

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 N ^o	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 $V_{\text{eq}} \approx 65.38 \text{ \AA}^3$. This corresponds to the equilibrium lattice parameter $a_{\text{eq}} \approx 4.029 \text{ \AA}$ at 0 K, which we took as the initial lattice parameter for the following tasks.

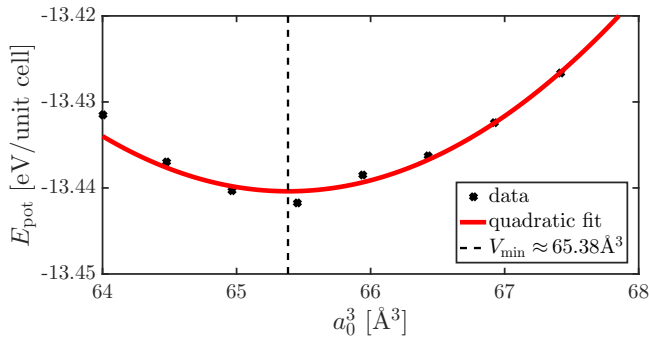


Figure 1: The potential energy per unit cell for aluminum as a function of the lattice parameter cubed.

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

Task 5

Equation (82) in MD lecture notes:

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

\Rightarrow

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

To determine M, we used mean of ... for t_i ...

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.

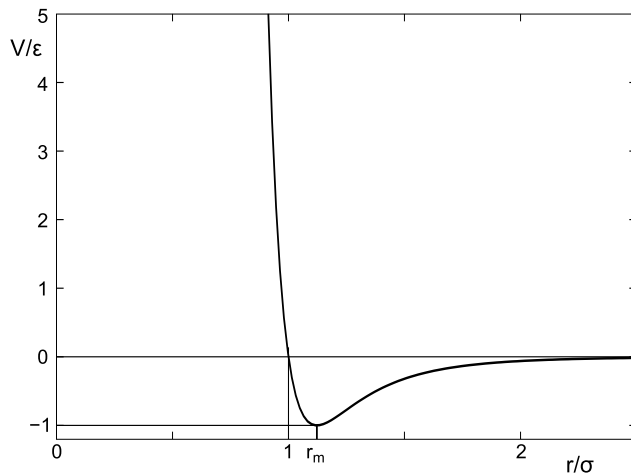


Figure 2: The Lennard–Jones potential. Make sure you label and have units on all axes! Also make sure that labels etc. are legible and that, if you print in black and white, that you use different line styles when required to differentiate between curves. In MATLAB you can export any figure to an .eps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

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

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, *LT_EX: 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

```
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
11 #include "initfcc.h"
12 #include "alpotential.h"
13
14 #define N_cells 4
15 #define N_lattice_params 25
16
17
18 /* Main program */
19 int main()
20 {
21
22     int N_atoms = 4*N_cells*N_cells*N_cells;
23     double a0;
24     double a0_min = 4.0;
25     double a0_max = 4.2;
26     double da0 = (a0_max - a0_min)/N_lattice_params;
27
28
29     double (*pos)[3] = malloc(sizeof(double[N_atoms][3]));
30     double *energy = malloc(sizeof(double[N_lattice_params]));
31
32
33     FILE *file_pointer;
34
35
36     /* ----- TASK 1 ----- */
37
38     for (int i=0; i<N_lattice_params; i++){
39         a0 = a0_min + i*da0;
40         init_fcc(pos, N_cells, a0);
41         // energy per unit cell
42         energy[i] = get_energy_AL(pos, N_cells*a0, N_atoms )*4/N_atoms;
43     }
44
45     file_pointer = fopen("../data/lattice_energies.tsv", "w");
46     for (int i=0; i<N_lattice_params; i++){
47         a0 = a0_min + i*da0;
48         fprintf(file_pointer, "%.8f \t %.8f \n", a0, energy[i]);
49     }
50     fclose(file_pointer);
51
52
53
54
55     free(pos); pos = NULL;
56     free(energy); energy = NULL;
57     return 0;
58 }
```

