# 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 22, 2018

| Task Nº ॒ | Points | Avail. points |
|-----------|--------|---------------|
|           |        |               |
|           |        |               |
|           |        |               |
|           |        |               |
|           |        |               |
|           |        |               |
|           |        |               |
|           |        |               |
| Σ         |        |               |

#### Introduction

Already in antiquity people studied the effect of particles impinging on other particles. Since then the art has developed ... (*If you like to do so, you may take the opportunity to put the methods in a wider perspective here.*) Here is a random reference.[1]

#### Task 1

We determined the theoretical lattice parameter ....

Figure 2 shows the potential energy as a function of the lattice parameter. We used a quadratic fit to find the minimum energy, and obtained  $V_{\rm eq} \approx 65.38\,{\rm Å}^3$ . This corresponds to the equilibrium lattice parameter  $a_{\rm eq} \approx 4.029\,{\rm Å}$  at 0 K, which we took as the initial lattice parameter for the following tasks.

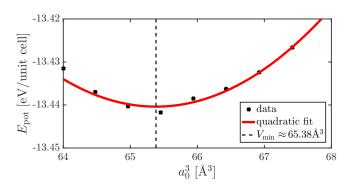


Figure 1: The potential energy per unit cell for aluminum as a function of the lattice parameter cubed.

We find that figure 2 looks similar to the figure 1 in the homework problem file.

#### Task 5

Equation (82) in MD lecture notes:

$$\Delta_{\rm MSD}(t) = \lim_{T \to \infty} \frac{1}{T} \int_0^T dt' \frac{1}{N_{\rm atoms}} \sum_{i=0}^{N_{\rm atoms}-1} \left[ \mathbf{r}_i(t+t') - \mathbf{r}_i(t') \right]^2 \tag{1}$$

\_

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

To determine M, we used mean of ... for t ¿ ...

#### Task 7

The average "power" content in avariable, X(t'), at some time, t, during some range of time, T, can be defined as

$$P_X(t,T) = \frac{1}{T} \int_{t-T/2}^{t+T/2} dt' \ X^2(t'). \tag{3}$$

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

$$P_X = \lim_{T \to \infty} P_X(T) = \lim_{T \to \infty} \frac{1}{T} \left\langle \int_0^T dt' \ X^2(t') \right\rangle. \tag{4}$$

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

$$\hat{f}(\omega) = \int_{-\infty}^{\infty} dt \, f(t) e^{i\omega t},\tag{5}$$

Using these two functions, we can define a power spectrum

$$\hat{P}(\omega) = \left\langle |\hat{\mathbf{v}}(\omega)|^2 \right\rangle_{\mathbf{A}} = \left\langle \hat{\mathbf{v}}(\omega) \cdot \bar{\hat{\mathbf{v}}}(\omega) \right\rangle_{\mathbf{A}}$$

$$= \left\langle \int_{-\infty}^{\infty} dt \ \mathbf{v}(t) e^{i\omega t} \int_{-\infty}^{\infty} dt' \ \mathbf{v}(t') e^{-i\omega t'} \right\rangle_{\mathbf{A}},$$
(6)

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

$$\hat{P}(\omega) = \int_{-\infty}^{\infty} d\tau \, e^{i\omega\tau} \int_{-\infty}^{\infty} dt' \, \langle v(t'+\tau) \cdot v(t') \rangle_{A}, \qquad (7)$$

## **Problem 1**

As a starting point we first look at scattering from a hard-sphere potential. We also consider the Lennard–Jones potential, which is depicted in Figure ??. (Always refer to Figures in the text.)

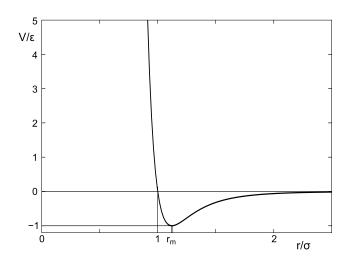


Figure 2: The Lennard–Jones potential. Make sure you label and have units on all axes! Also make sure that labels etc. are legible and that, if you print in black and white, that you use different line styles when required to differentiate between curves. In MATLAB you can export any figure to an .eps file from File  $\rightarrow$  Export... in the Figure window.

#### **Problem 2**

In the following we give an example of how to produce a table. Use the code for Table 1 as a template.

Table 1: A dummy table

| Col. 1 | Col. 2 | Col. 3 |
|--------|--------|--------|
| the    | quick  | brown  |
| fox    | jumps  | over   |
| the    | lazy   | dog    |

### **Problem 3**

If you find some part of the code particularly interesting you may include it in the text, otherwise it should be included in the appedix. If you do want to include code the following commands will print the text directly, with no LATEX commands executed:

```
% Hello world ten times in MATLAB
for i = 1 : 10
    fprintf('Hello world %d!\n',i);
end
```

```
# Hello world ten times in Python
for i in range(10):
   print 'Hello world %d!' % i
```

#### **Problem 4**

At some point it may be appropriate to include equations. It is done in the following way:

$$V(r) = 4\epsilon \left[ \left( \frac{\sigma}{r} \right)^{12} - \left( \frac{\sigma}{r} \right)^{6} \right] \tag{8}$$

Do number and reference all your equations.

# **Concluding discussion**

Use your favourite flavor of LATEX to compile the file:

```
xelatex template.tex
pdflatex template.tex
latex template.tex
```

should all work. If you use pdflatex or xelatex, included figures need to be in pdf, jpg, or png format. If you want to include eps figures, you can easily convert them to pdf using the command

ps2pdf -dEPSCrop figure.eps figure.pdf

# References

[1] Leslie Lamport, ETeX: A Document Preparation System. Addison Wesley, Massachusetts, 2nd Edition, 1994.

#### 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

```
MD_main.c
3
4
5
     Created by Anders Lindman on 2013-10-31.
6
     #include <stdio.h>
     #include <math.h>
    #include <stdlib.h>
10
    #include "initfcc.h"
#include "alpotential.h"
11
13
     #define N_cells 4
15
    #define N_lattice_params 25
16
17
     /* Main program */
18
    int main()
20
21
22
       int N_atoms = 4*N_cells*N_cells*N_cells;
23
       double a0;
double a0_min = 4.0;
24
       double a0_max = 4.2;
       double da0 = (a0_max - a0_min)/N_lattice_params;
27
28
29
       double (*pos)[3] = malloc(sizeof(double[N_atoms][3]));
double *energy = malloc(sizeof(double[N_lattice_params]));
30
31
33
       FILE *file_pointer;
34
35
       /* -----*/
36
37
       for (int i=0; i<N_lattice_params; i++){</pre>
39
         a0 = a0_min + i*da0;
         init_fcc(pos, N_cells, a0);
// energy per unit cell
40
41
42
         energy[i] = get_energy_AL(pos, N_cells*a0, N_atoms)*4/N_atoms;
43
45
       file_pointer = fopen("../data/lattice_energies.tsv", "w");
       for (int i=0; i<N_lattice_params; i++){
    a0 = a0_min + i*da0;</pre>
46
47
         fprintf(file_pointer, "%.8f \t %.8f \n", a0, energy[i]);
48
       fclose(file_pointer);
52
53
54
55
       free(pos); pos = NULL;
       free(energy); energy = NULL;
       return 0;
```

