Lab 4: Handout
Molecular dynamics.

In this lab, we will be using MOLDY and GULP as our molecular dynamics (MD) codes. The MOLDY manual is online and is located at this link:

http://www.earth.ox.ac.uk/~keithr/moldy-manual/moldy.html

It is a well-written manual, and is an excellent resource for learning about molecular dynamics in general.

The MD features of GULP are not quite as well documented since GULP is primarily a static energy code. General features of GULP are described in the GULP manual or in the GULP handout from Lab1.

Some other recommended links

http://www.fisica.uniud.it/~ercolessi/md/md/
http://www.fisica.uniud.it/~ercolessi/md/f90/

The first is link is an introduction to MD, written by Furio Ercolessi. It is a very good, and very easy-to-understand explanation of molecular dynamics and interatomic potentials. **It is highly recommended that you read this link**.

The second link is an example of a molecular dynamics code, written in Fortran 90. If you look at the sample MD code, you will see that the code itself is not too complicated (not counting the complications of energy methods).

There are many, other molecular dynamics codes available for public use. Many of these are listed on this webpage:

http://www.mrflip.com/resources/MDPackages.html

Available MD codes include CHARMM, MOSCITO, MARVIN, DYNAMO, and many others.

Quick summary of MD

Molecular dynamics follows classical mechanics, notably solving Newton's equations of motions for all atoms:

$$\vec{F}_i = m_i \vec{a}_i \quad ,$$

where i denotes the atom. Thus, *in principle*, for any initial parameters (positions, velocities), the positions, velocities, and accelerations are determined for all future times. The force on an atom, $\vec{F}_i$, (the left hand side) can be obtained from taking the derivative of the potential energy with respect to positions. That is, $\vec{F}_i = \frac{-dU}{d\vec{r}}$,

where U is the potential energy. When the form of U is analytical (such as with potentials), calculating the forces on an atom is relatively easy. Thus, we can solve for accelerations, $\vec{a}_i$ (right hand side), because the masses of the atoms are known. In theory, we could integrate accelerations to obtain velocities, and integrate velocities to obtain positions.

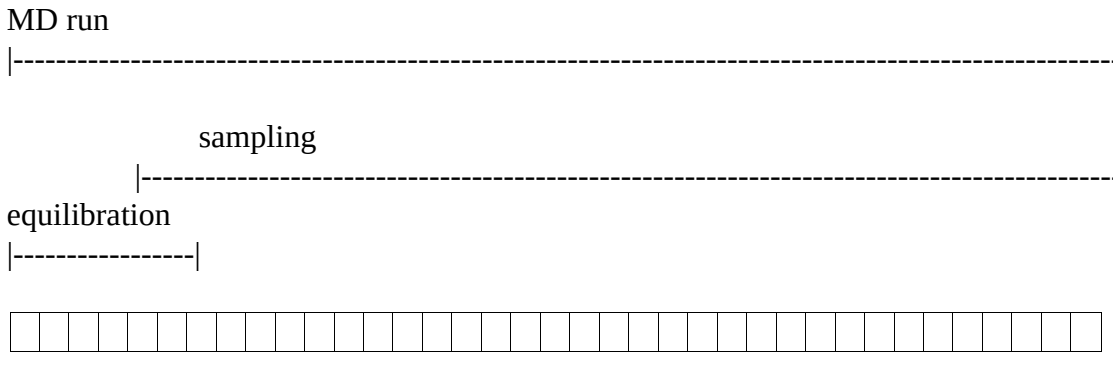
In practice, all integrations are carried out numerically. The parameter that controls the fineness of the integration is called the **timestep, δt**. Knowing positions, velocities, and accelerations at time t, we can integrate to obtain positions, velocities, and accelerations at time t+ δt.

