ESPResSo++ simulation package

Short introduction

P. Papež
T. Potisk

Ljubljana, October 2021

Outline

Requirements

- cmake ($\geq$ 3.12), make, g++ compiler, OpenMPI, Boost ($\geq$ 1.69), FFTW3
- python3 modules: mpi4py ($\geq$ 3.0.0), pyh5md, numpy, h5py, sphinx

Installation

```
git clone https://github.com/espressopp/espressopp.git
cd espressopp
mkdir build
cd build
cmake ..
make -j np
```

Running simulation

```
export PYTHONPATH=/home/kemijski/L17/espressopp/build:/home/
    kemijski/L17/mpi4py/build/lib.linux-x86_64-3.9:${PYTHONPATH}

mpirun -np 2 python3 script.py
```

Visualization

- VMD (`https://www.ks.uiuc.edu/Research/vmd`)
- OVITO (`https://www.ovito.org/linux-downloads`)

Part I

Lennard-Jones (LJ) fluid

Aims of the 1st part

We will **explain the `.py` script** for running simulation of the LJ fluid, focusing on:

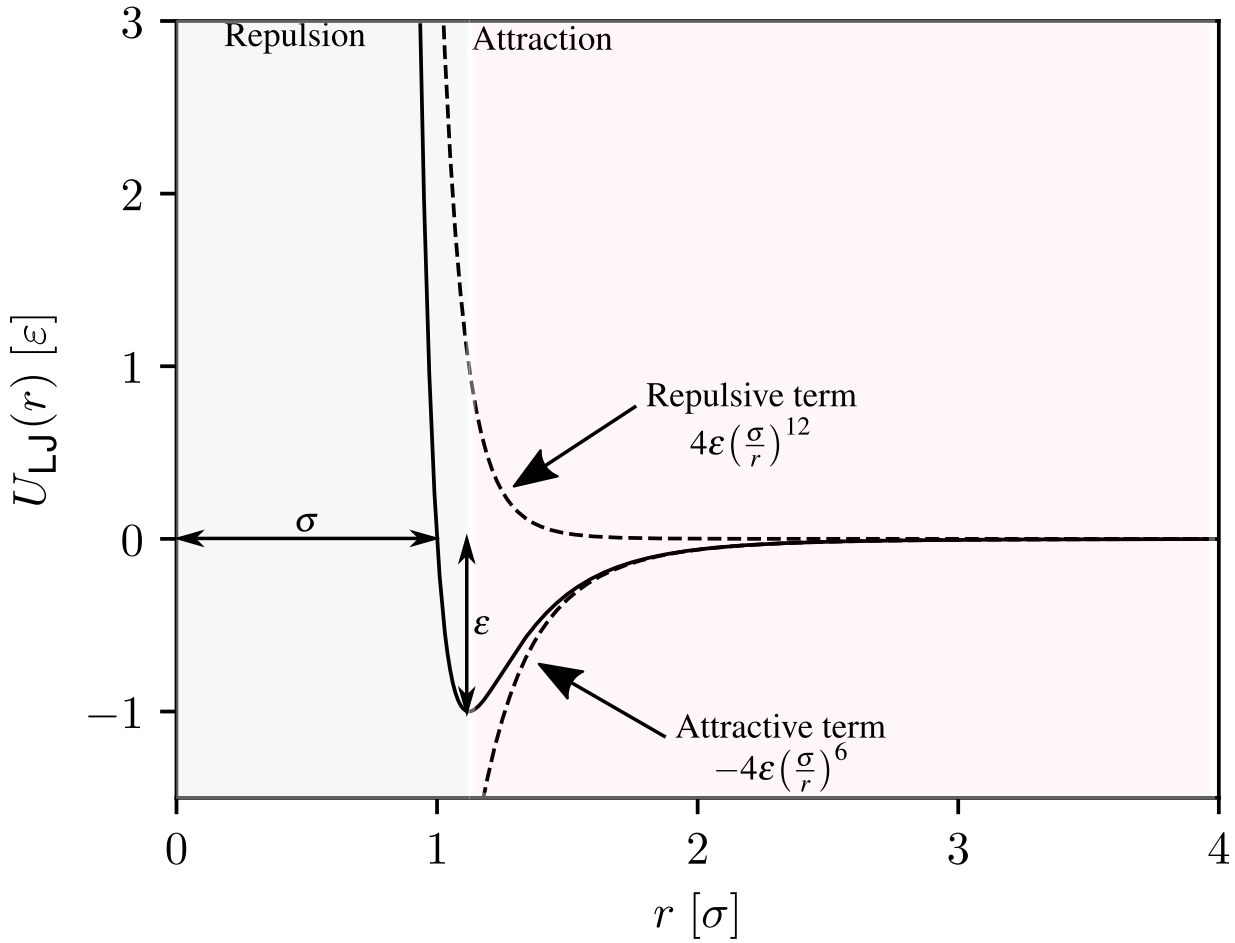
- periodic boundary conditions (PBC),
- velocity Verlet algorithm,
- Verlet List,
- interactions (i.e., LJ potential),
- radial distribution function (RDF),
- velocity autocorrelation function (VACF).

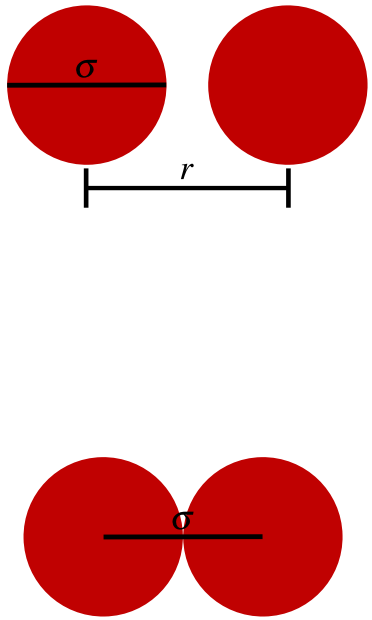
You will **perform simulation** of the LJ fluid and **analyze the results**. We will be interested in:

- *warmup run* $\rightarrow$ temperature (T), pressure (p), energy (E_k, E_p, E_{tot}),
- *production run* $\rightarrow$ temperature (T), pressure (p), energy (E_k, E_p, E_{tot}), RDF, VACF.

Lennard-Jones (LJ) fluid

$$U_{\mathrm{LJ}}(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$$

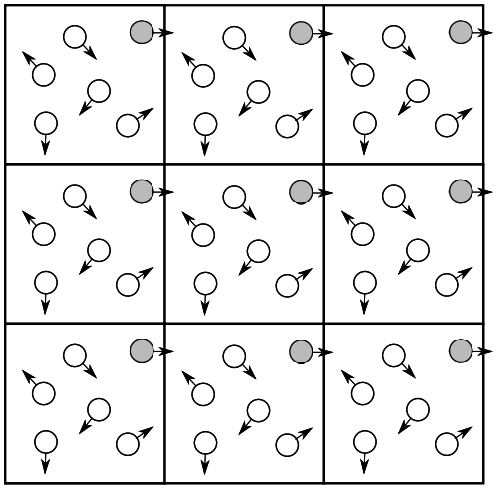




Py script

```
import espressopp
import numpy as np

# 1. specification of the main simulation parameters

Npart                = 1000
rho                  = 0.8442
L                    = pow(Npart/rho, 1.0/3.0)
box                  = (L, L, L)
r_cutoff             = 2.5
skin                 = 0.4
temperature          = 1.0
dt                   = 0.005
epsilon              = 1.0
sigma                = 1.0

warmup_cutoff        = pow(2.0, 1.0/6.0)
warmup_nloops        = 100
warmup_isteps        = 200
total_warmup_steps   = warmup_nloops * warmup_isteps
epsilon_start        = 0.1
epsilon_end          = 1.0
epsilon_delta        = (epsilon_end - epsilon_start) / warmup_nloops
capradius            = 0.6

equil_nloops         = 100
equil_isteps         = 100
```

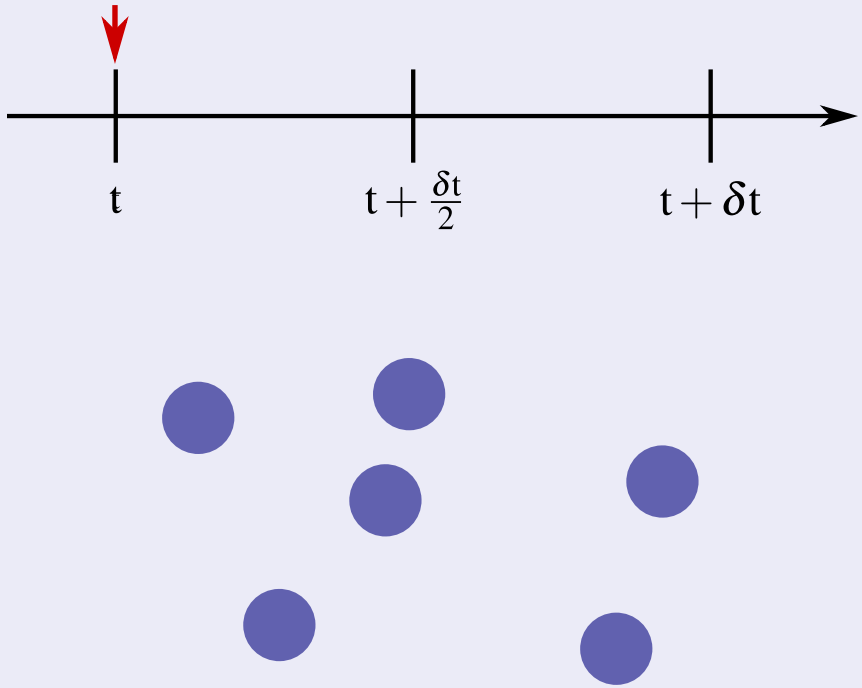
Py script



```
# 2. setup of the system, random number generator and
    parallelisation

system                = espressopp.System()
system.rng            = espressopp.esutil.RNG()
system.bc             = espressopp.bc.OrthorhombicBC(system.rng,box)
system.skin           = skin
NCPUs                 = espressopp.MPI.COMM_WORLD.size

nodeGrid              = espressopp.tools.decomp.nodeGrid(NCPUs,box,
    warmup_cutoff,skin)
cellGrid              = espressopp.tools.decomp.cellGrid(box,nodeGrid,
    warmup_cutoff,skin)
system.storage        = espressopp.storage.DomainDecomposition(system,
    nodeGrid,cellGrid)
```

