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 №	Points	Avail. points
$\sum$		

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

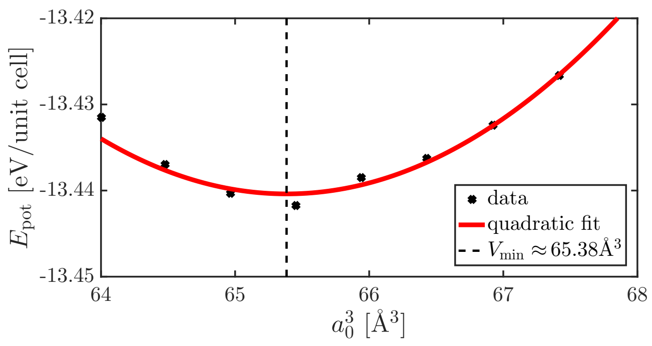

Figure 1: The potential energy per unit cell for aluminum as a function of the lattice parameter cubed.

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

Task 5

Equation (82) in MD lecture notes:

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

$$\Rightarrow$$

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

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

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

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

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

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

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

Using these two functions, we can define a power spectrum

$$\begin{aligned}\hat{P}(\omega) &= \left\langle |\hat{\boldsymbol{v}}(\omega)|^2 \right\rangle_{\mathrm{A}} = \left\langle \hat{\boldsymbol{v}}(\omega) \cdot \bar{\hat{\boldsymbol{v}}}(\omega) \right\rangle_{\mathrm{A}} \\ &= \left\langle \int_{-\infty}^{\infty} \mathrm{d}t\ \boldsymbol{v}(t)\mathrm{e}^{\mathrm{i}\omega t} \int_{-\infty}^{\infty} \mathrm{d}t'\ \boldsymbol{v}(t')\mathrm{e}^{-\mathrm{i}\omega t'} \right\rangle_{\mathrm{A}},\end{aligned} \tag{6}$$

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

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

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.*)

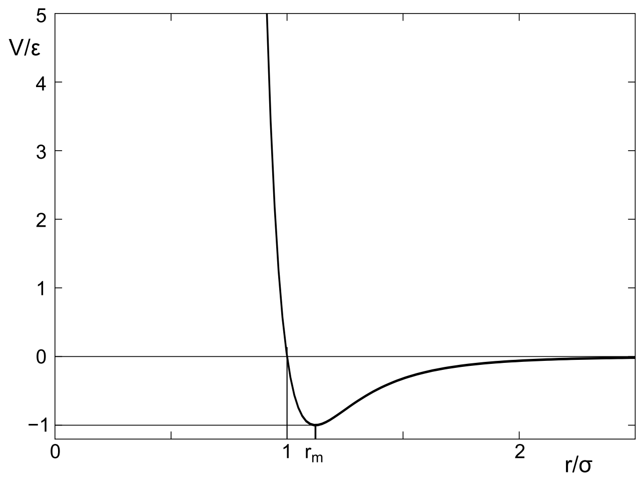

Figure 2: The Lennard–Jones potential. Make sure you label and have units on all axes! Also make sure that labels etc. are legible and that, if you print in black and white, that you use different line styles when required to differentiate between curves. In MATLAB you can export any figure to an .eps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

```c
/*
 MD_main.c

 Created by Anders Lindman on 2013-10-31.
 */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#include "initfcc.h"
#include "alpotential.h"

#define N_cells 4
#define N_lattice_params 25


/* Main program */
int main()
{

  int N_atoms = 4*N_cells*N_cells*N_cells;
  double a0;
  double a0_min = 4.0;
  double a0_max = 4.2;
  double da0 = (a0_max - a0_min)/N_lattice_params;


  double (*pos)[3] = malloc(sizeof(double[N_atoms][3]));
  double *energy = malloc(sizeof(double[N_lattice_params]));


  FILE *file_pointer;


  /* ---------------------------- TASK 1 ---------------------------------*/

  for (int i=0; i<N_lattice_params; i++){
    a0 = a0_min + i*da0;
    init_fcc(pos, N_cells, a0);
    // energy per unit cell
    energy[i] = get_energy_AL(pos, N_cells*a0, N_atoms )*4/N_atoms;
  }

  file_pointer = fopen("../data/lattice_energies.tsv", "w");
  for (int i=0; i<N_lattice_params; i++){
    a0 = a0_min + i*da0;
    fprintf(file_pointer, "%.8f \t %.8f \n", a0, energy[i]);
  }
  fclose(file_pointer);




  free(pos); pos = NULL;
  free(energy); energy = NULL;
  return 0;
}
```

A.2 Main program Task 2: `main_T2.c`

```c
/*
 MD_main.c

 Created by Anders Lindman on 2013-10-31.
 */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>

