# 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.

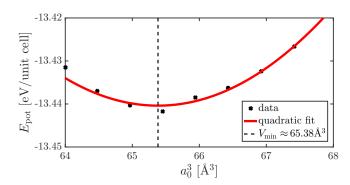


Figure 1: The potential energy per unit cell for aluminum as a function of the lattice parameter cubed.

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

#### Task 5

Equation (82) in MD lecture notes:

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

<u></u>

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

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

#### **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.)

#### **Problem 2**

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

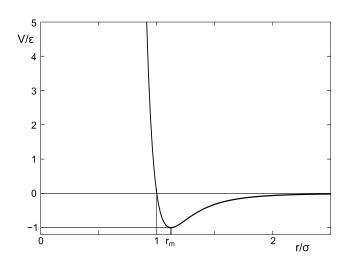


Figure 2: The Lennard–Jones potential. Make sure you label and have units on all axes! Also make sure that labels etc. are legible and that, if you print in black and white, that you use different line styles when required to differentiate between curves. In MATLAB you can export any figure to an .eps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

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:

$$V(r) = 4\epsilon \left[ \left( \frac{\sigma}{r} \right)^{12} - \left( \frac{\sigma}{r} \right)^{6} \right]$$
 (3)

Do number and reference all your equations.

## **Concluding discussion**

Use your favourite flavor of LATEX to compile the file:

xelatex template.tex odflatex 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←
                 (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>
     #include <time.h>
10
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;
- P_eq_bar*bar:
42
43
         double P_eq
                            = P_eq_bar*bar;
45
       double dt
                          = 5e-3;
                         = 30;
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
       double (*displacements)[N_save_atoms] =
63
                  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] =
                  malloc(sizeof(double[N save timesteps][N atoms][3])):
68
69
       double *temperature = malloc(sizeof(double[N_timesteps]));
70
       double *pressure
                                 = malloc(sizeof(double[N_timesteps]));
       double *msd
                                 = malloc(sizeof(double[N_save_timesteps]));
72
       double *vel corr
                                 = malloc(sizeof(double[N_save_timesteps]));
```

```
for (int i = 0; i<N_save_timesteps; i++){</pre>
75
         msd[i] = 0;
76
        FILE *file_pointer;
77
 78
 79
                                ----- TASK 3 -----
 80
        // read positions, momenta and cell_length
 81
        82
83
84
        fread(pos, sizeof(double), 3*N_atoms, file_pointer);
fread(momentum, sizeof(double), 3*N_atoms, file_pointer);
85
86
87
        fread(&cell_length, sizeof(double), 1, file_pointer);
88
        fclose(file_pointer);
89
        for (int i=0; i<N_atoms; i++) {
  for (int j=0; j<3; j++) {</pre>
90
91
 92
            pos_0[i][j]=pos[i][j];
 93
94
95
        inv_volume = pow(N_cells*cell_length, -3);
96
97
 98
99
        get_forces_AL( forces, pos, cell_length, N_atoms); //initial cond forces
100
101
        for (int i=0; i<N_timesteps; i++){</pre>
102
103
              The loop over the timesteps first takes a timestep according to the
104
              Verlet algorithm, then calculates the energies and temeperature.
105
106
          timestep_Verlet(N_atoms, pos, momentum, forces, m_Al, dt, cell_length);
107
          E_kin = get_kin_energy(N_atoms, momentum, m_Al );
virial = get_virial_AL(pos, cell_length, N_atoms);
108
109
110
111
            * 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);
113
114
115
116
          if (i % N_between_steps == 0){
                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]);
118
119
120
                copy_mat(N_atoms, 3, momentum, vel_all[k]);
scale_mat(N_atoms, 3, vel_all[k], 1/m_Al);
121
122
123
           }
124
125
126
127
        get_MSD(N_atoms, N_save_timesteps, pos_all, msd);
128
        get_vel_corr(N_atoms, N_save_timesteps, vel_all, vel_corr);
129
130
        /* Write tempertaure to file */
131
        132
133
        file_pointer = fopen(file_name, "w");
134
135
        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",
137
138
              t, temperature[i],pressure[i]);
139
        fclose(file_pointer);
140
141
142
        /* Write displacements to file */
        143
144
        (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);
    for (int j=0; j<N_save_atoms; j++){
        fprintf(file_pointer, "\t %.8f", displacements[i][j]);
}</pre>
145
146
147
149
150
151
            fprintf(file pointer. "\n"):
152
153
154
        fclose(file_pointer);
155
156
         /* Write MSD to file */
        157
158
        file_pointer = fopen(file_name, "w");
159
160
        // write header
        fprintf(file_pointer, "%% t[ps] \t MSD[A^2] \t vel_corr [A/ps]^2 \n");
161
        for (int i=0; i<N_save_timesteps; i++){</pre>
           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]);
163
164
```

