


POST-PROMD: AN OPEN-SOURCE POST-PROCESSING SOFTWARE FOR MOLECULAR DYNAMICS SIMULATIONS


A PREPRINT

 **Ashkan Shekaari**

 Department of Physics, K. N. Toosi University of Technology

 Tehran, 15875-4416, Iran

 shekaari.theory@gmail.com

 shekaari@email.kntu.ac.ir

April 21, 2024

Abstract

In the present work, a new software, named POST-PROMD, has been developed, which is capable of computing three main phase transition quantities including the Lindemann index, mean square displacement (MSD) of atoms, and radial distribution function (RDF) of systems of particles, only via post-processing the atomic-positions output file, containing atomic positions generated during molecular dynamics (MD) simulations. POST-PROMD runs independently (is not a module patched to any other software); is length-scale-independent (independent of whether the MD simulation has been quantum mechanical or classical); is released for the first time under the GNU GPL (Version 3) as a free and open-source package; has originally been written to post-process the atomic-positions output file "`cp.pos`" generated during Car-Parrinello MD simulations of QUANTUM ESPRESSO; and is software-independent nevertheless, capable of post-processing the MD output of any other software as well.

Keywords Software development · Molecular dynamics · Phase transition · Open source · QUANTUM ESPRESSO

PACS 71.15.Mb · 05.70.Fh

i All software names, as well as emails and URLs, have been hyperlinked and are then clickable.

1 Introduction

One of the most important capabilities of QUANTUM ESPRESSO [1, 2, 3] is to perform density functional molecular dynamics (DFMD) simulations within both Car-Parrinello (CP) [4] and Born-Oppenheimer (BO) [5, 6] frameworks. However, there is still a lack of required codes/modules to post-process raw outputs of such MD simulations generated by the CPV package. As an illustration, the output file "`cp.pos`" of CPMD simulations—including atomic positions of the system at every timestep—contains important information about phase transition/finite-temperature behavior/dynamics of that system, extracting information from which is left to the user.

To this end, i.e., to extract such important information contained in "`cp.pos`", we have written a post-processing package for MD simulations, called POST-PROMD—acronym for "an open-source **POST**-

PROcessing software for **Molecular Dynamics** simulations"—which computes three main phase transition indicators, including (i) the Lindemann index of the whole system, (ii) mean square displacement (MSD) of every atom/particle of the system, and (iii) radial distribution function (RDF) of the whole system.

The present package has been written in Fortran 90/95 [7] the modules of which are related to each other via Linux [8] Bash scripts, and is compatible with Intel Fortran Compiler ifort. The logo of POST-PROMD has been designed by the author using online logo designer <https://app.logo.com>, illustrated in Fig. 1. The present manuscript has also been written and rendered in T_EX studio 3.0.4.



Figure 1: Logo of POST-PROMD.

2 Phase Transition in Solids

Investigating phase transition/finite-temperature behavior of the condensed matter at the computational level of research using MD simulations is carried out via calculating a number of indicators the three important of which are the Lindemann index, MSD, and RDF, as mentioned before. These quantities are prevalently used to probe phase transition in solid materials, namely solid-to-liquid transition; however, they can also be applied to so-called few-body systems [9] of atoms/molecules whose bulk phases are not solid. The point at which phase transition takes place is the same as the melting temperature (T_m) of the solid; at this point, temperature dependence of the three phase transition indicators aforementioned undergo a sudden, substantial increase compared to their former behaviors. The facts aforementioned have clearly been demonstrated in an investigation on phase transition of B₃₆ nanocluster [10].

In the following, we describe the three phase transition quantities of interest, and illustrate/share the associated computer codes.