Velocity Verlet algorithm

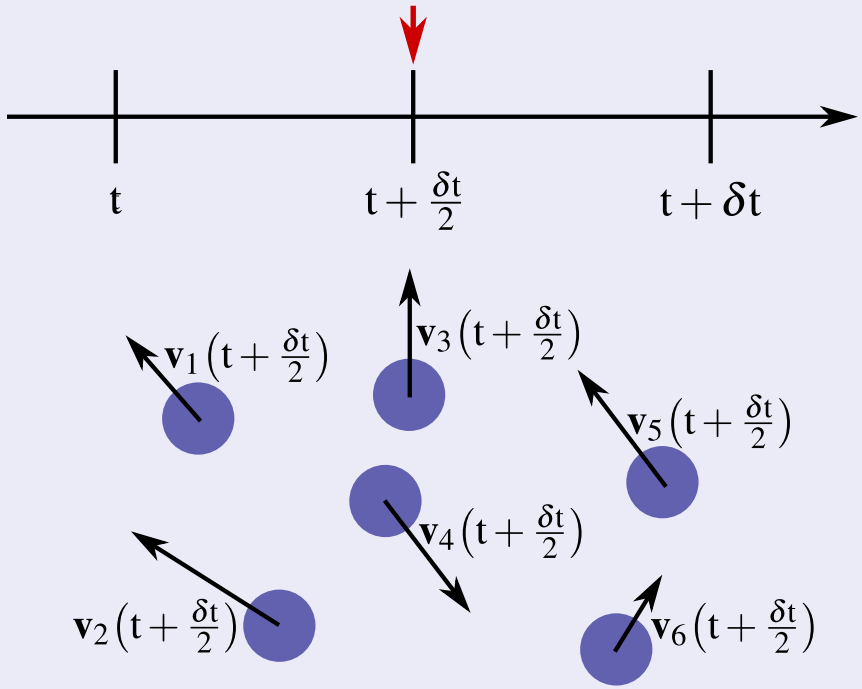
Initial configuration



Allen, M. P.; Tildesley, D. J. *Computer simulation of liquids*; Clarendon Press, 1987.

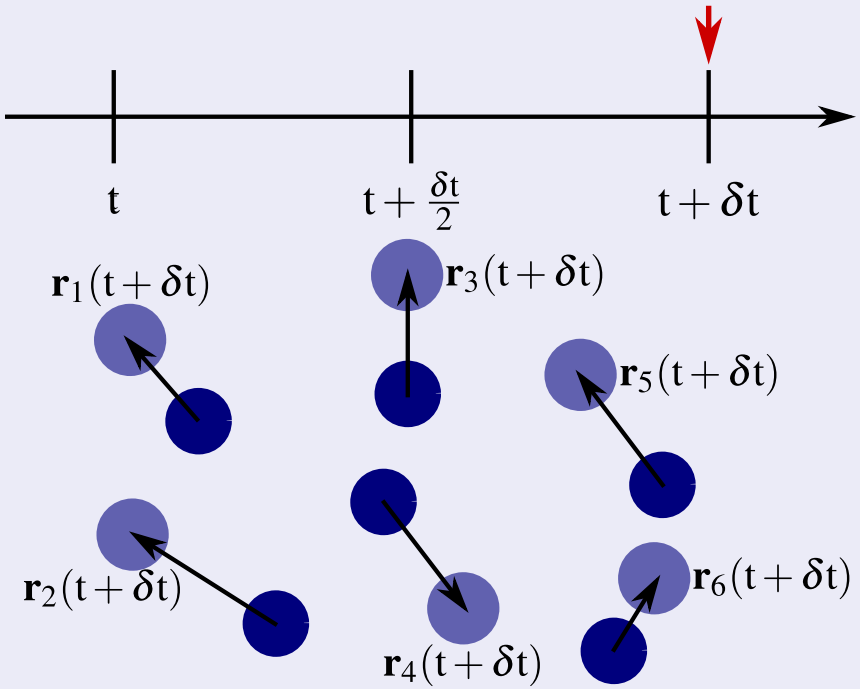
Velocity Verlet algorithm

$$\mathbf{p}_i(t + \tfrac{1}{2}\delta t) = \mathbf{p}_i(t) + \tfrac{1}{2}\delta t \mathbf{F}_i(t)$$



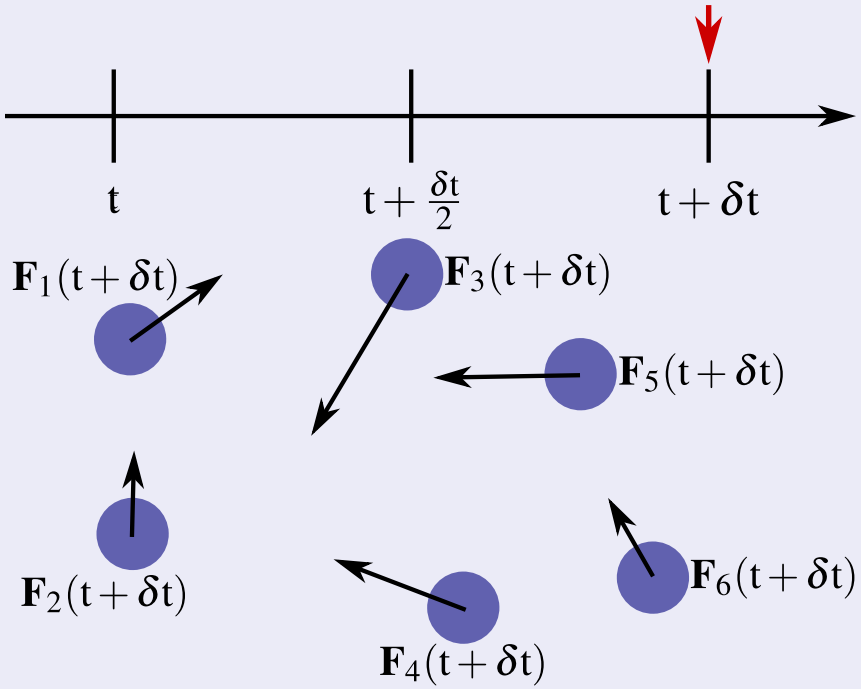
Velocity Verlet algorithm

$$\mathbf{r}_i(t+\delta t) = \mathbf{r}_i(t) + \delta t \mathbf{p}_i(t+\frac{1}{2}\delta t)/m_i$$



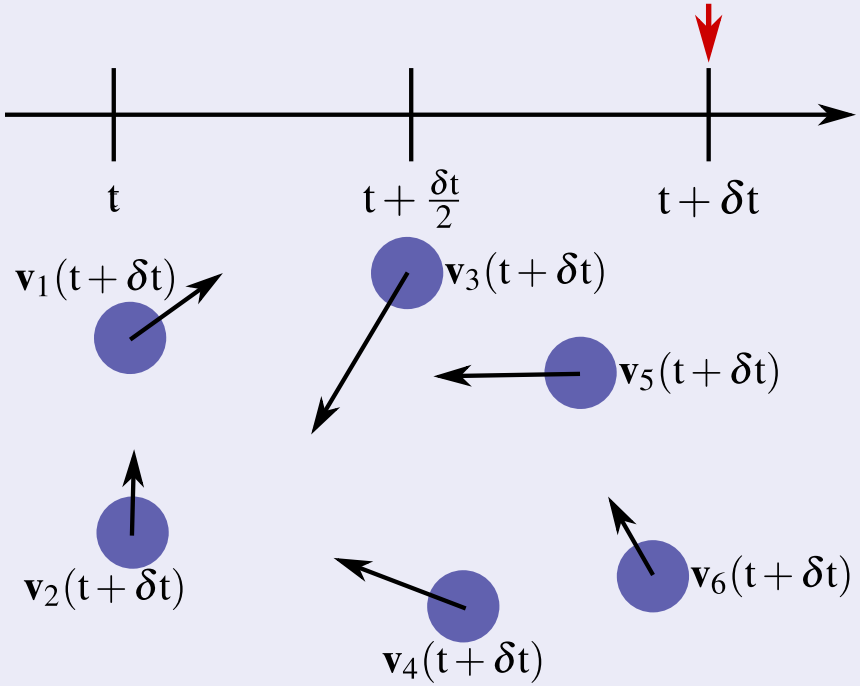
Velocity Verlet algorithm

$\mathbf{F}_i(t+\delta t)$



Velocity Verlet algorithm

$$\mathbf{p}_i(t+\delta t) = \mathbf{p}_i(t+\tfrac{1}{2}\delta t) + \tfrac{1}{2}\delta t \mathbf{F}_i(t+\delta t)$$