In class, you learned about the Verlet algorithm. The Verlet algorithm is a very common and easy to derive time integration algorithm. Moldy uses the Beeman algorithm, which is a predictor-corrector type algorithm. The algorithm is outlined below.

$$\left.\begin{array}{lll}
\text{i} & x(t+\delta t) & = x(t) + \delta t\,\dot{x}(t) + \frac{\delta t^2}{6}\left[4\ddot{x}(t) - \ddot{x}(t-\delta t)\right] \\
\text{ii} & \dot{x}^{(p)}(t+\delta t) & = \dot{x}(t) + \frac{\delta t}{2}\left[3\ddot{x}(t) - \ddot{x}(t-\delta t)\right] \\
\text{iii} & \ddot{x}(t+\delta t) & = F(\{x_i(t+\delta t), \dot{x}_i^{(p)}(t+\delta t)\}, i = 1 \ldots n) \\
\text{iv} & \dot{x}^{(c)}(t+\delta t) & = \dot{x}(t) + \frac{\delta t}{6}\left[2\ddot{x}(t+\delta t) + 5\ddot{x}(t) - \ddot{x}(t-\delta t)\right] \\
\text{v} & \text{Replace } \dot{x}^{(p)} \text{ with } \dot{x}^{(c)} \text{ and goto } \textit{iii}. \text{ Iterate to convergence} &
\end{array}\right\}$$

x represents coordinates, and $\dot{x}^{(p)}$ and $\dot{x}^{(c)}$ to represent ``predicted" and ``corrected" velocities respectively. GULP is capable of integrating using the leapfrog Verlet, velocity Verlet or Gear Predictor-Corrector algorithms. With initial parameters (and acceleration obtained from $\vec{a}_i = \frac{\vec{F}_i}{m_i}$) , and a chosen timestep, we can calculate system parameters at time t+ δt. Note that initial velocities $\vec{v}_i(t) = \dot{x}(t)$ are chosen randomly, but consistent with the Maxwell-Boltzmann distribution. Unfortunately, these initial velocities can be quite wrong at first. Thus, we want to

the system to run a several timesteps first before we actually **sample** thermodynamic quantities (such as potential energy or kinetic energy). This is called **equilibration time**. A schematic is shown below.



Temperature is controlled by scaling the velocities of the atoms. To see how, remember that from the equipartition theorem.

$$\text{Kinetic Energy} = \frac{3}{2}kT$$

Also, remember from classical mechanics that

$$\text{Kinetic Energy} = \frac{1}{2}mv^2$$

Thus, the temperature can be changed by scaling the velocities. Note, that when this happens, we are no exactly longer following Newton's laws of motion. Thus, often temperature scaling is turned on during the equilibration part of the simulation, but turned off during the sampling part of the simulation.

To do an MD simulation, we need a system to simulate. In our case, we will build supercells of Kr. Why are supercells necessary? Remember periodic boundary conditions from lab 1. We need supercells so that we can see long-wavelength fluctuations in atomic movements.

To summarize what one needs for this MD simulation

A. An energy method – (derivatives of energy are used to obtain accelerations)
B. An integration method (Beeman for Moldy, your choice in GULP)
C. A timestep for the integration method (you will need to find this).
D. A system to calculate (with supercell, atom positions, and atom masses)

Examining the different integrators:
All of the scripts and files for LAB 4 can be copied to a local directory:

```
username$ cp /home/nedward/LAB4/scripts/* .
```

We will be using GULP to compare the leapfrog Verlet, velocity Verlet and 5th order Gear Predictor-Corrector integration algorithms as they apply to EAM Ni.
The files for this Lab assignment can be found in /home/nedward/LAB4

gulp.in.example

```
(1)md conv
(2)
(3)cell
(4)   10.572000000    10.572000000    10.572000000 90 90 90
(5)fractional 108
(6)Ni     0.000000000     0.000000000     0.000000000
(7)...
(8)timestep 0.01
(9)temperature 300
(10)equilibration 3.50
(11)production 3.50
(12)integrator leapfrog verlet
(13)species
(14)Ni core  0.000
(15)manybody
(16)Ni core Ni core 0.0 12.0
(17)#attractive part
(18)eam_density power 6
(19)Ni core   729.6928296
(20)#repulsive part  second power is a dummy argument
(21)lennard 9 6
(22)Ni core Ni core 1303.09841 0.0 0.0 12.0
```

For the sake of brevity, I will only mention lines in the input file which are different from the descriptions used in Lab1. I've also omitted several lines which describe the atomic positions of atoms in a supercell.

