

NB: The graded, first version of the report must be returned if you hand in a second time!

H1b: MD simulation – dynamic properties

Andréas Sundström and Linnea Hesslow

November 23, 2018

Task N ^o	Points	Avail. points
Σ		

Introduction

The velocity Verlet algorithm is a semi-implicit and efficient method to simulate an ensemble of particles whose trajectories are governed by Newton's equation of motion. Accordingly, it is a suitable algorithm to study molecular dynamics and to determine statistical properties of a system. Here, we use the velocity Verlet algorithm to study a system of aluminum atoms in an fcc lattice. By scaling the positions, momenta and lattice parameter, we can equilibrate the temperature and pressure to prescribed values. We study the aluminum system at 500 °C and 700 °C, which correspond to the solid and liquid state respectively, and compute two dynamic quantities: mean square displacements and the velocity correlation function.

Task 1: potential energy

The theoretical lattice parameter for aluminum can be determined by calculating the minimum potential energy per unit cell in a lattice with zero initial momenta for all particles.

Figure 1 shows the potential energy as a function of the lattice parameter. We used a quadratic fit to find the minimum energy, and obtained $V_{eq} \approx 65.38 \text{ \AA}^3$. This corresponds to the equilibrium lattice parameter $a_{eq} \approx 4.029 \text{ \AA}$ at 0 K, which we took as the initial lattice parameter for the following tasks.

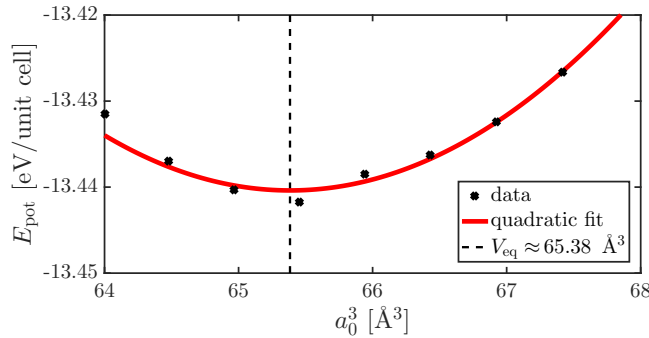


Figure 1: The potential energy per unit cell for aluminum as a function of the lattice parameter cubed.

We find that figure 1 looks similar to the figure 1 in the homework problem file.

Task 2: determine the timestep

With the random noise, the temperature and the energy differ between runs, but are in the same order of magnitude. From figure 2, we determine that $dt = 5 \cdot 10^{-3} \text{ ps} = 5 \text{ fs}$ is a sufficient time step. This is in line with the lecture notes, where it is stated that a suitable timestep would normally be a few femtoseconds, or somewhat larger for heavy atoms.

We note that the temperature is higher than desired value of 600-800 K. The temperatures and energies up to one standard deviation are quantified in table 1.

Table 1: Energies and temperatures with one standard deviation uncertainties for four different values of the time steps.

dt [ps]	T [K]	E_{tot} [eV/unit cell]
10^{-2}	$847 \pm 4.2\%$	$-12.55 \pm 1.2 \cdot 10^{-2}\%$
$5 \cdot 10^{-3}$	$1157 \pm 3.8\%$	$-12.24 \pm 3.6 \cdot 10^{-3}\%$
$2 \cdot 10^{-3}$	$1058 \pm 3.7\%$	$-12.35 \pm 6.6 \cdot 10^{-4}\%$
$1 \cdot 10^{-3}$	$841 \pm 3.7\%$	$-12.58 \pm 3.6 \cdot 10^{-4}\%$

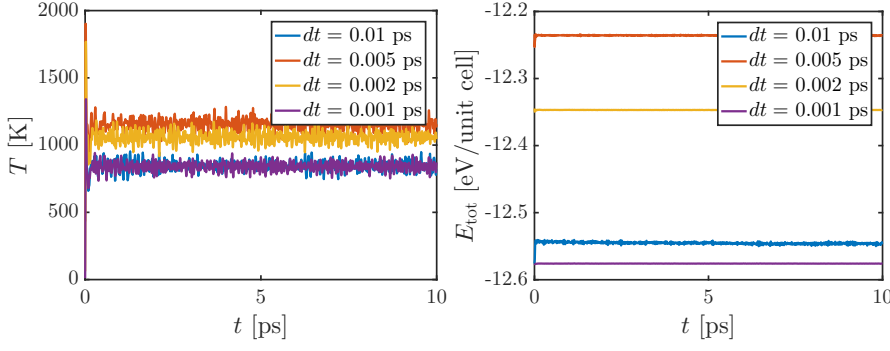


Figure 2: The temperature and kinetic energy per unit cell as a function of time for four different timesteps.

Tasks 3 and 4: Temperature and pressure equilibration

We set $\tau_P = \tau_T = 100dt$, where $dt = 5 \cdot 10^{-3}$ ps, and equilibrated the temperature and pressure by scaling the particle momenta and positions (and box size) respectively. Choosing a slower equilibration time did not affect the results qualitatively. Both temperature and pressure were equilibrated in the same Verlet loop, but for the higher temperature the the system was first melted by increasing the temperature to 900 °C. To determine the isothermal compressibility κ , the values of Young's modulus Y and shear modulus G were taken from Physics Handbook, table T 1.1. From F 1.15 in Physics Handbook, the bulk modulus can then calculated as

$$B = \frac{YG}{9G - 3Y} \quad \kappa_{Al} = \frac{1}{B} \approx 6.6444 \cdot 10^5 \text{ bar}, \quad (1)$$

where $1 \text{ bar} = 6.2415 \cdot 10^{-7} \text{ eV/\AA}^3$ in atomic units. **However, we set $\kappa = 100\kappa_{Al}$ since the pressure equilibration happened on a much longer timescale than τ_P with $\kappa = \kappa_{Al}$. We have not yet figured out why this is.**

The results are shown in figure 3, where we overlay the instantaneous values of \mathcal{T} and \mathcal{P} with a moving average using 250 time steps. The desired temperatures and pressures were approximately obtained in the equilibration process. Although the average pressure was slightly below zero, this is within the fluctuation error bars and is in line with the figure 2 in the homework problem document.

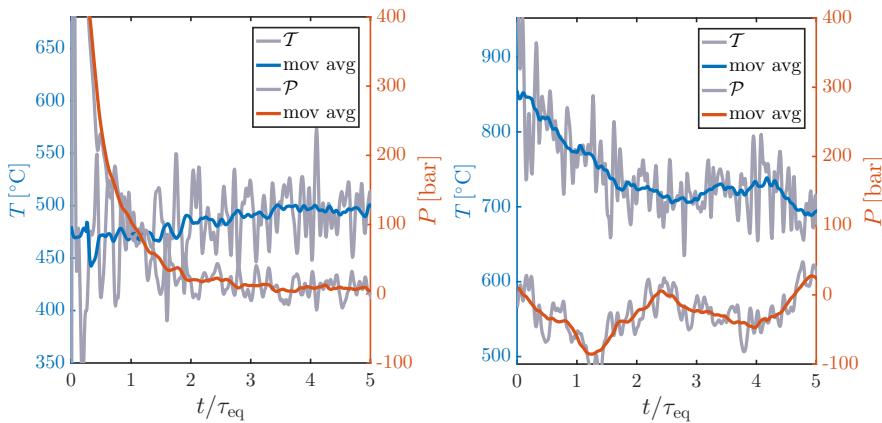


Figure 3: The instantaneous values of \mathcal{T} and \mathcal{P} overlaid with with a moving average using 100 time steps, which corresponds to $\Delta t = \tau_P/2$. Left panel: $T = 500^\circ\text{C}$, right panel: $T = 500^\circ\text{C}$.

The equilibrium values of the lattice parameter were found to be

$$a_0 \approx 4.10 \text{ \AA}, \quad T = 500^\circ\text{C}, \quad (2)$$

$$a_0 \approx 4.29 \text{ \AA}, \quad T = 700^\circ\text{C}. \quad (3)$$

These values are larger than the zero-temperature constants, and it is reasonable that the higher 700 °C case corresponds to a larger lattice parameter at constant pressure.

Tasks 3-5: particle trajectories

Starting with the temperature- and pressure equilibrated systems from the previous section, we study the particle trajectories for both systems. Here, we decrease the timestep to $dt = 5 \cdot 10^{-4}$ ps and the simulation length to $t_{\text{end}} = 5$ ps to get better statistics. This was mostly motivated by increasing the resolution in tasks 6-7.

Equation (82) in MD lecture notes:

$$\Delta_{\text{MSD}}(t) = \lim_{T \rightarrow \infty} \frac{1}{T} \int_0^T dt' \frac{1}{N_{\text{atoms}}} \sum_{i=0}^{N_{\text{atoms}}-1} [\mathbf{r}_i(t+t') - \mathbf{r}_i(t')]^2 \quad (4)$$

\Rightarrow

$$\Delta_{\text{MSD}}(t_k) \approx \frac{1}{N_T - k} \frac{1}{N_{\text{atoms}}} \sum_{j=0}^{N_T-k-1} \sum_{i=0}^{N_{\text{atoms}}-1} [\mathbf{r}_i(t_{k+j}) - \mathbf{r}_i(t_j)]^2 \quad (5)$$

First, we note that the cumulative averages of the instantaneous temperatures and pressures stayed close to their initial values. This is shown in figure 4.