A.2 Main program Task 2: main_T2.c

```
1  /*
2  MD_main.c
3
4  Created by Anders Lindman on 2013-10-31.
5  */
6
7  #include <stdio.h>
8  #include <math.h>
9  #include <stdlib.h>
10 #include <time.h>
11
```

```

12 #include "initfcc.h"
13 #include "alpotential.h"
14 #include "funcs.h"
15
16 #define N_cells 4
17 #define AMU 1.0364e-4
18 #define kB 8.6173303e-5
19
20 /* Main program */
21 int main()
22 {
23
24     int N_atoms = 4*N_cells*N_cells*N_cells;
25     double m_Al = 27*AMU;
26
27     double a_eq = 4.03;
28
29     double noise_amplitude = 6.5e-2 * a_eq;
30     double t_max=10;
31     double dt = 1e-3;
32     int N_timesteps = t_max/dt;
33     double t, E_kin;
34
35     double (*pos)[3] = malloc(sizeof(double[N_atoms][3]));
36     double (*momentum)[3] = malloc(sizeof(double[N_atoms][3]));
37     double (*forces)[3] = malloc(sizeof(double[N_atoms][3]));
38     double *temperature = malloc(sizeof(double[N_timesteps]));
39     double *E_tot = malloc(sizeof(double[N_timesteps]));
40
41     FILE *file_pointer;
42
43
44     /* ----- TASK 2 ----- */
45
46     init_fcc(pos, N_cells, a_eq); // initialize fcc lattice
47     add_noise( N_atoms, 3, pos, noise_amplitude ); // adds random noise to pos
48     set_zero( N_atoms, 3, momentum); // set momentum to 0
49     get_forces_AL( forces, pos, a_eq*N_cells, N_atoms); //initial cond forces
50
51     for (int i=0; i<N_timesteps; i++){
52         /*
53          The loop over the timesteps first takes a timestep according to the
54          Verlet algorithm, then calculates the energies and temeperature.
55          */
56         timestep_Verlet (N_atoms, pos, momentum, forces, m_Al, dt, a_eq*N_cells);
57
58         E_kin =get_kin_energy(N_atoms, momentum, m_Al );
59         E_tot[i] =E_kin + get_energy_AL(pos, a_eq*N_cells, N_atoms);
60
61         /*  $3N \cdot kB \cdot T/2 = 1/(2m) \cdot \sum_{i=1}^N p_i^2 = p_{sq}/(2m)$  */
62         temperature[i] = E_kin * 2/(3*N_atoms*kB);
63     }
64
65     /* Write tempertaure to file */
66     char file_name[100];
67     sprintf(file_name, "../data/temperature_dt-%0.0e_Task2.tsv", dt);
68     file_pointer = fopen(file_name, "w");
69     for (int i=0; i<N_timesteps; i++){
70         t = i*dt; // time at step i
71         fprintf(file_pointer, "%.4f \t %.8f \n", t, temperature[i]);
72     }
73     fclose(file_pointer);
74
75     /* Write total energy to file */
76     sprintf(file_name, "../data/total_energy_dt-%0.0e_Task2.tsv", dt);
77     file_pointer = fopen(file_name, "w");
78     for (int i=0; i<N_timesteps; i++){
79         t = i*dt; // time at step i
80         fprintf(file_pointer, "%.4f \t %.8f \n", t, E_tot[i]);
81     }
82     fclose(file_pointer);
83
84     free(pos); pos = NULL;
85     free(momentum); momentum = NULL;
86     free(forces); forces = NULL;
87     free(temperature); temperature = NULL;
88     free(E_tot); E_tot = NULL;
89     return 0;
90 }

```

A.3 Temperature and pressure equilibration for tasks 3-7 : main.T3.c

```

1 /*
2 MD_main.c
3
4 Created by Anders Lindman on 2013-10-31.

```

```

5  */
6
7  #include <stdio.h>
8  #include <math.h>
9  #include <stdlib.h>
10 #include <time.h>
11
12 #include "initfcc.h"
13 #include "alpotential.h"
14 #include "funcs.h"
15
16 #define N_cells 4
17 /* define constants in atomic units: eV, , ps, K */
18 #define AMU 1.0364e-4
19 #define degC_to_K 273.15
20 #define bar 6.2415e-07
21 #define kB 8.61733e-5
22
23 /* Main program */
24 int main()
25 {
26     char file_name[100];
27
28     int N_atoms = 4*N_cells*N_cells*N_cells;
29     double m_Al = 27*AMU;
30     /*
31      Values of Young's and shear modulus, Y and G resp., taken from
32      Physics Handbook, table T 1.1. Bulk modulus then calculated as
33      B = Y*G / (9*G - 3*Y) [F 1.15, Physics Handbook]
34      kappa = 1/B
35      */
36     double kappa_Al = 100/(6.6444e+05 * bar); // STRANGE FACTOR 100 OFF !!!
37     double a_eq = 4.03;
38     double cell_length = a_eq*N_cells;
39     double inv_volume = pow(N_cells*cell_length, -3);
40     double noise_amplitude = 6.5e-2 * a_eq;
41
42     double T_final_C= 700;
43     int nRuns = 2; //2 if melt, 1 otherwise
44     double T_melt_C = 900;
45
46     double P_final_bar= 1;
47
48     double T_eq;
49     double P_eq = P_final_bar*bar;
50     double dt = 5e-3;
51     double tau_T = 100*dt;
52     double tau_P = 100*dt;
53     //double t_Teq= 10*tau_T; //equilibration times
54     double t_eq= 15*tau_P; //equilibration times
55     int N_timesteps = t_eq/dt;
56
57     double alpha_T, alpha_P,alpha_P_cube_root;
58     double t, E_kin, virial;
59
60
61     double (*pos)[3] = malloc(sizeof(double[N_atoms][3]));
62     double (*momentum)[3] = malloc(sizeof(double[N_atoms][3]));
63     double (*forces)[3] = malloc(sizeof(double[N_atoms][3]));
64     double *temperature = malloc(sizeof(double[N_timesteps]));
65     double *pressure = malloc(sizeof(double[N_timesteps]));
66
67
68     FILE *file_pointer;
69
70     /* ----- TASK 3 ----- */
71
72
73     init_fcc(pos, N_cells, a_eq); // initialize fcc lattice
74     add_noise( N_atoms, 3, pos, noise_amplitude ); // add random noise to pos
75     set_zero( N_atoms, 3, momentum); // set momentum to 0
76     get_forces_AL( forces, pos, cell_length, N_atoms); //initial cond forces
77
78     /*
79     for (int i=0; i<N_timesteps_T_eq; i++){
80         //
81         The loop over the timesteps first takes a timestep according to the
82         Verlet algorithm, then calculates the energies and temeperature.
83         //
84         timestep_Verlet(N_atoms, pos, momentum, forces, m_Al, dt, cell_length);
85
86         E_kin = get_kin_energy(N_atoms, momentum, m_Al );
87         virial = get_virial_AL(pos, cell_length, N_atoms);
88
89         // PV = NkT + virial
90         pressure[i] = inv_volume * (1.5*E_kin + virial);
91         // 3N*kB*T/2 = 1/(2m) * \sum_{i=1}^N p_i^2 = p_sq/(2m)
92         temperature[i] = E_kin * 1/(1.5*N_atoms*kB);
93
94
95         alpha_T = 1 + 2*dt*(T_eq - temperature[i]) / (tau_T * temperature[i]);