3 Lindemann Index

The Lindemann index Δ_{rms} , also known as global Lindemann index, or root mean square bond length fluctuation [11], is indeed the extent of fluctuations in bond lengths between atoms/particles of the system, averaged both over all those atoms, and the total timespan of the MD simulation. This quantity is defined as

$$\Delta_{rms} = \frac{2}{N(N-1)} \sum_{i>j} \frac{\left(\langle r_{ij}^2 \rangle_t - \langle r_{ij} \rangle_t^2 \right)^{1/2}}{\langle r_{ij} \rangle_t}, \quad (1)$$

where N is total number of atoms, r_{ij} is distance between atoms i and j , and $\langle \dots \rangle_t$ is time (t) average over the entire trajectory/simulation. $2/N(N-1)$ also means average over total number of atomic pairs/bonds $N(N-1)/2$.

The two modules of POST-PROMD, namely "lind.sh" and "ln.source", are used to compute the Lindemann index of the system according to Eq. 1, being described as follows. Here, the files suffixed with ".sh" are purely Bash scripts, while those with suffix ".source" have mostly been written in Fortran 90/95 (the comments have also been removed in order for the codes to be looked less confusing/complicated).

3.1 lind.sh

```
1 #!/bin/bash
2 cp par.input par2.dat
3 cat ln.source >> par2.dat
4 mv par2.dat make_lind.sh
5 bash make_lind.sh
6 rm make_lind.sh
```

3.2 `ln.source` [↗](#)

```
1  eql=$(((ign0/iprint0)*n0))
2  m=$((n0 + 1))
3  cat cp.pos | sed '1~'$m'd' > pos.pos2
4  cat pos.pos2 | sed '1,'$eql'd' > pos.pos
5  rm pos.pos2
6  cat > module.f90 << EOF
7  module initial_setup
8  implicit none
9  save
10 integer, parameter ::      n = $n0
11 integer, parameter :: int_num = n*(n - 1)/2
12 integer, parameter ::      nstep = $nstep0
13 integer, parameter ::      iprint = $iprint0
14 integer, parameter ::      ign = $ign0
15 integer, parameter ::      rat = (nstep - ign)/iprint
16 integer, parameter :: line_f1 = rat*n
17 integer, parameter :: line_f2 = rat*int_num
18 real, parameter ::      dt = $dt0
19 real, parameter ::      t_tot = rat*dt
20 real, parameter ::      c = dt/t_tot
21 end module
22 EOF
23 ifort -c module.f90
24 cat > fluct1.f90 << EOF
25 program distance_pair
26 use initial_setup
27 implicit none
28 integer i, j, k
29 real*8, dimension(:, :), allocatable :: r
30 real*8, dimension(:, :), allocatable :: pos
31 allocate(pos(3, line_f1))
32 allocate(r(line_f1, line_f1))
33 open(1, file = 'pos.pos')
34 open(2, file = 'rij.data')
35 do i = 1, line_f1
36     read(1, *) pos(:, i)
37 end do
38 do i = 1, line_f1, n
39     do k = 1, n - 1
40         do j = i + k, i + n - 1
41             r(j, i + k - 1) = sqrt((pos(1, j) - pos(1, i + k - 1))**2 &
42                                     + (pos(2, j) - pos(2, i + k - 1))**2 &
43                                     + (pos(3, j) - pos(3, i + k - 1))**2)
44             write(2, *) r(j, i + k - 1)
45         end do
46     end do
47 end do
48 deallocate(pos)
49 deallocate(r)
50 end program
51 EOF
52 cat > fluct2.f90 << EOF
53 program rmsblf
54 use initial_setup
55 implicit none
56 integer i, k
57 real*8, dimension(:), allocatable :: rijsum, rij2sum, rij
```

```

58 real*8, dimension(:), allocatable :: rijtimeavg, rij2timeavg, term
59 real*8 s, delta_rms
60 allocate(rijsum(line_f2))
61 allocate(rij2sum(line_f2))
62 allocate(rij(line_f2))
63 allocate(rijtimeavg(line_f2))
64 allocate(rij2timeavg(line_f2))
65 allocate(term(line_f2))
66 open(1, file = 'rij.data')
67 do i = 1, line_f2
68     read(1, *) rij(i)
69 end do
70 do k = 1, int_num
71     rijsum(k) = 0.
72     rij2sum(k) = 0.
73     term(k) = 0.
74 end do
75 do k = 1, int_num
76     do i = k, line_f2, int_num
77         rijsum(k) = rijsum(k) + rij(i)
78         rij2sum(k) = rij2sum(k) + rij(i)*rij(i)
79     end do
80     rijtimeavg(k) = rijsum(k)*c
81     rij2timeavg(k) = rij2sum(k)*c
82     term(k) = (sqrt(rij2timeavg(k) - &
83         rijtimeavg(k)*rijtimeavg(k)))/rijtimeavg(k)
84 end do
85 s = sum(term)
86 deallocate(rijsum)
87 deallocate(rij2sum)
88 deallocate(rij)
89 deallocate(rijtimeavg)
90 deallocate(rij2timeavg)
91 deallocate(term)
92 delta_rms = s/real(int_num)
93 print*, '    >>> delta_rms =', delta_rms
94 end program
95 EOF
96 ifort module.o fluct1.f90 && ./a.out
97 ifort module.o fluct2.f90 && ./a.out
98 rm *.f90 *.mod a.out pos.pos rij.data *.o