```
165
166
        fclose(file_pointer);
167
                                   pos = NULL;
168
        free(pos);
                                   pos_0 = NULL;
169
        free(pos_0);
170
        free(momentum);
                                  momentum = NULL;
                                   forces = NULL;
        free(forces);
172
        free(temperature); temperature = NULL;
173
        free(pressure);
                                  pressure = NULL;
        free(displacements); displacements = NULL;
free(pos_all); pos_all = NULL;
free(vel_all); vel_all = NULL;
174
175
176
        return 0;
178
```

#### A.5 Production runs for tasks 3-7: main\_Prod.c

```
MD_main.c
      Created by Anders Lindman on 2013-10-31.
 5
 6
     #include <stdio.h>
#include <math.h>
     #include <stdlib.h>
     #include <time.h>
11
     #include "initfcc.h"
#include "alpotential.h"
#include "funcs.h"
12
13
14
15
     #define N_cells 4
17
      ^{\prime *} define constants in atomic units: eV, \,\, , ps, K ^*/
18
     #define AMU 1.0364e-4
     #define degC_to_K 273.15
#define bar 6.2415e-07
19
20
     #define kB 8.61733e-5
23
      /* Main program */
24
     int main()
25
       char file_name[100];
26
27
28
        int N_atoms = 4*N_cells*N_cells;
29
        double m_Al = 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]
31
32
33
34
          kappa = 1/B
35
36
         double kappa_A1 = 100/(6.6444e+05 * bar); // STRANGE FACTOR 100 OFF !!!
37
       double cell_length = 0;
       double inv_volume;
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 = P_eq_bar*bar;
42
43
44
45
       double dt
                          = 5e-3:
       double t_end
46
                          = 30;
         double tau_T = 100*dt;
47
48
     // double tau_P = 100*dt;
40
50
       int N_timesteps = t_end/dt;
51
52
        int N_between_steps = 1;
       int N_save_timesteps = N_timesteps / N_between_steps; //for the displacements
54
        int N_save_atoms = 5;
55
     // double alpha_T, alpha_P,alpha_P_cube_root;
double t, E_kin, virial;
56
57
58
        double (*pos)[3]
                                  = malloc(sizeof(double[N_atoms][3]));
        double (*pos_0)[3]
                                  = malloc(sizeof(double[N_atoms][3]));
61
        double (*momentum)[3] = malloc(sizeof(double[N_atoms][3]));
        double (*forces)[3] = malloc(sizeof(double[N_atoms][3]));
62
       double (*displacements)[N_save_atoms] =
    malloc(sizeof(double[N_save_timesteps][N_save_atoms]));
63
64
65
        double (*pos_all)[N_atoms][3] =
                    malloc(sizeof(double[N_save_timesteps][N_atoms][3]));
        double (*vel_all)[N_atoms][3] =
67
68
                   malloc(sizeof(double[N_save_timesteps][N_atoms][3]));
       double *temperature = malloc(sizeof(double[N_timesteps]));
69
```