#### 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>
```

```
#include "initfcc.h"
#include "alpotential.h"
#include "funcs.h"
 13
14
15
               #define N_cells 4
 16
               #define AMU 1.0364e-4
 17
               #define kB 8.6173303e-5
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;
                      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]));
                      double *temperature = malloc(sizeof(double[N_timesteps]));
                     double *E_tot = malloc(sizeof(double[N_timesteps]));
39
40
41
                     FILE *file pointer:
43
44
                       /* ----- TASK 2 ------
45
                     init_fcc(pos, N_cells, a_eq); // initialize fcc lattice
46
                     add_noise( N_atoms, 3, pos, noise_amplitude ); // adds random noise to pos set_zero( N_atoms, 3, momentum); // set momentum to 0
47
48
                     {\tt get\_forces\_AL(\ forces,\ pos,\ a\_eq^*N\_cells,\ N\_atoms);\ //initial\ cond\ forces}
50
51
                       for (int i=0; i<N_timesteps; i++){</pre>
52
                                      The loop over the timesteps first takes a timestep according to the % \left( 1\right) =\left( 1\right) \left( 1\right) +\left( 1\right) \left( 1\right) \left( 1\right) +\left( 1\right) \left( 1
53
                                      Verlet algorithm, then calculates the energies and temeperature.
54
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 );
                            \label{eq:loss_energy_AL} \texttt{E\_tot[i]} \ = \texttt{E\_kin} \ + \ \texttt{get\_energy\_AL(pos, a\_eq*N\_cells, N\_atoms);}
59
60
                            /* 3N*kB*T/2 = 1/(2m) * \sum_{i=1}^{N} p_i^2 = p_sq/(2m) */temperature[i] = E_kin * 2/(3*N_atoms*kB);
61
63
64
65
                       /* Write tempertaure to file */
                     char file_name[100];
66
                      sprintf(file_name,"../data/temperature_dt-%0.0e_Task2.tsv", dt);
67
                       file_pointer = fopen(file_name, "w");
68
                     for (int i=0; i<N_timesteps; i++){
  t = i*dt; // time at step i
  fprintf(file_pointer, "%.4f \t %.8f \n", t, temperature[i]);
 70
 71
72
 73
                     fclose(file_pointer);
 74
                     /* Write total energy to file */
sprintf(file_name,"../data/total_energy_dt-%0.0e_Task2.tsv", dt);
file_pointer = fopen(file_name, "w");
 75
76
77
                      for (int i=0; i<N_timesteps; i++){
  t = i*dt; // time at step i</pre>
 78
 79
                            fprintf(file_pointer, "%.4f \t %.8f \n", t, E_tot[i]);
80
82
                     fclose(file_pointer);
83
84
                      free(pos); pos = NULL;
                      free(momentum); momentum = NULL;
85
                      free(forces); forces = NULL;
86
                      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    Created by Anders Lindman on 2013-10-31.
```

```
#include <stdio.h>
     #include <math.h>
#include <stdlib.h>
     #include <time.h>
10
     #include "initfcc.h"
#include "alpotential.h"
#include "funcs.h"
13
14
15
     #define N_cells 4
16
        define constants in atomic units: eV, , ps, K */
18
     #define AMU 1.0364e-4
19
     #define degC_to_K 273.15
     #define bar 6.2415e-07
#define kB 8.61733e-5
20
21
22
      /* Main program */
24
     int main()
25
         char file_name[100];
26
27
         int N_atoms = 4*N_cells*N_cells*N_cells;
28
29
         double m_A1 = 27*AMU;
30
           Values of Young's and shear modulus, Y and G resp., taken from Physics Handbook, table T 1.1. Bulk mudulus then calculated as B = Y*G / (9*G - 3*Y) [F 1.15, Physics Handbook] kappa = 1/B
31
32
33
34
35
         double kappa_Al = 100/(6.6444e+05 * bar); // STRANGE FACTOR 100 OFF !!!
36
37
         double a_eq = 4.03;
         double cell_length = a_eq*N_cells;
38
         double inv_volume = pow(N_cells*cell_length, -3);
double noise_amplitude = 6.5e-2 * a_eq;
39
40
41
         double T_final_C= 700;
        int nRuns = 2; //2 if melt, 1 otherwise
double T_melt_C = 900;
43
44
45
         double P final bar= 1:
46
47
        double T_eq;
49
        double P_eq = P_final_bar*bar;
50
        double dt
                       = 5e-3;
51
        double tau_T = 100*dt;
       double tau_P = 100*dt;
52
53
        //double t_T_eq= 10*tau_T; //equlibration times
        double t_eq= 15*tau_P; //equlibration 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]));
        double (*momentum)[3] = malloc(sizeof(double[N_atoms][3]));
62
       double (*forces)[3] = malloc(sizeof(double[N_atoms][3]));
double *temperature = malloc(sizeof(double[N_timesteps]));
63
64
       double *pressure = malloc(sizeof(double[N_timesteps]));
65
66
67
68
       FILE *file_pointer;
69
70
        /* -----*/
71
72
73
        init_fcc(pos, N_cells, a_eq); // initialize fcc lattice
       add_noise( N_atoms, 3, pos, noise_amplitude ); // adds random noise to pos set_zero( N_atoms, 3, momentum); // set momentum to 0 get_forces_AL( forces, pos, cell_length, N_atoms); //initial cond forces
74
75
76
77
78
79
        for (int i=0; i<N_timesteps_T_eq; i++){</pre>
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
          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
          // 17 - Mai virial pressure[i] = inv_volume * (1.5*E_kin + virial); 
// 3N*kB*T/2 = 1/(2m) * \sum_{i=1}^{N} p_i^2 = p_sq/(2m)
90
91
          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]);
```