```

```

96     scale_mat(N_atoms, 3, momentum, sqrt(alpha_T));
97     temperature[i]*=alpha_T;
98 }
99 */
100
101
102 for (int irun=0; irun < nRuns; irun++){// last run: final, irun = 0
103     if (irun == nRuns - 1){ // final run
104         T_eq = T_final_C + degC_to_K;
105     }else{
106         T_eq = T_melt_C + degC_to_K;
107     }
108     for (int i=0; i<N_timesteps; i++){
109         /*
110          The loop over the timesteps first takes a timestep according to the
111          Verlet algorithm, then calculates the energies and temeperature.
112          */
113         timestep_Verlet(N_atoms, pos, momentum, forces, m_Al, dt, cell_length);
114
115
116         E_kin = get_kin_energy(N_atoms, momentum, m_Al );
117         virial = get_virial_AL(pos, cell_length, N_atoms);
118
119         /* 3N*kB*T/2 = 1/(2m) * \sum_{i=1}^N p_i^2 = p_sq/(2m) */
120         temperature[i] = E_kin * 1/(1.5*N_atoms*kB);
121         /* PV = NkT + virial */
122         pressure[i] = inv_volume * (1.5*E_kin + virial);
123
124         /* Equilibrate temperature by scaling momentum by a factor sqrt(alpha_T).
125            N.B. It is equally valid to scale the momentum instead of the velocity←
            since they only differ by a constant factor m.
            */
126         alpha_T = 1 + 2*dt*(T_eq - temperature[i]) / (tau_T * temperature[i]);
127         scale_mat(N_atoms, 3, momentum, sqrt(alpha_T));
128
129         // Equilibrate pressure by scaling the posistions by a factor of alpha_P←
130         ^{(1/3)}
131
132         alpha_P = 1 - kappa_Al* dt*(P_eq - pressure[i])/tau_P;
133         alpha_P_cube_root = pow(alpha_P, 1.0/3.0);
134         scale_mat(N_atoms, 3, pos, alpha_P_cube_root);
135
136         cell_length*=alpha_P_cube_root;
137         inv_volume*=1/alpha_P;
138
139         temperature[i]*=alpha_T;
140         pressure[i]*=alpha_P;
141     }
142 }
143
144
145
146 /* Write tempertaure to file */
147 sprintf(file_name, "../data/temp-%d_pres-%d_Task3.tsv",
148         (int) T_final_C, (int) P_final_bar);
149 file_pointer = fopen(file_name, "w");
150 for (int i=0; i<N_timesteps; i++){
151     t = i*dt; // time at step i
152     fprintf(file_pointer, "%.4f \t %.8f \t %.8f \n",
153             t, temperature[i], pressure[i]);
154 }
155 fclose(file_pointer);
156
157 /* Write phase space coordinates to file */
158 sprintf(file_name, "../data/phase-space-temp-%d_pres-%d.tsv",
159         (int) T_final_C, (int) P_final_bar);
160 file_pointer = fopen(file_name, "w");
161 for (int i=0; i<N_atoms; i++){
162     for (int j=0; j<3; j++){
163         fprintf(file_pointer, " %.16e \t", pos[i][j]);
164     }
165     for (int j=0; j<3; j++){
166         fprintf(file_pointer, " %.16e \t", momentum[i][j]);
167     }
168     fprintf(file_pointer, "\n");
169 }
170 fclose(file_pointer);
171
172 /* save equilibrated position and momentum as a binary file */
173 sprintf(file_name, "../data/INIDATA-temp-%d_pres-%d.bin",
174         (int) T_final_C, (int) P_final_bar);
175 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);
178 fwrite(&cell_length, sizeof(double), 1, file_pointer);
179 fclose(file_pointer);
180
181
182 /*
183 printf("T=%0.2f\tP=%0.2e\n",
184         temperature[N_timesteps-1], pressure[N_timesteps-1]);

```



```

185  */
186
187  free(pos); pos = NULL;
188  free(momentum); momentum = NULL;
189  free(forces); forces = NULL;
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

```

1  /*
2  MD_main.c
3
4  Created by Anders Lindman on 2013-10-31.
5  */
6
7  #include <stdio.h>
8  #include <math.h>
9  #include <stdlib.h>
10 #include <time.h>
11
12 #include "initfcc.h"
13 #include "alpotential.h"
14 #include "funcs.h"
15
16 #define N_cells 4
17 /* define constants in atomic units: eV, , ps, K */
18 #define AMU 1.0364e-4
19 #define degC_to_K 273.15
20 #define bar 6.2415e-07
21 #define kB 8.61733e-5
22
23 /* Main program */
24 int main()
25 {
26     char file_name[100];
27
28     int N_atoms = 4*N_cells*N_cells*N_cells;
29     double m_Al = 27*AMU;
30     /*
31      Values of Young's and shear modulus, Y and G resp., taken from
32      Physics Handbook, table T 1.1. Bulk modulus then calculated as
33      B = Y*G / (9*G - 3*Y) [F 1.15, Physics Handbook]
34      kappa = 1/B
35      */
36     // double kappa_Al = 100/(6.6444e+05 * bar); // STRANGE FACTOR 100 OFF !!!
37     double cell_length = 0;
38     double inv_volume;
39
40
41     double T_eq_C = 500;
42     double P_eq_bar = 1;
43     // double T_eq = T_eq_C + degC_to_K;
44     // double P_eq = P_eq_bar*bar;
45     double dt = 5e-3;
46     double t_end = 30;
47     // double tau_T = 100*dt;
48     // 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
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]));
60     double (*pos_0)[3] = malloc(sizeof(double[N_atoms][3]));
61     double (*momentum)[3] = malloc(sizeof(double[N_atoms][3]));
62     double (*forces)[3] = malloc(sizeof(double[N_atoms][3]));
63     double (*displacements)[N_save_atoms] =
64         malloc(sizeof(double[N_save_timesteps][N_save_atoms]));
65     double (*pos_all)[N_atoms][3] =
66         malloc(sizeof(double[N_save_timesteps][N_atoms][3]));
67     double (*vel_all)[N_atoms][3] =
68         malloc(sizeof(double[N_save_timesteps][N_atoms][3]));
69     double *temperature = malloc(sizeof(double[N_timesteps]));
70     double *pressure = malloc(sizeof(double[N_timesteps]));
71     double *msd = malloc(sizeof(double[N_save_timesteps]));
72     double *vel_corr = malloc(sizeof(double[N_save_timesteps]));
73

```