Line 1: Instead of the opti and single keywords, we will be using the md and conv keywords for these MD calculations. The keyword md signifies that this will be a molecular dynamics run and conv signifies that we will be doing constant volume calculations.

Line 8: Timestep increment is given in picoseconds. **You will be changing this line.**

Line 9: Temperature is given in Kelvin. You will not need to change the temperature in this exercise but if you choose to do the extra credit problem you will need to edit this line.

Lines 10-11: Equilibration and production times are given in picoseconds if the number contains a decimal point. If the number does not contain a decimal point then the equilibration and production times are integer multiples of the timestep. **You may want to vary both the equilibration times and production times to verify that you are acquiring averages from a well equilibrated configuration.**

Line 12: This integrator type to use for the calculation is declared here. Possible entries are leapfrog verlet, velocity verlet, and gear. **You will be changing this line**.

The critical script which you will be using in this exercise is **`run_gulp.j`**, it is explained here in two parts: first the header of the script that will allow you to use the queue properly and second the creation of a Gulp input file. These two pieces will be one file (**`run_gulp.j`**) on your machine: The header:

```
(1)#!/bin/bash -f
(2)#########################################################################
(3)#$ -N gulp_test
(4)#$ -cwd
(5)#$ -o $HOME/LAB4/err.out
(6)#$ -j y
(7)#$ -S /bin/sh
(8)
(9)#########################################################
(10)# This is a script to run GULP calculations in
(11)# the hpcbeo2 cluster.
(12)# P. Sit and N. Miller, 04-03-2005
(13)#########################################################
(14)# set the needed variables
(15)
(16)MYDIR="LAB4"
(17)USER=`whoami`
(18)RUNDIR="/state/partition1/$USER"
(19)OUTPUT="/$HOME/$MYDIR"
(20)if [ ! -d $RUNDIR ]; then
(21)        mkdir $RUNDIR
(22)fi
(23)
(24)if [ ! -d $OUTPUT ]; then
(25)        echo $MYDIR does not exist, please create it first
(26)        exit
(27)fi
(28)
(29)#########################################################
```

line1: calls the shell (don't change)
lines 3-7: For the queuing system, you can change (3) and (5) accordingly
line 16: Change to the directory you are in

lines 17-19: do not need to be changed
lines 20-26: Check for all of the correct directories (no need to change)

The body:

```
(1)# calculations
(2)
(3)cd $RUNDIR
(4)
(5)cell=3
(6)integrator=leapfrog
(7)integrator2=verlet
(8)for timestep in 0.001 0.005 0.01;
(9)do
(10)cat>gulp.wrap<<EOF
(11)3.524 Ni
(12)md conv
(13)timestep $timestep
(14)temperature 300
(15)equilibration 3.50
(16)production 3.50
(17)integrator $integrator $integrator2
(18)species
(19)Ni core  0.000
(20)manybody
(21)Ni core Ni core 0.0 12.0
(22)#attractive part
(23)eam_density power 6
(24)Ni core   729.6928296
(25)#repulsive part  second power is a dummy argument
(26)lennard 9 6
(27)Ni core Ni core 1303.09841 0.0 0.0 12.0
(28)EOF
(29)
(30)echo $cell $cell $cell |  /home/nedward/LAB4/bin/buildcell
(31) /home/nedward/LAB4/bin/gulp < gulp.in >$OUTPUT/gulp.$cell.$integrator.$timestep.out
(32)done
```

This script calls the **`buildcell`** routine from Lab1. The lines of importance are lines 5-9. You can change the cellsize, integrator and timesteps. Integrator is split into two words since GULP has two multiword integrators. For the Gear integrator, leave the right hand side of integrator2 set to nothing.

Output files will have the naming structure of
gulp.<cellsize>.<first word of integrator>.<timestep>.out

Key Utilities:
To use these utilities you will have to run this command:

```
username$ export PATH=$PATH:/home/nedward/LAB4/bin
```

`GULPMD_getEnergy.bash` – extracts the energy from a single file and stores the result in energy_average.out and energy_inst.out.
usage: `GULPMD_getEnergy.bash <outfile name>`

`GULPMD_grabEnergy.bash` – extracts energy from all .out files and stores the result in files of the format <outfilename -.out>.average.energy, <outfilename-.out>.inst.energy, <outfilename-.out>.summary.