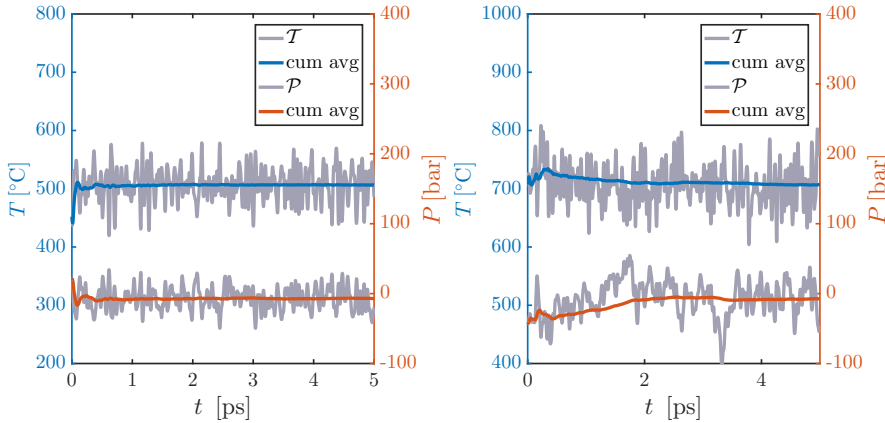


Figure 4: The instantaneous values, and the cumulative averages, of the temperature and the pressure in the production runs. Left panel: $T = 500$ °C, right panel: $T = 700$ °C

We now consider the particle trajectories. Figure 5 shows the trajectories of five individual particles along with the mean square displacement as determined in equation (5). We can clearly see that the particle trajectories are bounded in the left figure 5, while for the high-temperature case in the right panel, they increase as square root of time ($\Delta_{\text{MSD}} \propto t$). Consequently, the former is in a solid state while the latter is in a liquid state.

The self-diffusion coefficient as determined by the average slope of the mean square displacement, was calculated to $D_s \approx 0.52 \text{ Å}^2/\text{ps}$.

Task 7

The average “power” content in a variable, $X(t')$, at some time, t , during some range of time, T , can be defined as

$$P_X(t, T) = \frac{1}{T} \int_{t-T/2}^{t+T/2} dt' X^2(t'). \quad (6)$$

This quantity can (in physically relevant systems) also be defined for the process over all,

$$P_X = \lim_{T \rightarrow \infty} P_X(T) = \lim_{T \rightarrow \infty} \frac{1}{T} \left\langle \int_0^T dt' X^2(t') \right\rangle. \quad (7)$$

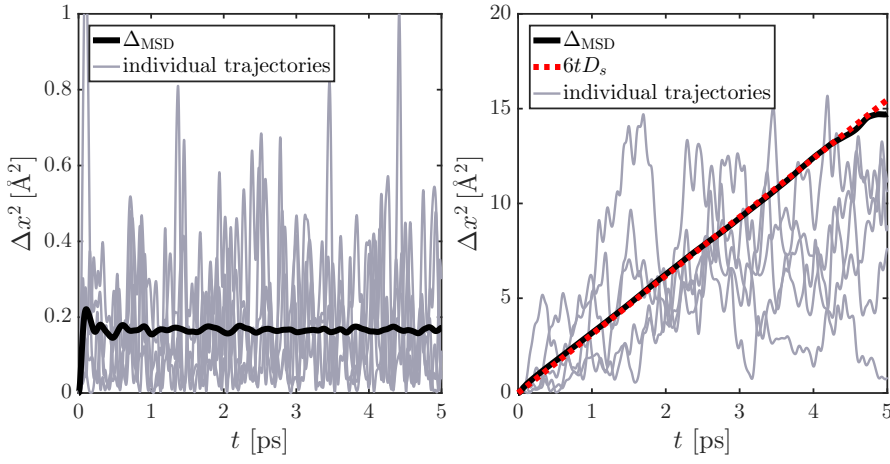


Figure 5: !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Left panel: $T = 500^\circ\text{C}$, right panel: $T = 500^\circ\text{C}$

At this stage, we can introduce a We have the Fourier transform

$$\hat{f}(\omega) = \int_{-\infty}^{\infty} dt f(t) e^{i\omega t}, \quad (8)$$

Using these two functions, we can define a power spectrum

$$\begin{aligned} \hat{P}(\omega) &= \langle |\hat{v}(\omega)|^2 \rangle_A = \langle \hat{v}(\omega) \cdot \hat{v}^*(\omega) \rangle_A \\ &= \left\langle \int_{-\infty}^{\infty} dt v(t) e^{i\omega t} \int_{-\infty}^{\infty} dt' v(t') e^{-i\omega t'} \right\rangle_A, \end{aligned} \quad (9)$$

where \hat{v}^* denotes the complex conjugate of \hat{v} . We can now change variables to $t = t' + \tau$ and note that the atom averages only falls on the velocities, which gives

$$\hat{P}(\omega) = \int_{-\infty}^{\infty} d\tau e^{i\omega\tau} \int_{-\infty}^{\infty} dt' \langle v(t' + \tau) \cdot v(t') \rangle_A, \quad (10)$$

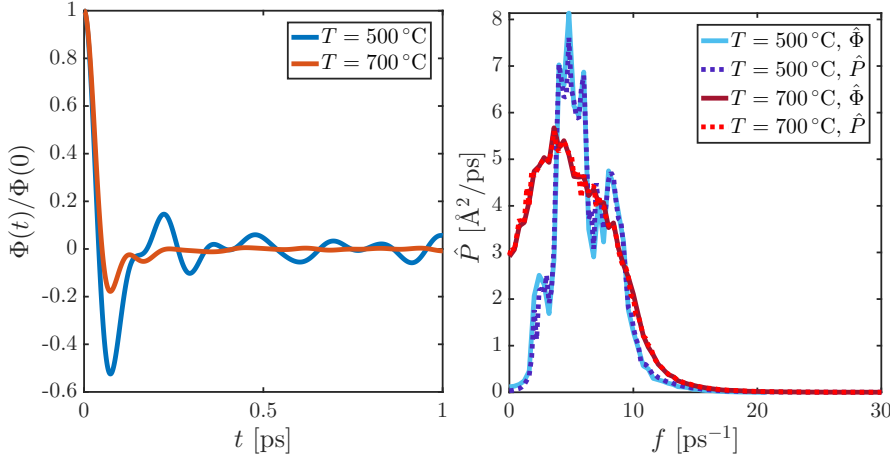


Figure 6: !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

The self-diffusion coefficient as determined by the power spectral density at $f = 0$, was found to be $D_s = 0.49 \text{ Å}^2/\text{ps}$, which is close to the value obtained from the mean square displacement, as expected.

Concluding discussion

We study a

From both the mean square displacements and the velocity correlation function, the solid state is

A Source Code

Include all source code here in the appendix. Keep the code formatting clean, use indentation, and comment your code to make it easy to understand. Also, break lines that are too long. (Keep them under 80 characters!)

A.1 Main program task 1: main_T1.c

```
1  /*
2   main_T1.c Task 1 H1b
3   */
4  #include <stdio.h>
5  #include <math.h>
6  #include <stdlib.h>
7
8  #include "initfcc.h"
9  #include "alpotential.h"
10
11 #define N_cells 4
12 #define N_lattice_params 25
13
14 /* Main program */
15 int main()
16 {
17     int N_atoms = 4*N_cells*N_cells*N_cells;
18     double a0;
19     double a0_min = 4.0;
20     double a0_max = 4.2;
21     double da0 = (a0_max - a0_min)/N_lattice_params;
22
23     double (*pos)[3] = malloc(sizeof(double[N_atoms][3]));
24     double *energy = malloc(sizeof(double[N_lattice_params]));
25
26     FILE *file_pointer;
27
28     for (int i=0; i<N_lattice_params; i++){
29         a0 = a0_min + i*da0;
30         init_fcc(pos, N_cells, a0);
31         // energy per unit cell
32         energy[i] = get_energy_AL(pos, N_cells*a0, N_atoms )*4/N_atoms;
33     }
34
35     file_pointer = fopen("../data/lattice_energies.tsv", "w");
36     for (int i=0; i<N_lattice_params; i++){
37         a0 = a0_min + i*da0;
38         fprintf(file_pointer, "%.8f \t %.8f \n", a0, energy[i]);
39     }
40     fclose(file_pointer);
41
42     free(pos); pos = NULL;
43     free(energy); energy = NULL;
44     return 0;
45 }
```