```

74 for (int i = 0; i<N_save_timesteps; i++){
75     msd[i] = 0;
76 }
77 FILE *file_pointer;
78
79 /* ----- TASK 3 ----- */
80
81 // read positions, momenta and cell_length
82 sprintf(file_name, "../data/INIDATA_temp-%d_pres-%d.bin",
83         (int) T_eq_C, (int) P_eq_bar);
84 file_pointer = fopen(file_name, "rb");
85 fread(pos, sizeof(double), 3*N_atoms, file_pointer);
86 fread(momentum, sizeof(double), 3*N_atoms, file_pointer);
87 fread(&cell_length, sizeof(double), 1, file_pointer);
88 fclose(file_pointer);
89
90 for (int i=0; i<N_atoms; i++){
91     for (int j=0; j<3; j++){
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++){
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
111     /* PV = NkT + virial */
112     pressure[i] = inv_volume * (1.5*E_kin + virial);
113     /*  $3N*k_B*T/2 = 1/(2m) * \sum_{i=1}^N p_i^2 = p_{sq}/(2m)$  */
114     temperature[i] = E_kin * 1/(1.5*N_atoms*kB);
115
116     if (i % N_between_steps == 0){
117         int k = i/N_between_steps; // number of saved timesteps so far
118         get_displacements (N_save_atoms, pos, pos_0, displacements[k]);
119         copy_mat(N_atoms, 3, pos, pos_all[k]);
120
121         copy_mat(N_atoms, 3, momentum, vel_all[k]);
122         scale_mat(N_atoms, 3, vel_all[k], 1/m_AL);
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 sprintf(file_name, "../data/temp-%d_pres-%d_Prod-test.tsv",
133         (int) T_eq_C, (int) P_eq_bar);
134 file_pointer = fopen(file_name, "w");
135 for (int i=0; i<N_timesteps; i++){
136     t = i*dt; // time at step i
137     fprintf(file_pointer, "%.4f \t %.8f \t %.8f \n",
138             t, temperature[i], pressure[i]);
139 }
140 fclose(file_pointer);
141
142 /* Write displacements to file */
143 sprintf(file_name, "../data/temp-%d_pres-%d_displacements.tsv",
144         (int) T_eq_C, (int) P_eq_bar);
145 file_pointer = fopen(file_name, "w");
146 for (int i=0; i<N_save_timesteps; i++){
147     t = i*dt*N_between_steps; // time at step i
148     fprintf(file_pointer, "%.4f", t);
149     for (int j=0; j<N_save_atoms; j++){
150         fprintf(file_pointer, "\t %.8f", displacements[i][j]);
151     }
152     fprintf(file_pointer, "\n");
153 }
154 fclose(file_pointer);
155
156 /* Write MSD to file */
157 sprintf(file_name, "../data/temp-%d_pres-%d_dynamicProperties.tsv",
158         (int) T_eq_C, (int) P_eq_bar);
159 file_pointer = fopen(file_name, "w");
160 // write header
161 fprintf(file_pointer, "%s \t [ps] \t MSD[A^2] \t vel_corr [A/ps]^2 \n");
162 for (int i=0; i<N_save_timesteps; i++){
163     t = i*dt*N_between_steps; // time at step i
164     fprintf(file_pointer, "%.4f \t %.8f \t %.8f \n", t, msd[i], vel_corr[i]);

```

```

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

```

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

```

1  /*
2  MD_main.c
3
4  Created by Anders Lindman on 2013-10-31.
5  */
6
7  #include <stdio.h>
8  #include <math.h>
9  #include <stdlib.h>
10 #include <time.h>
11
12 #include "initfcc.h"
13 #include "alpotential.h"
14 #include "funcs.h"
15
16 #define N_cells 4
17 /* define constants in atomic units: eV, , ps, K */
18 #define AMU 1.0364e-4
19 #define degC_to_K 273.15
20 #define bar 6.2415e-07
21 #define kB 8.61733e-5
22
23 /* Main program */
24 int main()
25 {
26     char file_name[100];
27
28     int N_atoms = 4*N_cells*N_cells*N_cells;
29     double m_Al = 27*AMU;
30     /*
31      Values of Young's and shear modulus, Y and G resp., taken from
32      Physics Handbook, table T 1.1. Bulk modulus then calculated as
33      B = Y*G / (9*G - 3*Y) [F 1.15, Physics Handbook]
34      kappa = 1/B
35      */
36     // double kappa_Al = 100/(6.6444e+05 * bar); // STRANGE FACTOR 100 OFF !!!
37     double cell_length = 0;
38     double inv_volume;
39
40
41     double T_eq_C = 500;
42     double P_eq_bar = 1;
43     // double T_eq = T_eq_C + degC_to_K;
44     // double P_eq = P_eq_bar*bar;
45     double dt = 5e-3;
46     double t_end = 30;
47     // double tau_T = 100*dt;
48     // 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
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]));
60     double (*pos_0)[3] = malloc(sizeof(double[N_atoms][3]));
61     double (*momentum)[3] = malloc(sizeof(double[N_atoms][3]));
62     double (*forces)[3] = malloc(sizeof(double[N_atoms][3]));
63     double (*displacements)[N_save_atoms] =
64         malloc(sizeof(double[N_save_timesteps][N_save_atoms]));
65     double (*pos_all)[N_atoms][3] =
66         malloc(sizeof(double[N_save_timesteps][N_atoms][3]));
67     double (*vel_all)[N_atoms][3] =
68         malloc(sizeof(double[N_save_timesteps][N_atoms][3]));
69     double *temperature = malloc(sizeof(double[N_timesteps]));

```