```
= malloc(sizeof(double[N_timesteps]));
        double *pressure
       double *msd
double *vel_corr
71
                                 = malloc(sizeof(double[N_save_timesteps]));
72
                               = malloc(sizeof(double[N_save_timesteps]));
 73
        for (int i = 0; i<N_save_timesteps; i++){</pre>
 74
 75
        msd[i] = 0;
 76
        FILE *file_pointer;
 77
78
 79
        /* ----- TASK 3 ------
80
        // read positions, momenta and cell_length
81
       82
83
84
        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);
85
86
87
        fclose(file_pointer);
88
 89
90
        for (int i=0; i<N_atoms; i++){</pre>
91
          for (int j=0; j<3; j++){</pre>
92
           pos_0[i][j]=pos[i][j];
         }
93
 95
        inv_volume = pow(N_cells*cell_length, -3);
96
97
98
99
        get forces AL( forces. pos. cell length. N atoms): //initial cond forces
100
        for (int i=0; i<N_timesteps; i++){</pre>
102
103
             The loop over the timesteps first takes a timestep according to the
104
             Verlet algorithm, then calculates the energies and temeperature.
105
106
          timestep_Verlet(N_atoms, pos, momentum, forces, m_Al, dt, cell_length);
107
108
          E_kin = get_kin_energy(N_atoms, momentum, m_Al );
109
          virial = get_virial_AL(pos, cell_length, N_atoms);
110
          /* PV = NkT + virial */
111
          pressure[i] = inv_volume * (1.5*E_kin + virial);
112
          114
115
116
          if (i % N_between_steps == 0){
               int k = i/N_between_steps; // number of saved timesteps so far
117
                get_displacements (N_save_atoms, pos, pos_0, displacements[k]);
118
               copy_mat(N_atoms, 3, pos, pos_all[k]);
119
               copy_mat(N_atoms, 3, momentum, vel_all[k]);
scale_mat(N_atoms, 3, vel_all[k], 1/m_Al);
121
122
123
           }
124
125
126
        get_MSD(N_atoms, N_save_timesteps, pos_all, msd);
127
128
        get_vel_corr(N_atoms, N_save_timesteps, vel_all, vel_corr);
129
130
        /* Write tempertaure to file */
131
        133
        file_pointer = fopen(file_name, "w");
134
       for (int i=0; i<N_timesteps; i++){
  t = i*dt; // time at step i
  fprintf(file_pointer, "%.4f \t %.8f \t %.8f \n",</pre>
135
136
137
138
              t, temperature[i],pressure[i]);
139
140
        fclose(file_pointer);
141
       /* Write displacements to file */
sprintf(file_name,"../data/temp-%d_pres-%d_displacements.tsv",
    (int) T_eq_C, (int) P_eq_bar);
142
143
        file_pointer = fopen(file_name, "w");
145
        for (int i=0; i<N_save_timesteps; i++){</pre>
146
          t = i*dt*N_between_steps; // time at step i
fprintf(file_pointer, "%.4f", t);
for (int j=0; j<N_save_atoms; j++){
    fprintf(file_pointer, "\t %.8f", displacements[i][j]);</pre>
147
148
149
150
151
152
           fprintf(file_pointer, "\n");
153
154
        fclose(file_pointer);
155
156
         /* Write MSD to file */
        sprintf(file_name,"../data/temp-%d_pres-%d_dynamicProperties.tsv",
            (int) T_eq_C, (int) P_eq_bar);
159
        file_pointer = fopen(file_name, "w");
       // write header
160
```

```
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++){
   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>
162
163
164
165
166
        fclose(file_pointer);
167
168
                                   pos = NULL;
         free(pos);
169
        free(pos_0);
                                   pos_0 = NULL;
                                   momentum = NULL;
forces = NULL;
170
        free(momentum);
171
        free(forces):
                                   temperature = NULL;
        free(temperature);
172
                                   pressure = NULL;
        free(pressure);
174
        free(displacements); displacements = NULL;
175
        free(pos_all); pos_all = NULL;
176
        free(vel_all); vel_all = NULL;
        return 0:
177
```

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