A.2 Main program Task 2: main_T2.c

```
1  /*
2   MD_main.c
3
4   Created by Anders Lindman on 2013-10-31.
5   */
6
7  #include <stdio.h>
8  #include <math.h>
9  #include <stdlib.h>
10 #include <time.h>
11
12 #include "initfcc.h"
13 #include "alpotential.h"
14 #include "funcs.h"
15
16 #define N_cells 4
17 #define AMU 1.0364e-4
18 #define kB 8.6173303e-5
19
20 /* Main program */
21 int main()
22 {
23
24     int N_atoms = 4*N_cells*N_cells*N_cells;
```

```

25 double m_Al = 27*AMU;
26
27 double a_eq = 4.03;
28
29 double noise_amplitude = 6.5e-2 * a_eq;
30 double t_max=10;
31 double dt = 1e-3;
32 int N_timesteps = t_max/dt;
33 double t, E_kin;
34
35 double (*pos)[3] = malloc(sizeof(double[N_atoms][3]));
36 double (*momentum)[3] = malloc(sizeof(double[N_atoms][3]));
37 double (*forces)[3] = malloc(sizeof(double[N_atoms][3]));
38 double *temperature = malloc(sizeof(double[N_timesteps]));
39 double *E_tot = malloc(sizeof(double[N_timesteps]));
40
41 FILE *file_pointer;
42
43
44 /* ----- TASK 2 ----- */
45
46 init_fcc(pos, N_cells, a_eq); // initialize fcc lattice
47 add_noise( N_atoms, 3, pos, noise_amplitude ); // adds random noise to pos
48 set_zero( N_atoms, 3, momentum); // set momentum to 0
49 get_forces_AL( forces, pos, a_eq*N_cells, N_atoms); //initial cond forces
50
51 for (int i=0; i<N_timesteps; i++){
52     /*
53      The loop over the timesteps first takes a timestep according to the
54      Verlet algorithm, then calculates the energies and temeperature.
55     */
56     timestep_Verlet (N_atoms, pos, momentum, forces, m_Al, dt, a_eq*N_cells);
57
58     E_kin = get_kin_energy(N_atoms, momentum, m_Al );
59     E_tot[i] = (E_kin + get_energy_AL(pos, a_eq*N_cells, N_atoms))*4/N_atoms;
60
61     /*  $3NkB^*T/2 = 1/(2m) * \sum_{i=1}^N p_i^2 = p_{sq}/(2m)$  */
62     temperature[i] = E_kin * 2/(3*N_atoms*kB);
63 }
64
65 /* Write tempertaure to file */
66 char file_name[100];
67 sprintf(file_name, "../data/temperature_dt-%0.0e_Task2.tsv", dt);
68 file_pointer = fopen(file_name, "w");
69 for (int i=0; i<N_timesteps; i++){
70     t = i*dt; // time at step i
71     fprintf(file_pointer, "%.4f \t %.8f \n", t, temperature[i]);
72 }
73 fclose(file_pointer);
74
75 /* Write total energy to file */
76 sprintf(file_name, "../data/total_energy_dt-%0.0e_Task2.tsv", dt);
77 file_pointer = fopen(file_name, "w");
78 for (int i=0; i<N_timesteps; i++){
79     t = i*dt; // time at step i
80     fprintf(file_pointer, "%.4f \t %.8f \n", t, E_tot[i]);
81 }
82 fclose(file_pointer);
83
84 free(pos); pos = NULL;
85 free(momentum); momentum = NULL;
86 free(forces); forces = NULL;
87 free(temperature); temperature = NULL;
88 free(E_tot); E_tot = NULL;
89 return 0;
90 }

```

A.3 Temperature and pressure equilibration for tasks 3-7 : main_T3.c

```

1  /*
2  MD_main.c
3
4  Created by Anders Lindman on 2013-10-31.
5  */
6
7  #include <stdio.h>
8  #include <math.h>
9  #include <stdlib.h>
10 #include <time.h>
11
12 #include "initfcc.h"
13 #include "alpotential.h"
14 #include "funcs.h"
15
16 #define N_cells 4
17 /* define constants in atomic units: eV, , ps, K */

```

```

18 #define AMU 1.0364e-4
19 #define degC_to_K 273.15
20 #define bar 6.2415e-07
21 #define kB 8.61733e-5
22
23 /* Main program */
24 int main()
25 {
26     char file_name[100];
27
28     int N_atoms = 4*N_cells*N_cells*N_cells;
29     double m_Al = 27*AMU;
30     /*
31      * Values of Young's and shear modulus, Y and G resp., taken from
32      * Physics Handbook, table T 1.1. Bulk modulus then calculated as
33      *  $B = Y \cdot G / (9 \cdot G - 3 \cdot Y)$  [F 1.15, Physics Handbook]
34      *  $\kappa = 1/B$ 
35      */
36     double kappa_Al = 100/(6.6444e+05 * bar); // STRANGE FACTOR 100 OFF !!!
37     double a_eq = 4.03;
38     double cell_length = a_eq*N_cells;
39     double inv_volume = pow(N_cells*cell_length, -3);
40     double noise_amplitude = 6.5e-2 * a_eq;
41
42     double T_final_C= 500;
43     int nRuns = 1; //2 if melt, 1 otherwise
44     double T_melt_C = 900;
45
46     double P_final_bar= 1;
47
48     double T_eq;
49     double P_eq = P_final_bar*bar;
50     double dt = 5e-3;
51     double tau_T = 100*dt;
52     double tau_P = 100*dt;
53     //double t_Teq= 10*tau_T; //equilibration times
54     double t_eq= 15*tau_P; //equilibration times
55     int N_timesteps = t_eq/dt;
56
57     double alpha_T, alpha_P, alpha_P_cube_root;
58     double t, E_kin, virial;
59
60
61     double (*pos)[3] = malloc(sizeof(double[N_atoms][3]));
62     double (*momentum)[3] = malloc(sizeof(double[N_atoms][3]));
63     double (*forces)[3] = malloc(sizeof(double[N_atoms][3]));
64     double *temperature = malloc(sizeof(double[N_timesteps]));
65     double *pressure = malloc(sizeof(double[N_timesteps]));
66
67
68     FILE *file_pointer;
69
70     /* ----- TASK 3 ----- */
71
72
73     init_fcc(pos, N_cells, a_eq); // initialize fcc lattice
74     add_noise( N_atoms, 3, pos, noise_amplitude ); // adds random noise to pos
75     set_zero( N_atoms, 3, momentum); // set momentum to 0
76     get_forces_AL( forces, pos, cell_length, N_atoms); //initial cond forces
77
78     /*
79     for (int i=0; i<N_timesteps_Teq; i++){
80         //
81         The loop over the timesteps first takes a timestep according to the
82         Verlet algorithm, then calculates the energies and temeperature.
83         //
84         timestep_Verlet(N_atoms, pos, momentum, forces, m_Al, dt, cell_length);
85
86         E_kin = get_kin_energy(N_atoms, momentum, m_Al );
87         virial = get_virial_AL(pos, cell_length, N_atoms);
88
89         // PV = NkT + virial
90         pressure[i] = inv_volume * (1.5*E_kin + virial);
91         //  $3N \cdot k_B \cdot T/2 = 1/(2m) \cdot \sum_{i=1}^N p_i^2 = p_{sq}/(2m)$ 
92         temperature[i] = E_kin * 1/(1.5*N_atoms*kB);
93
94
95         alpha_T = 1 + 2*dt*(T_eq - temperature[i]) / (tau_T * temperature[i]);
96         scale_mat(N_atoms, 3, momentum, sqrt(alpha_T));
97         temperature[i]*=alpha_T;
98     }
99     */
100
101
102     for (int irun=0; irun < nRuns; irun++){// last run: final, irun = 0
103         if (irun == nRuns - 1){ // final run
104             T_eq = T_final_C + degC_to_K;
105         }else{
106             T_eq = T_melt_C + degC_to_K;
107         }
108         for (int i=0; i<N_timesteps; i++){

```



```

109  /*
110     The loop over the timesteps first takes a timestep according to the
111     Verlet algorithm, then calculates the energies and temeperature.
112  */
113  timestep_Verlet(N_atoms, pos, momentum, forces, m_Al, dt, cell_length);
114
115
116  E_kin = get_kin_energy(N_atoms, momentum, m_Al );
117  virial = get_virial_AL(pos, cell_length, N_atoms);
118
119  /*  $3Nk_B T/2 = 1/(2m) * \sum_{i=1}^N p_i^2 = p_{sq}/(2m)$  */
120  temperature[i] = E_kin * 1/(1.5*N_atoms*kB);
121  /*  $PV = NkT + virial$  */
122  pressure[i] = inv_volume * (1.5*E_kin + virial);
123
124  /* Equilibrate temperature by scaling momentum by a factor sqrt(alpha_T).
125     N.B. It is equally valid to scale the momentum instead of the velocity↔
126     since they only differ by a constant factor m.
127  */
128  alpha_T = 1 + 2*dt*(T_eq - temperature[i]) / (tau_T * temperature[i]);
129  scale_mat(N_atoms, 3, momentum, sqrt(alpha_T));
130
131  // Equilibrate pressure by scaling the posistions by a factor of alpha_P↔
132     ^{(1/3)}
133
134  alpha_P = 1 - kappa_Al* dt*(P_eq - pressure[i])/tau_P;
135  alpha_P_cube_root = pow(alpha_P, 1.0/3.0);
136  scale_mat(N_atoms, 3, pos, alpha_P_cube_root);
137
138  cell_length*=alpha_P_cube_root;
139  inv_volume*=1/alpha_P;
140
141  temperature[i]*=alpha_T;
142  pressure[i]*=alpha_P;
143  }
144
145  printf("equilibrium a0 = %.4f A\n", cell_length/N_cells);
146
147  /* Write tempertaure to file */
148  sprintf(file_name, "../data/temp-%d_pres-%d_Task3.tsv",
149          (int) T_final_C, (int) P_final_bar);
150  file_pointer = fopen(file_name, "w");
151  for (int i=0; i<N_timesteps; i++){
152      t = i*dt; // time at step i
153      fprintf(file_pointer, "%.4f \t %.8f \t %.8f \n",
154              t, temperature[i], pressure[i]);
155  }
156  fclose(file_pointer);
157
158  /* Write phase space coordinates to file */
159  sprintf(file_name, "../data/phase-space-temp-%d_pres-%d.tsv",
160          (int) T_final_C, (int) P_final_bar);
161  file_pointer = fopen(file_name, "w");
162  for (int i=0; i<N_atoms; i++){
163      for (int j=0; j<3; j++){
164          fprintf(file_pointer, " %.16e \t", pos[i][j]);
165      }
166      for (int j=0; j<3; j++){
167          fprintf(file_pointer, " %.16e \t", momentum[i][j]);
168      }
169      fprintf(file_pointer, "\n");
170  }
171  fclose(file_pointer);
172
173  /* save equilibrated position and momentum as a binary file */
174  sprintf(file_name, "../data/INIDATA_temp-%d_pres-%d.bin",
175          (int) T_final_C, (int) P_final_bar);
176  file_pointer = fopen(file_name, "wb");
177  fwrite(pos, sizeof(double), 3*N_atoms, file_pointer);
178  fwrite(momentum, sizeof(double), 3*N_atoms, file_pointer);
179  fwrite(&cell_length, sizeof(double), 1, file_pointer);
180  fclose(file_pointer);
181
182
183  /*
184  printf("T=%0.2f\tP=%0.2e\n",
185         temperature[N_timesteps-1], pressure[N_timesteps-1]);
186  */
187
188  free(pos); pos = NULL;
189  free(momentum); momentum = NULL;
190  free(forces); forces = NULL;
191  free(temperature); temperature = NULL;
192  free(pressure); pressure = NULL;
193  //free(volume); volume = NULL;
194  return 0;
195  }