```

70 double *pressure      = malloc(sizeof(double[N_timesteps]));
71 double *msd           = malloc(sizeof(double[N_save_timesteps]));
72 double *vel_corr      = malloc(sizeof(double[N_save_timesteps]));
73
74 for (int i = 0; i<N_save_timesteps; i++){
75     msd[i] = 0;
76 }
77 FILE *file_pointer;
78
79 /* ----- TASK 3 ----- */
80
81 // read positions, momenta and cell_length
82 sprintf(file_name, "../data/INIDATA_temp-%d_pres-%d.bin",
83         (int) T_eq_C, (int) P_eq_bar);
84 file_pointer = fopen(file_name, "rb");
85 fread(pos, sizeof(double), 3*N_atoms, file_pointer);
86 fread(momentum, sizeof(double), 3*N_atoms, file_pointer);
87 fread(&cell_length, sizeof(double), 1, file_pointer);
88 fclose(file_pointer);
89
90 for (int i=0; i<N_atoms; i++){
91     for (int j=0; j<3; j++){
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++){
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
111     /* PV = NkT + virial */
112     pressure[i] = inv_volume * (1.5*E_kin + virial);
113     /* 3N*kB*T/2 = 1/(2m) * \sum_{i=1}^N p_i^2 = p_sq/(2m) */
114     temperature[i] = E_kin * 1/(1.5*N_atoms*kB);
115
116     if (i % N_between_steps == 0){
117         int k = i/N_between_steps; // number of saved timesteps so far
118         get_displacements (N_save_atoms, pos, pos_0, displacements[k]);
119         copy_mat(N_atoms, 3, pos, pos_all[k]);
120
121         copy_mat(N_atoms, 3, momentum, vel_all[k]);
122         scale_mat(N_atoms, 3, vel_all[k], 1/m_Al);
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 sprintf(file_name, "../data/temp-%d_pres-%d_Prod-test.tsv",
133         (int) T_eq_C, (int) P_eq_bar);
134 file_pointer = fopen(file_name, "w");
135 for (int i=0; i<N_timesteps; i++){
136     t = i*dt; // time at step i
137     fprintf(file_pointer, "%.4f \t %.8f \t %.8f \n",
138             t, temperature[i], pressure[i]);
139 }
140 fclose(file_pointer);
141
142 /* Write displacements to file */
143 sprintf(file_name, "../data/temp-%d_pres-%d_displacements.tsv",
144         (int) T_eq_C, (int) P_eq_bar);
145 file_pointer = fopen(file_name, "w");
146 for (int i=0; i<N_save_timesteps; i++){
147     t = i*dt*N_between_steps; // time at step i
148     fprintf(file_pointer, "%.4f", t);
149     for (int j=0; j<N_save_atoms; j++){
150         fprintf(file_pointer, "\t %.8f", displacements[i][j]);
151     }
152     fprintf(file_pointer, "\n");
153 }
154 fclose(file_pointer);
155
156 /* Write MSD to file */
157 sprintf(file_name, "../data/temp-%d_pres-%d_dynamicProperties.tsv",
158         (int) T_eq_C, (int) P_eq_bar);
159 file_pointer = fopen(file_name, "w");
160 // write header

```

```

161 fprintf(file_pointer, "%% t[ps] \t MSD[A^2] \t vel_corr [A/ps]^2 \n");
162 for (int i=0; i<N_save_timesteps; i++){
163     t = i*dt*N_between_steps; // time at step i
164     fprintf(file_pointer, "%.4f \t %.8f \t %.8f \n", t, msd[i], vel_corr[i]);
165 }
166 fclose(file_pointer);
167
168 free(pos);          pos = NULL;
169 free(pos_0);        pos_0 = NULL;
170 free(momentum);     momentum = NULL;
171 free(forces);        forces = NULL;
172 free(temperature);  temperature = NULL;
173 free(pressure);     pressure = NULL;
174 free(displacements); displacements = NULL;
175 free(pos_all);      pos_all = NULL;
176 free(vel_all);      vel_all = NULL;
177 return 0;
178 }

```

A.6 Misc functions : funcs.c