Melting of Krypton

again the files for this lab can copied as:
username$ **cp /home/nedward/LAB4/scripts/* .**

In this lab, we look at melting of Kr. Kr crystallizes in the FCC structure (the same structure as Cu and Al). Experimentally, Kr melts at 115 K. We will use Lennard-Jones (LJ) potentials, which we previously used in lab 1.

We are going to use two approaches:
[1] Melting of bulk Kr using supercells in Moldy
[2] Melting of Kr from a cell with a solid-liquid interfacce already introduced, similar to a method developed by **F. Ercolessi.** This method is used in the paper: PRB 49(5) 3109 which is available on the MIT server (thus, it is suggested you read it).

The following is the outline for the first method:
Finding the equilibrium lattice parameter:
This is done for you. The lattice parameter of Krypton that we will use is 5.920 Angstroms. Usually, you would have to do this yourself.

The input files:
Basically Moldy needs two input files: One that defines the system (**`Kr.in`**) and one that defines all of the external aspects of the system, T, P, etc. (**`control.Kr`**). You are provided with these two files:
The Kr.in file:

```
(1)Krypton  256
(2)1          0           0          0    83.80 0 Kr
(3)end
(4)Lennard-Jones
(5)1 1 5.4516 3.83
(6)end
(7)5.920 5.920 5.920 90.0 90.0 90.0 4 4 4
(8)Krypton 0.0 0.0 0.0
(9)Krypton 0.0 0.5 0.5
(10)Krypton 0.5 0.0 0.5
(11)Krypton 0.5 0.5 0.0
(12)end
```

Line 1: This has the species name (Krypton) and the number of atoms (256). You will not edit the species name. **On the other hand, depending on the supercell size, you will have to change the number of atoms**.

Line 2: This has an id number (1, first atom type), three flags you won't change, atomic mass (83.80), charge (0), and name(Krypton). You will not edit the species name. You do not need to edit this line.

Line 4-6: Specifies potential information. Line 4 tells the potential type (LJ). Line 5 gives what potential the atoms are between (type 1 and type 1), and the two potential parameters ε and σ. Epsilon and sigma are potential parameters. If you recall, the LJ potential is written:

$\phi(r_{ij}) = \frac{A}{r^{12}} - \frac{B}{r^{6}}$. This can be written equivalently

$$\phi(r_{ij}) = \epsilon\left[\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right]$$

Note that the above form is somewhat nonstandard in that epsilon is defined as four times the potential well depth.

Line 7-12: Specifies the cell. Line 7 has *a*, *b*, *c*, α, β, γ, n_x, n_y, n_z, where n_x, n_y, and n_z are how big you want to build the supercell in the x, y, and z directions**. You will need to change this to change the size of your supercell.** Lines 8-11 specify the atoms in the FCC structure.

The control.Kr file:

Note that the **`control.Kr`** will be generated by the script discussed in the next section.

Note: any lines can be commented out by putting a '#' at the front of the line.

```
(1)title=Krypton
(2)sys-spec-file=Kr.in
(3)# TIME-TIMESTEP (picoseconds)
(4)step=.01
(5)nsteps=5000
(6)# AVERAGES
(7)begin-average=500
(8)average-interval=100
(9)# Temperature
(10)temperature=10
(11)scale-interval=200
(12)scale-end=500
(13)scale-options=4
(14)#const-temp=1
(15)# Pressure
(16)#const-pressure=1
(17)#w=1000.0
(18)#pressure=0.
(19)#strain-mask=238
(20)# INITIAL CONDITION
(21)lattice-start=1
(22)# DUMP-RESTART
(23)begin-dump=1001
(24)dump-interval=50
(25)dump-level=3
(26)ndumps=100
(27)dump-file=Krdump
(28)restart-file =
(29)save-file= Kr.restart
(30)backup-file=
(31)# Output
(32)page-length=44
(33)page-width=80
(34)print-interval=200
(35)roll-interval=200
(36)# RDF
(37)rdf-interval=20
(38)begin-rdf=2000
(39)rdf-limit=10.0
(40)nbins=200
(41)rdf-out=2500
(42)# CUTOFF
(43)cutoff=10
(44)strict-cutoff=1
(45)end
```

Line 1-2: Title is title information (you can change, but not required). Sys-spec-file gives information about the potentials. This is set to Kr.in. You probably will not change these parameters.