```

```c
#include "initfcc.h"
#include "alpotential.h"
#include "funcs.h"

#define N_cells 4
#define AMU 1.0364e-4
#define kB 8.6173303e-5

/* Main program */
int main()
{

  int N_atoms = 4*N_cells*N_cells*N_cells;
  double m_Al = 27*AMU;

  double a_eq = 4.03;

  double noise_amplitude = 6.5e-2 * a_eq;
  double t_max=10;
  double dt = 1e-3;
  int N_timesteps = t_max/dt;
  double t, E_kin;

  double (*pos)[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]));
  double *temperature = malloc(sizeof(double[N_timesteps]));
  double *E_tot = malloc(sizeof(double[N_timesteps]));

  FILE *file_pointer;


  /* ---------------------------- TASK 2 -----------------------------------*/

  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, a_eq*N_cells, N_atoms); //initial cond forces

  for (int i=0; i<N_timesteps; i++){
    /*
       The loop over the timesteps first takes a timestep according to the
       Verlet algorithm, then calculates the energies and temeperature.
    */
    timestep_Verlet (N_atoms, pos,  momentum, forces, m_Al, dt, a_eq*N_cells);

    E_kin    =get_kin_energy(N_atoms, momentum, m_Al );
    E_tot[i] =E_kin + get_energy_AL(pos, a_eq*N_cells, N_atoms);

    /* 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);
  }

  /* Write tempertaure to file */
  char file_name[100];
  sprintf(file_name,"../data/temperature_dt-%0.0e_Task2.tsv", dt);
  file_pointer = fopen(file_name, "w");
  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]);
  }
  fclose(file_pointer);

  /* Write total energy to file */
  sprintf(file_name,"../data/total_energy_dt-%0.0e_Task2.tsv", dt);
  file_pointer = fopen(file_name, "w");
  for (int i=0; i<N_timesteps; i++){
    t = i*dt; // time at step i
    fprintf(file_pointer, "%.4f \t %.8f \n", t, E_tot[i]);
  }
  fclose(file_pointer);

  free(pos); pos = NULL;
  free(momentum); momentum = NULL;
  free(forces); forces = NULL;
  free(temperature); temperature = NULL;
  free(E_tot); E_tot = NULL;
  return 0;
}
```

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

```c
/*
 MD_main.c

 Created by Anders Lindman on 2013-10-31.
 */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>

#include "initfcc.h"
#include "alpotential.h"
#include "funcs.h"

#define N_cells 4
/* define constants in atomic units: eV,    , ps, K */
#define AMU 1.0364e-4
#define degC_to_K 273.15
#define bar 6.2415e-07
#define kB 8.61733e-5

/* Main program */
int main()
{
   char file_name[100];

   int N_atoms = 4*N_cells*N_cells*N_cells;
   double m_Al = 27*AMU;
   /*
     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
   */
   double kappa_Al = 100/(6.6444e+05 * bar); // STRANGE FACTOR 100 OFF !!!
   double a_eq = 4.03;
   double cell_length = a_eq*N_cells;
   double inv_volume = pow(N_cells*cell_length, -3);
   double noise_amplitude = 6.5e-2 * a_eq;

   double T_final_C= 700;
   int nRuns = 2; //2 if melt, 1 otherwise
   double T_melt_C = 900;

   double P_final_bar= 1;

  double T_eq;
  double P_eq  = P_final_bar*bar;
  double dt    = 5e-3;
  double tau_T = 100*dt;
  double tau_P = 100*dt;
  //double t_T_eq= 10*tau_T; //equlibration times
  double t_eq= 15*tau_P; //equlibration times
  int N_timesteps = t_eq/dt;

  double alpha_T, alpha_P,alpha_P_cube_root;
  double t, E_kin, virial;


  double (*pos)[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]));
  double *temperature = malloc(sizeof(double[N_timesteps]));
  double *pressure = malloc(sizeof(double[N_timesteps]));


  FILE *file_pointer;

  /* ----------------------------- TASK 3 ---------------------------------*/