```

1  #include "funcs.h"
2
3  void add_noise(int M, int N, double mat[M][N], double noise_amplitude )
4  {
5      const gsl_rng_type *T; /* static info about rngs */
6      gsl_rng *q; /* rng instance */
7      gsl_rng_env_setup (); /* setup the rngs */
8      T = gsl_rng_default; /* specify default rng */
9      q = gsl_rng_alloc(T); /* allocate default rng */
10     gsl_rng_set(q, time(NULL)); /* Initialize rng */
11
12     for (int i=0; i<N; i++){
13         for (int j=0; j<M; j++){
14             // adds uniformly distributed random noise in range +/-`noise_amplitude`
15             mat[i][j] += noise_amplitude * (2*gsl_rng_uniform(q)-1);
16         }
17     }
18     gsl_rng_free(q); /* deallocate rng */
19 }
20
21 void timestep_Verlet ( int N_atoms, double (*pos)[3], double (*momentum)[3],
22                       double (*forces)[3], double m, double dt,
23                       double cell_length){
24     for (int i = 0; i < N_atoms; i++) {
25         for (int j = 0; j < 3; j++) {
26             /* p(t+dt/2) */
27             momentum[i][j] += dt * 0.5 * forces[i][j];
28             /* q(t+dt) */
29             pos[i][j] += dt * momentum[i][j] / m;
30         }
31     }
32     /* F(t+dt) */
33     get_forces_AL( forces, pos, cell_length, N_atoms);
34     for (int i = 0; i < N_atoms; i++) {
35         for (int j = 0; j < 3; j++) {
36             /* p(t+dt/2) */
37             momentum[i][j] += dt * 0.5 * forces[i][j];
38         }
39     }
40 }
41
42 double get_kin_energy ( int N_atoms, double (*momentum)[3], double m ) {
43     double p_sq=0; // momentum squared
44     for (int i = 0; i < N_atoms; i++) {
45         for (int j = 0; j < 3; j++) {
46             p_sq += momentum[i][j] * momentum[i][j];
47         }
48     }
49     return p_sq / (2*m);
50 }
51
52 void get_displacements ( int N_atoms, double (*positions)[3],
53                         double (*initial_positions)[3], double disp[]) {
54     for (int i = 0; i < N_atoms; i++) {
55         for (int j = 0; j < 3; j++) {
56             disp[i] += (positions[i][j] - initial_positions[i][j])
57                 *(positions[i][j] - initial_positions[i][j]);
58         }
59         disp[i] = sqrt(disp[i]);
60     }
61 }
62
63
64 void get_MSD ( int N_atoms, int N_times, double all_pos[N_times][N_atoms][3],
65               double MSD[N_times]) {

```

```

66  /* all_pos = positions of all particles at all (saved) times */
67  /* outer time index it starts at outer it = 1, since MSD[0] = 0 */
68  for (int it = 1; it < N_times; it++) { //
69      for (int jt = 0; jt < N_times-it; jt++) { // summed time index
70          for (int kn = 0; kn < N_atoms; kn++) { // particle index
71              for (int kd = 0; kd < 3; kd++) { // three dimensions
72                  MSD[it] += (all_pos[it+jt][kn][kd] - all_pos[jt][kn][kd])
73                      *(all_pos[it+jt][kn][kd] - all_pos[jt][kn][kd]);
74              }
75          }
76      }
77      MSD[it] *= 1/( (double)N_atoms * (N_times-it));
78  }
79  }
80
81  void get_vel_corr ( int N_atoms, int N_times, double all_vel[N_times][N_atoms]←
82      [3],
83      double vel_corr[N_times]) {
84      /* all_vel = velocity of all particles at all (saved) times */
85      for (int it = 0; it < N_times; it++) { //
86          for (int jt = 0; jt < N_times-it; jt++) { // summed time index
87              for (int kn = 0; kn < N_atoms; kn++) { // particle index
88                  for (int kd = 0; kd < 3; kd++) { // three dimensions
89                      vel_corr[it] += (all_vel[it+jt][kn][kd] * all_vel[jt][kn][kd]);
90                  }
91              }
92          }
93      }
94      vel_corr[it] *= 1/( (double)N_atoms * (N_times-it));
95  }
96
97  void copy_mat (int M, int N, double mat_from[M][N], double mat_to[M][N]){
98      /* Copies matrix `mat_from` to `mat_to` */
99      for (int i = 0; i < M; i++) {
100          for (int j = 0; j < N; j++) {
101              mat_to[i][j] = mat_from[i][j];
102          }
103      }
104  }
105
106  void set_zero (int M, int N, double mat[M][N]){
107      /* Sets the matrix `mat` to zero */
108      for (int i = 0; i < M; i++) {
109          for (int j = 0; j < N; j++) {
110              mat[i][j] = 0;
111          }
112      }
113  }
114
115  void scale_mat (int M, int N, double mat[M][N], double alpha){
116      /* Scales the matrix `mat` by factor `alpha` */
117      for (int i = 0; i < M; i++) {
118          for (int j = 0; j < N; j++) {
119              mat[i][j] *= alpha;
120          }
121      }
122  }

```

B Auxiliary

B.1 Makefile

```

1
2  CC = gcc
3  CFLAGS = -O3 -Wall
4  LIBS = -lm -lgsl -lgslcblas
5
6  HEADERS = initfcc.h alpotential.h funcs.h
7  OBJECTS = initfcc.o alpotential.o funcs.o
8
9
10 %.o: %.c $(HEADERS)
11     $(CC) -c -o $@ $(CFLAGS)
12
13 all: Task1 Task2 Task3 main_Prod.c
14
15 Task1: $(OBJECTS) main_T1.c
16     $(CC) -o $@ $(CFLAGS) $(LIBS)
17
18 Task2: $(OBJECTS) main_T2.c
19     $(CC) -o $@ $(CFLAGS) $(LIBS)
20
21 Task3: $(OBJECTS) main_T3.c

```

```

22 $(CC) -o $@ $^ $(CFLAGS) $(LIBS)
23
24 Prod: $(OBJECTS) main_Prod.c
25 $(CC) -o $@ $^ $(CFLAGS) $(LIBS)
26
27 # $(PROGRAMS): $(OBJECTS) main_T1.c
28 # $(CC) -o $@ $^ $(CFLAGS) $(LIBS)
29
30 clean:
31 rm -f *.o
32 touch *.c

```

C Matlab scripts

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