```

4 Mean Square Displacement (MSD)

In contrast to the Lindemann index and radial distribution function, which are quantities related to the entire system, MSD is a single-particle quantity, and POST-PROMD consequently computes it for every individual atom of the system according to

$$\langle \mathbf{r}_I^2(t) \rangle = \frac{1}{n} \sum_{i=1}^n \left[\mathbf{R}_I(t_{0i} + t) - \mathbf{R}_I(t_{0i}) \right]^2, \quad (2)$$

where we average over n different time origins t_{0i} distributed over the entire trajectory with the interval $\mathbf{dt0}$ (included as an input in the file "par.input", described later on in section 6) between any two consecutive t_{0i} for the average. The I runs over number N of atoms, and \mathbf{R}_I is the position vector of the I th atom with respect to the center of mass (COM) of the system. The two modules "msd.sh" and "msd.source", computing MSD according to Eq. 2, are illustrated in the following.

4.1 msd.sh

```

1  #!/bin/bash
2  cp par.input par2.dat
3  cat msd.source >> par2.dat
4  mv par2.dat make_msd.sh
5  bash make_msd.sh
6  rm make_msd.sh

```

4.2 msd.source

```

1  eq1=$((ign0/iprint0)*n0))
2  m=$((n0 + 1))
3  cat cp.pos | sed '1~'$m'd' > pos.pos2
4  cat pos.pos2 | sed '1,'$eq1'd' > pos.pos
5  rm pos.pos2
6  cat > msd.f90 << EOF
7  program msd_maker
8  implicit none
9  integer c, i, j
10 integer, parameter ::      n = $n0
11 integer, parameter ::      ign = $ign0
12 integer, parameter ::      nstep = $nstep0
13 integer, parameter ::      iprint = $iprint0
14 real, parameter ::         dt = $dt00
15 real, parameter ::         au = 2.4189
16 real, parameter ::         scl = 0.00001
17 integer, parameter ::      line = ((nstep - ign)/iprint)*n
18 integer, parameter ::      den = (nstep - ign)/iprint
19 real, parameter :: dt_cp_dot_pos = dt*iprint*au*scl
20 real*8, allocatable, dimension(:) :: s, x, y, z
21 character*12 :: filename
22 allocate (x(line))
23 allocate (y(line))
24 allocate (z(line))
25 allocate (s(line))
26 open(0, file = 'pos.pos')
27 do i = 1, n
28   write(filename, '("msd",i2,".out")') i
29   open(unit=i,file=filename)
30 end do
31 do i = 1, line
32   read(0, *) x(i), y(i), z(i)
33 end do
34 do i = 1, n
35   c = 0
36   s(i) = 0.
37   do j = i, line, n
38     c = c + 1
39     s(i) = s(i) + ((x(j + n) - x(i))*(x(j + n) - x(i)) + &
40      (y(j + n) - y(i))*(y(j + n) - y(i)) + &
41      (z(j + n) - z(i))*(z(j + n) - z(i)))
42     write(i, *) c*dt_cp_dot_pos, s(i)/real(den)
43   end do
44 end do
45 deallocate (x)
46 deallocate (y)
47 deallocate (z)
48 deallocate (s)
49 end program
50 EOF