```
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</pre>
            if (irun == nRuns - 1){ // final run
  T_eq = T_final_C + degC_to_K;
103
104
105
            }else{
               T_eq = T_melt_C + degC_to_K;
106
107
             for (int i=0; i<N_timesteps; i++){</pre>
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
                         = 1/(2m) * \sum_{i=1}^{N} p_i^2 = p_sq/(2m) */
119
              3N*kB*T/2
120
           temperature[i] = E_{kin} * 1/(1.5*N_{atoms*kB});
           /* PV = NkT + virial */
121
122
           pressure[i] = inv_volume * (1.5*E_kin + virial);
123
124
           /* Equlibrate temperature by scaling momentum by a factor sqrt(alpha_T).
    N.B. It is equally valid to scale the momentum instead of the velocity ←
125
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
           // Equlibrate pressure by scaling the posistions by a factor of alpha_P\leftarrow
                 (1/3)
132
           alpha_P = 1 - kappa_Al* dt*(P_eq - pressure[i])/tau_P;
alpha_P_cube_root = pow(alpha_P, 1.0/3.0);
scale_mat(N_atoms, 3, pos, alpha_P_cube_root);
133
134
135
136
137
           cell_length*=alpha_P_cube_root;
138
           inv_volume*=1/alpha_P;
139
140
           temperature[i]*=alpha_T;
           pressure[i]*=alpha_P;
141
142
143
144
145
146
       /* Write tempertaure to file */
       147
148
150
       for (int i=0; i<N_timesteps; i++){</pre>
         151
152
153
154
155
       fclose(file_pointer);
157
       /* Write phase space coordinates to file */
       158
159
       file_pointer = fopen(file_name, "w");
for (int i=0; i<N_atoms; i++){</pre>
160
161
162
         for (int j=0; j<3; j++) {
           fprintf(file_pointer, " %.16e \t", pos[i][j]);
163
164
165
         for (int j=0; j<3; j++) {
           fprintf(file_pointer, " %.16e \t", momentum[i][j]);
166
167
168
         fprintf(file_pointer,"\n");
169
170
       fclose(file_pointer);
171
       ^{\primest} save equlibrated position and momentum as a binary file ^{st}/
172
       173
174
       file_pointer = fopen(file_name, "wb");
176
       fwrite(pos, sizeof(double), 3*N_atoms, file_pointer);
177
       fwrite(momentum, sizeof(double), 3*N_atoms, file_pointer);
       fwrite(&cell_length, sizeof(double), 1, file_pointer);
178
179
       fclose(file_pointer);
180
181
       printf("T=%0.2f\tP=%0.2e\n"
183
          temperature[N_timesteps-1], pressure[N_timesteps-1]);
184
```

```
185
186
187
       free(pos); pos = NULL;
188
       free(momentum); momentum = NULL;
       free(forces); forces = NULL;
189
190
       free(temperature); temperature = NULL;
191
       free(pressure); pressure = NULL;
192
       //free(volume); volume = NULL;
193
       return 0;
194
```

#### A.4 Production runs for tasks 3-7: main\_Prod.c

```
2
      MD_main.c
 3
      Created by Anders Lindman on 2013-10-31.
 6
     #include <stdio.h>
     #include <math.h>
     #include <stdlib.h>
10
     #include <time.h>
11
     #include "initfcc.h"
#include "alpotential.h"
#include "funcs.h"
12
13
15
16
     #define N_cells 4
     /* define constants in atomic units: eV, , ps, K */
17
     #define AMU 1.0364e-4
18
19
     #define degC_to_K 273.15
     #define bar 6.2415e-07
     #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;
29
       double m_Al = 27*AMU;
30
31
         Values of Young's and shear modulus, Y and G resp., taken from
         Physics Handbook, table T 1.1. Bulk mudulus then calculated as B = Y*G / (9*G - 3*Y) [F 1.15, Physics Handbook]
33
         kappa = 1/B
34
35
         double kappa_Al = 100/(6.6444e+05 * bar); // STRANGE FACTOR 100 OFF !!!
36
37
       double cell_length = 0;
       double inv_volume;
38
39
40
       double T_eq_C = 500;
41
       double P_eq_bar = 1;
/ double T_eq = T_eq_C + degC_to_K;
42
43
                            = P_eq_bar*bar;
         double P_eq
45
                         = 5e-4; // higher res for spectral function
       double dt
46
       double t_end
     // double tau_T = 100*dt;
// double tau_P = 100*dt;
47
48
49
50
       int N_timesteps = t_end/dt;
52
       int N_between_steps = 1;
53
       int N_save_timesteps = N_timesteps / N_between_steps; //for the displacements
54
       int N_save_atoms = 5;
55
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]));
       double (*pos_0)[3] = malloc(sizeof(double[N_atoms][3]));
double (*momentum)[3] = malloc(sizeof(double[N_atoms][3]));
double (*forces)[3] = malloc(sizeof(double[N_atoms][3]));
60
61
62
63
       double (*displacements)[N_save_atoms] =
                  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
                                 = malloc(sizeof(double[N_timesteps]));
       double *pressure
       double *msd
71
                                = malloc(sizeof(double[N_save_timesteps]));
       double *vel_corr
72
                                = malloc(sizeof(double[N_save_timesteps]));
       double *pow_spec
                           = malloc(sizeof(double[N_save_timesteps]));
```

```
double *freq = malloc(sizeof(double[N_save_timesteps]));
75
76
        for (int i = 0; i<N_save_timesteps; i++){</pre>
77
         msd[i] = 0;
pow_spec[i] = 0;
 78
 79
          vel_corr[i] = 0;
 80
 81
        FILE *file_pointer;
82
83
        /* ----- TASK 3 ------
84
        // read positions, momenta and cell_length
85
       86
87
88
        file_pointer = fopen(file_name, "rb");
       fread(pos, sizeof(double), 3*N_atoms, file_pointer);
fread(momentum, sizeof(double), 3*N_atoms, file_pointer);
fread(&cell_length, sizeof(double), 1, file_pointer);
89
 90
 91
 92
       fclose(file_pointer);
 93
94
        for (int i=0; i<N_atoms; i++){</pre>
95
          for (int j=0; j<3; j++){</pre>
           pos_0[i][j]=pos[i][j];
96
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
103
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);
113
          /* PV = NkT + virial */
          pressure[i] = inv_volume * (1.5*E_kin + virial);
/* 3N*kB*T/2 = 1/(2m) * \sum_{i=1}^{N} p_i^2 = p_sq/(2m) */
temperature[i] = E_kin * 1/(1.5*N_atoms*kB);
114
115
116
118
          if (i % N_between_steps == 0){
119
               int k = i/N_between_steps; // number of saved timesteps so far
               get_displacements (N_save_atoms, pos, pos_0, displacements[k]);
copy_mat(N_atoms, 3, pos, pos_all[k]);
120
121
122
123
               copy_mat(N_atoms, 3, momentum, vel_all[k]);
               scale_mat(N_atoms, 3, vel_all[k], 1/m_Al);
125
126
          if ((i*10) % N_timesteps == 0){
             printf("done %d0 %% of Verlet timestepping\n", (i*10)/N_timesteps);
127
         }
128
129
130
       printf("calculating MSD\n");
       get_MSD(N_atoms, N_save_timesteps, pos_all, msd);
131
132
        printf("calculating velocity correlation\n");
133
134
       get_vel_corr(N_atoms, N_save_timesteps, vel_all, vel_corr);
135
       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");
         * Write tempertaure to file */
143
144
       145
146
147
        for (int i=0; i<N_timesteps; i++){</pre>
          t = i*dt; // time at step i
fprintf(file_pointer, "%.4f \t %.8f \t %.8f \n",
149
150
151
              t, temperature[i],pressure[i]);
152
       fclose(file_pointer);
153
154
155
        /* Write displacements to file */
156
        sprintf(file_name,"../data/temp-%d_pres-%d_displacements.tsv",
       (int) T_eq_C, (int) P_eq_bar);
file_pointer = fopen(file_name, "w");
for (int i=0; i<N_save_timesteps; i++){
    t = i*dt*N_between_steps; // time at step i
    fprintf(file_pointer, "%.4f", t);</pre>
157
158
159
160
161
          for (int j=0; j<N_save_atoms; j++){</pre>
163
              fprintf(file_pointer, "\t %.8f", displacements[i][j]);
164
```