```

1 tmp = matlab.desktop.editor.getActive;
2 cd(fileparts(tmp.Filename));
3 set(0,'DefaultFigureWindowStyle','docked');
4 GRAY = 0.7*[0.9 0.9 1];
5
6 %% task 1
7 clc
8
9 energy_data = load('../data/lattice_energies.tsv');
10 a0 = energy_data(:,1);
11 v0 = a0.^3;
12
13 energy = energy_data(:,2);
14 figure(1);clf;
15 plot(v0,energy, 'xk');
16
17 start_v = 64;
18 end_v = 68;
19 indToInclude = (v0 > start_v) & (v0 < end_v);
20 p = polyfit(v0(indToInclude),energy(indToInclude),2);
21 hold on;
22
23 vvec = linspace(start_v, end_v);
24 plot(vvec, p(1)*vvec.^2 + p(2)*vvec + p(3), '-r');
25 xlim([64 68]);
26
27 v_min = -p(2)/(2*p(1));
28 a_min = v_min^(1/3);
29
30 ax = gca;
31 ax.YLim = [-13.45 -13.42];
32 h1 = plot(v_min*[1 1], ax.YLim, '--k'); % plot vertical line at v_min
33
34
35 ax.YTick = (-13.45:0.01:-13.42);
36 ylabel('$E_{\rm pot}$ [eV/unit cell]');
37 xlabel('$a_0^3$ [\AA^3]');
38 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]);
42 saveas(gcf, '../figures/potential_energy.eps', 'epsc')
43
44
45
46 %% task 2
47 %clc;
48 clf;clear
49
50 dt=[1e-2,5e-3,2e-3,1e-3];
51 for i=1:4
52 T_data = load(sprintf('../data/temperature_dt-%0.0e_Task2.tsv',dt(i)));
53 E_data = load(sprintf('../data/total_energy_dt-%0.0e_Task2.tsv',dt(i)));
54 t = T_data(:,1);
55 T = T_data(:,2);
56 E = E_data(:,2);
57
58 t_eq=0.5;
59
60
61 fprintf('dt = %0.0e\n',dt(i));
62
63 T_avg=mean(T(t>t_eq));
64 T_std=std(T(t>t_eq));
65 fprintf('\tT = %0.2f +- %0.1f %%\n', T_avg, abs(T_std/T_avg)*100);
66
67 E_avg=mean(E(t>t_eq));

```

```

68 E_std=std(E(t>t_eq));
69 fprintf('\tE = %0.2f +- %0.1e %%\n', E_avg, abs(E_std/E_avg)*100);
70
71 figure(i);clf
72 plot(t, T)
73 yyaxis right
74 plot(t, E)
75 ylim(E_avg*(1+0.001*[1,-1]));
76 end
77
78
79 %% test production pressure and temp
80 clc; clf;
81 %clear
82
83
84
85 %data = load(sprintf('../data/temperature_dt-1e-02_Task3.tsv'));
86 data = load('../data/temp-700_pres-1_Task3.tsv');
87 %data = load('../data/temp-700_pres-1_Prod-test.tsv');
88
89 bar = 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));
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
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
110 plot(t,T, 'color', GRAY),hold on
111 plot(t, cumsum(T)./(1:length(t))', '-k')
112 ylabel('$T \backslash, [\circ \rm C]$', 'r')
113
114 ylim([600,800])
115
116
117 yyaxis right
118 plot(t,P),hold on
119 plot(t,cumsum(P)./(1:length(t))', '-k')
120
121
122 ylabel('$P \backslash, [\rm bar]$', 'r')
123 ylim([-50,200])
124
125 xlabel('$t$', [ps])
126
127 ImproveFigureCompPhys(gcf, 'linewidth', 3, 'LineColor', {'MYORANGE', GRAY, 'MYBLUE', GRAY});
128
129 %% determine displacements and MSD
130
131 clc; clf;
132 figure(10); clf;
133 FILENAMES = strcat({'../data/temp-'}, num2str([500;700]), '_pres-1_displacements←
134 .tsv');
135 FILENAMES_Dyn = strcat({'../data/temp-'}, num2str([500;700]), '_pres-1←
136 _dynamicProperties.tsv');
137 for iFile = 1:numel(FILENAMES)
138     figure(iFile); clf;
139     data = load(FILENAMES{iFile});
140     t = data(:,1);
141     dx = data(:,2:end);
142
143     plot(t, dx.^2); hold on;
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;
152         D = MSD(t>tStart)./(6*(t>tStart));
153         selfDiffusionCoeff = mean(D); % in ^2 /ps
154         plot(t, 6*t*selfDiffusionCoeff, ':');
155     end

```



```

156     leg = legend( strcat({'$n=$'}, num2str((1:size(dx,2))'))');
157     leg.Location='northwest';
158     xlabel('$t$ [ps]')
159     ylabel('$\Delta x^2 \backslash, [\rm \AA^2] $')
160     if iFile ==1
161         ylim([ 0 1.0]);
162     else
163         ylim([0 200]);
164     end
165     ImproveFigureCompPhys(gcf);
166
167     figure(10)
168     plot(t, vel_corr/vel_corr(1)); hold on;
169     xlim([0 1])
170
171 end
172
173
174 figure(10)
175 for iFile = 1:numel(FILENAMES)
176     data = load(FILENAMES_Dyn{iFile});
177     t = data(:,1);
178     vel_corr = data(:,3);
179     plot(t, vel_corr/vel_corr(1)); hold on;
180
181 end
182
183 xlim([0 1])
184 leg = legend(strcat({'$T=$'}, num2str([500;700]), '\,`circ $C'));
185 leg.Location='northeast';
186 xlabel('$t$ [ps]')
187 ylabel('$\Phi(t)/\Phi(0)$')
188 ImproveFigureCompPhys();
189 %%
190 clc;clf;
191
192
193 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;
199 m_A1 = 27*AMU;
200 kB= 8.61733e-5;
201 N_atoms=4^4;
202
203 T=sum(sum(data1.^2,2),1) / (3*m_A1*N_atoms*kB)
204
205
206 data2=load('../data/phase-space-temp-500_pres-1.tsv');
207
208 T=sum(sum(data2(:,4:end).^2,2),1) / (3*m_A1*N_atoms*kB)

```

C.2 Improve figure appearance: ImproveFigureCompPhys.m

