Homework **LTTC Quantum dynamics** Mauro Gascon Navas
TCCM March 19, 2025

All scripts and files generated while completing this homework can be found in this Github repository.

1 Spherical confinement potential

A Spherical Confinement Potential (SCP) is added to as a functionality the program to ensure that the particles do not "escape" the simulation bulk. The potential energy is defined as:

$$E = 0.008 \cdot (dist_{COM} - R_{max})^4 = 0.008 \cdot (|\vec{r}| - R_{max})^4, \quad |\vec{r}| > R_{max}$$

Where the center of mass (COM) is defined as the origin. The force is then calculated as the negative gradient of the potential:

$$\begin{aligned} F = -\nabla E &= -\nabla \left(0.008 \cdot (|\vec{r}| - R_{max})^4\right) \\ &= -4 \cdot 0.008 \cdot (|\vec{r}| - R_{max})^3 \cdot \nabla |\vec{r}| \\ &= -0.032 \cdot (|\vec{r}| - R_{max})^3 \cdot \frac{\vec{r}}{|\vec{r}|} \end{aligned}$$

The subroutine *apply_scp* is added to the program to calculate the forces and potential energy due to the SCP. The subroutine takes as inputs the number of atoms, the positions, forces, potential energy and the radius of the confinement. The center of mass is calculated and the forces are updated for each atom. To activate the SCP, the program has to be called using the *-SCP* flag, followed by the R_{max} in Å.

At every step *apply_scp* is called: the COM is calculated and all particles are translated so it is centered at the origin. then de distances are computed and the forces and potential energies for the particles that are outside the confinement are updated. The subroutine is called at the end of the main loop, after the forces are calculated.

A sample run for the given system of 125 atoms (*liquid-argon.xyz*) confined with $R_{max} = 23$ Å is provided in Figure 1. To check the conservation of energy, an NVE simmulation is performed setting the initial temperature to 0 K to avoid random initial conditions and ensure reproducibility (see Section 2). All the kinetic energy of the system comes from the energy stored in the non-optimal initial geometry. The total energy remains constant through the simmulation, its small noise can be explained from the timestep (which needs to be large to sample geometries where the SCP is activated) and the very steep r dependence ($SCP \sim r^4$) of the potential. The command used to run this simulation is:

```
./md_program.exe -FILE liquid-argon.xyz -TIMESTEP 0.1 -SAVEFREQ 100 -
  MAX_MD_STEPS 100000 -TEMP 0 -SCP 23
```

Listing 1: Subroutine to apply a SCP.

```
subroutine apply_scp(forces,pot_ener,positions,nbr_atomes,R_scp)
    implicit none
```