```

```

51 ifort msd.f90 && ./a.out
52 rm msd.f90 a.out pos.pos
53 mkdir msd_output_files
54 mv msd*.out msd_output_files

```

5 Radial Distribution Function (RDF)

The quantity RDF $[g(r)]$, being calculated both in two and three spatial dimensions, computes the average number $\langle N \rangle$ of atoms within the shell volume V_s ($= 2\pi r dr$ in two, and $4\pi r^2 dr$ in three dimensions) of radius r around COM of the system, between $r - dr/2$ and $r + dr/2$, according to

$$g(r) = \frac{\langle N \rangle}{N} / V_s, \quad (3)$$

where dr is shell width. Indeed, POST-PROMD computes the average of atoms within each shell over the entire simulation, and then repeats the same for the next shell until reaching the outermost that encompasses the entire system. This is why RDF calculation takes a longer time compared to the Lindemann index or MSD. In the following, the two modules "rdf.sh" and "rdf.source", computing RDF according to Eq. 3, are represented.

5.1 rdf.sh

```

1 #!/bin/bash
2 cp par.input par2.dat
3 cat rdf.source >> par2.dat
4 mv par2.dat make_rdf.sh
5 bash make_rdf.sh
6 rm make_rdf.sh

```

5.2 rdf.source

```

1 eq1=$((ign0/iprint0)*n0))
2 m=$((n0 + 1))
3 cat cp.pos | sed '1~'$m'd' > pos.pos2
4 cat pos.pos2 | sed '1,'$eq1'd' > pos.pos
5 rm pos.pos2
6 cat > module.f90 << EOF
7 module shared_data
8 implicit none
9 save
10 integer, parameter ::      n = $n0
11 real, parameter ::      rmax = $rmax0
12 integer, parameter ::      nstep = $nstep0, iprint = $iprint0
13 integer, parameter ::      ign = $ign0
14 integer, parameter ::      snap = (nstep - ign)/iprint
15 real, parameter ::      cf = 0.529177010059357
16 real, parameter ::      delta_r = $delta_r0
17 real, parameter ::      pi = 4.*atan(1.)
18 integer, parameter ::      n2 = nint(rmax/delta_r)
19 integer, parameter ::      line_num = n2*snap
20 end module
21 EOF
22 ifort -c module.f90
23 cat > rdf.f90 << EOF
24 program radial_df
25 use shared_data
26 implicit none
27 integer :: i, j, cxy, cxz, cyz, c2
28 real :: r