  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

  /*
  for (int i=0; i<N_timesteps_T_eq; i++){
    //
       The loop over the timesteps first takes a timestep according to the
       Verlet algorithm, then calculates the energies and temeperature.
    //
    timestep_Verlet(N_atoms, pos,  momentum, forces, m_Al, dt, cell_length);

    E_kin  = get_kin_energy(N_atoms, momentum, m_Al );
    virial = get_virial_AL(pos, cell_length, N_atoms);

    // 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);


    alpha_T = 1 + 2*dt*(T_eq - temperature[i]) / (tau_T * temperature[i]);
    scale_mat(N_atoms, 3, momentum, sqrt(alpha_T));
    temperature[i]*=alpha_T;
  }
  */


    for (int irun=0; irun < nRuns; irun++){// last run: final, irun = 0
       if (irun == nRuns - 1){ // final run
          T_eq = T_final_C + degC_to_K;
       }else{
          T_eq = T_melt_C + degC_to_K;
       }
        for (int i=0; i<N_timesteps; i++){
      /*
         The loop over the timesteps first takes a timestep according to the
         Verlet algorithm, then calculates the energies and temeperature.
      */
      timestep_Verlet(N_atoms, pos,  momentum, forces, m_Al, dt, cell_length);


      E_kin  = get_kin_energy(N_atoms, momentum, m_Al );
      virial = get_virial_AL(pos, cell_length, N_atoms);

      /* 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);
      /* PV = NkT + virial */
      pressure[i] = inv_volume * (1.5*E_kin + virial);

      /* 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,
          since they only differ by a constant factor m.
      */
      alpha_T = 1 + 2*dt*(T_eq - temperature[i]) / (tau_T * temperature[i]);
      scale_mat(N_atoms, 3, momentum, sqrt(alpha_T));

      // Equlibrate pressure by scaling the posistions by a factor of alpha_P^(1/3)

      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);

      cell_length*=alpha_P_cube_root;
      inv_volume*=1/alpha_P;

      temperature[i]*=alpha_T;
      pressure[i]*=alpha_P;
      }
    }


  /* Write tempertaure to file */
  sprintf(file_name,"../data/temp-%d_pres-%d_Task3.tsv",
      (int) T_final_C, (int) P_final_bar);
  file_pointer = fopen(file_name, "w");
  for (int i=0; i<N_timesteps; i++){
    t = i*dt; // time at step i
    fprintf(file_pointer, "%.4f \t %.8f \t %.8f \n",
        t, temperature[i],pressure[i]);
  }
  fclose(file_pointer);

  /* Write phase space coordinates to file */
  sprintf(file_name,"../data/phase-space_temp-%d_pres-%d.tsv",
      (int) T_final_C, (int) P_final_bar);
  file_pointer = fopen(file_name, "w");
  for (int i=0; i<N_atoms; i++){
    for (int j=0;j<3;j++){
      fprintf(file_pointer, " %.16e \t", pos[i][j]);
    }
    for (int j=0;j<3;j++){
      fprintf(file_pointer, " %.16e \t", momentum[i][j]);
    }
    fprintf(file_pointer,"\n");
  }
  fclose(file_pointer);

  /* save equlibrated position and momentum as a binary file */
  sprintf(file_name,"../data/INIDATA_temp-%d_pres-%d.bin",
      (int) T_final_C, (int) P_final_bar);
  file_pointer = fopen(file_name, "wb");
  fwrite(pos, sizeof(double), 3*N_atoms, file_pointer);
  fwrite(momentum, sizeof(double), 3*N_atoms, file_pointer);
  fwrite(&cell_length, sizeof(double), 1, file_pointer);
  fclose(file_pointer);


  /*
  printf("T=%0.2f\tP=%0.2e\n",
     temperature[N_timesteps-1],pressure[N_timesteps-1]);
```

```c
  */

  free(pos); pos = NULL;
  free(momentum); momentum = NULL;
  free(forces); forces = NULL;
  free(temperature); temperature = NULL;
  free(pressure); pressure = NULL;
  //free(volume); volume = NULL;
  return 0;
}
```