```

1 function ImproveFigureCompPhys(varargin)
2 %ImproveFigureCompPhys Improves the figures of supplied handles
3 % Input:
4 % - none (improve all figures) or handles to figures to improve
5 % - optional:
6 %     LineWidth int
7 %     LineStyle column vector cell, e.g. {'-','--'}',
8 %     LineColor column vector cell, e.g. {'k',[0 1 1], 'MYBLUE'}'
9 %             colors: MYBLUE,MYORANGE,MYGREEN,MYPURPLE, MYYELLOW,
10 %             MYLIGHTBLUE, MYRED
11 %     Marker column vector cell, e.g. {'.', 'o', 'x'}'
12
13 % ImproveFigure was originally written by Adam Stahl, but has been heavily
14 % modified by Linnea Hesslow
15
16
17 %%% Handle inputs
18 % If no inputs or if the first argument is a string (a property rather than
19 % a handle), use all open figures
20 if nargin == 0 || ischar(varargin{1})
21     %Get all open figures
22     figHs = findobj('Type','figure');
23     nFigs = length(figHs);
24 else
25     % Check the supplied figure handles
26     figHs = varargin{1};
27     figHs = figHs(ishandle(figHs) == 1); %Keep only those handles that are ←
28     % proper graphics handles
29     nFigs = length(figHs);
30 end

```

```

30
31 % Define desired properties
32 titleSize = 24;
33 interpreter = 'latex';
34 lineWidth = 4;
35 axesWidth = 1.5;
36 labelSize = 22;
37 textSize = 20;
38 legTextSize = 18;
39 tickLabelSize = 18;
40 LineColor = {};
41 LineStyle = {};
42 Marker = {};
43
44 % define colors
45 co = [ 0      0.4470    0.7410
46       0.8500    0.3250    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 ];
52 colors = struct('MYBLUE', co(1,:),...
53 'MYORANGE', co(2,:),...
54 'MYYELLOW', co(3,:),...
55 'MYPURPLE', co(4,:),...
56 'MYGREEN', co(5,:),...
57 'MYLIGHTBLUE', co(6,:),...
58 'MYRED',co(7,:),...
59 'GERIBLUE', [0.3000    0.1500    0.7500],...
60 'GERIRED', [1.0000    0.2500    0.1500],...
61 'GERIYELLOW', [0.9000    0.7500    0.1000],...
62 'LIGHTGREEN', [0.4    0.85    0.4],...
63 'LINNEAGREEN', [7 184 4]/255);
64
65 % Loop through the supplied arguments and check for properties to set.
66 for i = 1:nargin
67     if ischar(varargin{i})
68         switch lower(varargin{i}) %Compare lower case strings
69             case 'linewidth'
70                 lineWidth = varargin{i+1};
71             case 'linestyle'
72                 LineStyle = varargin{i+1};
73             case 'linecolor'
74                 LineColor = varargin{i+1};
75                 for iLineColor = 1:numel(LineColor)
76                     if isfield(colors, LineColor{iLineColor})
77                         LineColor{iLineColor} = colors.(LineColor{iLineColor});
78                     end
79                 end
80             case 'marker'
81                 Marker = varargin{i+1};
82         end
83     end
84 end
85 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
86
87 %%% Improve the figure(s)
88
89 for iFig = 1:nFigs
90
91     fig = figHs(iFig);
92
93     lineObjects = findall(fig, 'Type', 'line');
94     textObjects = findall(fig, 'Type', 'text');
95     axesObjects = findall(fig, 'Type', 'axes');
96     legObjects = findall(fig, 'Type', 'legend');
97     contourObjects = findall(fig, 'Type', 'contour'); % not counted as lines
98
99     %%% TEXT APPEARANCE: first set all to textSize and then change the ones
100    %%% that need to be changed again
101
102    %Change size of any text objects in the plot
103    set(textObjects, 'FontSize', textSize);
104    set(legObjects, 'FontSize', legTextSize);
105
106    %%% FIX LINESTYLE, COLOR ETC. FOR EACH PLOT SEPARATELY
107    for iAx = 1:numel(axesObjects)
108        lineObjInAx = findall(axesObjects(iAx), 'Type', 'line');
109
110        %set line style and color style (only works if all figs have some
111        %number of line plots..)
112        if ~isempty(LineStyle)
113            set(lineObjInAx, {'LineStyle'}, LineStyle)
114            set(contourObjects, {'LineStyle'}, LineStyle); %%%
115        end
116        if ~isempty(LineColor)
117            set(lineObjInAx, {'Color'}, LineColor)
118            set(contourObjects, {'LineColor'}, LineColor); %%%
119        end
120        if ~isempty(Marker)

```

```

121         set(lineObjInAx, {'Marker'}, Marker)
122         set(lineObjInAx, {'Markersize'}, num2cell(10+22*strcmp(Marker, '.'))←
        )
123     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
131     axesObjects(iAx).XLabel.FontSize = labelSize;
132     axesObjects(iAx).YLabel.FontSize = labelSize;
133
134     %Change title size
135     axesObjects(iAx).Title.FontSize = titleSize;
136 end
137
138 %% LINE APPEARANCE
139 %Change line thicknesses
140 set(lineObjects, 'LineWidth', lineWidth);
141 set(contourObjects, 'LineWidth', lineWidth);
142 set(axesObjects, 'LineWidth', axesWidth)
143
144 % set interpreter: latex or tex
145 set(textObjects, 'interpreter', interpreter)
146 set(legObjects, 'Interpreter', interpreter)
147 set(axesObjects, 'TickLabelInterpreter', interpreter);
148 end
149 end

```

C.3 Change size of figures: setFigureSize.m

```

1 function [ fig ] = setFigureSize( fig )
2 %figureSizePaper1
3 fig.Units = 'points';
4 W = 600;
5 H = 300;
6
7 fig.WindowStyle = 'normal'; % undock
8 fig.Position(3:4) = [W H];
9
10 end

```