```
#include "funcs.h"
 2
     void add_noise(int M, int N, double mat[M][N], double noise_amplitude )
                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 */
10
        gsl_rng_set(q,time(NULL)); /* Initialize rng */
11
         for (int j=0; j<M; j++){
   // adds uniformly distributed random noise in range +-`noise_amplitude`
   mat[i][j] += noise_amplitude * (2*gsl_rng_uniform(q)-1);</pre>
13
14
15
16
17
        gsl_rng_free(q); /* deallocate rng */
19
20
     void timestep_Verlet ( int N_atoms, double (*pos)[3], double (*momentum)[3],
21
                        double (*forces)[3], double m, double dt,
double cell_length){
22
23
24
        for (int i = 0; i < N_atoms; i++) {</pre>
          for (int j = 0; j < 3; j++) {
25
26
             /* p(t+dt/2)
27
             momentum[i][j] += dt * 0.5 * forces[i][j];
28
             /* a(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
             /* p(t+dt/2)
36
             momentum[i][j] += dt * 0.5 * forces[i][j];
37
38
39
       }
40
41
42
     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++) {
   p_sq += momentum[i][j] * momentum[i][j];</pre>
45
46
          }
47
48
       return p_sq / (2*m);
50
     void get_displacements ( int N_atoms, double (*positions)[3],
52
        double (*initial_positions)[3], double disp[]) {
for (int i = 0; i < N_atoms; i++) {</pre>
53
54
          for (int j = 0; j < 3; j++) {
             disp[i] += (positions[i][j] - initial_positions[i][j])
56
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]) {
```

```
/* all_pos = positions of all particles at all (saved) times */
 67
              /* outer time index it starts at outer it = 1, since MSD[0] = 0*/
              for (int it = 1; it < N_times; it++) { //
for (int jt = 0; jt < N_times-it; jt++) { // summed time index
for (int kn = 0; kn < N_atoms; kn++) { // particle index
for (int kd = 0; kd < 3; kd++) { // three dimensions
 68
 69
 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]);
 73
 74
 75
                          }
 76
 77
                   MSD[it] *= 1/( (double) N_atoms * (N_times-it));
 79
 80
        void get_vel_corr ( int N_atoms, int N_times, double all_vel[N_times][N_atoms↔
81
                ][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++) { // summed time index
   for (int kn = 0; kn < N_atoms; kn++) { // particle index
      for (int kd = 0; kd < 3; kd++) { // three dimensions
      vel_corr[it] += (all_vel[it+jt][kn][kd] * all_vel[jt][kn][kd]);</pre>
 85
 86
87
 88
 89
 90
 91
 92
                   vel_corr[it] *= 1/( (double) N_atoms * (N_times-it));
 93
             }
 94
        }
 95
 97
        void copy_mat (int M, int N, double mat_from[M][N], double mat_to[M][N]){
              /* Copies matrix `mat_from` to `mat_to` */
for (int i = 0; i < M; i++) {
   for (int j = 0; j < N; j++) {
98
99
100
                   mat_to[i][j] = mat_from[i][j];
101
102
103
             }
104
        }
105
        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>
106
107
109
110
                  mat[i][j] = 0;
111
                }
112
           }
113
114
        void scale_mat (int M, int N, double mat[M][N], double alpha){
  /* Scales the matrix `mat` by factor `alpha` */
  for (int i = 0; i < M; i++) {
    for (int j = 0; j < N; j++) {
      mat[i][j] *= alpha;
    }
}</pre>
116
117
118
119
120
121
           }
```

# B Auxiliary

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

```
CFLAGS = -03 -Wall
LIBS = -lm -lgsl -lgslcblas
    HEADERS = initfcc.h alpotential.h funcs.h
    OBJECTS = initfcc.o alpotential.o funcs.o
9
10
    %.o: %.c $(HEADERS)
        $(CC) -c -o $@ $< $(CFLAGS)
11
12
    all: Task1 Task2 Task3 main_Prod.c
14
    15
16
17
    Task2: $(OBJECTS) main_T2.c
$(CC) -o $@ $^ $(CFLAGS) $(LIBS)
19
20
2.1
    Task3: $(OBJECTS) main_T3.c
```

```
22 | $(CC) -o $@ $^ $(CFLAGS) $(LIBS)

23 | Prod: $(OBJECTS) main_Prod.c
25 | $(CC) -o $@ $^ $(CFLAGS) $(LIBS)

26 | # $(PROGRAMS): $(OBJECTS) main_T1.c
28 | # $(CC) -o $@ $^ $(CFLAGS) $(LIBS)