Lines 3-5: Step is the timestep (in picoseconds), and nstep is the the total simulation length (in terms of timesteps). **You will have to test the timestep and change total number of steps. To obtain simulation time, multiply timestep by total number of steps**.

Lines 6-8: Begin-average tells the number of steps at which the code will start sampling to obtain thermodynamic quantities. **This is equivalent to "equilibration steps". You will have to change the begin-average parameter**. Average-interval tells how often to sample. You probably don't need to change this.

Lines 9-13: These lines control the temperature information. Temperature gives the temperature (in K), and scale-end gives when to stop temperature scaling. scale-interval gives how often to scale the temperature, const-temp and scale-options gives the method of scaling, You will have to change the temperature parameter. **You will also have to change the values for temperature and scale-end.** If the value of scale-end is not obvious, reread the "quick summary on MD" earlier in this handout.

Lines 14-18: These lines control applied pressure on the system. The applied pressure is 0 Mpa. You will not be editing these files unless you intend to examine the system with a constant pressure ensemble, in which case you would need to uncomment these lines.

Lines 19-20: Lattice-start controls the initial configuration. If lattice-start is set to 1, then it will read the Kr.in file and create supercells. **You will do your initial run with lattice-start=1, but all subsequent runs should be started from restart files, and lattice-start should be commented out.** Do not put lattice-start=1 unless you are doing a brand new supercell calculation.

Lines 21-29: These lines control the inputs and outputs. In MD simulations, we usually perform one initial run. All subsequent runs (in this case, at higher temperature) use the output of the previous run as input into the new run. The reason for this is that initial parameters are chosen randomly, but can be quite inaccurate at first. Using the output of a previous run gives better initial parameters for the new temperature. Lines 21-26 control the dumps (how often the program spits out data). begin-dump controls when dumping starts**.** dump-level controls how much information is dumped, and dump-interval controls how often the program dumps (in terms of timesteps), **dumpfile controls the name of the dumpfile – you will want to change this, depending on the conditions of your run**. Save-file is usually the name of the file that you want to save the final configuration to (positions, velocities, accelerations. **You will probably want to change this, depending on the conditions of your run.** Restart-file controls the restart file. **In this example it is empty, but you will want to change it to the name of the save file.**

Lines 30-44: These lines control output format, RDF output format, and potential cutoffs. You will not change any of these parameters.

Running Moldy:

Much like Gulp, we will have to run Moldy from a script in order to queue it properly. The script that you are provided with should allow for both the altering of timestep and temperature. Following these short descriptions is the script: (**`run_moldy.j`**)

Finding a timestep:

The first thing we need to do is find a good timestep. This is controlled by the **step** parameter in the control file. A rough rule of thumb is that the timestep is 1/100 the highest-frequency excitation that you are interested in. This is usually around 10^{-13}-10^{-15} seconds. You want as long a timestep as possible, so that you can perform long simulations. On the other hand, too long of a timestep will give nonsensical answers.

The standard test for testing the timestep is to measure the conservation of energy in the NVE ensemble. In the NVE ensemble, total energy (PE+KE) is constant. If the timestep is too long, one of two things may happen. First, instead of staying constant, the total energy may drift higher or lower. Second, the atoms may crash into each other. These are both indications of too large of a timestep. Note, we could also do this in the NPE ensemble also – in this case, the conserved quantity would be H =E +PV.

The timestep test should be done at a smaller-sized supercell but not too small (3x3x3 is fine). The temperature should be at LEAST the temperatures of interest. It should be done for ~3000 sampling timesteps. You can output the total energy by typing in **moldyext –f 4 <outputfile>.**

To recover the NVE ensemble, comment out everything that controls the pressure. To do this, put a pound (#) symbol in front of the pressure keywords. Next, make sure that temperature scaling is turned off at the same time you start sampling. If there is an option “const-temp”, you should comment this out. When temperature scaling is turned off, you have the NVE ensemble.