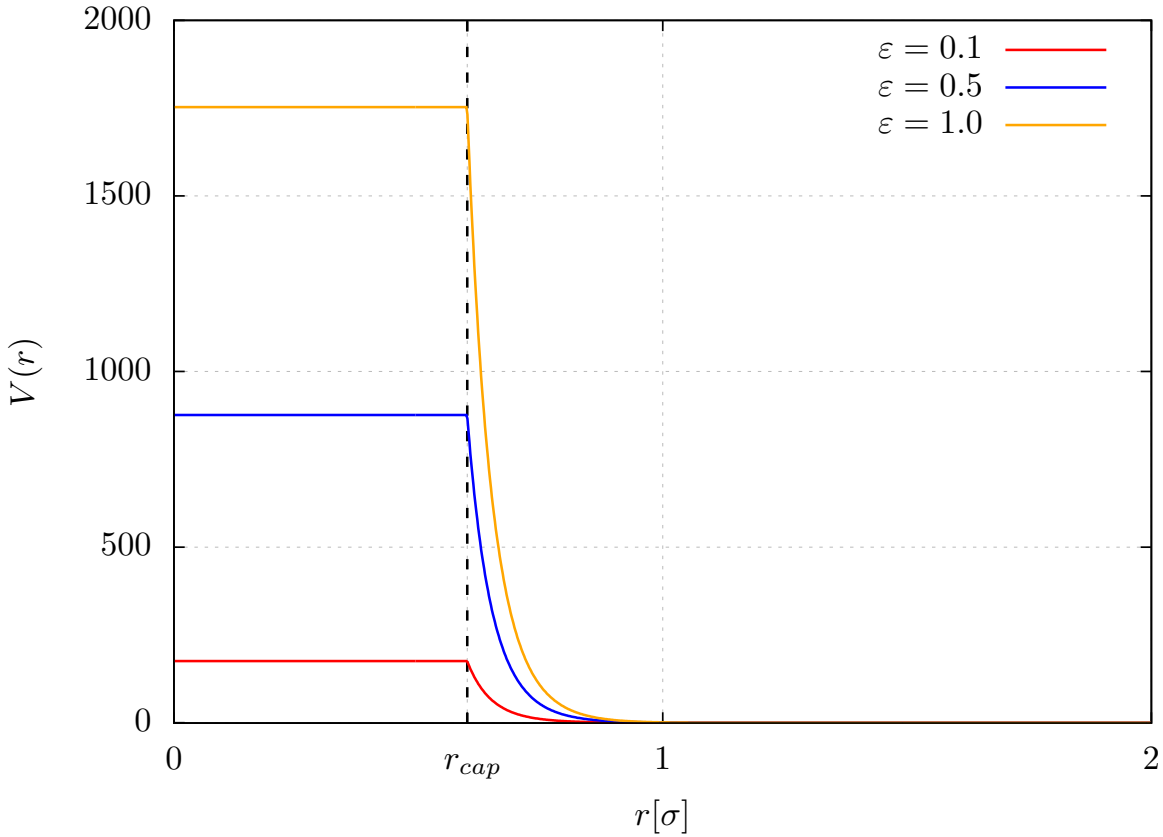


Py script

```
# 3. setup of the integrator and simulation ensemble

integrator      = espressopp.integrator.VelocityVerlet(system)
integrator.dt   = dt

if (temperature != None):
  thermostat             = espressopp.integrator.LangevinThermostat
    (system)
  thermostat.gamma       = 1.0
  thermostat.temperature = temperature
  integrator.addExtension(thermostat)

# 4. adding particles

for pid in range(Npart):
  pos = system.bc.getRandomPos()
  system.storage.addParticle(pid, pos)

system.storage.decompose()

# 5. setting up interaction potential for the warmup

verletlist  = espressopp.VerletList(system,warmup_cutoff)
LJpot       = espressopp.interaction.LennardJonesCapped(epsilon=
              epsilon_start,sigma=sigma,cutoff=warmup_cutoff,
              caprad=capradius,shift='auto')
```

Py script

```
interaction = espressopp.interaction.VerletListLennardJonesCapped(
                verletlist)
interaction.setPotential(type1=0,type2=0,potential=LJpot)
```



Py script

```
# 6. running the warmup loop

system.addInteraction(interaction)

fmt = '%5d %8.4f %8.4f %8.4f %8.4f %8.4f\n'
info_WarmUp = open('WarmUp/info_WarmUp.txt','w')

for step in range(1,warmup_nloops+1):
  integrator.run(warmup_isteps)

  LJpot.epsilon += epsilon_delta
  interaction.setPotential(type1=0, type2=0, potential=LJpot)

  espressopp.tools.analyse.info(system, integrator)
  T_WarmUp = espressopp.analysis.Temperature(system).compute()
  P_WarmUp = espressopp.analysis.Pressure(system).compute()
  Ek_WarmUp = espressopp.analysis.KineticEnergy(system).compute()
  Ep_WarmUp = espressopp.analysis.PotentialEnergy(system,
              interaction).compute()
  Etot_WarmUp = Ek_WarmUp+Ep_WarmUp

  info_WarmUp.write(fmt % (step*warmup_isteps,T_WarmUp,P_WarmUp,
                    Ek_WarmUp,Ep_WarmUp,Etot_WarmUp)+'\n')

info_WarmUp.close()

system.removeInteraction(0)
verletlist.disconnect()
```

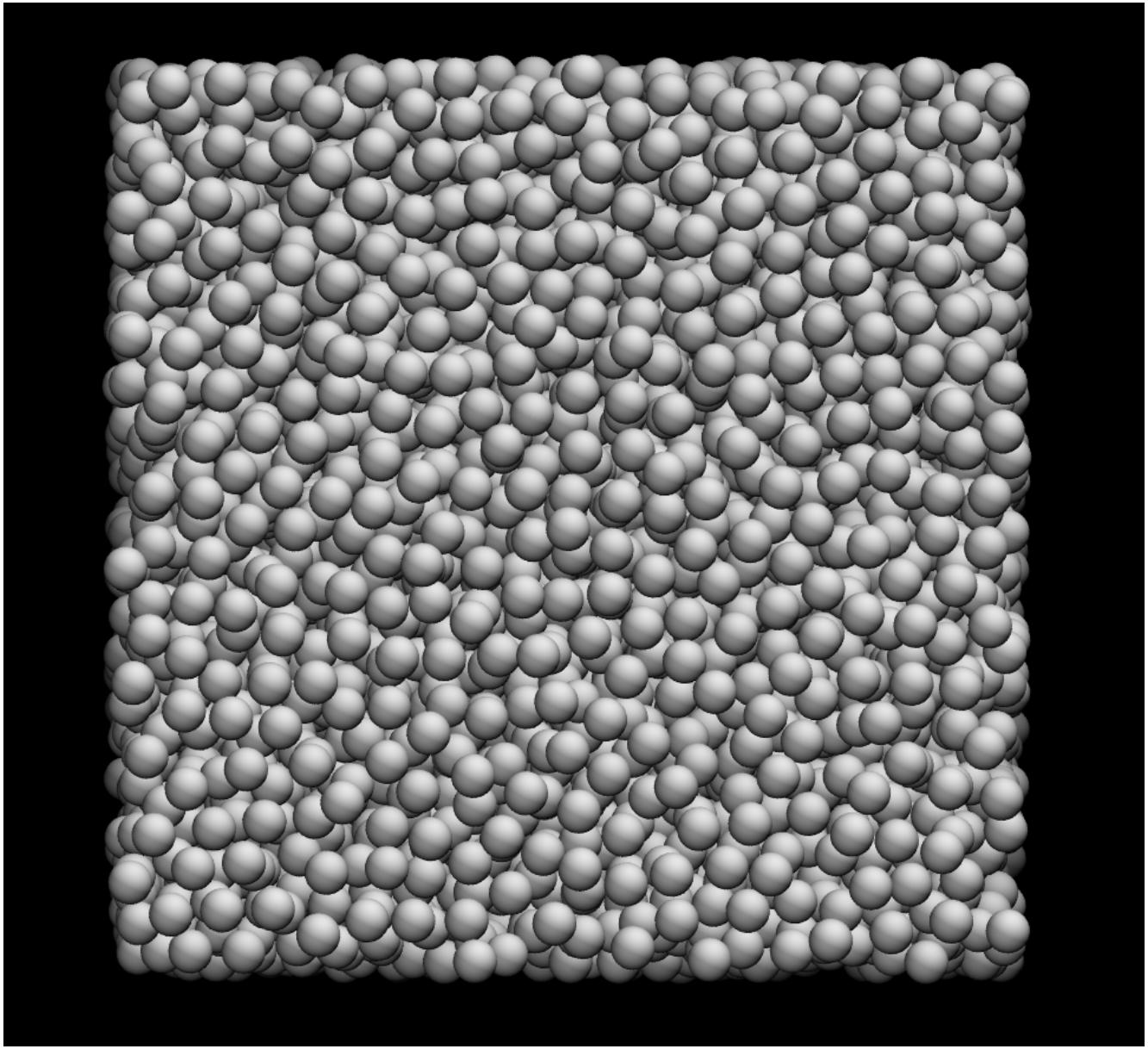
Py script

```
# 7. setting up interaction potential for production run

verletlist  = espressopp.VerletList(system,r_cutoff)
interaction = espressopp.interaction.VerletListLennardJones(
              verletlist)
potential   = interaction.setPotential(type1=0,type2=0,potential=
              espressopp.interaction.LennardJones(epsilon=epsilon,
              sigma=sigma,cutoff=r_cutoff,shift=0.0))

# 8. production run

fmt = '%5d %8.4f %8.4f %8.4f %8.4f %8.4f\n'
info_ProductionRun = open('ProductionRun/info_ProductionRun.txt',
                          'w')

system.addInteraction(interaction)
system.storage.cellAdjust()
integrator.resetTimers()
integrator.step = 0

dump_conf_xyz = espressopp.io.DumpXYZ(system,integrator,
                filename='./ProductionRun/traj.xyz')
```