```
fprintf(file_pointer, "\n");
166
167
          fclose(file_pointer);
168
          169
170
172
          file_pointer = fopen(file_name, "w");
          // write header
173
          fprintf(file_pointer, "%% t[ps] \t MSD[A^2] \t vel_corr [A/ps]^2 \n");
for (int i=0; i<N_save_timesteps; i++){</pre>
174
175
              t = i*dt*N_between_steps; // time at step i
fprintf(file_pointer, "%.4f \t %.8f \t %.8f \n", t, msd[i], vel_corr[i]);
176
177
178
179
          fclose(file_pointer);
180
          sprintf(file_name,"../data/temp-%d_pres-%d_power-spectrum.tsv",
          (int) T_eq_C, (int) P_eq_bar);
file_pointer = fopen(file_name, "w");
181
182
183
184
          // write header
          // write the definition of the first file_pointer, "%% f[1/ps] \t P[A/ps]^2 \n");
for (int i=0; i<N_save_timesteps/2; i++){ // only print from f=0 to f_crit fprintf(file_pointer, "%.4f \t %.8f \n", freq[i], pow_spec[i]);</pre>
185
186
187
188
189
          fclose(file_pointer);
190
191
                                        pos = NULL;
192
          free(pos_0);
                                        pos_0 = NULL;
193
          free(momentum);
                                        momentum = NULL;
forces = NULL;
194
          free(forces):
195
          free(temperature); temperature = NULL;
                                       pressure = NULL;
          free(pressure);
          free(displacements); displacements = NULL;
197
          free(usp.all); pos_all = NULL;
free(vel_all); vel_all = NULL;
free(msd); msd = NULL;
free(vel_corr); vel_corr = NULL;
198
199
200
201
          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

```
MD_main.c
 3
      Created by Anders Lindman on 2013-10-31.
 6
     #include <stdio.h>
     #include <math.h>
     #include <stdlib.h>
     #include <time.h>
10
11
     #include "initfcc.h"
#include "alpotential.h"
#include "funcs.h"
12
13
15
16
     #define N_cells 4
17
      /* define constants in atomic units: eV, \, , ps, K ^*/
     #define AMU 1.0364e-4
#define degC_to_K 273.15
18
19
     #define bar 6.2415e-07
21
     #define kB 8.61733e-5
23
      /* Main program */
24
     int main()
25
26
        char file_name[100];
27
       int N_atoms = 4*N_cells*N_cells*N_cells;
double m_A1 = 27*AMU;
28
29
30
          Values of Young's and shear modulus, Y and G resp., taken from Physics Handbook, table T 1.1.\ Bulk mudulus then calculated as
31
32
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 !!!
       double cell_length = 0;
double inv_volume;
37
38
39
40
41
        double T_eq_C = 500;
       double P_eq_bar = 1;
```

```
double T_eq = T_eq_C + degC_to_K;
             double P_eq
 44
                                           = P_eq_bar*bar;
 45
            double dt
                                        = 5e-4; // higher res for spectral function
 46
            double t_end
                                       = 5:
              double tau_T = 100*dt;
 47
         // double tau_P = 100*dt;
 49
 50
            int N_timesteps = t_end/dt;
 51
 52
            int N_between_steps = 1;
53
            int N_save_timesteps = N_timesteps / N_between_steps; //for the displacements
            int N save atoms = 5:
 56
               double alpha_T, alpha_P,alpha_P_cube_root;
 57
            double t, E_kin, virial;
 58
 59
                                                  = malloc(sizeof(double[N atoms][3])):
            double (*pos)[3]
            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]));
 63
            double (*displacements)[N_save_atoms] =
 64
                              malloc(sizeof(double[N_save_timesteps][N_save_atoms]));
 65
            double (*pos_all)[N_atoms][3] =
                              malloc(sizeof(double[N_save_timesteps][N_atoms][3]));
 66
 67
            double (*vel_all)[N_atoms][3] =
                              malloc(sizeof(double[N_save_timesteps][N_atoms][3]));
 68
 69
            double *temperature = malloc(sizeof(double[N_timesteps]));
            double *pressure
 70
                                                   = malloc(sizeof(double[N_timesteps]));
            double *msd
 71
                                                   = malloc(sizeof(double[N_save_timesteps]));
            double *vel_corr
                                                   = malloc(sizeof(double[N save timesteps])):
 72
 73
            double *pow_spec
                                                  = malloc(sizeof(double[N_save_timesteps]));
            double *freq
                                                     = malloc(sizeof(double[N_save_timesteps]));
 75
 76
            for (int i = 0; i<N_save_timesteps; i++){</pre>
 77
               msd[i] = 0;
pow_spec[i] = 0;
 78
 79
               vel_corr[i] = 0;
 80
 81
            FILE *file_pointer;
 82
 83
            /* ----- TASK 3 ------
 84
            // read positions, momenta and cell_length sprintf(file\_name,"../data/INIDATA\_temp-\%data/INIDATA\_temp-%data/INIDATA\_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_temp-%data/INIDATA_Temp-%data/INIDATA_Temp-%data/INIDATA_Temp-%data/INIDATA_Temp-%data/INIDATA_Temp-%data/INIDATA_Temp-%data/INIDATA_Temp-%data/INIDATA_Temp-%data/INIDATA_Temp-%data/INIDATA_Temp-%data/INIDATA_Temp-%data/INIDATA_Temp-%data/INIDATA_Temp-%data/INIDATA_Temp-%data/INIDATA_Temp-%data/INIDATA_Temp-%data/INIDATA_Temp-%data/INIDATA_Temp-%data/INIDATA_Temp-%data/INIDATA_Temp-%data/INIDATA_Temp-%data/INIDATA_Temp-%data/INIDATA_Temp-%data/INIDATA_Temp-%data/INIDATA_Temp-%data/INIDATA_Temp-%data/INIDATA_Temp-%dat
 85
                                              ../data/INIDATA_temp-%d_pres-%d.bin",
 86
 87
                   (int) T_eq_C, (int) P_eq_bar);
 88
            file_pointer = fopen(file_name, "rb");
            fread(pos, sizeof(double), 3*N_atoms, file_pointer);
fread(momentum, sizeof(double), 3*N_atoms, file_pointer);
fread(&cell_length, sizeof(double), 1, file_pointer);
89
 90
 91
 92
            fclose(file_pointer);
 93
 94
            for (int i=0; i<N_atoms; i++){</pre>
 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++){</pre>
104
                     The loop over the timesteps first takes a timestep according to the
105
                     Verlet algorithm, then calculates the energies and temeperature.
106
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);
113
                /* PV = NkT + virial */
               pressure[i] = inv_volume * (1.5*E_kin + virial);  
/* 3N*kB*T/2 = 1/(2m) * \sum_{i=1}^{N} p_i^2 = p_sq/(2m) */temperature[i] = E_kin * 1/(1.5*N_atoms*kB);
114
115
116
117
                if (i % N_between_steps == 0){
119
                        int k = i/N_between_steps; // number of saved timesteps so far
                        get_displacements (N_save_atoms, pos, pos_0, displacements[k]);
copy_mat(N_atoms, 3, pos, pos_all[k]);
120
121
122
123
                        copy\_mat(N\_atoms\,,\ 3\,,\ momentum\,,\ vel\_all[k])\,;
                        scale_mat(N_atoms, 3, vel_all[k], 1/m_Al);
124
125
126
                if ((i*10) % N_timesteps == 0){
                     printf("done %d0 %% of Verlet timestepping\n", (i*10)/N_timesteps);
127
               }
128
129
130
            printf("calculating MSD\n");
            get_MSD(N_atoms, N_save_timesteps, pos_all, msd);
131
132
133
            printf("calculating velocity correlation \verb|\n"|);
```

```
get_vel_corr(N_atoms, N_save_timesteps, vel_all, vel_corr);
135
136
       printf("calculating power spectrum\n");
       get_powerspectrum(N_atoms, N_save_timesteps, vel_all, pow_spec);
137
       fft_freq(freq, dt, N_save_timesteps);
138
139
140
141
142
       printf("writing to file\n");
       /* Write tempertaure to file */
143
144
       sprintf(file_name,"../data/temp-%d_pres-%d_Prod-test.tsv",
145
           (int) T_eq_C, (int) P_eq_bar);
147
       file_pointer = fopen(file_name, "w");
148
       for (int i=0; i<N_timesteps; i++){</pre>
         t = i*dt; // time at step i
fprintf(file_pointer, "%.4f \t %.8f \t %.8f \n",
149
150
             t, temperature[i],pressure[i]);
151
152
153
       fclose(file_pointer);
154
       /* Write displacements to file */
155
       156
157
159
       for (int i=0; i<N_save_timesteps; i++){</pre>
         160
161
162
163
164
165
          fprintf(file_pointer, "\n");
166
167
       fclose(file_pointer);
168
        /* Write MSD to file */
169
       170
171
       file_pointer = fopen(file_name,
172
173
       // write header
       fprintf(file\_pointer, "%% t[ps] \ \ MSD[A^2] \ \ \ vel\_corr \ [A/ps]^2 \ \ \ \ \ ");
174
       for (int i=0; i<N_save_timesteps; i++){
   t = i*dt*N_between_steps; // time at step i
   fprintf(file_pointer, "%.4f \t %.8f \t %.8f \n", t, msd[i], vel_corr[i]);</pre>
175
176
177
178
179
       fclose(file_pointer);
180
       sprintf(file_name,"../data/temp-%d_pres-%d_power-spectrum.tsv",
          (int) T_eq_C, (int) P_eq_bar);
file_pointer = fopen(file_name, "w");
181
182
183
       // write header
       fprintf(file_pointer, "%% f[1/ps] \t P[A/ps]^2 \n");
185
       186
187
188
189
       fclose(file_pointer);
190
191
                             pos = NULL;
       free(pos);
192
       free(pos_0);
                             pos_0 = NULL;
193
       free(momentum):
                             momentum = NULL:
                             forces = NULL;
194
       free(forces):
195
       free(temperature); temperature = NULL;
                            pressure = NULL;
       free(pressure);
197
       free(displacements); displacements = NULL;
198
       free(pos_all); pos_all = NULL;
       free(vel_all); vel_all = NULL;
free(msd); msd = NULL;
free(vel_corr); vel_corr = NULL;
199
200
201
       free(pow_spec); pow_spec = NULL;
       free(freq); freq = NULL;
203
204
       return 0;
    }
205
```