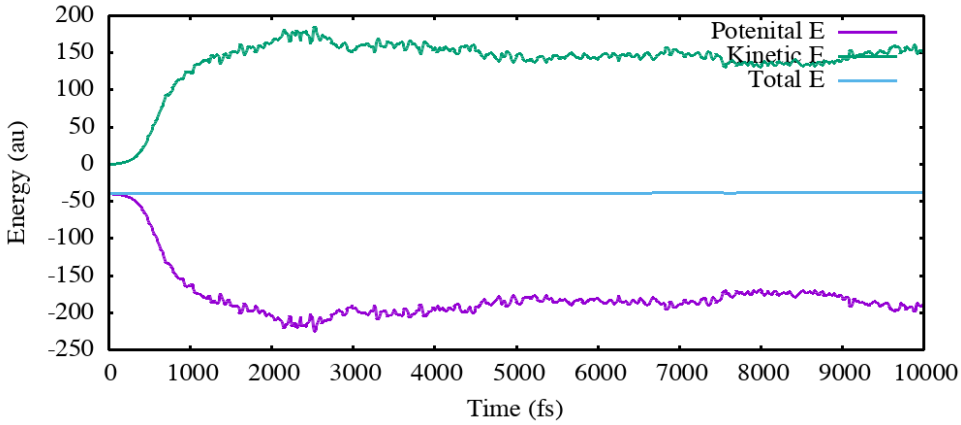


Figure 1: NVE simmulation of 125 Ar atoms in a SCP with $R_{max} = 23$ Å

```
    integer,intent(in)                                          :: nbr_atomes
    real*8, intent(in)                                          :: R_scp
    real*8, allocatable, dimension (:,:),intent(inout)          :: positions
    real*8, allocatable, dimension (:,:),intent(inout)          :: forces
    real*8,intent(inout)                                        :: pot_ener
    real*8, dimension(3)  :: com
    real*8  :: d_com
    integer :: i,j

    com = 0.0d0 ! Get center of Mass
    do i=1,nbr_atomes
        do j=1,3
        com(j)=com(j)+positions(i,j) !if we had diffrent masses we would have to
     "mass-avarage"
        enddo
    enddo
    com=com/dble(nbr_atomes)

    do i=1,nbr_atomes !!Apply SCP
        d_com = 0.0d0
        do j=1,3
        positions(i,j) = positions(i,j) - com(j) ! Center the system
        d_com = d_com + positions(i,j)**2 ! Distance to origin (COM)
        enddo
        d_com = sqrt(d_com)-R_scp
        if (d_com .gt. 0.0d0) then
        pot_ener = pot_ener + 0.008d0*(d_com)**4 ! update potential energy
        do j=1,3
            forces(i,j) = forces(i,j) -0.032*(d_com)**3*positions(i,j)/(d_com+
   R_scp) !update forces calculateed previously
        enddo
        end if
    enddo
end subroutine apply_SCP
```

2 Initial random velocities

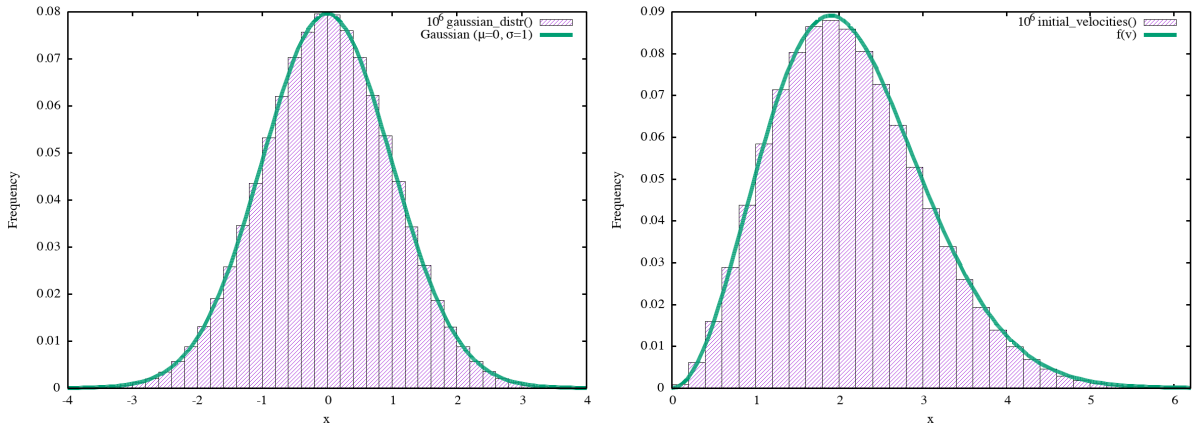
In this section, a subroutine to genereate the initial velocities following the Boltzman distribution is implemented. The given subroutine *gaussian_distr* is used to generate random numbers

sampled from a normal distribution:

$$v_i = \sqrt{\frac{k_b T}{m}} \cdot gauss\_distr()$$

The subroutine *initial_velocities* is added to the program to initialize each velocity component of every particle independently. The subroutine takes as inputs the number of atoms, the velocities, the temperature and the mass of the particles.

In Figure 2 the results of 10^6 points generated by *initial_velocities* and *gaussian_distr()* are compared to their expectations, showing its correct functioning.



(a) *gauss_distr()* (b) *initial_velocities*

Figure 2: Comparison between the generated and expected distributions. Left, gaussian distribution with $\mu = 0, \sigma = 1$. Right, velocity boltzamnn distribution with $\frac{K_b T}{m} = 1$.