Py script

```
for step in range(1,equil_nloops+1):
  integrator.run(equil_isteps)

  espressopp.tools.analyse.info(system, integrator)
  T_ProductionRun = espressopp.analysis.Temperature(system).
                    compute()
  P_ProductionRun = espressopp.analysis.Pressure(system).compute()
  Ek_ProductionRun = espressopp.analysis.KineticEnergy(system).
                     compute()
  Ep_ProductionRun = espressopp.analysis.PotentialEnergy(system,
                     interaction).compute()
  Etot_ProductionRun = Ek_ProductionRun+Ep_ProductionRun

  info_ProductionRun.write(fmt % (step*equil_isteps,
                    T_ProductionRun,P_ProductionRun,
                    Ek_ProductionRun,Ep_ProductionRun,
                    Etot_ProductionRun) '\n')
  dump_conf_xyz.dump()

  out_CoordVel = './ProductionRun/xyz/CoordVel_' '{num:05d}'.format
                 (num=integrator.step)+'.txt'
  espressopp.tools.writexyz(out_CoordVel,system,velocities=True,
                    unfolded=False)

info_ProductionRun.close()
```

Snapshot of the simulated system



Perform simulation of the LJ fluid and analyze the results

- simulation:

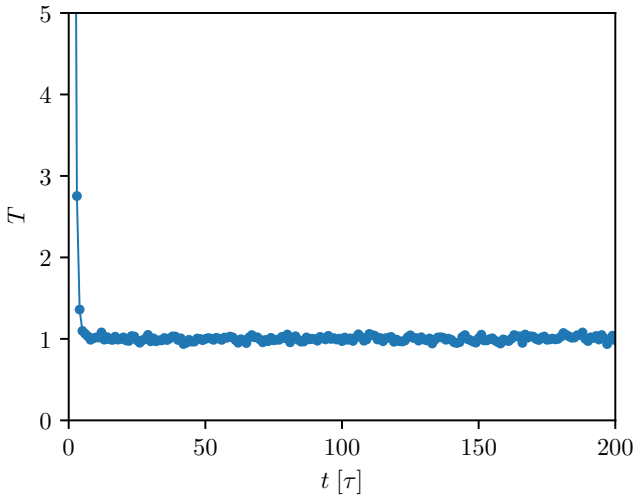
```
cd /home/kemijski/L17/espressopp/examples/lennard_jones

# vim lennard_jones.py # in info_ProductionRun.write() ::: change ::: step*equil_isteps

bash run.sh # run bash script
```

- analysis:

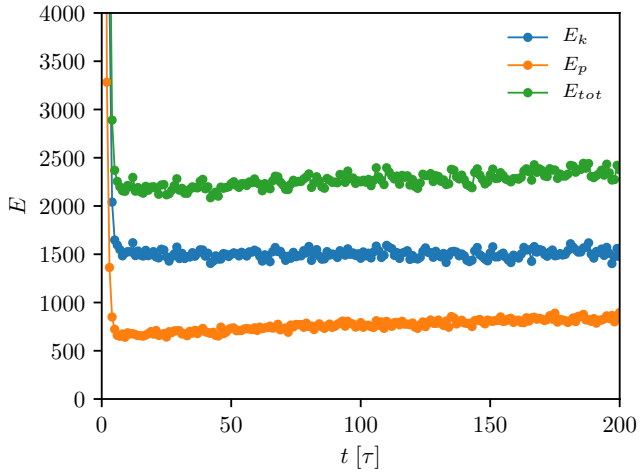
```
# 1. warmup run
cd /home/kemijski/L17/espressopp/examples/lennard_jones/WarmUp

python3 ploting.py # [T(t), P(t), Ek(t), Ep(t), Etot(t)]

 # 2. production run
cd /home/kemijski/L17/espressopp/examples/lennard_jones/ProductionRun

# rsync -avP ./../WarmUp/ploting.py . # ::: some changes

python3 rdf.py # RDF

python3 green_kubo.py # VACF and D ::: vim green_kubo.py ::: change ::: dt = 5*0.005
```

LJ fluid - results

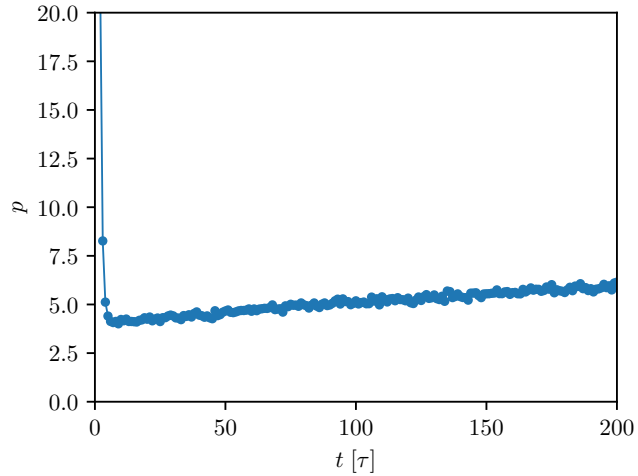
- temperature



- energy

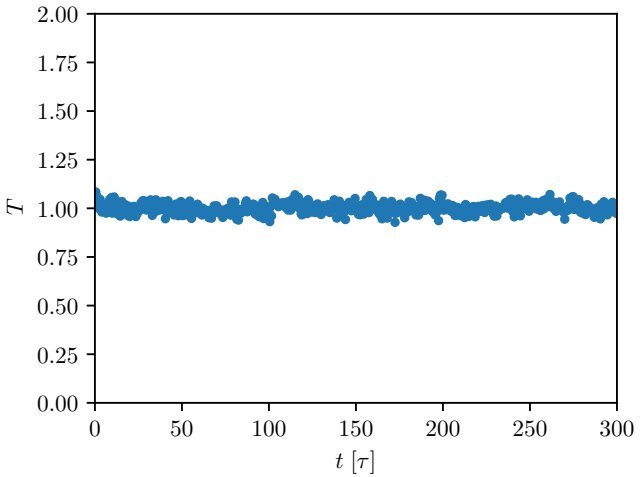


- pressure

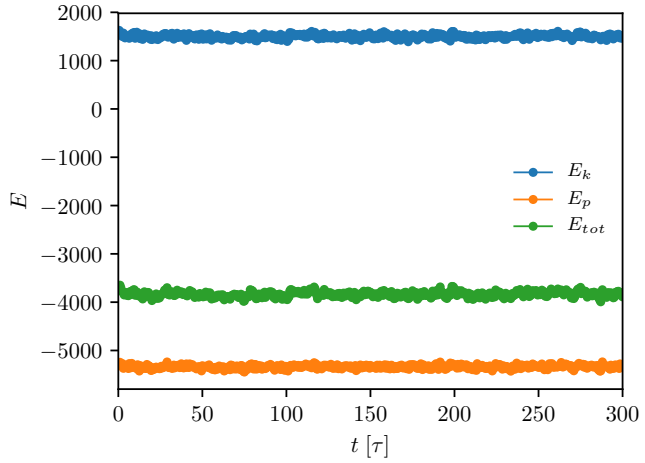


LJ fluid - results

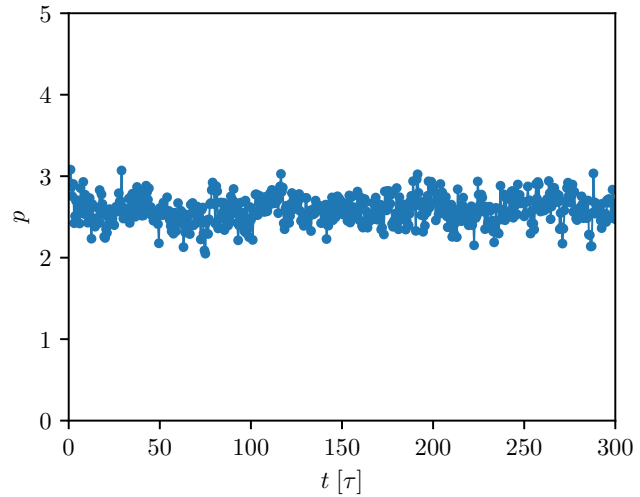
- temperature



- energy

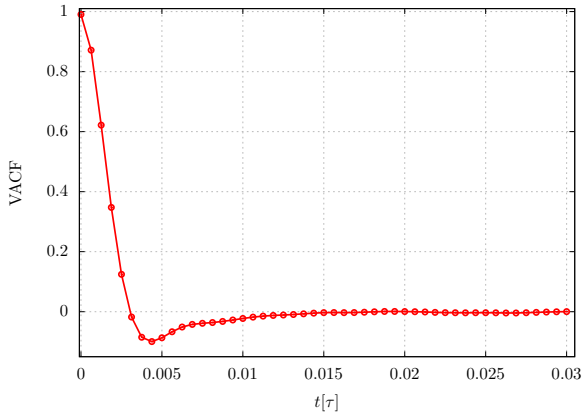


- pressure



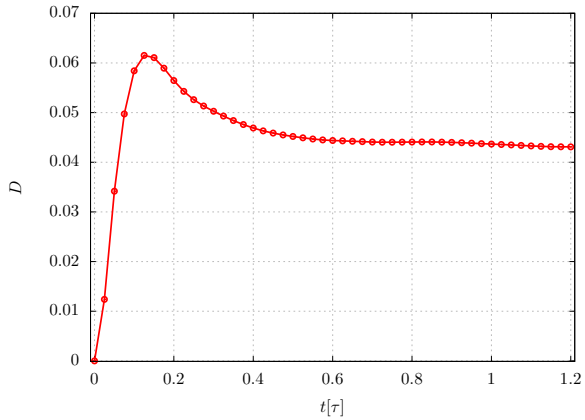
LJ fluid - results

- VACF

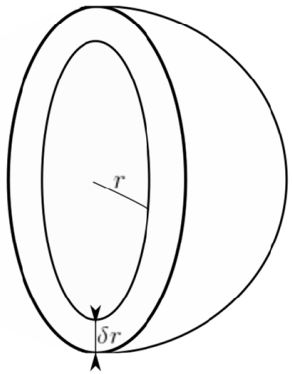


- diffusion coefficient

$$D = \frac{1}{3}\int_0^\infty dt \frac{1}{N}\sum_{i=1}^{N} \langle \dot{\mathbf{r}}_i(0) \cdot \dot{\mathbf{r}}_i(t) \rangle$$