#### A.6 Misc functions: funcs.c

```
#include "funcs.h"

void add_noise(int M, int N, double mat[M][N], double noise_amplitude )

{
    const gsl_rng_type *T; /* static info about rngs */
    gsl_rng *q; /* rng instance */
    gsl_rng_env_setup (); /* setup the rngs */
    T = gsl_rng_default; /* specify default rng */
    q = gsl_rng_alloc(T); /* allocate default rng */
    gsl_rng_set(q,time(NULL)); /* Initialize rng */
}
```

```
for (int i=0: i<N: i++){</pre>
 13
           for (int j=0; j<M; j++){</pre>
             // adds uniformly distributed random noise in range +-`noise_amplitude`
mat[i][j] += noise_amplitude * (2*gsl_rng_uniform(q)-1);
 14
15
          }
 16
 17
        gsl_rng_free(q); /* deallocate rng */
 19
20
     21
22
                        double cell_length){
        for (int i = 0; i < N_atoms; i++)
 24
25
           for (int j = 0; j < 3; j++) {
             /* p(t+dt/2) */
26
             momentum[i][j] += dt * 0.5 * forces[i][j];
27
28
             /* q(t+dt) */
 29
             pos[i][j] += dt * momentum[i][j] / m;
 30
          }
 31
        /* F(t+dt) */
32
33
        get_forces_AL( forces, pos, cell_length, N_atoms);
        for (int i = 0; i < N_atoms; i++) {
  for (int j = 0; j < 3; j++) {
    /* p(t+dt/2) */</pre>
 34
 35
36
             momentum[i][j] += dt * 0.5 * forces[i][j];
 37
 38
39
        }
40
     }
 41
      double get kin energy ( int N atoms. double (*momentum)[3]. double m ) {
        double p_sq=0; // momentum squared
for (int i = 0; i < N_atoms; i++) {</pre>
43
44
          for (int j = 0; j < 3; j++) {
45
             p_sq += momentum[i][j] * momentum[i][j];
46
          }
47
 48
        return p_sq / (2*m);
 50
 51
52
      void get_displacements ( int N_atoms, double (*positions)[3],
                     double (*initial_positions)[3], double disp[]) {
53
        for (int i = 0; i < N_atoms; i++) {
  for (int j = 0; j < 3; j++) {
 54
 55
 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
63
64
      void get_MSD ( int N_atoms, int N_times, double all_pos[N_times][N_atoms][3],
                          double MSD[N_times]) {
65
         /* all_pos = positions of all particles at all (saved) times *//* outer time index it starts at outer it = 1, since MSD[0] = 0*/
66
67
 68
         for (int it = 1; it < N_times; it++) { //</pre>
 69
             for (int jt = 0; jt < N_times-it; jt++) { // summed time index</pre>
                 for (int kn = 0; kn < N_atoms; kn++) { // particle index
for (int kd = 0; kd < 3; kd++) { // three dimensions</pre>
 70
 71
                        MSD[it] += (all_pos[it+jt][kn][kd] - all_pos[jt][kn][kd])
    *(all_pos[it+jt][kn][kd] - all_pos[jt][kn][kd]);
 72
 73
 75
                  }
 76
 77
             MSD[it] *= 1/( (double) N_atoms * (N_times-it));
 78
         }
 79
     }
 80
      void get_vel_corr ( int N_atoms, int N_times, double all_vel[N_times][N_atoms↔
           ][3],
82
                           double vel_corr[N_times]) {
         /* all_vel = velocity of all particles at all (saved) times */
for (int it = 0; it < N_times; it++) { //</pre>
83
 84
             for (int jt = 0; jt < N_times-it; jt++) { // summed time index</pre>
85
                 for (int kn = 0; kn < N_atoms; kn++) { // particle index for (int kd = 0; kd < 3; kd++) { // three dimensions
 86
87
88
                        vel_corr[it] += (all_vel[it+jt][kn][kd] * all_vel[jt][kn][kd]);
                     }
89
90
 91
             vel_corr[it] *= 1/( (double) N_atoms * (N_times-it));
93
94
     }
95
96
      void get_powerspectrum ( int N_atoms, int N_times, double all_vel[N_times][←
           N_atoms][3],
97
                           double pow_spec[N_times]) {
         /* all_vel = velocity of all particles at all (saved) times */
double vel_component[N_times]; // "all_vel[:][i][j]"
98
99
100
         double pow_spec_component[N_times];
```

```
double normalization_factor = 1/( (double)N_atoms * (N_times));
            for (int kn = 0; kn < N_atoms; kn++) {    // particle index
for (int kd = 0; kd < 3; kd++) {    // three dimensions
    for (int it = 0; it < N_times; it++) {    //</pre>
102
103
104
                         vel_component[it] = all_vel[it][kn][kd];
105
106
107
                       powerspectrum(vel_component, pow_spec_component, N_times);
                       for (int iw = 0; iw < N_times; iw++) { // for all frequencies
   pow_spec[iw] += pow_spec_component[iw];</pre>
108
109
                       }
110
                  }
111
112
            for (int iw = 0; iw < N_times; iw++) { // for all frequencies
    pow_spec[iw] *= normalization_factor;</pre>
114
115
116
       }
117
118
119
       121
122
123
124
125
126
127
128
       void set_zero (int M, int N, double mat[M][N]){
   /* Sets the matrix `mat` to zero */
for (int i = 0; i < M; i++) {
   for (int j = 0; j < N; j++) {</pre>
129
130
131
132
133
                mat[i][j] = 0;
              }
134
135
          }
136
       }
137
       void scale_mat (int M, int N, double mat[M][N], double alpha){
   /* Scales the matrix `mat` by factor `alpha` */
138
139
           for (int i = 0; i < M; i++) {
  for (int j = 0; j < N; j++) {
    mat[i][j] *= alpha;</pre>
140
141
142
143
             }
145
```