```

```

29 real, allocatable, dimension(:) :: x, y, z, d2xy, d2xz, d2yz, d3
30 real xcom, ycom, zcom
31 open(1, file = 'pos.pos')
32 open(2, file = 'rdf2.data')
33 allocate(x(n*snap))
34 allocate(y(n*snap))
35 allocate(z(n*snap))
36 allocate(d2xy(n*snap))
37 allocate(d2xz(n*snap))
38 allocate(d2yz(n*snap))
39 allocate(d3(n*snap))
40 do j = 1, snap
41     xcom = 0.; ycom = 0.; zcom = 0.
42     do i = (j - 1)*n + 1, j*n
43         read(1, *) x(i), y(i), z(i)
44         xcom = xcom + x(i)
45         ycom = ycom + y(i)
46         zcom = zcom + z(i)
47     end do
48     xcom = xcom/real(n)
49     ycom = ycom/real(n)
50     zcom = zcom/real(n)
51     do i = (j - 1)*n + 1, j*n
52         x(i) = x(i) - xcom
53         y(i) = y(i) - ycom
54         z(i) = z(i) - zcom
55         d2xy(i) = sqrt(x(i)*x(i) + y(i)*y(i))*cf
56         d2xz(i) = sqrt(x(i)*x(i) + z(i)*z(i))*cf
57         d2yz(i) = sqrt(y(i)*y(i) + z(i)*z(i))*cf
58         d3(i) = sqrt(x(i)*x(i) + y(i)*y(i) + z(i)*z(i))*cf
59     end do
60     r = 0.
61     do while (r <= rmax)
62         r = r + delta_r
63         cxy = 0; cxz = 0; cyz = 0; c2 = 0
64         do i = (j - 1)*n + 1, j*n
65             if(d2xy(i) < r + 0.5*delta_r .and. d2xy(i) >= r - 0.5*delta_r) then
66                 cxy = cxy + 1
67             end if
68             if(d2xz(i) < r + 0.5*delta_r .and. d2xz(i) >= r - 0.5*delta_r) then
69                 cxz = cxz + 1
70             end if
71             if(d2yz(i) < r + 0.5*delta_r .and. d2yz(i) >= r - 0.5*delta_r) then
72                 cyz = cyz + 1
73             end if
74             if(d3(i) < r + 0.5*delta_r .and. d3(i) >= r - 0.5*delta_r) then
75                 c2 = c2 + 1
76             end if
77         end do
78         write(2, *) r, cxy/(2.*pi*r*delta_r*n), &
79             cxz/(2.*pi*r*delta_r*n), cyz/(2.*pi*r*delta_r*n), &
80             c2/(4.*pi*r*r*delta_r*n)
81     end do
82 end do
83 deallocate(x)
84 deallocate(y)
85 deallocate(z)
86 deallocate(d2xy)
87 deallocate(d2xz)

```

```
88 deallocate(d2yz)
89 deallocate(d3)
90 end program
91 EOF
92 ifort module.o rdf.f90 && ./a.out
93 cat > rdf2.f90 << EOF
94 program radial_df2
95 use shared_data
96 implicit none
97 integer i, j
98 real, allocatable, dimension(:) :: r, rdf2xy, rdf2xz, rdf2yz, rdf3
99 real :: s1xy, s1xz, s1yz, s2
100 open(1, file = 'rdf2.data')
101 open(2, file = 'rdf5.out')
102 allocate(r(line_num))
103 allocate(rdf2xy(line_num))
104 allocate(rdf2xz(line_num))
105 allocate(rdf2yz(line_num))
106 allocate(rdf3(line_num))
107 do i = 1, line_num
108     read(1, *) r(i), rdf2xy(i), rdf2xz(i), rdf2yz(i), rdf3(i)
109 end do
110 do j = 1, n2
111     s1xy = 0.; s1xz = 0.; s1yz = 0.; s2 = 0.
112     do i = j, line_num, n2
113         s1xy = s1xy + rdf2xy(i)
114         s1xz = s1xz + rdf2xz(i)
115         s1yz = s1yz + rdf2yz(i)
116         s2 = s2 + rdf3(i)
117     end do
118     write(2, *) r(j), s1xy/snap, s1xz/snap, s1yz/snap, s2/snap
119 end do
120 deallocate(r)
121 deallocate(rdf2xy)
122 deallocate(rdf2xz)
123 deallocate(rdf2yz)
124 deallocate(rdf3)
125 end program
126 EOF
127 ifort module.o rdf2.f90 && ./a.out
128 rm *.f90 a.out *.o *.mod rdf2.data pos.pos
129 mkdir rdf_output_file
130 mv rdf5.out rdf_output_file
```