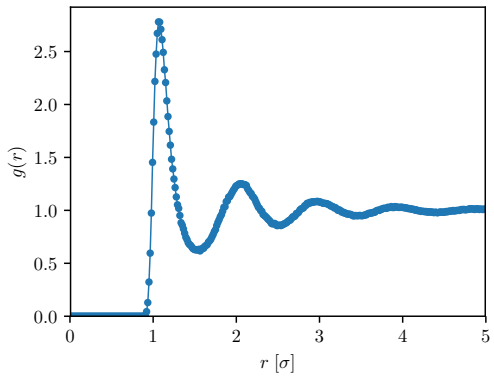
- RDF



$$V = \frac{4}{3}\pi(r+\partial r)^3 - \frac{4}{3}\pi r^3 = 4\pi r^2 \partial r + 4\pi r \partial r^2 + \frac{4}{3}\pi \partial r^3$$

$$V \approx 4\pi r^2 \partial r$$

Number of particles: $4\pi\rho r^2 \partial r$



(1) Tuckerman, M. E. *Statistical Mechanics: Theory and Molecular Simulation*; Oxford University Press, 2010. **(2)** Leach, A. R. *Molecular modelling: principles and applications*; Harlow: Prentice Hall, 2001.

Part II

AdResS simulation
a liquid of tetrahedral molecules

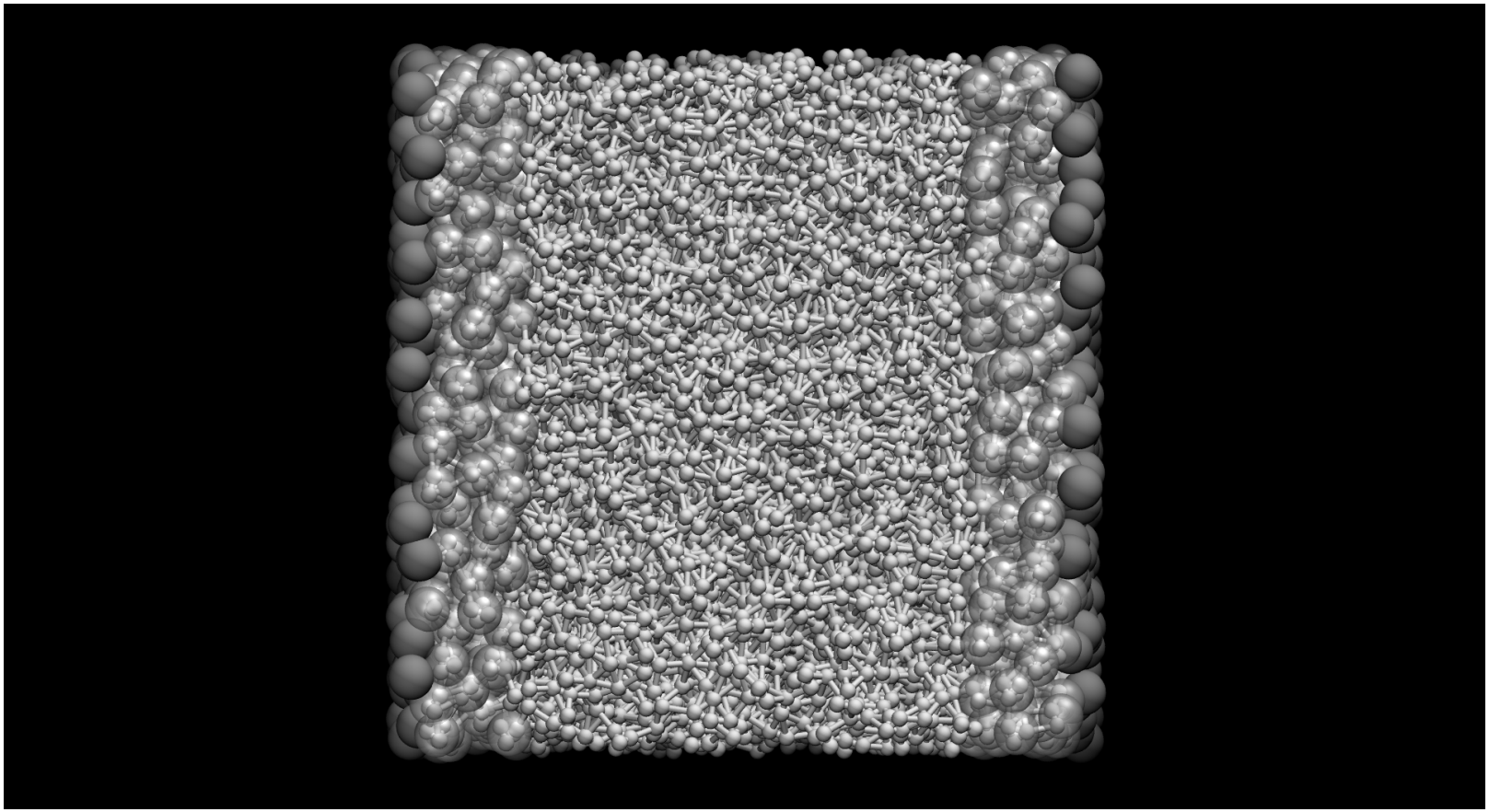
Aims of the 2nd part

We will **explain the `.py` script** for running multiscale simulation of a liquid of tetrahedral molecules, focusing on:

- AdResS scheme,
- system design,
- interactions (e.g., Weeks-Chandler-Andersen (WCA) potential, finite extensible nonlinear elastic (FENE) potential),
- calculation of the coarse-grained (CG) potential (using iterative Boltzmann inversion (IBI) method).

Adaptive resolution scheme

$$\mathbf{F}_{\alpha\beta} = w(\mathbf{r}_\alpha)w(\mathbf{r}_\beta)\mathbf{F}^{\mathrm{AT}}_{\alpha\beta} + [1 - w(\mathbf{r}_\alpha)w(\mathbf{r}_\beta)]\mathbf{F}^{\mathrm{CG}}_{\alpha\beta}$$



(1) Praprotnik, M.; Delle Site, L.; Kremer K. Adaptive resolution molecular-dynamics simulation: Changing the degrees of freedom on the fly. *J. Comput. Chem.* **2005**, *36*, 467. **(2)** Praprotnik, M.; Site Delle, L.; Kremer, K. Multiscale Simulation of Soft Matter: From Scale Bridging to Adaptive Resolution. *Annu. Rev. Phys. Chem.* **2008**, *59*, 545. **(3)** Cortes-Huerto, R.; Praprotnik, M.; Kremer, K.; Delle Site, L. From adaptive resolution to molecular dynamics of open systems. *Eur. Phys. J. B* **2021**, *94*, 189.

Py script

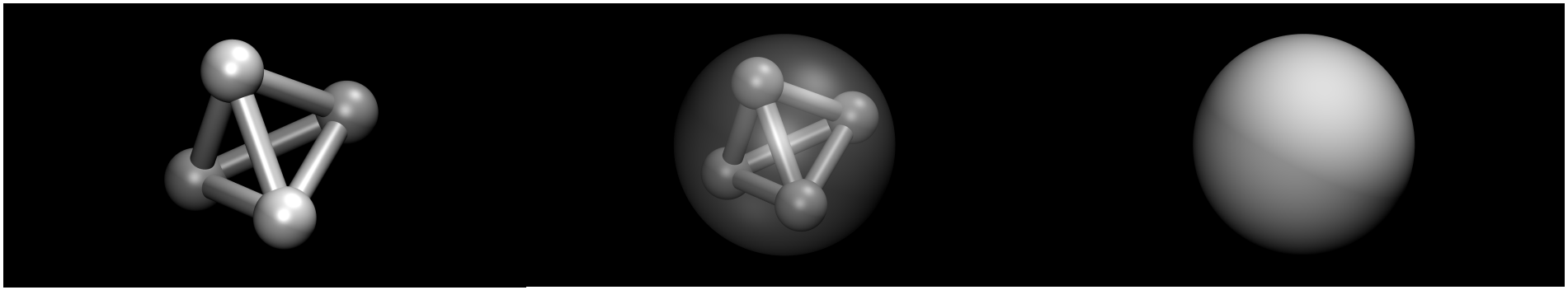
```
import sys
import time
import espressopp
import mpi4py.MPI as MPI
import logging
from espressopp import Real3D, Int3D
from espressopp.tools import decomp
from espressopp.tools import timers

# integration steps, cutoff, skin and thermostat flag
steps = 5000
timestep = 0.0001
intervals = 500

rc = 2.31
rca = 1.122462048309373
skin = 0.4

gamma = 0.5
temp = 1.0

ex_size = 12.5
hy_size = 5.0
```

(1) Praprotnik, M.; Delle Site, L.; Kremer K. Adaptive resolution molecular-dynamics simulation: Changing the degrees of freedom on the fly. *J. Comput. Chem.* **2005**, *36*, 467. **(2)** Praprotnik, M.; Delle Site, L.; Kremer K. Adaptive resolution scheme for efficient hybrid atomistic-mesoscale molecular dynamics simulations of dense liquids. *Phys. Rev. E* **2006**, *73*, 066701.