# B Auxiliary

#### **B.1** Makefile

```
CFLAGS = -03 -Wall -Wno-unused-result
     LIBS = -lm - lgsl - lgslcblas
     HEADERS = initfcc.h alpotential.h funcs.h fft_func.h
     OBJECTS = initfcc.o alpotential.o funcs.o fft_func.o
10
     %.o: %.c $(HEADERS)
11
          $(CC) -c -o $@ $< $(CFLAGS)
     all: Task1 Task2 Task3 main_Prod.c
15
     Task1: $(OBJECTS) main_T1.c
    $(CC) -o $@ $^ $(CFLAGS) $(LIBS)
16
17
18
     Task2: $(OBJECTS) main_T2.c
$(CC) -o $@ $^ $(CFLAGS) $(LIBS)
20
2.1
     Task3: $(OBJECTS) main_T3.c
$(CC) -o $@ $^ $(CFLAGS) $(LIBS)
22
23
24
     Prod: $(OBJECTS) main_Prod.c
26
          $(CC) -o $@ $^ $(CFLAGS) $(LIBS)
27
     # $(PROGRAMS): $(OBJECTS) main_T1.c
# $(CC) -o $@ $^ $(CFLAGS) $(LIBS)
28
29
30
31
          rm -f *.o
          touch *.c
```