Making a script to change the temperature:

The `run_moldy.j` script: Again the scrip is explained in two parts, the header and the run section, if you need and explanation of the header refer to the GULP section of this handout. This is the run section of the script which creates the **`control.Kr`**:

```
(1) ##########################################################
(2) # calculations
(3) # oldtemp=10
(4) cp $OUTPUT/$INPUT_FILE $RUNDIR/
(5) cd $RUNDIR
(6) #cp $OUTPUT/Kr.restart.oldtemp $RUNDIR
(7) for tstep in 0.001 0.01
(8) do
(9) for temp in 100
(10)do
(11)
(12)cat > $RUNDIR/control.Kr << EOF
(13)title=KR
(14) sys-spec-file=Kr.in
(15)
(16)# TIME-TIMESTEP (picoseconds)
(17)step=$tstep
(18)nsteps=4000
(19)
(20)# AVERAGES
(21)begin-average=500
(22)average-interval=100
(23)
(24)# Temperature
(25)temperature=$temp
(26)scale-interval=200
(27)scale-end=500
(28)scale-options=4
(29)#const-temp=1
(30)# Pressure
(31)#const-pressure=1
(32)#w=1000.0
(33)#pressure=0.
(34)#strain-mask=238
(35)
(36)# INITIAL CONDITION
(37)lattice-start=1
(38)# DUMP-RESTART
(39)begin-dump=1001
(40)dump-interval=50
(41)dump-level=3
(42)ndumps=100
(43)dump-file=Krdump$temp
(44) #restart-file = Kr.restart.$oldtemp
(45)save-file= Kr.restart.$temp
(46)backup-file=
(47)
(48)# Output
(49)page-length=44
(50)page-width=80
(51)print-interval=200
(52)roll-interval=200
(53)
(54)# RDF
(55)rdf-interval=20
(56)begin-rdf=2000
(57)rdf-limit=10.0
(58)nbins=200
(59)rdf-out=2500
(60)
(61)# CUTOFF
(62)cutoff=10
(63)strict-cutoff=1
(64)end
(65)EOF
(66)/home/nedward/LAB4/bin/moldy<control.Kr>$RUNDIR/Kr.out.$tstep.$temp
(67)oldtemp=$temp
(68)done
(69)done
(70)mv $RUNDIR/*Kr* $OUTPUT/
```

Basically this script will do both temperature and timestep sweeping, however you should do them separately. First, find a usable timestep and then start changing the temperature. Note, be careful about what is commented out and what is not:

Line 3: Discussed later (the first temperature run you will do will relate to this)

Line 4: This copies the **`Kr.in`** file (defined in the header) to the local disk.

Line 5: This enters the directory on the local disk.

Line 6: This will copy the reference restart file to the working directory (leave it commented out until you start the full temperature iterations)

Lines 7-8: Timestep iterating, you can change this, and it is recommended that you optimize this first.

Lines 9-10: Temperature iterating, you will be using this while seeking the melting temp. Note for optimizing the timestep, just choose one temperature.

Line 11: Starts to make the control.Kr file

Lines 12-64: The body of the control.Kr file, which was discussed before. Make note of the following:

Line 36: When running the timestep iterations and the first reference temperature iteration, you will leave this uncommented. As soon as you start reading from the restart files you will need to comment this line out.

Line 43: Uncomment this line according to the above instruction, note that you will have to do one temperature run with it commented (try temp=10) for a reference to start from. Then when you finally do start the temperature iteration uncomment lines 3 and 10 and comment out line 36.

Line 66: Runs the moldy executable.

Line 67: Changes the reference Kr.restrart file to the current one for the next run to start from.

Line 70: Moves any files that were created on the local disk back to your home directory.

The second method: The solid-liquid interface:

This method is a bit more tricky, thus we will provide you with an input file **`Kr_SL.in`** : This file contain 1024 Kr atoms, half solid and half liquid. It is a rectangular unit 4x4x16 unit cells of Kr.

You will need to:

1. Alter the script (**`run_moldy.j`**)to run from the restart file that we provide to you: (**`Kr.restart.store`**)
   Hint: You will want to use this restart file from to start all of the different temperatures.