29 | | clean:
31 | rm -f *.o
32 | 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, 'DefaultFigureWindowStyle', 'docked');
    GRAY = 0.7*[0.9 0.9 1];
    %% task 1
    clc
    energy_data = load('../data/lattice_energies.tsv');
    a0 = energy_data(:,1);
    v0 = a0.^3;
11
12
13
    energy = energy_data(:,2);
    figure(1); clf;
14
    plot(v0, energy, 'xk');
16
17
    start_v = 64;
    end_v = 68:
18
    indToInclude = (v0 > start_v) & (v0 < end_v);</pre>
19
    p = polyfit(v0(indToInclude), energy(indToInclude),2);
20
    hold on;
23
    vvec = linspace(start_v, end_v);
24
25
    plot(vvec, p(1)*vvec.^2 + p(2)*vvec + p(3), '-r');
    xlim([64 68]);
26
    v_min = -p(2)/(2*p(1));
a_min = v_min^(1/3);
29
30
    ax.YLim = [-13.45 - 13.42];
31
    h1 = plot(v_min*[1 1], ax.YLim, '--k'); % plot vertical line at v_min
32
33
35
    ax.YTick = (-13.45:0.01:-13.42);
    36
37
38
39
40
    ImproveFigureCompPhys(gcf); h1.LineWidth = 2; setFigureSize(gcf);
41
    %axis([63 68 ylim(1) 0]);
saveas(gcf, '../figures/potential_energy.eps', 'epsc')
42
43
    %% task 2
47
    %clc;
48
    clf; clear
49
50
    dt=[1e-2,5e-3,2e-3,1e-3];
    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)));
53
    t = T_data(:,1);
55
    T = T_data(:,2);
    E = E data(:.2):
56
57
    t_eq=0.5;
59
60
61
    fprintf('dt = %0.0e\n',dt(i));
62
63
    T_avg=mean(T(t>t_eq));
    T_std=std(T(t>t_eq));
    fprintf('\tT = %0.2f +- %0.1f %%\n', T_avg, abs(T_std/T_avg)*100);
    E_avg=mean(E(t>t_eq));
```

```
 \begin{split} &E\_std = std(E(t) + L_eq)); \\ &fprintf(' \setminus tE = \%0.2f +- \%0.1e \ \%\% \setminus n', \ E\_avg, \ abs(E\_std/E\_avg)*100); \end{split} 
69
 70
 71
      figure(i);clf
      plot(t, T)
yyaxis right
 72
 73
 74
      plot(t, E)
 75
      ylim(E_avg*(1+0.001*[1,-1]));
 76
      end
 77
 78
 79
      %% test production pressure and temp
      clc; clf;
81
      %clear
82
83
84
      %data = load(sprintf('../data/temperature_dt-1e-02_Task3.tsv'));
85
      data = load('../data/temp-700_pres-1_Task3.tsv');
%data = load('../data/temp-700_pres-1_Prod-test.tsv');
 86
 87
88
89
      har = 6.2415e-07:
 90
 91
      t = data(:,1);
 92
      T = data(:,2)-273.15;
 93
      P = data(:,3)/bar;
 94
 95
 96
      t_eq=0.5;
 97
 98
 99
      %fprintf('dt = %0.0e\n',dt(i));
100
101
      T_avg=mean(T(t>t_eq));
      T_std=std(T(t>t_eq));
102
103
      fprintf('\tT = %0.2f +- %0.1f %%\n', T_avg, abs(T_std/T_avg)*100);
104
105
      P_avg=mean(P(t>t_eq));
106
      P_std=std(P(t>t_eq));
107
      fprintf('\tP = \%0.2f +- \%0.1f \%\n', P_avg, abs(P_std/P_avg)*100);
108
109
      yyaxis 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
114
      ylim([600,800])
115
116
117
      yyaxis right
      plot(t,P),hold on
119
      plot(t,cumsum(P)./(1:length(t))','-k')
120
121
      ylabel('$P \,[\rm bar]$')
ylim([-50,200])
122
123
124
125
      xlabel('$t$\, [ps]')
126
      ImproveFigureCompPhys(gcf, 'linewidth', 3, 'LineColor', {'MYORANGE', GRAY, '←
127
           MYBLUE', GRAY}');
128
129
      %% determine displacements and MSD
130
131
      clc; clf;
132
      figure(10); clf;
      FILENAMES = strcat({'../data/temp-'}, num2str([500;700]), '_pres-1_displacements \cdots tsv');
133
134
      FILENAMES\_Dyn = strcat(\{'../data/temp-'\}, num2str([500;700]), '\_pres-1 \leftarrow
             dynamicProperties.tsv');
135
      for iFile = 1:numel(FILENAMES)
136
           figure(iFile); clf;
137
           data = load(FILENAMES{iFile});
138
           t = data(:,1);
139
140
           dx = data(:,2:end);
141
           plot(t, dx.^2); hold on;
142
143
144
145
           data = load(FILENAMES_Dyn{iFile});
146
           MSD = data(:,2);
147
           vel_corr = data(:,3);
148
           plot(t, MSD, 'k')
149
150
           if iFile ==2 % liquid
151
                tStart = 1;
                D = MSD(t>tStart)./(6*t(t>tStart));
152
                selfDiffusionCoeff = mean(D); % in
plot(t, 6*t*selfDiffusionCoeff, ':');
153
154
155
```