6 The Input

So far, we have demonstrated POST-PROMD modules that perform calculation (placed in `<post-promd_root_directory/src>`). The Bash scripts suffixed with ".sh" use the same input files "cp.pos" and "par.input" to do calculation. The former ("cp.pos") is indeed an output file, formerly generated by the CPV package of QUANTUM ESPRESSO; the latter ("par.input"), on the other hand, is completely user-defined, containing a number of numerical parameters some of which have already been defined in MD input file of QUANTUM ESPRESSO. These input parameters are described in detail, as follows:

- `n0` = Total number of particles (also included in MD input file of QUANTUM ESPRESSO).
- `nstep0` = Total number of steps (also included in MD input file of QUANTUM ESPRESSO).
- `iprint0` = Number of steps between successive output writings (also included in MD input file of QUANTUM ESPRESSO).

- `ign0` = Number of steps for thermalization (thermal equilibration), to have reliable statistical averages. In fact, whether or not the initial atomic positions are random, it takes a time for the entire system to reach thermal equilibrium according to the target temperature defined in the simulation. Over this time interval, statistical averages, such as total energy, are not valid because they are only meaningful at thermal equilibrium according to theory (thermodynamics/statistical mechanics). Usually, integer values between 50 and 500 steps are suitable. Evidently, systems with larger numbers of particles take longer times/steps to reach thermal equilibrium (to be more accurate, the user has to check the diagram of total energy vs. simulation step/index to determine the time/step at which total energy begins to get converged).
- `dt0` = Timestep, included in (1st row, 2nd column) of the MD output file "`cp.pos`".
- `dt00` = Timestep in MD input file of QUANTUM ESPRESSO.
- `delta_r0` = Increment value in radius r ranging from zero (COM of the system) to the outermost shell encompassing the whole system. It is a very sensitive parameter; the value 0.0005 Å works well.
- `rmax0` = Distance from COM to the outermost shell of the system. For example, the value 5 Å is suitable for B₃₆ nanocluster [10].

Note that in "`par.input`", there must not be any blank space before and after the equal signs (=). The sample input files ("`cp.pos`" and "`par.input`") have also been placed in `<post-promd_root_directory/input>`.

An example of "`par.input`" is as follows:

```
1 n0=14
2 nstep0=20000
3 iprint0=10
4 ign0=500
5 dt0=0.00120944
6 dt00=5
7 delta_r0=0.0005
8 rmax0=5
```

7 Release Information and Download

POST-PROMD is released for the first time ever, as a free and open-source package, with assigned version v.1.0.0, under the GNU General Public License (Version 3, 29 June 2007); also been uploaded to a number of leading repositories including GitHub, GitLab, Zenodo with doi:10.5281/zenodo.10982023, and Software Heritage. All the material included in this distribution is free and open-source software, meaning that one is allowed to redistribute/modify it under the GNU GPL (Version 3). This package is a new feature; without any warranty, even the implied warranty, of merchantability/fitness for a particular purpose; runs independently; and is not a module patched to any other software such as QUANTUM ESPRESSO; however, the author's main motivation to write the package has been to fill the gap in post-processing the CPMD outputs of QUANTUM ESPRESSO, and this is why the specific format of the file "`cp.pos`" has been considered as a benchmark (to post-process the MD outputs of other computational packages, the user has to apply only minor changes to come up with compatible input-file-formatting).

