Lab 4: Handout
Molecular dynamics.

In this lab, we will be using MOLDY as our molecular dynamics (MD) code. The MOLDY manual is online.

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

Some other recommended links

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

The first 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 GULP (used in lab 1), CHARMM, MOLDY, MOSCITO, MARVIN, DYNAMO, and many others.

In this lab, we look at melting of Ar. Ar crystallizes in the FCC structure (the same structure as Cu and Al). Experimentally, Ar melts at 84 K. We will use Lennard-Jones (LJ) potentials, which we previously used in lab 1.

The lattice parameter of Ar is given to you. To find the melting point of Ar, we will need to build Ar supercells. Supercells are necessary because of periodic boundary conditions. Without supercells, we can not see long-wavelength fluctuations in the materials. Next, we will need to find a suitable timestep, equilibration time, and sampling time. Then, we will investigate melting, in a few different ways.

Singapore students:

Your instructors, will give you the appropriate files

MIT students:

Your instructors will go over the location of the lab.

Cambridge students:

Your instructors, will help you log onto Armageddon and locate the appropriate files.

Quick summary of MD

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

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

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, $\vec{F}_i = -\frac{d}{d\vec{r}} U$,

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.

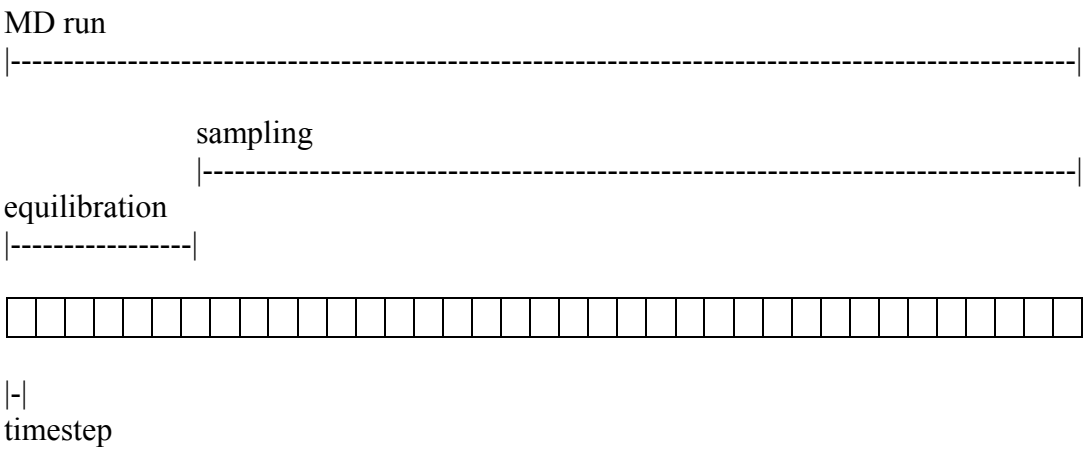
In class, you learned about the Verlet algorithm. This is the best-known time integration algorithm. Moldy uses the Beeman algorithm, which is a predictor-corrector type algorithm. The algorithm is outlined below.

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

x represents coordinates, and $\dot{x}^{(p)}$ and $\dot{x}^{(c)}$ to represent ``predicted" and ``corrected" velocities respectively. 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 bunch of 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.

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

Also, remember from classical mechanics that

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

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 Ar. 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 in this case)
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)

Finding the equilibrium lattice parameter:

This is done for you. The lattice parameter of Argon that we will use is 5.263 Angstroms. Usually, you would have to do this yourself.

The input files

There are two input files given to you. They **Ar.in** and **control.Ar**.

The Ar.in file

```
1     Argon 256
2     1          0          0          0    36 0 Argon
3     end
4     Lennard-Jones
5     1 1 3.984 3.41
6     end
7     5.263 5.263 5.263 90 90 90 4 4 4
8     Argon 0 0 0
9     Argon .5 .5 0
10    Argon .5 0 .5
11    Argon 0 .5 .5
12    end
```

Line 1

This has the species name (Argon) 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 (36), charge (0), and name.(Argon) and the number of atoms (256). 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

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

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.Ar file

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

The other input file is control.Ar

```
1   Title=Argon
2   sys-spec-file=Ar.in
3   # TIME-TIMESTEP (picoseconds)
4   step=1
5   nsteps=5000
6   # AVERAGES
7   begin-average=500
8   average-interval=100
9   # Temperature
10  temperature=90
11  scale-interval=200
12  scale-end=1000
13  scale-options=4
14  # Pressure
15  const-pressure=1
16  w=1000.0
17  pressure=0.
18  strain-mask=238
19  # INITIAL CONDITION
20  lattice-start=1
21  # DUMP-RESTART
22  begin-dump=1001
23  dump-interval=50
24  dump-level=3
25  ndumps=100
26  dump-file=Ardump
27  restart-file =
28  save-file= Ar.restart
29  backup-file=
30  # Output
31  page-length=44
32  page-width=80
33  print-interval=200
34  roll-interval=200
35  # RDF
36  rdf-interval=20
37  begin-rdf=2000
38  rdf-limit=10.0
39  nbins=200
40  rdf-out=2500
41  # CUTOFF
42  cutoff=10
43  strict-cutoff=1
44  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 Ar.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 quanitites. **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.

Lines 19-20

Lattice-start controls the initial configuration. If lattice-start is set to 1, then it will read the Ar.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

Moldy is run by typing

```
mit3320@master$ moldy< (control file here) > (output file here)
```

for example,

```
mit3320@master$ moldy< control.Ar > Ar.out.10
```

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

We have provided a script for you to change the temperature of your simulations, **tempscript**. Note that this file has a few parameters that are not correctly set (such as timestep!!) so you should look over it before you use it**. As input, it requires the output of an old run with output Ar.restart.<temperature here>.** This means you may have to copy your Ar.restart to Ar.restart.<temperature here>. For instance, you might do a run at 90 K first, before using tempscript to find MD data at 95, 100, 110, and 140 K. Remember that you use the output of your old temperature as input into your new temperature, because this procedure gives better initial conditions. The script is briefly outlined below.