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

```c
/*
 MD_main.c

 Created by Anders Lindman on 2013-10-31.
 */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>

#include "initfcc.h"
#include "alpotential.h"
#include "funcs.h"

#define N_cells 4
/* define constants in atomic units: eV,    , ps, K */
#define AMU 1.0364e-4
#define degC_to_K 273.15
#define bar 6.2415e-07
#define kB 8.61733e-5

/* Main program */
int main()
{
  char file_name[100];

  int N_atoms = 4*N_cells*N_cells*N_cells;
  double m_Al = 27*AMU;
  /*
    Values of Young's and shear modulus, Y and G resp., taken from
    Physics Handbook, table T 1.1. Bulk modulus then calculated as
    B = Y*G / (9*G - 3*Y)   [F 1.15, Physics Handbook]
    kappa = 1/B
  */
//  double kappa_Al = 100/(6.6444e+05 * bar); // STRANGE FACTOR 100 OFF !!!
  double cell_length = 0;
  double inv_volume;


  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;
  double dt       = 5e-4; // higher res for spectral function
  double t_end    = 5;
//  double tau_T = 100*dt;
//  double tau_P = 100*dt;

  int N_timesteps = t_end/dt;

  int N_between_steps = 1;
  int N_save_timesteps = N_timesteps / N_between_steps; //for the displacements
  int N_save_atoms = 5;

//  double alpha_T, alpha_P,alpha_P_cube_root;
  double t, E_kin, virial;

  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]));
  double (*displacements)[N_save_atoms] =
          malloc(sizeof(double[N_save_timesteps][N_save_atoms]));
  double (*pos_all)[N_atoms][3] =
          malloc(sizeof(double[N_save_timesteps][N_atoms][3]));
  double (*vel_all)[N_atoms][3] =
          malloc(sizeof(double[N_save_timesteps][N_atoms][3]));
  double *temperature   = malloc(sizeof(double[N_timesteps]));
  double *pressure      = malloc(sizeof(double[N_timesteps]));
  double *msd           = malloc(sizeof(double[N_save_timesteps]));
  double *vel_corr      = malloc(sizeof(double[N_save_timesteps]));
  double *pow_spec      = malloc(sizeof(double[N_save_timesteps]));
  double *freq             = malloc(sizeof(double[N_save_timesteps]));

  for (int i = 0; i<N_save_timesteps; i++){
    msd[i] = 0;
    pow_spec[i] = 0;
    vel_corr[i] = 0;
  }
  FILE *file_pointer;

  /* ----------------------------- TASK 3 -----------------------------------*/

  // read positions, momenta and cell_length
  sprintf(file_name,"../data/INIDATA_temp-%d_pres-%d.bin",
      (int) T_eq_C, (int) P_eq_bar);
  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);
  fclose(file_pointer);

  for (int i=0; i<N_atoms; i++){
    for (int j=0; j<3; j++){
      pos_0[i][j]=pos[i][j];
    }
  }
  inv_volume = pow(N_cells*cell_length, -3);
  get_forces_AL( forces, pos, cell_length, N_atoms); //initial cond forces

  printf("Initialized. Starting with Verlet timestepping.\n");
  for (int i=0; i<N_timesteps; i++){
    /*
       The loop over the timesteps first takes a timestep according to the
       Verlet algorithm, then calculates the energies and temeperature.
    */
    timestep_Verlet(N_atoms, pos,  momentum, forces, m_Al, dt, cell_length);

    E_kin  = get_kin_energy(N_atoms, momentum, m_Al );
    virial = get_virial_AL(pos, cell_length, N_atoms);

    /* 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);

    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]);

         copy_mat(N_atoms, 3, momentum, vel_all[k]);
         scale_mat(N_atoms, 3, vel_all[k], 1/m_Al);
     }
    if ((i*10) % N_timesteps == 0){
       printf("done %d0 %% of Verlet timestepping\n", (i*10)/N_timesteps);
    }
  }
  printf("calculating MSD\n");
  get_MSD(N_atoms, N_save_timesteps, pos_all, msd);

  printf("calculating velocity correlation\n");
  get_vel_corr(N_atoms, N_save_timesteps, vel_all, vel_corr);

  printf("calculating power spectrum\n");
  get_powerspectrum(N_atoms, N_save_timesteps, vel_all, pow_spec);
  fft_freq(freq, dt, N_save_timesteps);



  printf("writing to file\n");
  /* Write tempertaure to file */

  sprintf(file_name,"../data/temp-%d_pres-%d_Prod-test.tsv",
      (int) T_eq_C, (int) P_eq_bar);
  file_pointer = fopen(file_name, "w");
  for (int i=0; i<N_timesteps; i++){
    t = i*dt; // time at step i
    fprintf(file_pointer, "%.4f \t %.8f \t %.8f \n",
        t, temperature[i],pressure[i]);
  }
  fclose(file_pointer);

  /* Write displacements to file */
  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);
    for (int j=0; j<N_save_atoms; j++){
        fprintf(file_pointer, "\t %.8f", displacements[i][j]);
     }
```

```
    fprintf(file_pointer, "\n");
  }
  fclose(file_pointer);

   /* Write MSD to file */
  sprintf(file_name,"../data/temp-%d_pres-%d_dynamicProperties.tsv",
      (int) T_eq_C, (int) P_eq_bar);
  file_pointer = fopen(file_name, "w");
  // write header
  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]);
  }
  fclose(file_pointer);

  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");
  // write header
  fprintf(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]);
  }
  fclose(file_pointer);

  free(pos);           pos = NULL;
  free(pos_0);         pos_0 = NULL;
  free(momentum);      momentum = NULL;
  free(forces);        forces = NULL;
  free(temperature);   temperature = NULL;
  free(pressure);      pressure = NULL;
  free(displacements); displacements = NULL;
  free(pos_all); pos_all = NULL;
  free(vel_all); vel_all = NULL;
  free(msd); msd = NULL;
  free(vel_corr); vel_corr = NULL;
  free(pow_spec); pow_spec = NULL;
  free(freq); freq = NULL;
  return 0;
}
```