Each of the repositories aforementioned, as well as each "POST-PROMD" word, is clickable and navigates the user to a download page. Here, we provide again the download links nevertheless:

- <https://zenodo.org/records/10982023/files/shekaari-theory/POSTPROMD-v.1.0.0.zip?download=1>
- <https://github.com/shekaari-theory/POSTPROMD/tree/main>
- https://gitlab.com/shekaari-theory/POSTPROMD/-/raw/main/post-promd-v.1.0.0.tar.gz?ref_type=heads&inline=false
- <https://zenodo.org/records/10982023/files/shekaari-theory/POSTPROMD-v.1.0.0.zip?download=1>
- https://archive.softwareheritage.org/browse/origin/directory/?origin_url=https://github.com/shekaari-theory/POSTPROMD



8 How To Cite

Please, in publications or presentations arising from this work, cite:

- A. Shekaari. "Shekaari-theory/postpromd: POST-PROMD". Zenodo, April 16, 2024. <https://doi.org/10.5281/zenodo.10982023>.

and

- A. Shekaari. "POST-PROMD: an open-source POST-PROcessing software for Molecular Dynamics simulations". ResearchGate, April 16, 2024. <http://dx.doi.org/10.13140/RG.2.2.21312.34569>.

More formats can also be generated at the bottom of this webpage.

9 How To Run ⚙️

No need to install, POST-PROMD would be ready to run only via unzipping the source file `post-promd-v.1.0.0.tar.gz`. To this end, the user needs to pursue the following simple steps:

1. Unzip the source file.
2. Place "`cp.pos`" in `<post-promd_root_directory/src>`.
3. Edit "`par.input`" according to your MD setup, then place it in `<post-promd_root_directory/src>`.
4. Open a terminal in `<post-promd_root_directory/src>`.
5. To compute the Lindemann index, run in terminal: `>_ bash lind.sh`.
6. To compute MSD, run in terminal: `>_ bash msd.sh`.
7. To compute RDF, run in terminal: `>_ bash rdf.sh`.

Note that the Fortran part of POST-PROMD, namely the files suffixed with `".source"`, are compatible with the Intel Fortran Compiler ifort; therefore, the user either has to have ifort installed, or modify the `".source"` files simply via replacing the word `"ifort"` with any other available Fortran compiler, such as `gfortran`, `mpif90`, etc. The output files generated by POST-PROMD are also described as follows:

1. The output of `bash lind.sh` is displayed in terminal.
2. Those of `bash msd.sh` will be in the directory `<msd_output_files>`, each of which belongs to one atom of the system. Each of these files, named with the template `"msd***.out"`, contain two columns of numerical data, which, in order, are: timestep, and MSD. The user has to plot these files.
3. The output of `bash rdf.sh`, namely `"rdf5.out"`, will be in the directory `<rdf_output_file>`; the number '5' stands for the five columns of numerical data contained in that file, being, in order: r (radius), rdf (RDF values) on xy plane, rdf on xz plane, rdf on yz plane, and rdf in three dimensions. The user also has to plot `"rdf5.out"`.

10 Example Files 📄

The present distribution of POST-PROMD also contains example files, placed in the directory `<post-promd_root_directory/example>`, which are related to the CPMD simulation of the unit cell of SLSiN (single-layer silicon nitride; chemical formula: Si_3N_4 ; containing 14 atoms) [12], at $T = 5$ K. The related outputs have also been placed in the directory `<post-promd_root_directory/example/reference>`.

11 Concluding Remarks

We have developed a new software named POST-PROMD, capable of post-processing the atomic-positions output file "cp.pos", generated by Car-Parrinello molecular dynamics (CPMD) simulations of QUANTUM ESPRESSO, in order to compute three main phase transition indicators including the Lindemann index, per-atom mean square displacement (MSD), and radial distribution function (RDF) of the system. Analyzing the temperature dependence of such quantities is widely used to investigate the finite-temperature behaviors, and to estimate the melting points of systems of atoms/particles. POST-PROMD is an independent package, not a module patched to any other software, and can then independently run to post-process any file containing atomic positions of the system stored at all timesteps as an input file generated by any MD software. Nevertheless, the author's main motivation has been to post-process the atomic-positions file ("cp.pos") generated by QUANTUM ESPRESSO; this is why the specific format of this file has been considered as a benchmark in POST-PROMD. This package is free and open-source, released for the first time under the GNU GPL (Version 3), being technically independent of whether the MD simulation is performed classically or quantum mechanically, so is also applicable to classical simulations as well.