Listing 2: Velocity initalization

```
subroutine initial_velocities(velocities,nbr_atomes,temp,mass)
  implicit none

  real*8, dimension (:,:),intent(inout) :: velocities
  real*8, intent(in) :: temp,mass
  integer, intent(in) :: nbr_atomes
  real*8 :: kb=3.166811d-6
  integer :: i,j

  do i=1,nbr_atomes
    do j=1,3
      velocities(i,j)=sqrt(kb*temp/mass)*gaussian_distr() ! Initial velocities
    following the Boltzmann distribution
    enddo
  enddo
end subroutine initial_velocities
```

3 Periodic boundary conditions

To implement periodic boundry conditions (PBC) the subroutine *get_distances_forces_PBC* is added to the program. The subroutine takes as inputs the number of atoms, the positions, dis-

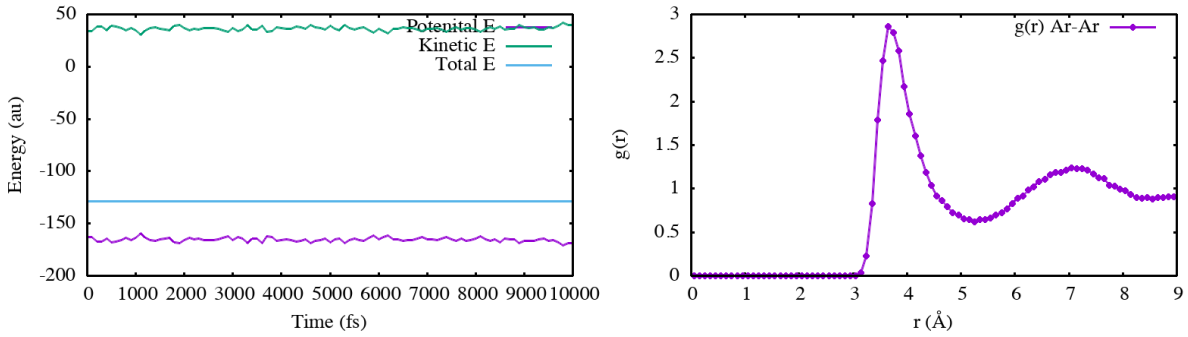
tances, forces, the box sidelength, and σ and ϵ of the Lennard-Jones potential. The functionality is activated by calling the program with the *-PBC* flag, followed by the box sidelength in Å. If the user tries to PBC together with SCP, the program returns an error informing of the incapability of doing so.

At every step, the subroutine calculates the distances and the forces using the nearest image of each particle pair. This translates to each unique particle only interacting once with each other unique particles (and not with itself). The downside of this approach is that the code will not be able to model a system with a small number of particles, like the provided example input of 10 Ar atoms. Arguably, PBC with such small systems are going to be unreallistic in any case. The subsequent calculations are performed with a bigger system of 125 Ar atoms.
Additionally, for better visualization, the atoms are constantly wrapped to the unit cell.

To test the performance of subroutine, the results from [1] are reproduced and compared. The radial distribution function is calculated for liquid argon at 94.4 K, $\epsilon = 0.24$ K and $\sigma = 3.4$ Å. The simulation is run with 125 particles (in [1] they use 864 particles) in a cubic box with periodic boundary conditions of 18.2 Å:

```
./md_program -FILE liquid-argon.xyz -TIMESTEP 1 -PBC 18.206778313424067 -
SAVEFREQ 100 -MAX_MD_STEPS 10000 -THERMO BEREN 10 -TEMP 94.4 -EPS
0.23846451108000002 -SIGMA 3.4
```

The trajectory is then analyzed using VMD to obtain the radial distribution function. It contains two peaks, g(3.7) = 2.9 and g(7.1)=1.3, which are in agreement with the results of [1].



(a) *E(t)* (b) *g(r)*

Figure 3: Left, energy evolution of the PBC simmulation. Right, radial distribution function for liquid argon at same conditions as in [1].