```

A.4 Production runs for tasks 3-7 : main_Prod.c

```
1  /*
2  MD_main.c
3
4  Created by Anders Lindman on 2013-10-31.
5  */
6
7  #include <stdio.h>
8  #include <math.h>
9  #include <stdlib.h>
10 #include <time.h>
11
12 #include "initfcc.h"
13 #include "alpotential.h"
14 #include "funcs.h"
15
16 #define N_cells 4
17 /* define constants in atomic units: eV, , ps, K */
18 #define AMU 1.0364e-4
19 #define degC_to_K 273.15
20 #define bar 6.2415e-07
21 #define kB 8.61733e-5
22
23 /* Main program */
24 int main()
25 {
26     char file_name[100];
27
28     int N_atoms = 4*N_cells*N_cells*N_cells;
29     double m_Al = 27*AMU;
30     /*
31      Values of Young's and shear modulus, Y and G resp., taken from
32      Physics Handbook, table T 1.1. Bulk modulus then calculated as
33      B = Y*G / (9*G - 3*Y) [F 1.15, Physics Handbook]
34      kappa = 1/B
35      */
36     // double kappa_Al = 100/(6.6444e+05 * bar); // STRANGE FACTOR 100 OFF !!!
37     double cell_length = 0;
38     double inv_volume;
39
40     double T_eq_C = 500;
41     double P_eq_bar = 1;
42     // double T_eq = T_eq_C + degC_to_K;
43     // double P_eq = P_eq_bar*bar;
44     double dt = 5e-4; // higher res for spectral function
45     double t_end = 5;
46     // double tau_T = 100*dt;
47     // double tau_P = 100*dt;
48
49     int N_timesteps = t_end/dt;
50
51     int N_between_steps = 1;
52     int N_save_timesteps = N_timesteps / N_between_steps; //for the displacements
53     int N_save_atoms = 5;
54
55     // double alpha_T, alpha_P, alpha_P_cube_root;
56     double t, E_kin, virial;
57
58     double (*pos)[3] = malloc(sizeof(double[N_atoms][3]));
59     double (*pos_0)[3] = malloc(sizeof(double[N_atoms][3]));
60     double (*momentum)[3] = malloc(sizeof(double[N_atoms][3]));
61     double (*forces)[3] = malloc(sizeof(double[N_atoms][3]));
62     double (*displacements)[N_save_atoms] =
63         malloc(sizeof(double[N_save_timesteps][N_save_atoms]));
64     double (*pos_all)[N_atoms][3] =
65         malloc(sizeof(double[N_save_timesteps][N_atoms][3]));
66     double (*vel_all)[N_atoms][3] =
67         malloc(sizeof(double[N_save_timesteps][N_atoms][3]));
68     double *temperature = malloc(sizeof(double[N_timesteps]));
69     double *pressure = malloc(sizeof(double[N_timesteps]));
70     double *msd = malloc(sizeof(double[N_save_timesteps]));
71     double *vel_corr = malloc(sizeof(double[N_save_timesteps]));
72     double *pow_spec = malloc(sizeof(double[N_save_timesteps]));
73     double *freq = malloc(sizeof(double[N_save_timesteps]));
74
75     for (int i = 0; i < N_save_timesteps; i++){
76         msd[i] = 0;
77         pow_spec[i] = 0;
78         vel_corr[i] = 0;
79     }
80     FILE *file_pointer;
81
82     /* ----- TASK 3 ----- */
83
84     // read positions, momenta and cell_length
85     sprintf(file_name, "../data/INIDATA_temp-%d_pres-%d.bin",
86
```

```

87     (int) T_eq_C, (int) P_eq_bar);
88     file_pointer = fopen(file_name, "rb");
89     fread(pos, sizeof(double), 3*N_atoms, file_pointer);
90     fread(momentum, sizeof(double), 3*N_atoms, file_pointer);
91     fread(&cell_length, sizeof(double), 1, file_pointer);
92     fclose(file_pointer);
93
94     for (int i=0; i<N_atoms; i++){
95         for (int j=0; j<3; j++){
96             pos_0[i][j]=pos[i][j];
97         }
98     }
99     inv_volume = pow(N_cells*cell_length, -3);
100     get_forces_AL( forces, pos, cell_length, N_atoms); //initial cond forces
101
102     printf("Initialized. Starting with Verlet timestepping.\n");
103     for (int i=0; i<N_timesteps; i++){
104         /*
105          * The loop over the timesteps first takes a timestep according to the
106          * Verlet algorithm, then calculates the energies and temeperature.
107          */
108         timestep_Verlet(N_atoms, pos, momentum, forces, m_Al, dt, cell_length);
109
110         E_kin = get_kin_energy(N_atoms, momentum, m_Al );
111         virial = get_virial_AL(pos, cell_length, N_atoms);
112
113         /* PV = NkT + virial */
114         pressure[i] = inv_volume * (1.5*E_kin + virial);
115         /* 3N*kB*T/2 = 1/(2m) * \sum_{i=1}^N p_i^2 = p_sq/(2m) */
116         temperature[i] = E_kin * 1/(1.5*N_atoms*kB);
117
118         if (i % N_between_steps == 0){
119             int k = i/N_between_steps; // number of saved timesteps so far
120             get_displacements (N_save_atoms, pos, pos_0, displacements[k]);
121             copy_mat(N_atoms, 3, pos, pos_all[k]);
122
123             copy_mat(N_atoms, 3, momentum, vel_all[k]);
124             scale_mat(N_atoms, 3, vel_all[k], 1/m_Al);
125         }
126         if ((i*10) % N_timesteps == 0){
127             printf("done %d%% of Verlet timestepping\n", (i*10)/N_timesteps);
128         }
129     }
130     printf("calculating MSD\n");
131     get_MSD(N_atoms, N_save_timesteps, pos_all, msd);
132
133     printf("calculating velocity correlation\n");
134     get_vel_corr(N_atoms, N_save_timesteps, vel_all, vel_corr);
135
136     printf("calculating power spectrum\n");
137     get_powerspectrum(N_atoms, N_save_timesteps, vel_all, pow_spec);
138     fft_freq(freq, dt, N_save_timesteps);
139
140
141
142     printf("writing to file\n");
143     /* Write tempertaure to file */
144
145     sprintf(file_name, "../data/temp-%d_pres-%d_Prod-test.tsv",
146             (int) T_eq_C, (int) P_eq_bar);
147     file_pointer = fopen(file_name, "w");
148     for (int i=0; i<N_timesteps; i++){
149         t = i*dt; // time at step i
150         fprintf(file_pointer, "%.4f \t %.8f \t %.8f \n",
151                 t, temperature[i], pressure[i]);
152     }
153     fclose(file_pointer);
154
155     /* Write displacements to file */
156     sprintf(file_name, "../data/temp-%d_pres-%d_displacements.tsv",
157             (int) T_eq_C, (int) P_eq_bar);
158     file_pointer = fopen(file_name, "w");
159     for (int i=0; i<N_save_timesteps; i++){
160         t = i*dt*N_between_steps; // time at step i
161         fprintf(file_pointer, "%.4f", t);
162         for (int j=0; j<N_save_atoms; j++){
163             fprintf(file_pointer, "\t %.8f", displacements[i][j]);
164         }
165         fprintf(file_pointer, "\n");
166     }
167     fclose(file_pointer);
168
169     /* Write MSD to file */
170     sprintf(file_name, "../data/temp-%d_pres-%d_dynamicProperties.tsv",
171             (int) T_eq_C, (int) P_eq_bar);
172     file_pointer = fopen(file_name, "w");
173     // write header
174     fprintf(file_pointer, "%\n t[ps] \t MSD[A^2] \t vel_corr [A/ps]^2 \n");
175     for (int i=0; i<N_save_timesteps; i++){
176         t = i*dt*N_between_steps; // time at step i
177         fprintf(file_pointer, "%.4f \t %.8f \t %.8f \n", t, msd[i], vel_corr[i]);