Py script

```
# read ESPResSo++ configuration file
Lx,Ly,Lz,x,y,z,type,q,vx,vy,vz,fx,fy,fz,bonds = espresso_old.read("
                                                adress.espressopp")
num_particlesCG = 5001
num_particles = len(x)

density = num_particles/(Lx*Ly*Lz)
size = (Lx,Ly,Lz)

system = espressopp.System()
system.rng = espressopp.esutil.RNG()
system.bc = espressopp.bc.OrthorhombicBC(system.rng,size)
system.skin = skin

comm = MPI.COMM_WORLD
nodeGrid = decomp.nodeGrid(comm.size,size,rc,skin)
cellGrid = decomp.cellGrid(size,nodeGrid,rc,skin)

# AdResS domain decomposition
system.storage = espressopp.storage.DomainDecompositionAdress(
                 system,nodeGrid,cellGrid)
```

Py script



```
# prepare AT particles
allParticlesAT = []
allParticles = []
tuples = []
for pidAT in range(num_particles):
    allParticlesAT.append([pidAT,
                           Real3D(x[pidAT],y[pidAT],z[pidAT]),
                           Real3D(vx[pidAT],vy[pidAT],vz[pidAT]),
                           Real3D(fx[pidAT],fy[pidAT],fz[pidAT]),
                           1,1.0,1])
```

Py script

```
# create CG particles from center of mass
for pidCG in range(num_particlesCG):
    cmp = [0,0,0]
    cmv = [0,0,0]
    tmptuple = [pidCG+num_particles]
    # com calculation
    for pidAT in range(4):
        pid = pidCG*4+pidAT
        tmptuple.append(pid)
        pos = (allParticlesAT[pid])[1]
        vel = (allParticlesAT[pid])[2]
        for i in range(3):
            cmp[i] += pos[i]
            cmv[i] += vel[i]
    for i in range(3):
        cmp[i] /= 4.0
        cmv[i] /= 4.0

    allParticles.append([pidCG+num_particles,
                         Real3D(cmp[0],cmp[1],cmp[2]),
                         Real3D(cmv[0],cmv[1],cmv[2]),
                         Real3D(0, 0, 0),
                         0,4.0,0])

    pidCG_out.append(pidCG+num_particles)
```

Py script

```
    for pidAT in range(4):
        pid = pidCG*4+pidAT
        allParticles.append([pid,
                             (allParticlesAT[pid])[1],
                             (allParticlesAT[pid])[2],
                             (allParticlesAT[pid])[3],
                             (allParticlesAT[pid])[4],
                             (allParticlesAT[pid])[5],
                             (allParticlesAT[pid])[6]])

        pidAT_out.append(pid)

    tuples.append(tmptuple)

# add particles
system.storage.addParticles(allParticles,"id","pos","v","f",
                            "type","mass","adrat")

# add tuples
ftpl = espressopp.FixedTupleListAdress(system.storage)
ftpl.addTuples(tuples)
system.storage.setFixedTuplesAdress(ftpl)

# add bonds between AT particles
fpl = espressopp.FixedPairListAdress(system.storage,ftpl)
fpl.addBonds(bonds)
```

Py script

```
# decompose after adding tuples and bonds
system.storage.decompose()

# AdResS Verlet list
vl = espressopp.VerletListAdress(system,cutoff=rc+skin,
     adrcut=rc+skin,dEx=ex_size,dHy=hy_size,
     adrCenter=[18.42225,18.42225,18.42225])

# non-bonded potentials
# LJ Capped WCA between AT and tabulated Morse between CG particles
tabMorse = "pot-morse.txt"
potMorse = espressopp.interaction.Morse(epsilon=0.105,alpha=2.4,
           rMin=rc,cutoff=rc,shift="auto")
writeTabFile(potMorse,tabMorse,N=512,low=0.005,high=4.5)

interNB = espressopp.interaction.VerletListAdressLennardJonesCapped
          (vl,ftpl)
potWCA  = espressopp.interaction.LennardJonesCapped(epsilon=1.0,
          sigma=1.0,shift=True,caprad=0.27,cutoff=rca)
potMorse = espressopp.interaction.Tabulated(itype=2,filename=
           tabMorse,cutoff=rc)
interNB.setPotentialAT(type1=1,type2=1,potential=potWCA)
interNB.setPotentialCG(type1=0,type2=0,potential=potMorse)
system.addInteraction(interNB)
```

Iterative Boltzmann inversion (IBI) method

- **Inversion step**

$$\boxed{U_0^{eff}(r) = -k_B T \ln[g_{\mathrm{AT}}^{cm}(r)]}$$

$k_B \cdots$ Boltzmann constant
$T \cdots$ temperature
$g_{\mathrm{AT}}^{cm} \cdots$ RDF of the COM of CG particles (computed from the all-atom simulation)

  - run CG simulation
  - compute RDF (i.e., g_{CG}^{i})
  - correct the guess for the CG potential

- **Iterative procedure**

$$\boxed{U_{i+1}^{eff}(r) = U_i^{eff}(r) - k_B T \ln\left[\frac{g_{\mathrm{CG}}^{i}(r)}{g_{\mathrm{AT}}^{cm}(r)}\right] \pm V_0\left[1 - \frac{r}{r_{cut}}\right]}$$

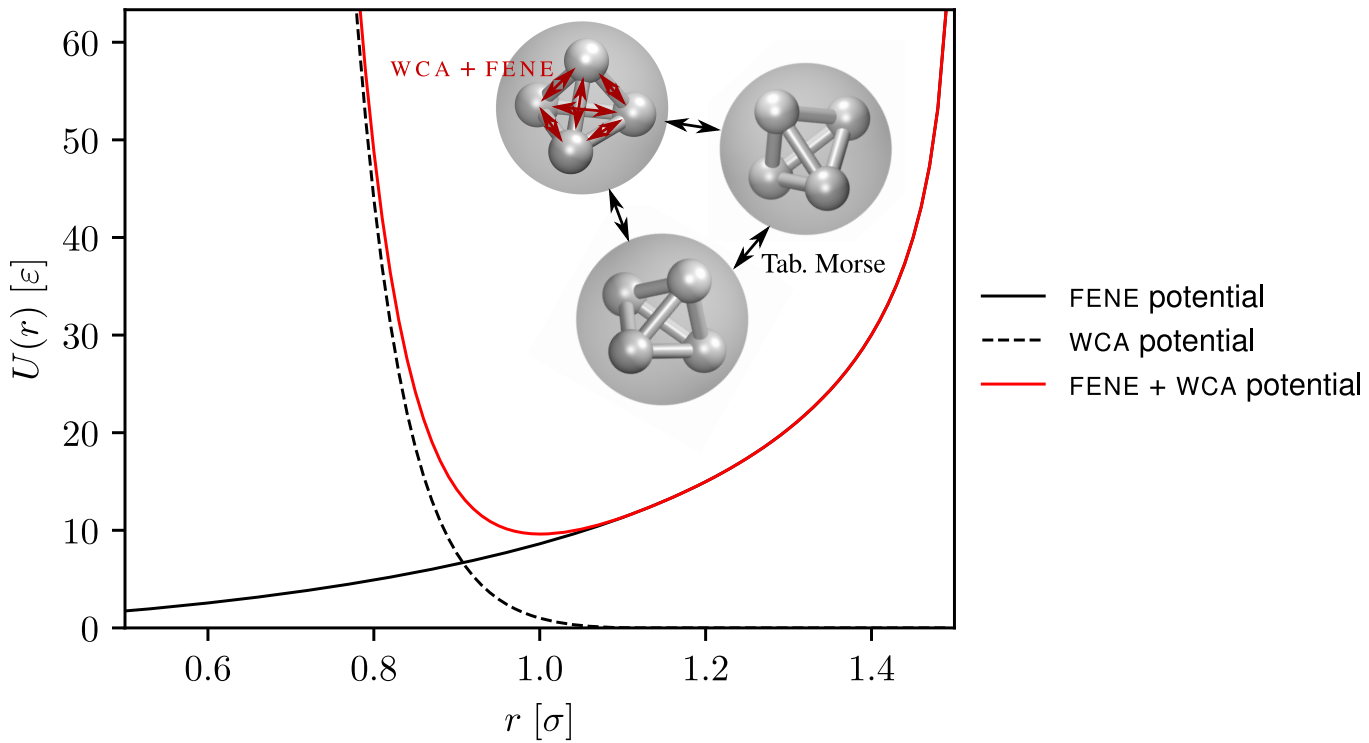
  - iterate until the CG RDF matches the g_{AT}^{cm}

(1) Reith, D.; Pütz, M.; Müller-Plathe, F. Deriving effective mesoscale potentials from atomistic simulations. *J. Comput. Chem.* **2003**, *24*, 1624. **(2)** Bevc, S.; Junghans, C.; Praprotnik, M. STOCK: Structure mapper and online coarse-graining kit for molecular simulations. *J. Comput. Chem.* **2015**, *36*, 467.