# C Matlab scripts

#### C.1 Analysis scripts for tasks 3-7: Al\_energies.m

```
tmp = matlab.desktop.editor.getActive;
     cd(fileparts(tmp.Filename));
     set(0, 'DefaultrigureWindowStyle', 'docked');
GRAY = 0.7*[0.9 0.9 1];
     %% task 1
     energy_data = load('../data/lattice_energies.tsv');
     a0 = energy_data(:,1);
10
     v0 = a0.^3;
11
12
     energy = energy_data(:,2);
     figure(1); clf;
15
     plot(v0,energy, 'xk');
16
     start_v = 64;
end_v = 68;
17
18
     indToInclude = (v0 > start_v) & (v0 < end_v);</pre>
     p = polyfit(v0(indToInclude), energy(indToInclude), 2);
     hold on;
2.1
     vvec = linspace(start_v, end_v);
plot(vvec, p(1)*vvec.^2 + p(2)*vvec + p(3), '-r');
xlim([64 68]);
26
27
     v_{min} = -p(2)/(2*p(1));
     a_min = v_min^(1/3);
29
30
     ax = gca;
ax.YLim = [-13.45 -13.42];
31
     h1 = plot( v_min*[1 1], ax.YLim, '--k'); % plot vertical line at v_min
33
34
     ax.YTick = (-13.45:0.01:-13.42);
35
     ylabel('$E_{\rm pot}$ [eV/unit cell]');
xlabel('$a_0^3$ [\AA$^3$]');
36
37
     legend('data', 'quadratic fit', ['$V_{\rm eq} \approx \, $' num2str(round(v_min↔,2)) '\, \AA$^3$'], ...
'location', 'southeast')
39
     ImproveFigureCompPhys(gcf); h1.LineWidth = 2; setFigureSize(gcf);
40
41
     %axis([63 68 ylim(1) 0]);
saveas(gcf, '../figures/potential_energy.eps', 'epsc')
44
45
46
     %% task 2
47
     %clc:
48
     clf:clear
     dt=[1e-2,5e-3,2e-3,1e-3];
51
     for i=1:4
     T_data = load(sprintf('../data/temperature_dt-%0.0e_Task2.tsv',dt(i)));
E_data = load(sprintf('../data/total_energy_dt-%0.0e_Task2.tsv',dt(i)));
     t = T_data(:,1);
T = T_data(:,2);
     E = E_data(:,2);
58
     t_eq=0.5;
59
60
     fprintf('dt = %0.0e\n',dt(i));
61
63
     T_avg=mean(T(t>t_eq));
64
     T_std=std(T(t>t_eq));
     fprintf('\tT = %0.2f +- %0.1f %%\n', T_avg, abs(T_std/T_avg)*100);
65
66
67
     E_avg=mean(E(t>t_eq));
     E_std=std(E(t>t_eq));
     fprintf('\tE = \%0.2f +- \%0.1e \%\n', E_avg, abs(E_std/E_avg)*100);
70
71
     figure(i); clf
     plot(t, T)
yyaxis right
72
73
     plot(t, E)
     ylim(E_avg*(1+0.001*[1,-1]));
78
79
     %% test production pressure and temp
     clc; clf;
     %clear
```

```
%data = load(sprintf('../data/temperature_dt-1e-02_Task3.tsv'));
%data = load('../data/temp-700_pres-1_Task3.tsv');
data = load('../data/temp-500_pres-1_Prod-test.tsv');
%data = load('../data/temp-700_pres-1_Prod-test.tsv');
85
 86
 87
 88
      bar = 6.2415e-07;
 90
      t = data(:,1);
T = data(:,2)-273.15;
P = data(:,3)/bar;
 91
 92
 93
 94
 96
      t_eq=0.5;
97
98
      %fprintf('dt = %0.0e\n',dt(i));
99
100
101
      T_avg=mean(T(t>t_eq));
102
      T_std=std(T(t>t_eq));
103
       fprintf('\tT = \%0.2f +- \%0.1f \%\n', T_avg, abs(T_std/T_avg)*100);
104
      P_avg=mean(P(t>t_eq));
105
      P_std=std(P(t>t_eq));
fprintf('\tP = %0.2f +- %0.1f %%\n', P_avg, abs(P_std/P_avg)*100);
106
107
108
109
      yyans left
plot(t,T, 'color', GRAY),hold on
plot(t, cumsum(T)./(1:length(t))','-k')
ylabel('$T \, [^\circ \rm C]$')
110
111
112
113
      ylim([400,800])
115
116
117
      yyaxis right
      plot(t,P),hold on
118
119
      plot(t,cumsum(P)./(1:length(t))','-k')
120
121
      ylabel('$P \,[\rm bar]$')
ylim([-50,200])
122
123
124
       xlabel('$t$\, [ps]')
125
126
127
      ImproveFigureCompPhys(gcf, 'linewidth', 3, 'LineColor', {'MYORANGE', GRAY, '←
             MYBLUE', GRAY}');
128
      %% determine displacements and MSD
temperatures = num2str([500;700]);
129
130
       clc; clf;
131
       figure(10); clf;
133
      FILENAMES = strcat(\{'../data/temp-'\}, temperatures, '\_pres-1\_displacements.tsv') \hookleftarrow
      FILENAMES_Dyn = strcat({'.../data/temp-'}, temperatures, '_pres-1↔
    __dynamicProperties.tsv');
FILENAMES_Pow = strcat({'.../data/temp-'}, temperatures, '_pres-1_power-spectrum.↔
134
135
136
       for iFile = 1:numel(FILENAMES)
137
            figure(iFile); clf;
138
139
            data = load(FILENAMES{iFile});
140
            t = data(:,1);
141
            dx = data(:,2:end);
142
143
            plot(t, dx.^2); hold on;
144
145
146
            data = load(FILENAMES_Dyn{iFile});
            MSD = data(:,2);
148
            vel_corr = data(:,3);
140
            plot(t, MSD, 'k')
150
            if iFile ==2 % liquid
151
                  tStart = 1:
152
153
                  D = MSD(t>tStart)./(6*t(t>tStart));
                 selfDiffusionCoeff = mean(D); % in
plot(t, 6*t*selfDiffusionCoeff, ':');
154
155
156
            end
157
            leg = legend( strcat({'$n=$'}, num2str((1:size(dx,2))'))');
158
            leg.Location='northwest';
xlabel('$t$ [ps]')
ylabel('$\Delta x^2 \,[\rm \AA^2]$')
159
160
161
162
            if iFile ==1
                 ylim([ 0 1.0]);
163
            else
164
                 ylim([0 200]);
165
166
167
            ImproveFigureCompPhys(gcf);
168
            figure(10)
169
```

```
170
          plot(t, vel_corr/vel_corr(1)); hold on;
171
          xlim([0 1])
172
173
     end
174
     % velocity correlation
      figure(10);clf; figure(11);clf;
      n_average_points = 1;%30;
177
178
      for iFile = 1:numel(FILENAMES)
          data = load(FILENAMES_Dyn{iFile});
179
          t = data(:,1);
180
181
          vel_corr = data(:,3);
183
          data = load(FILENAMES_Pow{iFile});
          freq = data(:,1);
184
185
          pow_spec = data(:,2);
186
187
          figure(10);
188
          plot(t, vel_corr/vel_corr(1)); hold on;
189
190
          N_{times} = \frac{round(length(t)/2)}{round(length(t)/2)} % we have too bad statistics at later times. deltaf = 1/(N_{times} * dt);
191
192
          omegavec = 0:deltaf:(1/(2*dt));
193
          %PhiHat = 2 * trapz(t(1:N_times), (vel_corr(1:N_times) * ones(size(omegavec) \leftarrow (1:N_times)) 
          )) .* \cos(t(1:N_{times}) * omegavec), 1); %dimension 1
PhiHat = 1/2 * 1/N_times * 2 * sum( (vel_corr(1:N_times) * ones(size(\leftrightarrow omegavec))) .* \cos(t(1:N_{times}) * omegavec), 1); %dimension 1
195
196
197
          figure(11):
198
199
          plot(omegavec/(2*pi), PhiHat); hold on;
          plot(freq, pow_spec,
if iFile ==2 % liquid
200
                                   ':'); hold on;
201
202
               tStart = 1:
203
               selfDiffusionCoeff spectral = PhiHat(1)/6: % in ^2 /ps
204
205
206
207
     disp([selfDiffusionCoeff selfDiffusionCoeff spectral]):
208
209
210
      figure(10)
211
     xlim([0 1])
212
     leg = legend(strcat({'$T='}, num2str([500;700]), '\,^\circ $C'));
213
     leg.Location='northeast';
     xlabel('$t$ [ps]')
ylabel('$\Phi (t)/\Phi(0)$')
214
215
216
217
219
     hat v|^2$',..
220
           '$T= 700 \,
                         `\circ $C, $ \hat \Phi$', '$T= 700 \, ^\circ $C, $|\hat v|^2$');
     xlim([0 30])
ylim([0 Inf])
221
223
     ImproveFigureCompPhys('LineColor', {'r', 'MYRED', 'GERIBLUE','MYLIGHTBLUE'}');
224
225
     clc;clf;
226
227
228
      FILENAME = '.
                    ./data/INIDATA_temp-700_pres-1.bin';
     fID=fopen(FILENAME, 'rb');
230
      data1=fread(fID,[3,inf],'real*8').';
231
      fclose(fID);
232
233
     AMU = 1.0364e-4:
     m_A1 = 27*AMU;
234
     kB= 8.61733e-5;
236
     N_atoms=4^4;
237
238
     T=sum(sum(data1.^2,2),1) / (3*m_Al*N_atoms*kB)
239
240
     data2=load('../data/phase-space_temp-500_pres-1.tsv');
242
     T=sum(sum(data2(:,4:end).^2,2),1) / (3*m_Al*N_atoms*kB)
```