```

```

178 }
179 fclose(file_pointer);
180
181 sprintf(file_name, "../data/temp-%d_pres-%d_power-spectrum.tsv",
182         (int) T_eq_C, (int) P_eq_bar);
183 file_pointer = fopen(file_name, "w");
184 // write header
185 fprintf(file_pointer, "%% f[1/ps] \t P[A/ps]^2 \n");
186 for (int i=0; i<N_save_timesteps/2; i++){ // only print from f=0 to f_crit
187     fprintf(file_pointer, "%.4f \t %.8f \n", freq[i], pow_spec[i]);
188 }
189 fclose(file_pointer);
190
191 free(pos);          pos = NULL;
192 free(pos_0);        pos_0 = NULL;
193 free(momentum);     momentum = NULL;
194 free(forces);        forces = NULL;
195 free(temperature);   temperature = NULL;
196 free(pressure);      pressure = NULL;
197 free(displacements); displacements = NULL;
198 free(pos_all); pos_all = NULL;
199 free(vel_all); vel_all = NULL;
200 free(msd); msd = NULL;
201 free(vel_corr); vel_corr = NULL;
202 free(pow_spec); pow_spec = NULL;
203 free(freq); freq = NULL;
204 return 0;
205 }

```

A.5 Production runs for tasks 3-7 : main_Prod.c

```

1  /*
2  MD_main.c
3
4  Created by Anders Lindman on 2013-10-31.
5  */
6
7  #include <stdio.h>
8  #include <math.h>
9  #include <stdlib.h>
10 #include <time.h>
11
12 #include "initfcc.h"
13 #include "alpotential.h"
14 #include "funcs.h"
15
16 #define N_cells 4
17 /* define constants in atomic units: eV, , ps, K */
18 #define AMU 1.0364e-4
19 #define degC_to_K 273.15
20 #define bar 6.2415e-07
21 #define kB 8.61733e-5
22
23 /* Main program */
24 int main()
25 {
26     char file_name[100];
27
28     int N_atoms = 4*N_cells*N_cells*N_cells;
29     double m_Al = 27*AMU;
30     /*
31      Values of Young's and shear modulus, Y and G resp., taken from
32      Physics Handbook, table T 1.1. Bulk modulus then calculated as
33      B = Y*G / (9*G - 3*Y) [F 1.15, Physics Handbook]
34      kappa = 1/B
35      */
36     // double kappa_Al = 100/(6.6444e+05 * bar); // STRANGE FACTOR 100 OFF !!!
37     double cell_length = 0;
38     double inv_volume;
39
40     double T_eq_C = 500;
41     double P_eq_bar = 1;
42     // double T_eq = T_eq_C + degC_to_K;
43     // double P_eq = P_eq_bar*bar;
44     double dt = 5e-4; // higher res for spectral function
45     double t_end = 5;
46     // double tau_T = 100*dt;
47     // double tau_P = 100*dt;
48
49     int N_timesteps = t_end/dt;
50
51     int N_between_steps = 1;
52     int N_save_timesteps = N_timesteps / N_between_steps; //for the displacements
53     int N_save_atoms = 5;

```

```

56 // double alpha_T, alpha_P, alpha_P_cube_root;
57 double t, E_kin, virial;
58
59 double (*pos)[3] = malloc(sizeof(double[N_atoms][3]));
60 double (*pos_0)[3] = malloc(sizeof(double[N_atoms][3]));
61 double (*momentum)[3] = malloc(sizeof(double[N_atoms][3]));
62 double (*forces)[3] = malloc(sizeof(double[N_atoms][3]));
63 double (*displacements)[N_save_atoms] =
64     malloc(sizeof(double[N_save_timesteps][N_save_atoms]));
65 double (*pos_all)[N_atoms][3] =
66     malloc(sizeof(double[N_save_timesteps][N_atoms][3]));
67 double (*vel_all)[N_atoms][3] =
68     malloc(sizeof(double[N_save_timesteps][N_atoms][3]));
69 double *temperature = malloc(sizeof(double[N_timesteps]));
70 double *pressure = malloc(sizeof(double[N_timesteps]));
71 double *msd = malloc(sizeof(double[N_save_timesteps]));
72 double *vel_corr = malloc(sizeof(double[N_save_timesteps]));
73 double *pow_spec = malloc(sizeof(double[N_save_timesteps]));
74 double *freq = malloc(sizeof(double[N_save_timesteps]));
75
76 for (int i = 0; i < N_save_timesteps; i++){
77     msd[i] = 0;
78     pow_spec[i] = 0;
79     vel_corr[i] = 0;
80 }
81 FILE *file_pointer;
82
83 /* ----- TASK 3 ----- */
84
85 // read positions, momenta and cell_length
86 sprintf(file_name, "../data/INIDATA-temp-%d-pres-%d.bin",
87     (int) T_eq_C, (int) P_eq_bar);
88 file_pointer = fopen(file_name, "rb");
89 fread(pos, sizeof(double), 3*N_atoms, file_pointer);
90 fread(momentum, sizeof(double), 3*N_atoms, file_pointer);
91 fread(&cell_length, sizeof(double), 1, file_pointer);
92 fclose(file_pointer);
93
94 for (int i=0; i<N_atoms; i++){
95     for (int j=0; j<3; j++){
96         pos_0[i][j]=pos[i][j];
97     }
98 }
99 inv_volume = pow(N_cells*cell_length, -3);
100 get_forces_AL( forces, pos, cell_length, N_atoms); //initial cond forces
101
102 printf("Initialized. Starting with Verlet timestepping.\n");
103 for (int i=0; i<N_timesteps; i++){
104     /*
105      * The loop over the timesteps first takes a timestep according to the
106      * Verlet algorithm, then calculates the energies and temeperature.
107      */
108     timestep_Verlet(N_atoms, pos, momentum, forces, m_AL, dt, cell_length);
109
110     E_kin = get_kin_energy(N_atoms, momentum, m_AL);
111     virial = get_virial_AL(pos, cell_length, N_atoms);
112
113     /* PV = NkT + virial */
114     pressure[i] = inv_volume * (1.5*E_kin + virial);
115     /* 3N*kB*T/2 = 1/(2m) * \sum_{i=1}^N p_i^2 = p_sq/(2m) */
116     temperature[i] = E_kin * 1/(1.5*N_atoms*kB);
117
118     if (i % N_between_steps == 0){
119         int k = i/N_between_steps; // number of saved timesteps so far
120         get_displacements(N_save_atoms, pos, pos_0, displacements[k]);
121         copy_mat(N_atoms, 3, pos, pos_all[k]);
122
123         copy_mat(N_atoms, 3, momentum, vel_all[k]);
124         scale_mat(N_atoms, 3, vel_all[k], 1/m_AL);
125     }
126     if ((i*10) % N_timesteps == 0){
127         printf("done %d%% of Verlet timestepping\n", (i*10)/N_timesteps);
128     }
129 }
130 printf("calculating MSD\n");
131 get_MSD(N_atoms, N_save_timesteps, pos_all, msd);
132
133 printf("calculating velocity correlation\n");
134 get_vel_corr(N_atoms, N_save_timesteps, vel_all, vel_corr);
135
136 printf("calculating power spectrum\n");
137 get_powerspectrum(N_atoms, N_save_timesteps, vel_all, pow_spec);
138 fft_freq(freq, dt, N_save_timesteps);
139
140
141
142 printf("writing to file\n");
143 /* Write tempertaure to file */
144
145 sprintf(file_name, "../data/temp-%d-pres-%d-Prod-test.tsv",
146     (int) T_eq_C, (int) P_eq_bar);

```

```

147 file_pointer = fopen(file_name, "w");
148 for (int i=0; i<N_timesteps; i++){
149     t = i*dt; // time at step i
150     fprintf(file_pointer, "%.4f \t %.8f \t %.8f \n",
151         t, temperature[i], pressure[i]);
152 }
153 fclose(file_pointer);
154
155 /* Write displacements to file */
156 sprintf(file_name, "../data/temp-%d_pres-%d_displacements.tsv",
157     (int) T_eq_C, (int) P_eq_bar);
158 file_pointer = fopen(file_name, "w");
159 for (int i=0; i<N_save_timesteps; i++){
160     t = i*dt*N_between_steps; // time at step i
161     fprintf(file_pointer, "%.4f", t);
162     for (int j=0; j<N_save_atoms; j++){
163         fprintf(file_pointer, "\t %.8f", displacements[i][j]);
164     }
165     fprintf(file_pointer, "\n");
166 }
167 fclose(file_pointer);
168
169 /* Write MSD to file */
170 sprintf(file_name, "../data/temp-%d_pres-%d_dynamicProperties.tsv",
171     (int) T_eq_C, (int) P_eq_bar);
172 file_pointer = fopen(file_name, "w");
173 // write header
174 fprintf(file_pointer, "%s t[ps] \t MSD[A^2] \t vel_corr [A/ps]^2 \n");
175 for (int i=0; i<N_save_timesteps; i++){
176     t = i*dt*N_between_steps; // time at step i
177     fprintf(file_pointer, "%.4f \t %.8f \t %.8f \n", t, msd[i], vel_corr[i]);
178 }
179 fclose(file_pointer);
180
181 sprintf(file_name, "../data/temp-%d_pres-%d_power-spectrum.tsv",
182     (int) T_eq_C, (int) P_eq_bar);
183 file_pointer = fopen(file_name, "w");
184 // write header
185 fprintf(file_pointer, "%s f[1/ps] \t P[A/ps]^2 \n");
186 for (int i=0; i<N_save_timesteps/2; i++){ // only print from f=0 to f_crit
187     fprintf(file_pointer, "%.4f \t %.8f \n", freq[i], pow_spec[i]);
188 }
189 fclose(file_pointer);
190
191 free(pos);          pos = NULL;
192 free(pos_0);        pos_0 = NULL;
193 free(momentum);     momentum = NULL;
194 free(forces);        forces = NULL;
195 free(temperature);  temperature = NULL;
196 free(pressure);     pressure = NULL;
197 free(displacements); displacements = NULL;
198 free(pos_all);      pos_all = NULL;
199 free(vel_all);      vel_all = NULL;
200 free(msd);          msd = NULL;
201 free(vel_corr);     vel_corr = NULL;
202 free(pow_spec);     pow_spec = NULL;
203 free(freq);         freq = NULL;
204 return 0;
205 }