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

```
/*
 MD_main.c

 Created by Anders Lindman on 2013-10-31.
 */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>

#include "initfcc.h"
#include "alpotential.h"
#include "funcs.h"

#define N_cells 4
/* define constants in atomic units: eV,    , ps, K */
#define AMU 1.0364e-4
#define degC_to_K 273.15
#define bar 6.2415e-07
#define kB 8.61733e-5

/* Main program */
int main()
{
  char file_name[100];

  int N_atoms = 4*N_cells*N_cells*N_cells;
  double m_Al = 27*AMU;
  /*
    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
  */
//  double kappa_Al = 100/(6.6444e+05 * bar); // STRANGE FACTOR 100 OFF !!!
  double cell_length = 0;
  double inv_volume;


  double T_eq_C   = 500;
  double P_eq_bar = 1;
```

```c
//  double T_eq     = T_eq_C + degC_to_K;
//  double P_eq     = P_eq_bar*bar;
  double dt       = 5e-4; // higher res for spectral function
  double t_end    = 5;
//  double tau_T = 100*dt;
//  double tau_P = 100*dt;

  int N_timesteps = t_end/dt;

  int N_between_steps = 1;
  int N_save_timesteps = N_timesteps / N_between_steps; //for the displacements
  int N_save_atoms = 5;

//  double alpha_T, alpha_P,alpha_P_cube_root;
  double t, E_kin, virial;

  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]));
  double (*displacements)[N_save_atoms] =
            malloc(sizeof(double[N_save_timesteps][N_save_atoms]));
  double (*pos_all)[N_atoms][3] =
            malloc(sizeof(double[N_save_timesteps][N_atoms][3]));
  double (*vel_all)[N_atoms][3] =
            malloc(sizeof(double[N_save_timesteps][N_atoms][3]));
  double *temperature   = malloc(sizeof(double[N_timesteps]));
  double *pressure      = malloc(sizeof(double[N_timesteps]));
  double *msd           = malloc(sizeof(double[N_save_timesteps]));
  double *vel_corr      = malloc(sizeof(double[N_save_timesteps]));
  double *pow_spec      = malloc(sizeof(double[N_save_timesteps]));
  double *freq            = malloc(sizeof(double[N_save_timesteps]));

  for (int i = 0; i<N_save_timesteps; i++){
    msd[i] = 0;
    pow_spec[i] = 0;
    vel_corr[i] = 0;
  }
  FILE *file_pointer;

  /* ----------------------------- TASK 3 ----------------------------------*/

  // read positions, momenta and cell_length
  sprintf(file_name,"../data/INIDATA_temp-%d_pres-%d.bin",
      (int) T_eq_C, (int) P_eq_bar);
  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);
  fclose(file_pointer);

  for (int i=0; i<N_atoms; i++){
    for (int j=0; j<3; j++){
      pos_0[i][j]=pos[i][j];
    }
  }
  inv_volume = pow(N_cells*cell_length, -3);
  get_forces_AL( forces, pos, cell_length, N_atoms); //initial cond forces

  printf("Initialized. Starting with Verlet timestepping.\n");
  for (int i=0; i<N_timesteps; i++){
    /*
       The loop over the timesteps first takes a timestep according to the
       Verlet algorithm, then calculates the energies and temeperature.
    */
    timestep_Verlet(N_atoms, pos,  momentum, forces, m_Al, dt, cell_length);

    E_kin  = get_kin_energy(N_atoms, momentum, m_Al );
    virial = get_virial_AL(pos, cell_length, N_atoms);

    /* 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);

    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]);

         copy_mat(N_atoms, 3, momentum, vel_all[k]);
         scale_mat(N_atoms, 3, vel_all[k], 1/m_Al);
     }
    if ((i*10) % N_timesteps == 0){
       printf("done %d0 %% of Verlet timestepping\n", (i*10)/N_timesteps);
    }
  }
  printf("calculating MSD\n");
  get_MSD(N_atoms, N_save_timesteps, pos_all, msd);

  printf("calculating velocity correlation\n");
  get_vel_corr(N_atoms, N_save_timesteps, vel_all, vel_corr);

  printf("calculating power spectrum\n");
  get_powerspectrum(N_atoms, N_save_timesteps, vel_all, pow_spec);
  fft_freq(freq, dt, N_save_timesteps);



  printf("writing to file\n");
  /* Write tempertaure to file */

  sprintf(file_name,"../data/temp-%d_pres-%d_Prod-test.tsv",
      (int) T_eq_C, (int) P_eq_bar);
  file_pointer = fopen(file_name, "w");
  for (int i=0; i<N_timesteps; i++){
    t = i*dt; // time at step i
    fprintf(file_pointer, "%.4f \t %.8f \t %.8f \n",
        t, temperature[i],pressure[i]);
  }
  fclose(file_pointer);

  /* Write displacements to file */
  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);
    for (int j=0; j<N_save_atoms; j++){
        fprintf(file_pointer, "\t %.8f", displacements[i][j]);
     }
     fprintf(file_pointer, "\n");
  }
  fclose(file_pointer);

   /* Write MSD to file */
  sprintf(file_name,"../data/temp-%d_pres-%d_dynamicProperties.tsv",
      (int) T_eq_C, (int) P_eq_bar);
  file_pointer = fopen(file_name, "w");
  // write header
  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]);
  }
  fclose(file_pointer);

  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");
  // write header
  fprintf(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]);
  }
  fclose(file_pointer);

  free(pos);           pos = NULL;
  free(pos_0);         pos_0 = NULL;
  free(momentum);      momentum = NULL;
  free(forces);        forces = NULL;
  free(temperature);   temperature = NULL;
  free(pressure);      pressure = NULL;
  free(displacements); displacements = NULL;
  free(pos_all); pos_all = NULL;
  free(vel_all); vel_all = NULL;
  free(msd); msd = NULL;
  free(vel_corr); vel_corr = NULL;
  free(pow_spec); pow_spec = NULL;
  free(freq); freq = NULL;
  return 0;
}
```