2. Use velocity scaling for longer than before

3. You may need hundreds of picoseconds of total simulation to see the effect

4. Make sure that the final system is two phase

Tools for data analysis:

Data analysis: visualizing

XCRYSDEN

run the script:

username$**mdxyz > <target.xyz>**
example: username$ **mdxyz > Kr.100.xyz**

follow the directions:

choose **(2)** for restart file
enter the file name (ex. **Kr.restart.100**)
choose **(1)**

Data analysis: Radial distribution function.

The Radial distribution function gives the density of an atom ?, at a particular distance d. It is calculated by summing the number of atoms found at a given distance in all directions. In a solid, the radial distribution function will consist of sharp peaks. As a material melts, these peaks will broaden.

To find the radial distribution function:
username$ **makerdf.pl <output filename>**
For example,
username$ **makerdf.pl Kr.out**

This will create a file rdf.out that contains rdf information.
If you forget how to use this script, type
username$ **makerdf.pl**
with no arguments. This will give you a reminder on how to use the script.

If you want to generate the RDF for all of your current output files, type
username$ **getallRDF.bash**
This will calculate the RDF for all of your output files and store the results in the from <outputfilename>.rdf

Data analysis: Potential energy vs. temperature.

U vs. T at one temperature:

To find potential energy and temperature (averaged throughout the run) type
username$ **getUvT.pl <output filename>**

For example,
username$ **getUvT.pl Kr.out.110**
109.7360 -2562.3160

This will give you an output of the averaged temperature and potential energy for that run. In this case, it outputs an average temperature of 109.7 K.

Note, the INPUT temperature is not necessarily the same as the temperature of the run! This is because temperature scaling is not a 100% foolproof procedure. it will try to set the temperature of the run to the temperature you choose, but it will not be able to set it exactly. The two temperatures should be close, but to be consistent you should use the temperature of the run, rather than the input temperature.

If you forget how to use this script, type

username$ **getUvT.pl**

with no arguments. This will give you a reminder on how to use the script.

U vs. T at all temperatures:

To find potential energy and temperature (averaged throughout the run) type

username$ **getallUvT.pl**

This script takes all output files (if they are named Kr.out.<temperature here>), finds the average temperature and potential energy, and puts them in the file UvTall.out

Data analysis: Mean squared displacements.
Mean squared displacements can give a lot of useful information. MSD are calculated from <|r(t)-r(0)|>. One can obtain a lot of useful information from msd. For instance, one can calculate diffusion from MSD from the relation:

6Dt= <|r(t)-r(0)|>.

To calculate msd, type

```
username$ getmsd.pl <dump filename> <spec file>
```

For example:

```
username$ getmsd.pl Kr.dump.1500 Kr.in
```

for a run at 150 K. Important: use the dumpfile (*.dump) instead of the output file (*.out).

This script will output the mean squared displacements as a function of time to the file msd.out. The first column is mean squared displacements in x, the second column is mean squared displacements in y, the third column is mean squared displacements in z, and the fourth column is total mean squared displacements.(angstroms sq.). If you forget how to use this command, type

```
username$ getmsd.pl
```

with no arguments.

To generate mean squared displacement files for all of your calculations in the current directory, type

```
username$ getallMSD.bash
```
The results will be stored in the form of files called <dumpfilename>.msd

You will have to insert the time axis yourself.

Examining fluctuation of temperature vs system size: Related to problem 4:
We will use the same script file as problem 1. By changing cellsize=3, 4, 5, 6 etc, we can change the totoal number of atoms in the simulations.

Key utilities
`GULPMD_getTemperature.bash` – extracts the temperature from a single file and stores the result in temperature_average.out and temperature_inst.out.
usage:

```
username$ GULPMD_getTemperature.bash <outfile name>
username$ GULPMD_grabTemperature.bash
```

extracts temperature from all .out files and stores the result in files of format <outfilename -.out>.average.energy, <outfilename-.out>.inst.energy, <outfilename-.out>.summary. The summary files contains the standard deviation which you will need for data analysis.

FAQ for runs:

What is a "normal" timestep?