```

A.6 Misc functions : funcs.c

```

1 #include "funcs.h"
2
3 void add_noise(int M, int N, double mat[M][N], double noise_amplitude )
4 {
5     const gsl_rng_type *T; /* static info about rngs */
6     gsl_rng *q; /* rng instance */
7     gsl_rng_env_setup (); /* setup the rngs */
8     T = gsl_rng_default; /* specify default rng */
9     q = gsl_rng_alloc(T); /* allocate default rng */
10    gsl_rng_set(q, time(NULL)); /* Initialize rng */
11
12    for (int i=0; i<N; i++){
13        for (int j=0; j<M; j++){
14            // adds uniformly distributed random noise in range +/- noise_amplitude
15            mat[i][j] += noise_amplitude * (2*gsl_rng_uniform(q)-1);
16        }
17    }
18    gsl_rng_free(q); /* deallocate rng */
19 }
20
21 void timestep_Verlet ( int N_atoms, double (*pos)[3], double (*momentum)[3],
22     double (*forces)[3], double m, double dt,
23     double cell_length){
24     for (int i = 0; i < N_atoms; i++) {

```

```

25     for (int j = 0; j < 3; j++) {
26         /* p(t+dt/2) */
27         momentum[i][j] += dt * 0.5 * forces[i][j];
28         /* q(t+dt) */
29         pos[i][j] += dt * momentum[i][j] / m;
30     }
31 }
32 /* F(t+dt) */
33 get_forces_AL( forces, pos, cell_length, N_atoms);
34 for (int i = 0; i < N_atoms; i++) {
35     for (int j = 0; j < 3; j++) {
36         /* p(t+dt/2) */
37         momentum[i][j] += dt * 0.5 * forces[i][j];
38     }
39 }
40 }
41
42 double get_kin_energy ( int N_atoms, double (*momentum)[3], double m ) {
43     double p_sq=0; // momentum squared
44     for (int i = 0; i < N_atoms; i++) {
45         for (int j = 0; j < 3; j++) {
46             p_sq += momentum[i][j] * momentum[i][j];
47         }
48     }
49     return p_sq / (2*m);
50 }
51
52 void get_displacements ( int N_atoms, double (*positions)[3],
53                         double (*initial_positions)[3], double disp[]) {
54     for (int i = 0; i < N_atoms; i++) {
55         for (int j = 0; j < 3; j++) {
56             disp[i] += (positions[i][j] - initial_positions[i][j])
57                     * (positions[i][j] - initial_positions[i][j]);
58         }
59         disp[i] = sqrt(disp[i]);
60     }
61 }
62
63 void get_MSD ( int N_atoms, int N_times, double all_pos[N_times][N_atoms][3],
64              double MSD[N_times]) {
65     /* all_pos = positions of all particles at all (saved) times */
66     /* outer time index it starts at outer it = 1, since MSD[0] = 0 */
67     for (int it = 1; it < N_times; it++) { //
68         for (int jt = 0; jt < N_times-it; jt++) { // summed time index
69             for (int kn = 0; kn < N_atoms; kn++) { // particle index
70                 for (int kd = 0; kd < 3; kd++) { // three dimensions
71                     MSD[it] += (all_pos[it+jt][kn][kd] - all_pos[jt][kn][kd])
72                             * (all_pos[it+jt][kn][kd] - all_pos[jt][kn][kd]);
73                 }
74             }
75         }
76         MSD[it] *= 1/( (double)N_atoms * (N_times-it));
77     }
78 }
79
80 void get_vel_corr ( int N_atoms, int N_times, double all_vel[N_times][N_atoms][3],
81                   double vel_corr[N_times]) {
82     /* all_vel = velocity of all particles at all (saved) times */
83     for (int it = 0; it < N_times; it++) { //
84         for (int jt = 0; jt < N_times-it; jt++) { // summed time index
85             for (int kn = 0; kn < N_atoms; kn++) { // particle index
86                 for (int kd = 0; kd < 3; kd++) { // three dimensions
87                     vel_corr[it] += (all_vel[it+jt][kn][kd] * all_vel[jt][kn][kd]);
88                 }
89             }
90         }
91         vel_corr[it] *= 1/( (double)N_atoms * (N_times-it));
92     }
93 }
94
95 void get_powerspectrum ( int N_atoms, int N_times, double all_vel[N_times][N_atoms][3],
96                        double pow_spec[N_times]) {
97     /* all_vel = velocity of all particles at all (saved) times */
98     double vel_component[N_times]; // "all_vel[:,i][j]"
99     double pow_spec_component[N_times];
100     double normalization_factor = 1/( (double)N_atoms * (N_times));
101     for (int kn = 0; kn < N_atoms; kn++) { // particle index
102         for (int kd = 0; kd < 3; kd++) { // three dimensions
103             for (int it = 0; it < N_times; it++) { //
104                 vel_component[it] = all_vel[it][kn][kd];
105             }
106             powerspectrum(vel_component, pow_spec_component, N_times);
107             for (int iw = 0; iw < N_times; iw++) { // for all frequencies
108                 pow_spec[iw] += pow_spec_component[iw];
109             }
110         }
111     }
112 }
113 for (int iw = 0; iw < N_times; iw++) { // for all frequencies

```

```

114     pow_spec[iw] *= normalization_factor;
115 }
116 }
117
118
119
120 void copy_mat (int M, int N, double mat_from[M][N], double mat_to[M][N]){
121     /* Copies matrix `mat_from` to `mat_to` */
122     for (int i = 0; i < M; i++) {
123         for (int j = 0; j < N; j++) {
124             mat_to[i][j] = mat_from[i][j];
125         }
126     }
127 }
128
129 void set_zero (int M, int N, double mat[M][N]){
130     /* Sets the matrix `mat` to zero */
131     for (int i = 0; i < M; i++) {
132         for (int j = 0; j < N; j++) {
133             mat[i][j] = 0;
134         }
135     }
136 }
137
138 void scale_mat (int M, int N, double mat[M][N], double alpha){
139     /* Scales the matrix `mat` by factor `alpha` */
140     for (int i = 0; i < M; i++) {
141         for (int j = 0; j < N; j++) {
142             mat[i][j] *= alpha;
143         }
144     }
145 }

```

B Auxiliary

B.1 Makefile

```

1 CC = gcc
2 CFLAGS = -O3 -Wall -Wno-unused-result
3
4
5 LIBS = -lm -lgsl -lgslcblas
6
7 HEADERS = initfcc.h alpotential.h funcs.h fft_func.h
8 OBJECTS = initfcc.o alpotential.o funcs.o fft_func.o
9
10
11 %.o: %.c $(HEADERS)
12     $(CC) -c -o $@ $< $(CFLAGS)
13
14 all: Task1 Task2 Task3 main_Prod.c
15
16 Task1: $(OBJECTS) main_T1.c
17     $(CC) -o $@ $^ $(CFLAGS) $(LIBS)
18
19 Task2: $(OBJECTS) main_T2.c
20     $(CC) -o $@ $^ $(CFLAGS) $(LIBS)
21
22 Task3: $(OBJECTS) main_T3.c
23     $(CC) -o $@ $^ $(CFLAGS) $(LIBS)
24
25 Prod: $(OBJECTS) main_Prod.c
26     $(CC) -o $@ $^ $(CFLAGS) $(LIBS)
27
28 # $(PROGRAMS): $(OBJECTS) main_T1.c
29 #     $(CC) -o $@ $^ $(CFLAGS) $(LIBS)
30
31 clean:
32     rm -f *.o
33     touch *.c

```

C Matlab scripts

C.1 Analysis scripts for tasks 3-7: A1_energies.m

```

1 tmp = matlab.desktop.editor.getActive; %% cd to current path
2 cd(fileparts(tmp.Filename));
3 set(0, 'DefaultFigureWindowStyle', 'docked');