Py script

```
# bonded potentials
# FENE and LJ potential between AT particles
potFENE = espressopp.interaction.FENE(K=30.0,r0=0.0,rMax=1.5)
potLJ = espressopp.interaction.LennardJones(epsilon=1.0,sigma=1.0,
        shift=True,cutoff=rca)
interFENE = espressopp.interaction.FixedPairListFENE(system,fpl,
            potFENE)
interLJ = espressopp.interaction.FixedPairListLennardJones(system,
          fpl,potLJ)
system.addInteraction(interFENE)
system.addInteraction(interLJ)

# VV integrator
integrator = espressopp.integrator.VelocityVerlet(system)
integrator.dt = timestep

# add AdResS extension
adress = espressopp.integrator.Adress(system,vl,ftpl)
integrator.addExtension(adress)

# add Langevin thermostat extension
langevin = espressopp.integrator.LangevinThermostat(system)
langevin.gamma = gamma
langevin.temperature = temp
langevin.adress = True
integrator.addExtension(langevin)
```

WCA and FENE potential



Weeks-Chandler-Andersen (WCA) potential

$$U(r) = \begin{cases} 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} + \frac{1}{4}\right] & \text{if } r \leq 2^{1/6}\sigma \\ 0 & \text{if } r > 2^{1/6}\sigma \end{cases}$$

Finite extensible nonlinear elastic (FENE) potential

$$U(r) = -\frac{1}{2} r_{max}^2 K \log\left[1 - \left(\frac{r - r_0}{r_{max}}\right)^2\right]$$

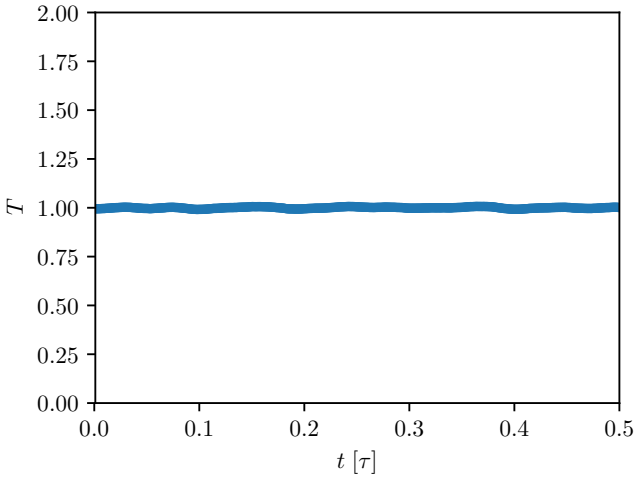
(1) Grest, G. S.; Kremer, K. Molecular dynamics simulation for polymers in the presence of a heat bath. *Phys. Rev. A* **1986**, *33*, 3628.

Py script

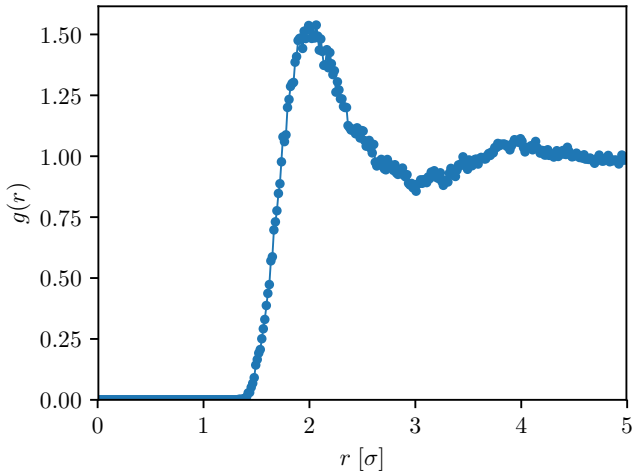
```
# analysis
temperature = espressopp.analysis.Temperature(system)
pressure = espressopp.analysis.Pressure(system)

fmt = '%5d %8.4f %10.5f %12.3f %12.3f %12.3f %12.3f\n'

T = temperature.compute()
P = pressure.compute()
Ek = 0.5 * T * (3 * num_particles)
Ep = interNB.computeEnergy()
Eb = interFENE.computeEnergy()
sys.stdout.write(' step     T           P         Pxy        etotal
           epotential      ebonded     ekinetic\n')
sys.stdout.write(fmt % (0,T,P,Ek+Ep+Eb,Ep,Eb,Ek))

nsteps = steps // intervals
for s in range(1, intervals + 1):
    integrator.run(nsteps)
    step = nsteps*s
    T = temperature.compute()
    P = pressure.compute()
    Ek = 0.5 *T*(3*num_particles)
    Ep = interNB.computeEnergy()
    Eb = interFENE.computeEnergy()
    sys.stdout.write(fmt % (step,T,P,Ek+Ep+Eb,Ep,Eb,Ek))
    system.storage.decompose()
```

Liquid of tetrahedral molecules - results

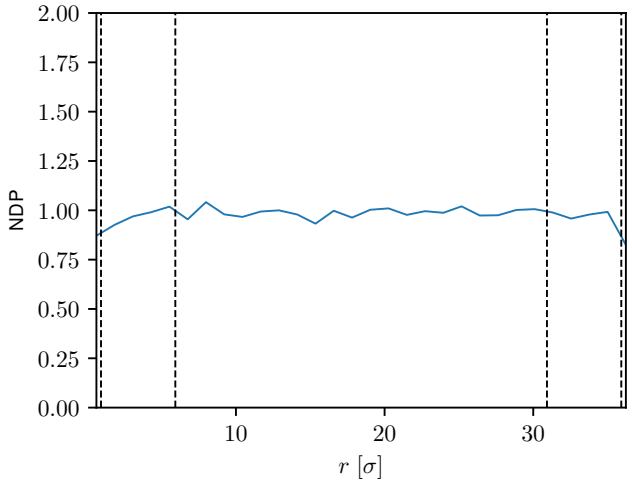
- temperature



- RDF



- normalized density profile (NDP)



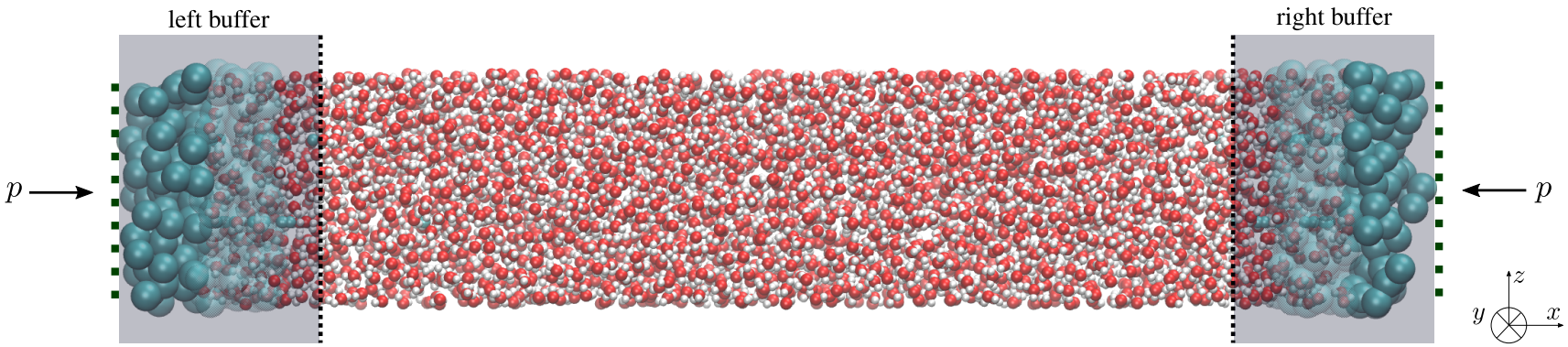
Part III

Open molecular dynamics simulation
SPC water

Aims of the 3rd part

We will **explain** the **role of buffers** and **imposition of the external boundray conditions** (e.g., constant normal load) in the open boundray molecular dynamics method (OBMD).

OBMD setup



- *insertion and deletion of particles*

$$\Delta N_B = \frac{\delta t}{\tau_B}(\langle N_B \rangle - N_B)$$

$\langle N_B \rangle \cdots$ desired number of particles inside the buffer
$N_B \cdots$ number of particles inside the buffer
$\tau_B \cdots$ relaxation time of the buffer

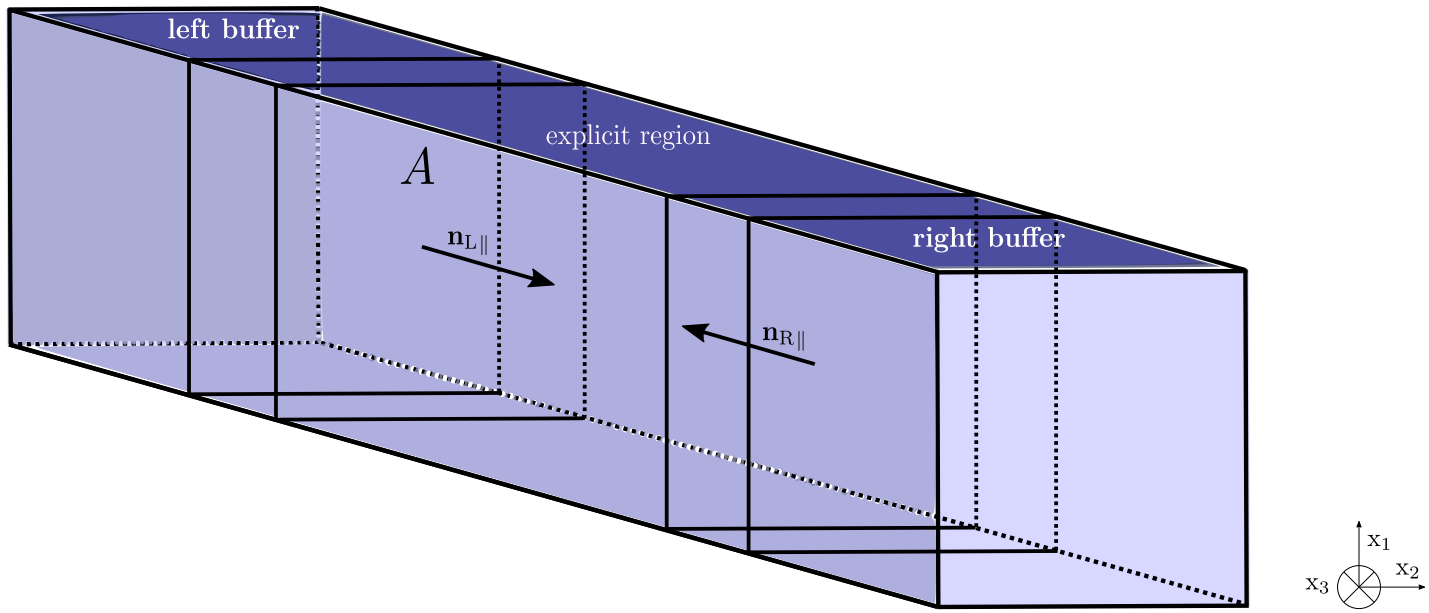
$\Delta N_B < 0 \rightarrow$ particles need to be deleted from the system
$\Delta N_B > 0 \rightarrow$ particles need to be inserted into the system (USHER)

(1) Flekkøy, E. G.; Delgado-Buscalioni, R.; Coveney, P. V. Flux boundary conditions in particle simulations. *Phys. Rev. E* **2005**, *72*, 026703. **(2)** Delgado-Buscalioni, R.; Sablić, J.; Praprotnik, M. Open boundary molecular dynamics. *Eur. Phys. J. Spec. Top.* **2015**, *224*, 2331. **(3)** Delgado-Buscalioni, R.; Coveney, P. V. USHER: An algorithm for particle Insertion in dense fluids. *J. Chem. Phys.* **2003**, *119*, 978.

