the particles. We put this system in a heat bath, i.e.

$$Z = \int \prod_{i=1}^{N} [d\vec{r}_i] \exp\left[-\frac{1}{T}\sum_{i<j} V_{ij}\right]$$

(in dimensionless units). This is a simple model for ionic crystal $\leftrightarrow$ liquid $\leftrightarrow$ gas transition.




Note that we neglect here the kinetic energy (and equation of motion) of the particles. Thus, this corresponds to a particle system in a heat bath; i.e. the particles can get or lose energy through other channes than particle-particle interactions (radiation, interactions with some other (neutral) particles, etc.).

The number of particles is given beforehand; the degrees of freedom are the particle positions. Now it is easy to write a Monte Carlo program which updates the positions of the particles using the Metropolis update. Let $S$ be a tunable scale, and $G_1, G_2$ gaussian distributed random numbers. Now one update sweep is the following:

*For each particle $i$ do*

1. $x_i' = x_i + S\,G_1$
2. $y_i' = y_i + S\,G_2$
3. *accept* $(x,y)_i \to (x',y')_i$ *with probability*

$$p = \max(1, \exp[-\delta V/T]).$$

*If not accepted, leave $(x,y)$ as it was.*

For concreteness, let us enclose the system in a finite box of size `size`. This could be periodic, but for simplicity we consider hard walls. One should not have infinite volume, because then the average density $= 0$, and evaporating particles never meet another particle. Thus, density is one of the thermodynamic variables here.

## 5.1 Metropolis update code for ions with Coulomb potential

```
  ...
  double x[n_atoms],y[n_atoms];  /* coordinates of the atoms */
  int q[n_atoms];                /* charge of atoms +-1 */
  double scale;                  /* Metropolis scale */
  double T;                      /* temperature */
  double size;                   /* box size */
  double xn,yn,e1,e2;
  int accept,try,loop;
  ...
  /* initialize etc. here */
  ...
```

The potential is given by the function

```
double V(double x,double y,int q1q2)
{
   double r2 = x*x + y*y;
   double r4 = r2*r2;

   return( q1q2/sqrt(r2) + 1.0/(r4*r4));
}
```

Here q1q2 is the product of the charges of the particles 1 and 2 ($\pm 1$), and x, y are the components of $\vec{r}_1 - \vec{r}_2$. The update section of the program is

```
  for (loop=0; loop<n_loops; loop++){
    accept = try = 0;
    /* modify the location, acc/rej */
    for (i=0; i<n_atoms; i++) {

      /* calculate the potential energy for i */
      for (e1=j=0; j<n_atoms; j++) if (j != i)
        e1 += V( x[i]-x[j], y[i]-y[j], q[i]*q[j] );

      /* update position -- keep within box! */
      do xn = x[i] + scale * gaussian_ran(); while (xn < 0 || xn > size);
      do yn = y[i] + scale * gaussian_ran(); while (yn < 0 || yn > size);

      /* and calculate new potential energy */
      for (e2=j=0; j<n_atoms; j++) if (j != i)
        e2 += V( xn-x[j], yn-y[j], q[i]*q[j] );

      /* now Metropolis accept/reject */
      if ( exp( (e1-e2)/T ) > mersenne() ) {
        accept++;
        x[i] = xn;
        y[i] = yn;
      }
      try++;
```

```
    }
    /* other stuff (measurements etc.) here ... */
  }
```

Here `scale` is the adjustable Metropolis step size, which is tuned for acceptance.

*NOTE:* when the position of particle $i$ is updated, only the part of the total energy dependent on $i$ is calculated, i.e.

$$E_i = \sum_{j \neq i} V_{ij}.$$

This has to be calculated both before and after the modification.

$\Rightarrow$ Update of all $N$ atom positions requires $\propto N^2$ operations! With large $N$ this becomes very slow.