```



```

4 GRAY = 0.7*[0.9 0.9 1];
5 warning('off','MATLAB:handle_graphics:exceptions:SceneNode'); % interpreter ↩
6 warning
7 %% task 1: lattice energies
8 clc
9 AMU = 1.0364e-4;
10 m_Al = 27*AMU;
11
12 energy_data = load('./data/lattice_energies.tsv');
13 a0 = energy_data(:,1);
14 v0 = a0.^3;
15
16 energy = energy_data(:,2);
17 figure(1);clf;
18 plot(v0,energy, 'xk');
19
20 start_v = 64;
21 end_v = 68;
22 indToInclude = (v0 > start_v) & (v0 < end_v);
23 p = polyfit(v0(indToInclude),energy(indToInclude),2);
24 hold on;
25
26 vvec = linspace(start_v, end_v);
27 plot(vvec, p(1)*vvec.^2 + p(2)*vvec + p(3), '-r');
28 xlim([64 68]);
29
30 v_min = -p(2)/(2*p(1));
31 a_min = v_min^(1/3);
32 omega_res = sqrt(2*p(1)*a_min^4/m_Al);
33 f_res = omega_res/(2*pi);
34
35
36
37 ax = gca;
38 ax.YLim = [-13.45 -13.42];
39 h1 = plot(v_min*[1 1], ax.YLim, '--k'); % plot vertical line at v_min
40
41
42 ax.YTick = (-13.45:0.01:-13.42);
43 ylabel('$E_{\rm pot}$ [eV/unit cell]');
44 xlabel('$a_0^3$ [\AA^3]');
45 legend('data', 'quadratic fit', ['$V_{\rm eq}$ \approx \, $' num2str(round(v_min↩
46 ,2)) '\, \AA^3$'], ...
47 'location', 'southeast')
48 ax = gca; ax.Children = ax.Children(3:-1:1);
49 ImproveFigureCompPhys(gcf); h1.LineWidth = 2; setFigureSize(gcf, 300, 600);
50 %axis([63 68 ylim(1) 0]);
51 saveas(gcf, '../figures/potential_energy.eps', 'eps')
52
53 %% task 2: find a suitable timestep
54 clc;clf;
55
56 dt=[1e-2,5e-3,2e-3,1e-3];
57 figure(1);clf;figure(2);clf;
58 for i=1:4
59     T_data = load(sprintf('./data/temperature_dt-%0.0e_Task2.tsv',dt(i)));
60     E_data = load(sprintf('./data/total_energy_dt-%0.0e_Task2.tsv',dt(i)));
61     t = T_data(:,1);
62     T = T_data(:,2);
63     E = E_data(:,2);
64
65     t_eq=0.5;
66
67     fprintf('dt = %0.0e\n',dt(i));
68
69     T_avg=mean(T(t>t_eq));
70     T_std=std(T(t>t_eq));
71     fprintf('\tT = %0.2f +- %0.1f %%\n', T_avg, abs(T_std/T_avg)*100);
72
73     E_avg=mean(E(t>t_eq));
74     E_std=std(E(t>t_eq));
75     fprintf('\tE = %0.2f +- %0.1e %%\n', E_avg, abs(E_std/E_avg)*100);
76
77     figure(1)
78     plot(t, T); hold on;
79
80     %yyaxis right
81     figure(2)
82     plot(t, E);hold on;
83     ylim(E_avg*(1+0.001*[1,-1]));
84 end
85 for ifig = 1:2
86     figure(ifig);
87     h = legend(strcat({'dt = $ '}, num2str(round(dt',4)) , ' ps'));
88     xlabel('$t$ [ps]');
89     if ifig ==1
90         ylabel('$T$ [K]')
91     else
92         ylabel('$E_{\rm tot}$ [eV/unit cell]');

```

```

93     ax = gca; ax.YTick = (-13:0.1:-10);
94     ax.YLim = [-12.6 -12.2];
95     %h.Location = 'best';
96 end
97 ImproveFigureCompPhys(gcf,'Linewidth', 2);setFigureSize(gcf, 400, 400);
98 end
99 saveas(1, '../figures/dt-scan-temperature.eps', 'epsc')
100 saveas(2, '../figures/dt-scan-energy.eps', 'epsc')
101
102 %% task 3: temperature and pressure equilibration,
103 % and task4: test production pressure and temperature
104
105 clc; clf;
106 temps = [500 700 500 700];
107 temperatures_str = num2str([500;700]);
108 FILENAMES = [strcat({'../data/temp-'}, temperatures_str, '_pres-1_Task3.tsv');
109             strcat({'../data/temp-'}, temperatures_str, '_pres-1_Prod-test.tsv')];
110 bar = 6.2415e-07;
111 Kelvin_to_degC = -273.15;
112 t_eqs = [1 1 0.5 0.5]; % approximate equilibration time
113 N_average_points = 50;
114 dt = 5e-3;
115 tau_eqilibration = 100*dt;
116
117 for iFile = 1:numel(FILENAMES)
118     figure(iFile);clf;
119     data = load(FILENAMES{iFile});
120
121     t = data(:,1);
122     T = data(:,2)+Kelvin_to_degC;
123     P = data(:,3)/bar;
124
125     t_eq=t_eqs(iFile);
126
127     %fprintf('dt = %0.0e\n',dt(i));
128     T_avg=mean(T(t>t_eq));
129     T_std=std(T(t>t_eq));
130     fprintf('\tT = %0.2f +- %0.1f K\n', T_avg, abs(T_std));
131
132     P_avg=mean(P(t>t_eq));
133     P_std=std(P(t>t_eq));
134     fprintf('\tP = %0.2f +- %0.1f bar\n', P_avg, abs(P_std));
135
136     yyaxis left
137
138     if iFile <=2 % equilibration run, otherwise production
139         plot(t./tau_eqilibration,T, 'color', GRAY),hold on;
140         plot(t./tau_eqilibration, movmean(T,N_average_points),'-k')
141     else
142         plot(t,T, 'color', GRAY),hold on;
143         plot(t, cumsum(T)./(1:length(t)),'-k')
144     end
145     ylabel('$T \backslash, [\^{\circ} \rm C]$')
146
147
148     if iFile <=2 % equilibration run, otherwise production
149         ylim(temps(iFile)*(1+ 0.3*[-1,1.2]))
150         yyaxis right
151         plot(t./tau_eqilibration,P),hold on;
152         plot(t./tau_eqilibration, movmean(P,N_average_points),'-k')
153         legend('$\mathcal{T}$', 'mov avg', '$\mathcal{P}$', 'mov avg');
154         xlabel('$t \backslash \tau_{\rm eq}$')
155         xlim([0 5])
156     else
157         ylim(temps(iFile)+ 100*[-3,3])
158         yyaxis right
159         plot(t,P),hold on;
160         plot(t, cumsum(P)./(1:length(t)),'-k')
161         legend('$\mathcal{T}$', 'cum avg', '$\mathcal{P}$', 'cum avg');
162         xlabel('$t \backslash, [\rm ps]$')
163     end
164     ylabel('$P \backslash, [\rm bar]$')
165     ylim([-100,400])
166     ImproveFigureCompPhys(gcf, 'linewidth', 3, 'LineColor', {'MYORANGE', GRAY, '←
167         MYBLUE', GRAY});
168     setFigureSize(gcf, 400, 400);
169 end
170
171 saveas(1, '../figures/TP-eq-500.eps', 'epsc')
172 saveas(2, '../figures/TP-eq-700.eps', 'epsc')
173 saveas(3, '../figures/TP-prod-500.eps', 'epsc')
174 saveas(4, '../figures/TP-prod-700.eps', 'epsc')
175
176 %% determine displacements and MSD
177 temperatures_str = num2str([500;700]);
178 clc; clf;
179 figure(10); clf;
180 FILENAMES = strcat({'../data/temp-'}, temperatures_str, '_pres-1_displacements.←
    tsv');

```

```

181 FILENAMES_Dyn = strcat({'../data/temp-'}, temperatures_str, '_pres-1←
    _dynamicProperties.tsv');
182 FILENAMES_Pow = strcat({'../data/temp-'}, temperatures_str, '_pres-1←
    spectrum.tsv');
183 for iFile = 1:numel(FILENAMES)
184
185     figure(iFile); clf;
186     data = load(FILENAMES{iFile});
187     t = data(:,1);
188     dx = data(:,2:end);
189
190
191
192     data = load(FILENAMES_Dyn{iFile});
193     MSD = data(:,2);
194     vel_corr = data(:,3);
195     plot(t, MSD, 'k'); hold on;
196
197     if iFile == 2 % liquid
198         tStart = 1;
199         D = MSD(t>tStart)./(6*t(t>tStart));
200         selfDiffusionCoeff = mean(D); % in  $\text{\AA}^2/\text{ps}$ 
201         plot(t, 6*t*selfDiffusionCoeff, 'r');
202     end
203
204     plot(t, dx.^2, 'color', GRAY); hold on;
205
206     xlabel('$t$ [ps]')
207     ylabel('$\Delta x^2$, [\rm \AA^2]$')
208     if iFile == 1
209         ylim([0 1.0]);
210         leg = legend('$\Delta_{\rm MSD}$', 'individual trajectories');
211     else
212         ylim([0 20]);
213         leg = legend('$\Delta_{\rm MSD}$', '$6 t D_s$', 'individual trajectories←
    ');
214     end
215
216     leg.Location='northwest';
217     ImproveFigureCompPhys(gcf, 'Linewidth', 2);
218     ax = gca; [ax.Children(6:end).LineWidth] = deal(5);
219     ax.Children = ax.Children([6:end 1:5]);
220
221     setFigureSize(gcf, 400, 400);
222
223     figure(10)
224     plot(t, vel_corr/vel_corr(1), 'color', GRAY); hold on;
225     xlim([0 0.8])
226
227 end
228
229 % % velocity correlation
230 figure(10);clf; figure(11);clf;
231 n_average_points = 1;%30;
232 for iFile = 1:numel(FILENAMES)
233     data = load(FILENAMES_Dyn{iFile});
234     t = data(:,1);
235     vel_corr = data(:,3);
236
237     data = load(FILENAMES_Pow{iFile});
238     freq = data(:,1);
239     pow_spec = data(:,2);
240
241     figure(10);
242     plot(t, vel_corr/vel_corr(1)); hold on;
243
244     dt = t(2)-t(1);
245     N_times = round(length(t)/2); % we have too bad statistics at later times.
246     deltaf = 1/(N_times * dt);
247     freqvec = 0:deltaf:(1/(2*dt));
248     PhiHat = 2 * trapz(t(1:N_times), (vel_corr(1:N_times) * ones(size(freqvec)))←
    .* cos(2*pi*t(1:N_times) * freqvec ), 1); %dimension 1
249     %PhiHat = 1/2 * 1/N_times * 2 * sum( (vel_corr(1:N_times) * ones(size(←
    freqvec))) .* cos(2*pi*t(1:N_times) * freqvec ), 1); %dimension 1
250
251     figure(11);
252
253     plot(freqvec, PhiHat); hold on;
254     plot(freq, pow_spec*t(end), ':'); hold on;
255     if iFile == 2 % liquid
256         tStart = 1;
257         selfDiffusionCoeff_spectral = PhiHat(1)/6; % in  $\text{\AA}^2/\text{ps}$ 
258     end
259
260 end
261
262 disp([selfDiffusionCoeff selfDiffusionCoeff_spectral]);
263
264 figure(10)
265 xlim([0 1])
266 leg = legend(strcat({'$T=$'}, num2str([500;700]), '\, "\circ $C$'));