References

- [1] P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G.L. Chiarotti, M. Cococcioni, I. Dabo, A.D. Corso, S. de Gironcoli, S. Fabris, G. Fratesi, R. Gebauer, U. Gerstmann, C. Gougousis, A. Kokalj, M. Lazzeri, L. Martin-Samos, N. Marzari, F. Mauri, R. Mazzarello, S. Paolini, A. Pasquarello, L. Paulatto, C. Sbraccia, S. Scandolo, G. Sclauzero, A.P. Seitsonen, A. Smogunov, P. Umari, R.M. Wentzcovitch, Quantum ESPRESSO: a modular and open-source software project for quantum simulations of materials, *J. Phys.: Cond. Matter* 21 (2009) 395502.
- [2] P. Giannozzi, O. Andreussi, T. Brumme, O. Bunau, M. Buongiorno Nardelli, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, M. Cococcioni, N. Colonna, I. Carnimeo, A. Dal Corso, S. de Gironcoli, P. Delugas, R.A. DiStasio Jr, A. Ferretti, A. Floris, G. Fratesi, G. Fugallo, R. Gebauer, U. Gerstmann, F. Giustino, T. Gorni, J. Jia, M. Kawamura, H.-Y. Ko, A. Kokalj, E. Küçükbenli, M. Lazzeri, M. Marsili, N. Marzari, F. Mauri, N.L. Nguyen, H.-V. Nguyen, A. Otero-de-la-Roza, L. Paulatto, S. Poncé, D. Rocca, R. Sabatini, B. Santra, M. Schlipf, A.P. Seitsonen, A. Smogunov, I. Timrov, T. Thonhauser, P. Umari, N. Vast, X. Wu, S. Baroni, Advanced capabilities for materials modelling with Quantum ESPRESSO, *J. Phys.: Condens. Matter* 29 (2017) 465901.
- [3] P. Giannozzi, O. Baseggio, P. Bonfá, D. Brunato, R. Car, I. Carnimeo, C. Cavazzoni, S. de Gironcoli, P. Delugas, F. Ferrari Ruffino, A. Ferretti, N. Marzari, I. Timrov, A. Urru, S. Baroni, Quantum ESPRESSO toward the exascale, *J. Chem. Phys.* 152 (2020) 154105.
- [4] R. Car, M. Parrinello, The unied approach to density functional and molecular dynamics in real space, *Solid State Commun.* 62 (1987) 403.
- [5] D. Marx, J. Hutter, *Ab Initio Molecular Dynamics: Basic Theory and Advanced Methods*, 1st edn., Cambridge University Press, Cambridge, 2009.
- [6] M. Born, R. Oppenheimer, Zur quantentheorie der molekeln, *Ann. Phys.* 389 (1927) 457.
- [7] S.J. Chapman, *Fortran for Scientists and Engineers*, 4th edn., McGraw-Hill Education, 2 Penn Plaza, New York, NY 10121, 2018.
- [8] L. Torvalds, The Linux edge, *Commun. ACM* 42 (1999) 38.
- [9] A. Shekaari, M. Jafari, Phase-transition behavior of $(\text{H}_2\text{O})_{n=1-4}$ few-body systems from Car-Parrinello molecular dynamics, *Phase Transit.* 94 (2021) 889.
- [10] A. Shekaari, M. Jafari, Finite temperature properties and phase transition behavior of quasi-planar B_{36} nanocluster from first principles, *Mater. Res. Express* 6 (2019) 025014.
- [11] F.A. Lindemann, The calculation of molecular vibration frequencies, *Phys. Z.* 11 (1910) 609.
- [12] A. Shekaari, M. Jafari, Unveiling the first post-graphene member of silicon nitrides: A novel 2D material, *Comput. Mater. Sci.* 180 (2020) 109693.