```
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
          if iFile ==1
161
              ylim([ 0 1.0]);
163
          ylim([0 200]);
end
164
165
          ImproveFigureCompPhys(gcf);
166
167
          plot(t, vel_corr/vel_corr(1)); hold on;
xlim([0 1])
169
170
171
172
173
174
     figure(10)
      for iFile = 1:numel(FILENAMES)
176
          data = load(FILENAMES_Dyn{iFile});
177
          t = data(:,1);
          vel corr = data(:.3):
178
          plot(t, vel_corr/vel_corr(1)); hold on;
179
180
181
182
183
     xlim([0 1])
     leg = legend(strcat({'$T='}, num2str([500;700]), '\,^\circ $C'));
leg.Location='northeast';
184
185
          xlabel('$t$ [ps]')
ylabel('$\Phi (t)/\Phi(0)$')
186
188
     ImproveFigureCompPhys();
189
190
     clc;clf;
191
192
     FILENAME = '.
                     ./data/INIDATA_temp-700_pres-1.bin';
194
     fID=fopen(FILENAME, 'rb');
195
     data1=fread(fID,[3,inf],'real*8').';
196
     fclose(fID):
197
198
     AMU = 1.0364e-4;
     m_A1 = 27*AMU;
200
     kB= 8.61733e-5;
201
     N_atoms=4^4;
202
     T=sum(sum(data1.^2,2),1) / (3*m_Al*N_atoms*kB)
203
204
205
     data2=load('../data/phase-space_temp-500_pres-1.tsv');
207
     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)
    %ImproveFigureCompPhys Improves the figures of supplied handles
       Input:
 4
    \mbox{\%} - none (improve all figures) or handles to figures to improve
      - optional:
 5
             LineWidth int
             LineStyle column vector cell, e.g. {'-','--'}',
LineColor column vector cell, e.g. {'k',[0 1 1], 'MYBLUE'}'
 8
                                    colors: MYBLUE, MYORANGE, MYGREEN, MYPURPLE, MYYELLOW,
             MYLIGHTBLUE, MYRED
Marker column vector cell, e.g. {'.', 'o', 'x'}'
10
11
12
13
    % ImproveFigure was originally written by Adam Stahl, but has been heavily
    % modified by Linnea Hesslow
15
16
17
    %%% Handle inputs
    % If no inputs or if the first argument is a string (a property rather than
18
    % a handle), use all open figures
19
     if nargin == 0 || ischar(varargin{1})
         %Get all open figures
22
         figHs = findobj('Type','figure');
         nFigs = length(figHs);
23
24
25
         % 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);
    end
29
```