A.6 Misc functions : funcs.c

```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++){
    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);
    }
  }
  gsl_rng_free(q); /*  deallocate  rng */
}

void timestep_Verlet ( int N_atoms, double (*pos)[3],  double (*momentum)[3],
               double (*forces)[3], double m, double dt,
               double cell_length){
  for (int i = 0; i < N_atoms; i++) {
    for (int j = 0; j < 3; j++) {
      /* p(t+dt/2) */
      momentum[i][j] += dt * 0.5 * forces[i][j];
      /* q(t+dt) */
      pos[i][j] += dt * momentum[i][j] / m;
    }
  }
  /* F(t+dt) */
  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) */
      momentum[i][j] += dt * 0.5 * forces[i][j];
    }
  }
}

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++) {
    for (int j = 0; j < 3; j++) {
      p_sq += momentum[i][j] * momentum[i][j];
    }
  }
  return p_sq / (2*m);
}

void get_displacements ( int N_atoms,  double (*positions)[3],
             double (*initial_positions)[3], double disp[]) {
  for (int i = 0; i < N_atoms; i++) {
    for (int j = 0; j < 3; j++) {
      disp[i] += (positions[i][j] - initial_positions[i][j])
            *(positions[i][j] - initial_positions[i][j]);
    }
    disp[i] = sqrt(disp[i]);
  }
}


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 */
   /* 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
               MSD[it] += (all_pos[it+jt][kn][kd] - all_pos[jt][kn][kd])
                          *(all_pos[it+jt][kn][kd] - all_pos[jt][kn][kd]);
             }
          }
      }
      MSD[it] *= 1/( (double)N_atoms * (N_times-it));
   }
}

void get_vel_corr ( int N_atoms,  int N_times, double all_vel[N_times][N_atoms↩
     ][3],
                 double vel_corr[N_times]) {
   /* all_vel = velocity of all particles at all (saved) times */
   for (int it = 0; 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
               vel_corr[it] += (all_vel[it+jt][kn][kd] * all_vel[jt][kn][kd]);
             }
          }
      }
      vel_corr[it] *= 1/( (double)N_atoms * (N_times-it));
   }
}

void get_powerspectrum ( int N_atoms,  int N_times, double all_vel[N_times][↩
     N_atoms][3],
                 double pow_spec[N_times]) {
   /* all_vel = velocity of all particles at all (saved) times */
   double vel_component[N_times]; // "all_vel[:][i][j]"
   double pow_spec_component[N_times];
```

```c
   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++) { //
            vel_component[it] = all_vel[it][kn][kd];
          }
          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];
          }
       }
   }
   for (int iw = 0; iw < N_times; iw++) { // for all frequencies
       pow_spec[iw] *= normalization_factor;
    }
}



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++) {
      mat_to[i][j] = mat_from[i][j];
      }
   }
}

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++) {
      mat[i][j] = 0;
    }
  }
}

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;
    }
  }
}
```