```
1   #!/bin/bash
2  oldtemp=90
3
4  for temp in   95  100 110   140
5  do
6  cat>control.Ar.$temp<<EOF
7.8.9….
10  dump-file=Ardump.$temp
11  restart-file =Ar.restart.$oldtemp
12  save-file= Ar.restart.$temp
13  backup-file=
14   EOF
15  moldy<control.Ar.$temp>Ar.out.$temp
16  oldtemp=$temp
17     done
```

In this case, the lines number are for labeling only: they do not correspond to actual lines of the script that you have.

Line 1

This gives the type of script. Do not change.

Line 2

This gives the temperature of the initial run that you performed. In this case, we did a run at 90 K

Line 4,5,6,14,18

This is the for..do..done and cat..EOF constructions you have seen in the other labs. Note, this script will create a control file with the temperature at the end. That is, at 95 K, the control file will be named control.Ar.95.

Line 7.8.9…

This is the stuff that you will put in your control file. This has been left out here, but you can look at the control.Ar explanation earlier in this handout for an example of what it might be.

Lines 10-13

This controls the input/output. The dump-file instruction means the run will send dump information to Ardump.temperature (for instance, Ardump.95**. The program sticks an extra 0 at the end of dumpfiles**). The save-file instruction means it will also save information from the current run (for instance Ar.restart.95). The restart-file means that the program will obtain initial positions, velocities, and accelerations from the input file (for instance Ar.restart.90).

Lines 15-16

Here, the script runs the program (moldy), and sets the current temperature to the old temperature before restarting the loop.

Tools for data analysis

Data analysis: visualizing

Note, the visualizer only works from the master node (for Cambridge, and MIT. Singapore has the PC version). **You must be logged onto the master node for the visualizer to work.**

We will use a program called gdis. Gdis is a free visualizer, which can read a variety of input files. Gulp is one of the input types supported by gdis.
The webpage is http://gdis.seul.org

Gdis is only on xterminals, or requires Xwindows. This means if you are using secureCRT from a PC, it will not work (unless you have a an Xwindows system with X-forwarding enabled).

To prepare a file for gdis input, use the script **makegin.pl**

To run this script, type

```
mit3320@master$ makegin.pl <restart file>
```

For example,

```
mit3320@master$ makegin.pl Ar.restart.100
```

If you forget how to use this script, type

```
mit3320@master$ makegin.pl
```

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

The script works on restart files, so it will give you a picture of what the run looks like AFTER ther run.

Now, you are ready to use the GDIS visualizer.

To run GDIS, type

```
mit3320@master$gdis
```

If this does not work, you may have to export your display. Type

```
Lambic$: export DISPLAY=<your machine name here>:0.0
```

You can find your machine name by typing "uname –n".

To open a file:
On the top row, go to File->load and open the file you want. Again, all input files for gdis must end in ".gin".

To rotate the picture
Move the mouse to the picture, hold down the right mouse button, and move the mouse around

To change from atoms as balls to atoms as points
To speed up the visualization, you may want to change the way atoms are portrayed.

To save and print

Click on the thing that looks like a yellow ball in a cube. It will say "display properties" if you hold your mouse over it. Go to POVRAY->Render Image. Hold right click, go to "save", and save the file you want, with the format you want (pick format after hitting "save" to change formats).

Data analysis: Radial distribution function.

The Radial distribution function gives the density of an atom ρ, 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

```
mit3320@master$ makerdf.pl <output filename>
```

For example,

```
mit3320@master$ makerdf.pl Ar.out
```

This will create a file rdf.out that contains rdf information.

If you forget how to use this script, type

```
mit3320@master$ makerdf.pl
```

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

Data analysis: Potential energy vs. temperature.

U vs. T at one temperature:

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

```
mit3320@master$ getUvT.pl <output filename>
```

For example,

```
mit3320@master$ getUvT.pl Ar.out.100
99.4720   -625.4192
```

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 99.4720 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

```
mit3320@master$ 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

```
mit3320@master$ getallUvT.pl
```

This script takes all output files (if they are named Ar.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:

$$6Dt = \langle |r(t)-r(0)| \rangle .$$

To calculate msd, type

```
mit3320@master$ getmsd.pl <dump filename> <number of atoms)
```

For example:

```
mit3320@master$ getmsd.pl Ar.dump.1000 108
```

for a run at 100 K with 108 atoms. 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 time (ps), the second column is mean squared displacements (angstroms sq.). If you forget how to use this command, type

```
mit3320@master$ getmsd.pl
```

with no arguments.

FAQ for runs:

What is a "normal" timestep?

It depends on the material. Usually, you take the type of excitation you are interested in, and divide by ~100. In a metal, 1-10 ps is typically chosen. For a lighter material, or for higher temperatures, a shorter timestep is 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. Since the code doesn't directly output H, NVE is easier.

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 iternations 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 timestepx3000 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 and 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 big 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.

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 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.