```
31
      % Define desired properties
32
      titleSize = 24;
      interpreter = 'latex':
33
      lineWidth = 4;
34
      axesWidth = 1.5;
35
      labelSize = 22;
 37
      textSize = 20;
38
      legTextSize = 18;
      tickLabelSize = 18;
39
40
      LineColor = {};
      LineStyle = {};
41
      Marker = {};
43
44
      % define colors
                      0.4470
45
      co = [ 0
                                   0.7410
           0.8500
                        0.3250
46
                                     0.0980
47
           0.9290
                        0.6940
                                     0.1250
 48
           0.4940
                        0.1840
                                     0.5560
49
           0.4660
                        0.6740
                                     0.1880
50
           0.3010
                        0.7450
                                     0.9330
51
           0.6350
                       0.0780
                                    0.1840 ];
      colors = struct('MYBLUE', co(1,:),...
52
           'MYORANGE', co(2,:),...
'MYYELLOW', co(3,:),...
'MYPURPLE', co(4,:),...
53
 55
           'MYPURPLE', co(4,:),...
'MYGREN', co(5,:),...
'MYLIGHTBLUE', co(6,:),...
'MYRED',co(7,:),...
'GERIBLUE', [0.3000 0.1500 0.7500],...
'GERIRED', [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);
 56
57
58
59
                                                     0.75001....
60
                                                       0.1000],...
61
62
63
64
      % Loop through the supplied arguments and check for properties to set.
65
      for i = 1:nargin
66
67
           if ischar(varargin{i})
                68
69
                     case 'linewidth'
                          lineWidth = varargin{i+1};
 70
 71
                     case
                            'linestvle'
                          LineStyle = varargin{i+1};
e 'linecolor'
 72
 73
                          LineColor = varargin{i+1};
 74
 75
                           for iLineColor = 1:numel(LineColor)
                               if isfield(colors, LineColor{iLineColor})
 76
                                    LineColor{iLineColor} = colors.(LineColor{iLineColor});
 77
 78
 79
                          end
 80
                     case 'marker'
81
                          Marker = varargin{i+1};
                end
82
           end
83
      end
84
85
      86
87
      %%% Improve the figure(s)
88
89
      for iFig = 1:nFigs
90
91
           fig = figHs(iFig);
           lineObjects = findall(fig, 'Type', 'line');
textObjects = findall(fig, 'Type', 'text');
axesObjects = findall(fig, 'Type', 'axes');
legObjects = findall(fig, 'Type', 'legend');
contourObjects = findall(fig, 'Type', 'contour'); % not counted as lines
93
94
95
96
 97
 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
106
           %%% FIX LINESTYLE, COLOR ETC. FOR EACH PLOT SEPARATELY
107
           for iAx = 1:numel(axesObjects)
                lineObjInAx = findall(axesObjects(iAx), 'Type', 'line');
108
109
                %set line style and color style (only works if all figs have some
110
                %number of line plots..)
112
                if ~isempty(LineStyle)
                     set(lineObjInAx, {'LineStyle'}, LineStyle)
set(contourObjects, {'LineStyle'}, LineStyle); %%%%%
113
114
                end
if ~isempty(LineColor)
115
116
                     set(lineObjInAx, {'Color'}, LineColor)
117
                     set(contourObjects, {'LineColor'}, LineColor); %%%%%
119
                if ~isempty(Marker)
120
```

```
set(lineObjInAx, {'Marker'}, Marker)
set(lineObjInAx, {'Markersize'}, num2cell(10+22*strcmp(Marker, '.'))←
122
123
                    end
124
125
                    %%% change font sizes.
126
                    % Tick label size
127
                    xLim = axesObjects(iAx).XLim;
                    axesObjects(iAx).FontSize = tickLabelSize;
axesObjects(iAx).XLim = xLim;
%Change label size
128
129
130
                    axesObjects(iAx).XLabel.FontSize = labelSize;
131
                    axesObjects(iAx).YLabel.FontSize = labelSize;
133
                    %Change title size
axesObjects(iAx).Title.FontSize = titleSize;
134
135
              end
136
137
              %%% LINE APPEARANCE
138
139
              %Change line thicknesses
              set(lineObjects,'LineWidth',lineWidth);
set(contourObjects, 'LineWidth', lineWidth);
set(axesObjects, 'LineWidth',axesWidth)
140
141
142
143
              % set interpreter: latex or tex
set(text0bjects, 'interpreter', interpreter)
set(leg0bjects, 'Interpreter', interpreter)
set(axes0bjects,'TickLabelInterpreter', interpreter);
145
146
147
148
        end
149
        end
```

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

```
function [ fig ] = setFigureSize( fig )
figureSizePaper1
fig.Units = 'points';
W = 600;
H = 300;
fig.WindowStyle = 'normal'; % undock
fig.Position(3:4) = [W H];
end
```