It depends on the material and the integration method. Usually, you take the type of excitation you are interested in, and divide by ~100. For lighter materials, or for higher temperatures, a shorter timestep may be needed. To get an idea of "bad" timesteps, you should check both long timesteps and short timesteps.

Why am I testing my timestep in the NVE ensemble? Why do I have to turn pressure control off?

You could test your timestep in the NPE ensemble too. In this case, your conserved quantity would be H=E+PV. Most MD codes were originally written for the NVE ensemble and report E rather than H.

How do I know the timestep is good?

You need to guess and check. If your timestep is too long, a few things can happen. One, your atoms may crash into each other. Two, in the NVE ensemble, your total energy may show a constant drift higher or lower. If either of these two things happen, shorten the timestep. In first-principles MD, a timestep that is too long will usually mean that the electronic iterations will not converge.

How long should my runs be to check timesteps?

You need to set both equilibration and production time to be long, say timestep x 1000 for equilibration and timestep x 3000 for production. The NVE ensemble takes some time to equilibrate, so don't worry about large fluctuations during equilibration time.

What temperature should I use to check timesteps?

Your "temperature" should be at least as high than the temperatures of simulation interest. Different temperatures can require different timesteps.

In principle you can do runs with longer timesteps at lower temperatures and shorter timesteps at higher temperatures– this is sometimes done in first-principles MD.

What size supercell should I use to check timestep?

The supercell does not have to be enormous, but it does have to be large enough to see enough of the different vibrational modes. A 3x3x3 supercell, or even a 2x2x2 supercell is probably good enough (a larger supercell is fine too, but will take a little longer).

How long (simulation time) should my runs be?

A good time is between 1000-3000x the timestep for equilibration and 2000-4000x the timestep for production. Longer is better, but then the runs will take longer.

My runs take way too long (in terms of my time).

Make sure your timestep isn't too small and total simulation time isn't too long. Don't worry about making the most enormous supercell size. The basic concepts of timestep, scaling, simulation time, etc.. are the most important here.

What size and dimension supercell should I use?

In the previous homeworks, we made our supercells along the *z* direction, because we did not care about periodic boundary conditions along the *x* and *y* directions. In this case, we will see long wavelength fluctuations in all directions. Thus, instead of a 1x1x10 supercell (for example) a 3x3x3 supercell is more suitable. A 4x4x4 supercell is probably sufficient but the transition temperature will change a function of supercell size – this is one of the issues you can investigate.

Remember – your melting temperature will change with supercell size. An extremely small cell doesn't allow for long-wavelength fluctuations of atoms. This effect will definitely change the melting temperature. Always consider your simulation before picking your supercell size.

The MOLDY website says that the code is unconventional in that it doesn't use the minimum image convention for force calculations. What is the 'minimum image convention'?

With periodic boundary conditions, conventional codes will only allow an atom to interact with a single periodic image of another atom. (e.g. for a one dimensional periodic boundary condition, an atom at the origin will see an image of an atom at L/4 at x=L/4 and at x=-3L/4. By the minimum image convention, the interaction with the atom at x=-3L/4 is not considered.) For large enough cell sizes (i.e. L/2 greater than r_cutoff), the minimum image convention has little effect on the calculation. For smaller cells, codes which use the minimum image convention typically reduce the cutoff radius accordingly.

Why am I feeding the inputs of the previous temperature into the next temperature instead of starting over at each temperature?

One of the problems with MD simulations is getting good initial system parameters (positions, velocities, and accelerations). It takes some time for the system to equilibrate these quantities in a "new" run. The output of a previous temperature is a better starting point than a brand new run. For example, if there is premelting at a 550 K, the 550 K configuration is a better starting point for a calculation at 600 K than a 0 K configuration.

How do I know when I have a phase transition?

A few ways. One thing you can do is look at the potential energy or the total energy vs. temperature. A melting reaction is usually a *first-order* phase transition. (This means you see a discontinuity in the *first-order* derivatives of the FREE energy (F), not internal energy, U). Details of how the internal energy is a derivative of the free energy can be found in any introductory statistical mechanics book. You should think about the other ways yourself.

My run isn't working.

When asking for help from the TA, please show him your input files. Sometimes, the problem can be diagnosed right away.