OBMD setup

- *imposition of the external boundary condition (equilibrium)*

$$\frac{\partial(\rho \mathbf{v})}{\partial t} = -\nabla \cdot \mathsf{J}^P$$
$$\mathsf{J}^P = \rho \mathbf{v} \otimes \mathbf{v} + \Pi + \tilde{\Pi}$$

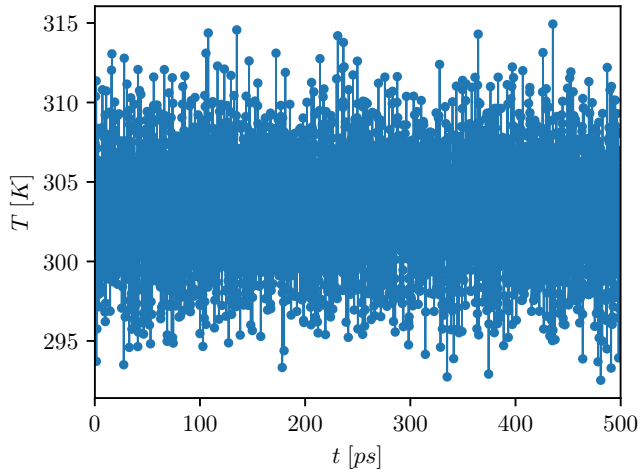


$$\Pi = (p + \pi)\mathsf{I} + \Pi^S$$
$$\Pi^S_{\alpha\beta} = -\eta(\partial_\alpha v_\beta + \partial_\beta v_\alpha - 2/3 \partial_\gamma v_\gamma \delta_{\alpha\beta})$$

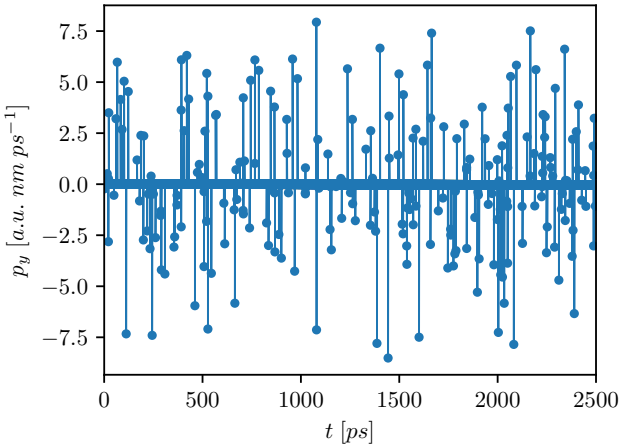
(1) De Fabritiis, G.; Serrano, M.; Delgado-Buscalioni, R.; Coveney, P. V. Fluctuating hydrodynamic modeling of fluids at the nanoscale. *Phys. Rev. E* **2007**, *75*, 026307. **(2)** Delgado-Buscalioni, R.; Coveney, P. V. Continuum-particle hybrid coupling for mass, momentum, and energy transfers in unsteady fluid flow. *Phys. Rev. E* **2003**, *67*, 046704. **(3)** Delle Site, L.; Praprotnik, M. Molecular systems with open boundaries: Theory and simulation. *Phys. Rep.* **2017**, *693*, 1.

SPC water - results

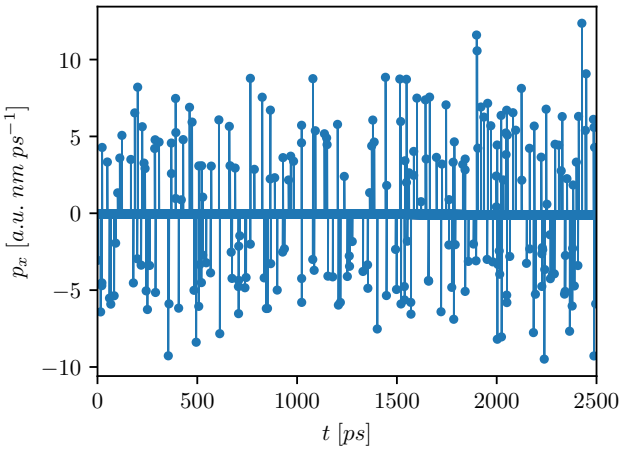
- temperature



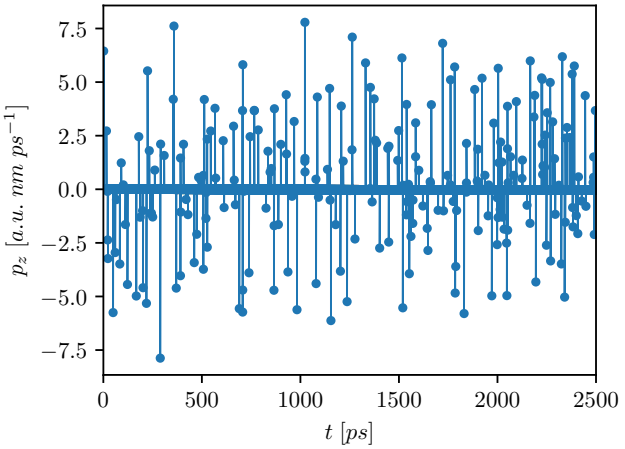
- momentum (p_y)



- momentum (p_x)

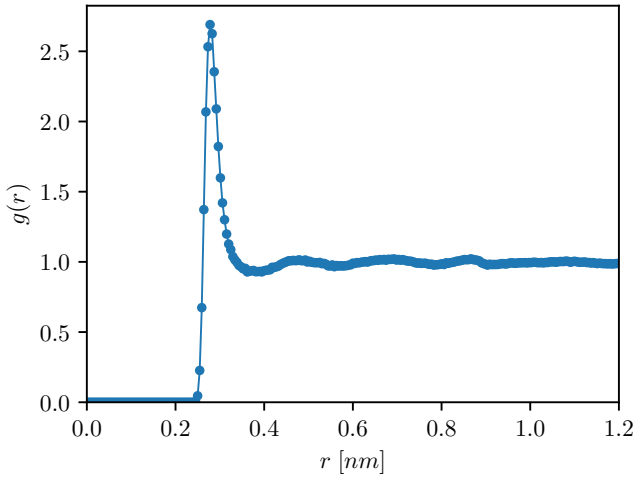


- momentum (p_z)

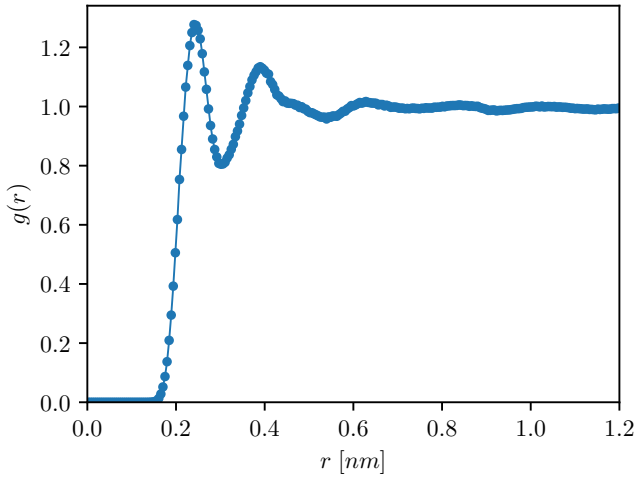


SPC water - results

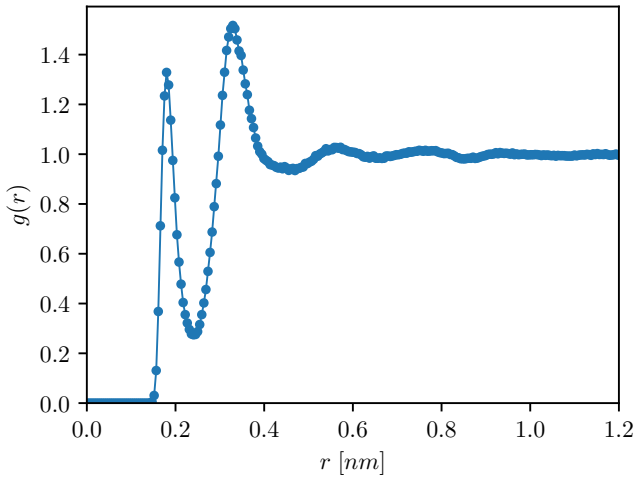
- RDF (OW-OW)



- RDF (HW-HW)



- RDF (OW-HW)



- (NDP)