B Auxiliary

B.1 Makefile

```make

CC = gcc
CFLAGS = -O3 -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


%.o: %.c $(HEADERS)
	$(CC) -c -o $@ $< $(CFLAGS)

all: Task1 Task2 Task3 main_Prod.c

Task1: $(OBJECTS) main_T1.c
	$(CC) -o $@ $^ $(CFLAGS) $(LIBS)

Task2: $(OBJECTS) main_T2.c
	$(CC) -o $@ $^ $(CFLAGS) $(LIBS)

Task3: $(OBJECTS) main_T3.c
	$(CC) -o $@ $^ $(CFLAGS) $(LIBS)

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

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

clean:
	rm -f *.o
	touch *.c
```

C Matlab scripts

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

```matlab
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;

energy = energy_data(:,2);
figure(1);clf;
plot(v0,energy, 'xk');

start_v = 64;
end_v = 68;
indToInclude = (v0 > start_v) & (v0 < end_v);
p = polyfit(v0(indToInclude),energy(indToInclude),2);
hold on;

vvec = linspace(start_v, end_v);
plot(vvec, p(1)*vvec.^2 + p(2)*vvec + p(3), '-r');
xlim([64 68]);

v_min = -p(2)/(2*p(1));
a_min = v_min^(1/3);

ax = gca;
ax.YLim = [-13.45 -13.42];
h1 = plot( v_min*[1 1], ax.YLim, '--k'); % plot vertical line at v_min


ax.YTick = (-13.45:0.01:-13.42);
ylabel('$E_{\rm pot}$ [eV/unit cell]');
xlabel('$a_0^3$ [\AA$^3$]');
legend('data', 'quadratic fit', ['$V_{\rm eq} \approx \, $' num2str(round(v_min↩
     ,2)) '\, \AA$^3$'], ...
    'location', 'southeast')
ImproveFigureCompPhys(gcf); h1.LineWidth = 2; setFigureSize(gcf);

%axis([63 68 ylim(1) 0]);
saveas(gcf, '../figures/potential_energy.eps', 'epsc')


%% task 2
%clc;
clf;clear

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)));
t = T_data(:,1);
T = T_data(:,2);
E = E_data(:,2);

t_eq=0.5;


fprintf('dt = %0.0e\n',dt(i));

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));
E_std=std(E(t>t_eq));
fprintf('\tE = %0.2f +- %0.1e %%\n', E_avg, abs(E_std/E_avg)*100);

figure(i);clf
plot(t, T)
yyaxis right
plot(t, E)
ylim(E_avg*(1+0.001*[1,-1]));
end


%% 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');
bar = 6.2415e-07;

t = data(:,1);
T = data(:,2)-273.15;
P = data(:,3)/bar;


t_eq=0.5;


%fprintf('dt = %0.0e\n',dt(i));

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);

P_avg=mean(P(t>t_eq));
P_std=std(P(t>t_eq));
fprintf('\tP = %0.2f +- %0.1f %%\n', P_avg, abs(P_std/P_avg)*100);

yyaxis left
plot(t,T, 'color', GRAY),hold on
plot(t, cumsum(T)./(1:length(t))','-k')
ylabel('$T \, [^\circ \rm C]$')

ylim([400,800])


yyaxis right
plot(t,P),hold on
plot(t,cumsum(P)./(1:length(t))','-k')


ylabel('$P \,[\rm bar]$')
ylim([-50,200])

xlabel('$t$\, [ps]')

ImproveFigureCompPhys(gcf, 'linewidth', 3, 'LineColor', {'MYORANGE', GRAY, 'MYBLUE', GRAY}');

%% determine displacements and MSD
temperatures = num2str([500;700]);
clc; clf;
figure(10); clf;
FILENAMES = strcat({'../data/temp-'}, temperatures, '_pres-1_displacements.tsv');
FILENAMES_Dyn = strcat({'../data/temp-'}, temperatures, '_pres-1_dynamicProperties.tsv');
FILENAMES_Pow = strcat({'../data/temp-'}, temperatures, '_pres-1_power-spectrum.tsv');
for iFile = 1:numel(FILENAMES)

    figure(iFile); clf;
    data = load(FILENAMES{iFile});
    t = data(:,1);
    dx = data(:,2:end);

    plot(t, dx.^2); hold on;


    data = load(FILENAMES_Dyn{iFile});
    MSD = data(:,2);
    vel_corr = data(:,3);
    plot(t, MSD, 'k')

    if iFile ==2 % liquid
        tStart = 1;
        D = MSD(t>tStart)./(6*t(t>tStart));
        selfDiffusionCoeff = mean(D); % in   ^2 /ps
        plot(t, 6*t*selfDiffusionCoeff, ':');
    end

    leg = legend( strcat({'$n=$'}, num2str((1:size(dx,2))'))');
    leg.Location='northwest';
    xlabel('$t$ [ps]')
    ylabel('$\Delta x^2 \,[\rm \AA^2]$')
    if iFile ==1
        ylim([ 0 1.0]);
    else
        ylim([0 200]);
    end
    ImproveFigureCompPhys(gcf);

    figure(10)
    plot(t, vel_corr/vel_corr(1)); hold on;
    xlim([0 1])

end

% velocity correlation
figure(10);clf; figure(11);clf;
n_average_points = 1;%30;
for iFile = 1:numel(FILENAMES)
    data = load(FILENAMES_Dyn{iFile});
    t = data(:,1);
    vel_corr = data(:,3);

    data = load(FILENAMES_Pow{iFile});
    freq = data(:,1);
    pow_spec = data(:,2);

    figure(10);
    plot(t, vel_corr/vel_corr(1)); hold on;

    dt = t(2)-t(1);
    N_times = round(length(t)/2); % we have too bad statistics at later times.
    deltaf = 1/(N_times * dt);
    omegavec = 0:deltaf:(1/(2*dt));
    %PhiHat = 2 * trapz(t(1:N_times), (vel_corr(1:N_times) * ones(size(omegavec))) .* cos(t(1:N_times) * omegavec ), 1); %dimension 1
    PhiHat =  1/2 * 1/N_times * 2 * sum( (vel_corr(1:N_times) * ones(size(omegavec))) .* cos(t(1:N_times) * omegavec ), 1); %dimension 1

    figure(11);

    plot(omegavec/(2*pi), PhiHat); hold on;
    plot(freq, pow_spec, ':'); hold on;
    if iFile ==2 % liquid
        tStart = 1;
        selfDiffusionCoeff_spectral = PhiHat(1)/6; % in   ^2 /ps
    end

end

disp([selfDiffusionCoeff selfDiffusionCoeff_spectral]);

figure(10)
xlim([0 1])
leg = legend(strcat({'$T='}, num2str([500;700]), '\,^\circ $C'));
leg.Location='northeast';
xlabel('$t$ [ps]')