#### C.2 Improve figure appearance: ImproveFigureCompPhys.m

```
function ImproveFigureCompPhys(varargin)
function ImproveFigureCompPhys (varargin)
function ImproveFigureCompPhys Improves the figures of supplied handles
function ImproveFigureCompPhys(varargin)
function ImproveFigureCompPhys Improves the figures of supplied handles
function ImproveFigureCompPhys Improve the figures of supplied handles
function Improve function Improve the figures of supplied handles
function Improve fu
```

```
LineStyle column vector cell, e.g. \{'-','--'\}', LineColor column vector cell, e.g. \{'k',[0\ 1\ 1],\ 'MYBLUE'\}'
 9
     %
                                          colors: MYBLUE, MYORANGE, MYGREEN, MYPURPLE, MYYELLOW,
               MYLIGHTBLUE, MYRED
Marker column vector cell, e.g. {'.', 'o', 'x'}'
10
     %
11
     %
13
     % ImproveFigure was originally written by Adam Stahl, but has been heavily
     % modified by Linnea Hesslow
15
16
     %%% Handle inputs
17
     % If no inputs or if the first argument is a string (a property rather than
18
     % a handle), use all open figures
20
     if nargin == 0 || ischar(varargin{1})
21
          %Get all open figures
          figHs = findobj('Type','figure');
nFigs = length(figHs);
22
23
24
           % Check the supplied figure handles
26
           figHs = varargin{1};
27
           figHs = figHs(ishandle(figHs) == 1); %Keep only those handles that are \leftarrow
               proper graphics handles
28
           nFigs = length(figHs);
29
     end
30
     % Define desired properties
31
     titleSize = 24;
interpreter = 'latex';
33
34
     lineWidth = 4;
axesWidth = 1.5;
35
     labelSize = 22;
36
     textSize = 20;
37
38
     legTextSize = 18;
39
     tickLabelSize = 18;
     LineColor = {};
LineStyle = {};
40
41
42
     Marker = {};
44
     % define colors
45
     co = [ 0
0.8500
                     0.4470
                                   0.7410
                       0.3250
46
                                    0.0980
           0.9290
47
                       0.6940
                                    0.1250
           0.4940
                       0.1840
48
                                    0.5560
                                    0.1880
49
           0.4660
                       0.6740
50
           0.3010
                       0.7450
                                     0.9330
51
           0.6350
                       0.0780
                                    0.1840 ];
     colors = struct('MYBLUE', co(1,:),...
52
          'MYORANGE', CO(2,:),...
'MYORANGE', CO(3,:),...
'MYYELLOW', CO(3,:),...
'MYPURPLE', CO(4,:),...
'MYGREEN', CO(5,:),...
53
54
55
57
           'MYLIGHTBLUE', co(6,:),...
58
           'MYRED', co(7,:),..
          'GERIBLUE', [0.3000 0.1500 0.7500],...
'GERTRED', [1.0000 0.2500 0.1500],...
'GERIYELLOW', [0.9000 0.7500 0.1000],...
'LIGHTGREEN', [0.4 0.85 0.4],...
'LINNEAGREEN', [7 184 4]/255);
59
60
61
62
63
64
65
     % Loop through the supplied arguments and check for properties to set.
     for i = 1:nargin
66
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'
                         LineStyle = varargin{i+1};
e 'linecolor'
72
73
                     case
74
                          LineColor = varargin{i+1};
                          for iLineColor = 1:numel(LineColor)
76
                               if isfield(colors, LineColor{iLineColor})
77
                                    LineColor{iLineColor} = colors.(LineColor{iLineColor});
78
                               end
79
                          end
80
                           'marker'
                     case
81
                          Marker = varargin{i+1};
82
          end
83
     end
84
     85
86
     %%% Improve the figure(s)
88
89
     for iFig = 1:nFigs
90
91
           fig = figHs(iFig);
92
          lineObjects = findall(fig, 'Type', 'line');
textObjects = findall(fig, 'Type', 'text');
axesObjects = findall(fig, 'Type', 'axes');
legObjects = findall(fig, 'Type', 'legend');
94
95
96
```

```
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
           set(textObjects,'FontSize',textSize);
set(legObjects,'FontSize',legTextSize);
103
104
105
          %%% FIX LINESTYLE, COLOR ETC. FOR EACH PLOT SEPARATELY
for iAx = 1:numel(axesObjects)
106
107
108
               lineObjInAx = findall(axesObjects(iAx), 'Type', 'line');
110
               %set line style and color style (only works if all figs have some
111
               %number of line plots..)
112
               if ~isempty(LineStyle)
                    set(lineObjInAx, {'LineStyle'}, LineStyle)
set(contourObjects, {'LineStyle'}, LineStyle); %%%%%
113
114
115
               end
               if ~isempty(LineColor)
                     set(lineObjInAx, {'Color'}, LineColor)
117
                     set(contourObjects, {'LineColor'}, LineColor); %%%%%
118
119
                   ~isempty(Marker)
120
                     set(lineObjInAx, {'Marker'}, Marker)
set(lineObjInAx, {'Markersize'}, num2cell(10+22*strcmp(Marker, '.'))↔
121
122
123
               end
124
125
               %%% change font sizes.
126
               % Tick label size
127
               xLim = axesObjects(iAx).XLim;
128
                axesObjects(iAx).FontSize = tickLabelSize;
129
               axesObjects(iAx).XLim = xLim;
130
               %Change label size
               axesObjects(iAx).XLabel.FontSize = labelSize;
131
               axesObjects(iAx).YLabel.FontSize = labelSize;
132
133
134
               %Change title size
135
               axesObjects(iAx).Title.FontSize = titleSize;
           end
136
137
           %%% LINE APPEARANCE
138
           %Change line thicknesses
140
           set(lineObjects,'LineWidth',lineWidth);
          set(contourObjects, 'LineWidth', lineWidth);
set(axesObjects, 'LineWidth', axesWidth)
141
142
143
144
           % set interpreter: latex or tex
          set(textObjects, 'interpreter', interpreter)
set(legObjects, 'Interpreter', interpreter)
set(axesObjects, 'TickLabelInterpreter', interpreter);
146
147
148
      end
149
      end
```

#### C.3 Change size of figures: setFigureSize.m

```
function [ fig ] = setFigureSize( fig )
fig.Units = 'points';
W = 600;
H = 300;
fig.WindowStyle = 'normal'; % undock
fig.Position(3:4) = [W H];
end
```