```

```

267 leg.Location='northeast';
268 xlabel('$t$ [ps]')
269 ylabel('$\Phi(t)/\Phi(0)$')
270 ImproveFigureCompPhys(gcf);
271 setFigureSize(gcf, 400, 400);
272
273 figure(11)
274 leg = legend('$T= 500 \, , \, \circlearrowleft $C, \, \hat{\Phi}$', '$T= 500 \, , \, \circlearrowleft $C, \, \hat{\Phi}$',...
275             '$T= 700 \, , \, \circlearrowleft $C, \, \hat{\Phi}$', '$T= 700 \, , \, \circlearrowleft $C, \, \hat{\Phi}$');
276 xlim([0 30])
277 ylim([0 Inf])
278 xlabel('$f$ [ps$^{-1}$]')
279 ylabel('$\hat{\Phi}$ [AA$^2$/ps] ')
280 setFigureSize(gcf, 400, 400);
281
282 ImproveFigureCompPhys(gcf, 'LineColor', {'r', 'MYRED', 'GERIBLUE', 'MYLIGHTBLUE'←
283             });
284
285
286
287 saveas(1, '../figures/MSD-500.eps', 'eps')
288 saveas(2, '../figures/MSD-700.eps', 'eps')
289 saveas(10, '../figures/Phi-t.eps', 'eps')
290 saveas(11, '../figures/P-freq.eps', 'eps')

```

C.2 Improve figure appearance: ImproveFigureCompPhys.m

```

1 function ImproveFigureCompPhys(varargin)
2 %ImproveFigureCompPhys Improves the figures of supplied handles
3 % Input:
4 % - none (improve all figures) or handles to figures to improve
5 % - optional:
6 %     LineWidth int
7 %     LineStyle column vector cell, e.g. {'-', '--'}',
8 %     LineColor column vector cell, e.g. {'k', [0 1 1], 'MYBLUE'}'
9 %     colors: MYBLUE, MYORANGE, MYGREEN, MYPURPLE, MYYELLOW,
10 %     MYLIGHTBLUE, MYRED
11 %     Marker column vector cell, e.g. {'.', 'o', 'x'}'
12
13 % ImproveFigure was originally written by Adam Stahl, but has been heavily
14 % modified by Linnea Hesslow
15
16
17 %% Handle inputs
18 % If no inputs or if the first argument is a string (a property rather than
19 % a handle), use all open figures
20 if nargin == 0 || ischar(varargin{1})
21     %Get all open figures
22     figHs = findobj('Type', 'figure');
23     nFigs = length(figHs);
24 else
25     % Check the supplied figure handles
26     figHs = varargin{1};
27     figHs = figHs(ishandle(figHs) == 1); %Keep only those handles that are ←
28     % proper graphics handles
29     nFigs = length(figHs);
30 end
31
32 % Define desired properties
33 titleSize = 24;
34 interpreter = 'latex';
35 lineWidth = 4;
36 axesWidth = 1.5;
37 labelSize = 22;
38 textSize = 20;
39 legTextSize = 18;
40 tickLabelSize = 18;
41 LineColor = {};
42 LineStyle = {};
43 Marker = {};
44
45 % define colors
46 co = [ 0 0.4470 0.7410
47       0.8500 0.3250 0.0980
48       0.9290 0.6940 0.1250
49       0.4940 0.1840 0.5560
50       0.4660 0.6740 0.1880
51       0.3010 0.7450 0.9330
52       0.6350 0.0780 0.1840 ];
53 colors = struct('MYBLUE', co(1,:), ...
54               'MYORANGE', co(2,:), ...
55               'MYYELLOW', co(3,:), ...
56               'MYPURPLE', co(4,:), ...
57               'MYGREEN', co(5,:), ...

```

```

57 'MYLIGHTBLUE', co(6,:),...
58 'MYRED',co(7,:),...
59 'GERIBLUE', [0.3000 0.1500 0.7500],...
60 'GERIRED', [1.0000 0.2500 0.1500],...
61 'GERIYELLOW', [0.9000 0.7500 0.1000],...
62 'LIGHTGREEN', [0.4 0.85 0.4],...
63 'LINNEAGREEN', [7 184 4]/255);
64
65 % Loop through the supplied arguments and check for properties to set.
66 for i = 1:nargin
67     if ischar(varargin{i})
68         switch lower(varargin{i}) %Compare lower case strings
69             case 'linewidth'
70                 lineWidth = varargin{i+1};
71             case 'linestyle'
72                 LineStyle = varargin{i+1};
73             case 'linecolor'
74                 LineColor = varargin{i+1};
75                 for iLineColor = 1:numel(LineColor)
76                     if isfield(colors, LineColor{iLineColor})
77                         LineColor{iLineColor} = colors.(LineColor{iLineColor});
78                     end
79                 end
80             case 'marker'
81                 Marker = varargin{i+1};
82         end
83     end
84 end
85 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
86
87 %%% Improve the figure(s)
88
89 for iFig = 1:nFigs
90
91     fig = figHs(iFig);
92
93     lineObjects = findall(fig, 'Type', 'line');
94     textObjects = findall(fig, 'Type', 'text');
95     axesObjects = findall(fig, 'Type', 'axes');
96     legObjects = findall(fig, 'Type', 'legend');
97     contourObjects = findall(fig, 'Type', 'contour'); % not counted as lines
98
99     %%% TEXT APPEARANCE: first set all to textSize and then change the ones
100    %%% that need to be changed again
101
102    %Change size of any text objects in the plot
103    set(textObjects, 'FontSize', textSize);
104    set(legObjects, 'FontSize', legTextSize);
105
106    %%% FIX LINESTYLE, COLOR ETC. FOR EACH PLOT SEPARATELY
107    for iAx = 1:numel(axesObjects)
108        lineObjInAx = findall(axesObjects(iAx), 'Type', 'line');
109
110        %set line style and color style (only works if all figs have some
111        %number of line plots..)
112        if ~isempty(LineStyle)
113            set(lineObjInAx, {'LineStyle'}, LineStyle)
114            set(contourObjects, {'LineStyle'}, LineStyle); %%%%%%
115        end
116        if ~isempty(LineColor)
117            set(lineObjInAx, {'Color'}, LineColor)
118            set(contourObjects, {'LineColor'}, LineColor); %%%%%%
119        end
120        if ~isempty(Marker)
121            set(lineObjInAx, {'Marker'}, Marker)
122            set(lineObjInAx, {'Markersize'}, num2cell(10+22*strcmp(Marker, '.'))←
123                )
124        end
125
126        %%% change font sizes.
127        % Tick label size
128        xLim = axesObjects(iAx).XLim;
129        axesObjects(iAx).FontSize = tickLabelSize;
130        axesObjects(iAx).XLim = xLim;
131        %Change label size
132        axesObjects(iAx).XLabel.FontSize = labelSize;
133        axesObjects(iAx).YLabel.FontSize = labelSize;
134
135        %Change title size
136        axesObjects(iAx).Title.FontSize = titleSize;
137    end
138
139    %%% LINE APPEARANCE
140    %Change line thicknesses
141    set(lineObjects, 'LineWidth', lineWidth);
142    set(contourObjects, 'LineWidth', lineWidth);
143    set(axesObjects, 'LineWidth', axesWidth)
144
145    % set interpreter: latex or tex
146    set(textObjects, 'interpreter', interpreter)
147    set(legObjects, 'Interpreter', interpreter)

```

```
147     set(axesObjects, 'TickLabelInterpreter', interpreter);
148 end
149 end
```

C.3 Change size of figures: setFigureSize.m

```
1 function [ fig ] = setFigureSize( fig, H, W )
2 fig.Units = 'points';
3 fig.WindowStyle = 'normal'; % undock
4 fig.Position(3:4) = [W H];
5 end
```