ylabel('$\Phi (t)/\Phi(0)$')


figure(11)
leg = legend('$T= 500 \, ^\circ $C, $ \hat \Phi$' , '$T= 500 \, ^\circ $C, $|\hat v|^2$',...
    '$T= 700 \, ^\circ $C, $ \hat \Phi$', '$T= 700 \, ^\circ $C, $|\hat v|^2$');
xlim([0 30])
ylim([0 Inf])
ImproveFigureCompPhys('LineColor', {'r', 'MYRED', 'GERIBLUE','MYLIGHTBLUE'}');
%%
clc;clf;


FILENAME = '../data/INIDATA_temp-700_pres-1.bin';
fID=fopen(FILENAME,'rb');
data1=fread(fID,[3,inf],'real*8').';
fclose(fID);

AMU = 1.0364e-4;
m_Al = 27*AMU;
kB= 8.61733e-5;
N_atoms=4^4;

T=sum(sum(data1.^2,2),1) / (3*m_Al*N_atoms*kB)


data2=load('../data/phase-space_temp-500_pres-1.tsv');

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:
% - none (improve all figures) or handles to figures to improve
% - optional:
%       LineWidth  int
```

```matlab
%       LineStyle  column vector cell, e.g. {'-','--'}',
%       LineColor  column vector cell, e.g. {'k',[0 1 1], 'MYBLUE'}'
%                           colors: MYBLUE,MYORANGE,MYGREEN,MYPURPLE, MYYELLOW,
%                           MYLIGHTBLUE, MYRED
%       Marker column vector cell, e.g. {'.', 'o', 'x'}'

% ImproveFigure was originally written by Adam Stahl, but has been heavily
% modified by Linnea Hesslow


%%% Handle inputs
% If no inputs or if the first argument is a string (a property rather than
% a handle), use all open figures
if nargin == 0 || ischar(varargin{1})
    %Get all open figures
    figHs = findobj('Type','figure');
    nFigs = length(figHs);
else
    % Check the supplied figure handles
    figHs = varargin{1};
    figHs = figHs(ishandle(figHs) == 1); %Keep only those handles that are ↩
         proper graphics handles
    nFigs = length(figHs);
end

% Define desired properties
titleSize = 24;
interpreter = 'latex';
lineWidth = 4;
axesWidth = 1.5;
labelSize = 22;
textSize = 20;
legTextSize = 18;
tickLabelSize = 18;
LineColor = {};
LineStyle = {};
Marker = {};

% define colors
co =  [ 0    0.4470    0.7410
    0.8500    0.3250    0.0980
    0.9290    0.6940    0.1250
    0.4940    0.1840    0.5560
    0.4660    0.6740    0.1880
    0.3010    0.7450    0.9330
    0.6350    0.0780    0.1840 ];
colors = struct('MYBLUE', co(1,:),...
    'MYORANGE', co(2,:),...
    'MYYELLOW', co(3,:),...
    'MYPURPLE', co(4,:),...
    'MYGREEN', 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);

% Loop through the supplied arguments and check for properties to set.
for i = 1:nargin
    if ischar(varargin{i})
        switch lower(varargin{i})   %Compare lower case strings
            case 'linewidth'
                lineWidth = varargin{i+1};
            case 'linestyle'
                LineStyle = varargin{i+1};
            case 'linecolor'
                LineColor = varargin{i+1};
                for iLineColor = 1:numel(LineColor)
                    if isfield(colors, LineColor{iLineColor})
                        LineColor{iLineColor} = colors.(LineColor{iLineColor});
                    end
                end
            case 'marker'
                Marker = varargin{i+1};
        end
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Improve the figure(s)

for iFig = 1:nFigs

    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

    %%% TEXT APPEARANCE: first set all to textSize and then change the ones
    %%% that need to be changed again

    %Change size of any text objects in the plot
    set(textObjects,'FontSize',textSize);
    set(legObjects,'FontSize',legTextSize);

    %%% FIX LINESTYLE, COLOR ETC. FOR EACH PLOT SEPARATELY
    for iAx =  1:numel(axesObjects)
        lineObjInAx = findall(axesObjects(iAx), 'Type', 'line');

        %set line style and color style (only works if all figs have some
        %number of line plots..)
        if ~isempty(LineStyle)
            set(lineObjInAx, {'LineStyle'}, LineStyle)
            set(contourObjects, {'LineStyle'}, LineStyle); %%%%%%
        end
        if ~isempty(LineColor)
            set(lineObjInAx, {'Color'}, LineColor)
            set(contourObjects, {'LineColor'}, LineColor); %%%%%%
        end
        if ~isempty(Marker)
            set(lineObjInAx, {'Marker'}, Marker)
            set(lineObjInAx, {'Markersize'}, num2cell(10+22*strcmp(Marker, '.')))
        end

        %%% change font sizes.
        % Tick label size
        xLim = axesObjects(iAx).XLim;
        axesObjects(iAx).FontSize = tickLabelSize;
        axesObjects(iAx).XLim = xLim;
        %Change label size
        axesObjects(iAx).XLabel.FontSize = labelSize;
        axesObjects(iAx).YLabel.FontSize = labelSize;

        %Change title size
        axesObjects(iAx).Title.FontSize = titleSize;
    end

    %%% LINE APPEARANCE
    %Change line thicknesses
    set(lineObjects,'LineWidth',lineWidth);
    set(contourObjects, 'LineWidth', lineWidth);
    set(axesObjects, 'LineWidth',axesWidth)

    % set interpreter: latex or tex
    set(textObjects, 'interpreter', interpreter)
    set(legObjects, 'Interpreter', interpreter)
    set(axesObjects,'TickLabelInterpreter', interpreter);
end
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
```