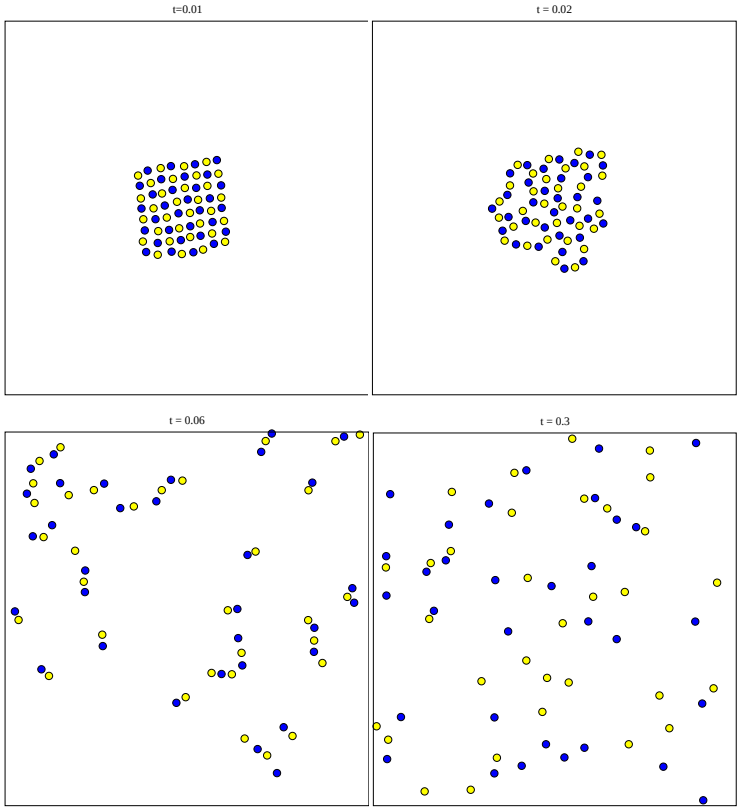
$\Rightarrow$ Often only interactions with nearest atoms (up to some range) are calculated exactly. For atoms further away, average charge is *coarse-grained* (not discussed in this course).

## Phase diagram

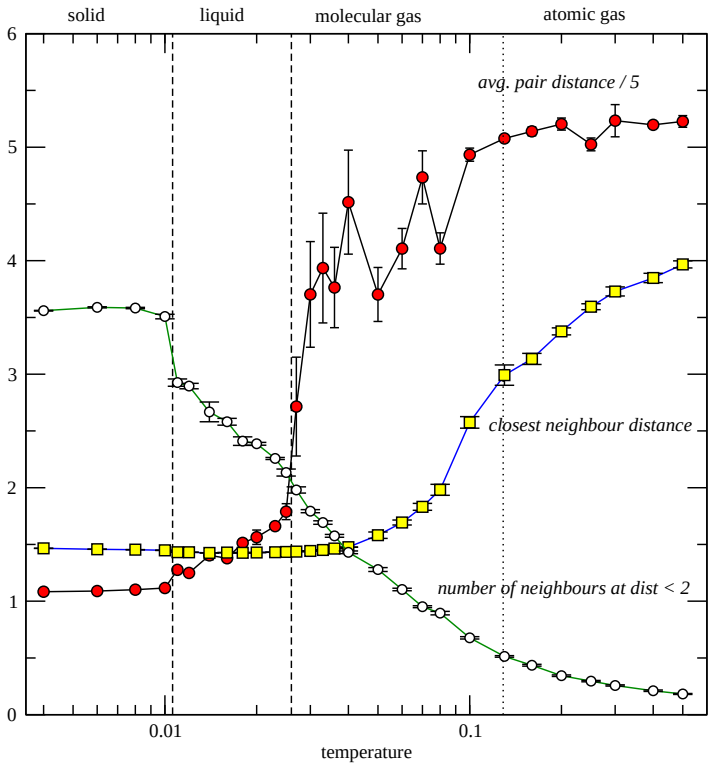
Consider 64 ions in a box of size $60^2$:

- At low temperatures $T \lesssim 0.01$ solid square crystal
- $0.01 \lesssim T \lesssim 0.03$ liquid
- $0.03 \lesssim T \lesssim 0.2$ gas which consists mostly of charge neutral molecules
- $0.2 \lesssim T$ gas which consists mostly of individual ions

The transitions are not sharp at finite system; especially gas of molecules $\leftrightarrow$ ions is continuous even at infinite systems.



The properties of the phases can be monitored by using various observables. Shown here in the plot:




- Average pair distance $\frac{2}{N(N-1)} \sum_{i<j} |\vec{r}_i - \vec{r}_j|$.
  Distinguishes between gaseous and non-gaseous phases.
- average of the distance to the *nearest* neighbour atom of each of the atom. Becomes large when gas consists of single ions.
- average number of ions at distance $< 2$, measured from each ion. Close to 4 for crystals (boundary effects!), becomes $\sim 0$ for individual ions.

The program `ions.c` is available in the course web pages. If compiled with the `grace_np` -library, it can show animations of the Monte Carlo evolution. See instructions in the program. (This requires that grace/xmgrace -program is installed. It is available on most linux distributions, or from `http://plasma-gate.weizmann.ac.il/Grace/` .)

The method is easy to modify for other potentials or 3-dimensional space.