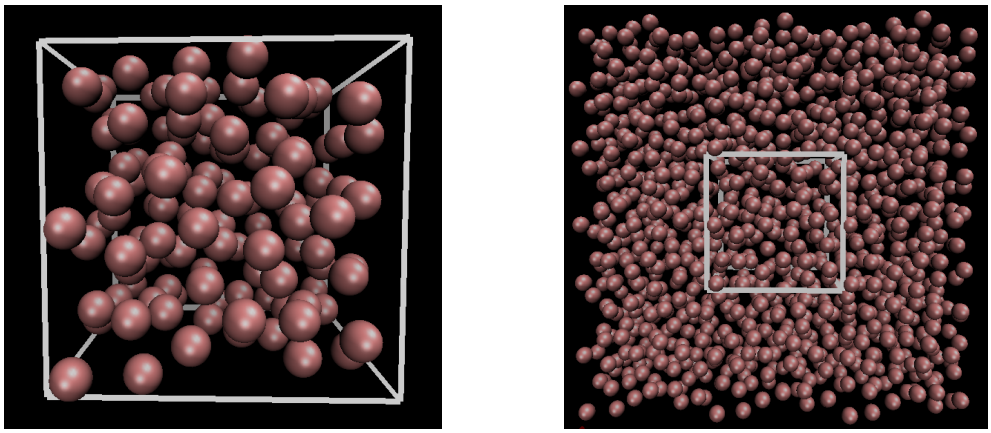
Listing 3: Subroutine to apply a PBC.

```
subroutine get_distances_forces_PBC(positions,distances,forces,nbr_atomes,L,
sigma,eps)

implicit none

real*8, allocatable, dimension (:,:),intent(inout) :: positions
real*8, allocatable, dimension (:,:),intent(inout) :: distances
real*8, allocatable, dimension (:,:),intent(inout) :: forces

real*8, intent(in)    :: L, sigma, eps
integer, intent(in)   :: nbr_atomes
```



(a) *(1,1,1) box* (b) (3,3,1) box

Figure 4: Snapshots of the PBC calculation. Left, unit cell ((1,1,1) box). Right, (3,3,1) superbox showing the system continuity. The unit cell is shown with white lines.

```
    real*8              :: dist
    real*8              :: sigma_12,sigma_6
    real*8              :: dist2
    real*8              :: dr4,dr8,dr14
    real*8              :: part_6,part_12
    integer             :: i,j,k

    forces=0.0d0
    distances=0.0d0

    do i=1,nbr_atomes-1 !calculate distances**2
      do j=i+1,nbr_atomes
        do k=1,3
          dist = positions(i,k)-positions(j,k)
          distances(i,j)=distances(i,j) + (dist - L*dble(nint(dist/L)))**2 !We
   use the nearest image of each particle pair
        enddo
       distances(j,i)=distances(i,j)
      enddo
    enddo

    sigma_6=sigma**6*6.0d0 !calculate forces
    sigma_12=sigma**12*6.0d0

    do i=1,nbr_atomes-1
      do k=i+1,nbr_atomes
        do j=1,3
          dist2=positions(i,j)-positions(k,j) ! Need dist component for the F
   direction
          dist2=dist2-L*dble(nint(dist2/L)) ! closest image for PBC
          dr4=distances(i,k)*distances(i,k)
          dr8=dr4*dr4
          dr14=dr8*dr4*distances(i,k)
          part_6  = sigma_6*(-(dist2)/dr8)
          part_12 = sigma_12*(-(2.0d0*dist2)/dr14)
          forces(i,j)=forces(i,j)-4.0d0*eps*(part_12-part_6)
          forces(k,j)=forces(k,j)+4.0d0*eps*(part_12-part_6)
        enddo
      enddo
```

```
    enddo
    endsubroutine get_distances_forces_PBC
```

Listing 4: Forbid using SCP together with PBC.

```
if (pbc .eqv. .TRUE.) then  !Apply PBC if activated
  call get_distances_forces_PBC(positions,distances,forces,nbr_atomes,Box_L,
    sigma,eps)
else
  call get_distances(positions,distances,nbr_atomes)
  call get_forces(forces,distances,positions,nbr_atomes,sigma,eps)
  if (scp .eqv. .TRUE.) then ! Apply SCP if activated
    call apply_scp(forces,energy_pot,positions,nbr_atomes,R_scp)
  endif
endif
```

References

[1] Aneesur Rahman. "Correlations in the motion of atoms in liquid argon". In: *Physical review* 136.2A (1964), A405.