Determination of Magnetic Transitions: Monte Carlo Simulations

Magnetic Properties from First Principles, TÜBİTAK Workshop, Nov 24-26, 2021

This notebook presents a case study of monolayer CrI_3 to illustrate a typical workflow for determining finite-temperature magnetic properties.
Yen Lee Loh, 2021-11-20

> This session will teach you how to:
> – use VASP to do collinear and non-collinear magnetic calculations
> – find exchange couplings that "best fit" DFT energies using least squares (linear regression)
> – avoid common pitfalls
> – appreciate Ising, Heisenberg, Mermin-Wagner, etc.

1 Code (please run this section before doing anything else)

2 Code: example usage

3 Attempt 1: Fit spin configurations of Cr_2I_6 unit cell with honeycomb Ising model with J_1

3.1 Relax structure

Begin by setting up the crystal structure of monolayer CrI_3. Make sure that you know how to construct the four VASP input files (`INCAR`, `POSCAR`, `POTCAR`, `KPOINTS`) as well as a script file (`script`) for launching jobs on a cluster that uses SLURM or PBS for scheduling.

– Establish working directory: `CrI3/1RELAX`
– Download bulk CrI_3 structure (.cif) from https://materialsproject.org/materials/mp-1213805/
– Convert CIF file to POSCAR file (using VESTA)
– Exfoliate to obtain monolayer CrI_3 (by editing POSCAR manually)
– Symmetrize the structure (optional)
– Build POTCAR for chromium and iodine atoms: `buildPOTCAR.py Cr I`
– Relax ions using standard protocols (various ISIF settings).
  – Prepare INCAR, KPOINTS
  – Launch VASP (prepare script file, rsync everything to cluster, and `sbatch script`)
  – As many times as desired, `cp CONTCAR POSCAR` , change ISIF settings, and rerun VASP
  – If using `ISIF=3`, unit cell may begin to collapse along c direction. Watch out for this.

My input files look something like this. This is for illustration purposes. For serious work you should probably use a higher ENCUT.

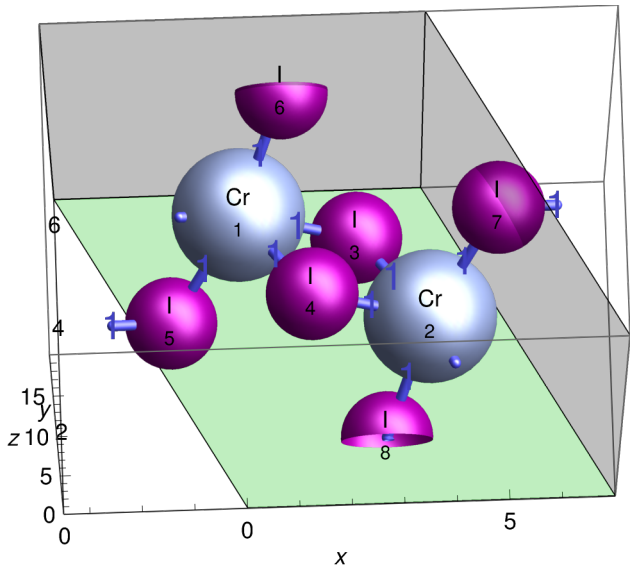
```
-------- INCAR --------
SYSTEM = CrI3 monolayer
ISTART = 0        # fresh start
IBRION = 2        # relax ions using conjugate gradient method
#ISIF   = 3       # relax positions, relax shape, relax volume
```

```
ISIF    = 4        # relax  positions,  relax  shape,  fix  volume
NSW     = 100      # maximum  number  of  ionic  steps
ENCUT   = 300
EDIFF   = 1e-8
EDIFFG  = -1e-4
-------- POSCAR  --------
 Cr2  I6
   1.00000000000000
     7.0016664865505183      -0.0000368755465378       0.0003713610401982
    -3.5008651784353364       6.0636026084056160       0.0003713610401982
     0.0010616927552912       0.0018389057941922      19.9906024167027709
   Cr    I
     2     6
Direct
  0.3333098275383799     0.6666901969700723     0.5000000000002416
  0.6666414958816236     0.3333584796099245     0.4999999999997584
  0.6239074716026258     0.6239564096497289     0.4242266463927612
  0.3760435498062408     0.3760925048506276     0.5757733214312347
  0.0000182951352539     0.3761849517682309     0.4242262602922209
  0.6238150676947070     0.9999816813359261     0.5757737075323648
  0.9999321766263749     0.6238661688504205     0.5757743472356924
  0.3761338406177815     0.0000678248441074     0.4242256205890422
-------- KPOINTS  --------
COMMENT  LINE
 0
Monkhorst-Pack
 4  4  1
 0.  0.  0.
-------- script  --------
...
mpirun /home/export/apps/  vasp.5.4.1/bin/vasp_std   >& script.out
```

Below is the unit cell of the relaxed structure (after copying CONTCAR to POSCAR).

In[333]:=
```
unitCell = readPOSCAR ["1RELAX/POSCAR "];
drawStructure [unitCell , BondCutoffDistance → 2.9 , ImageSize → 324]
```

Out[334]=

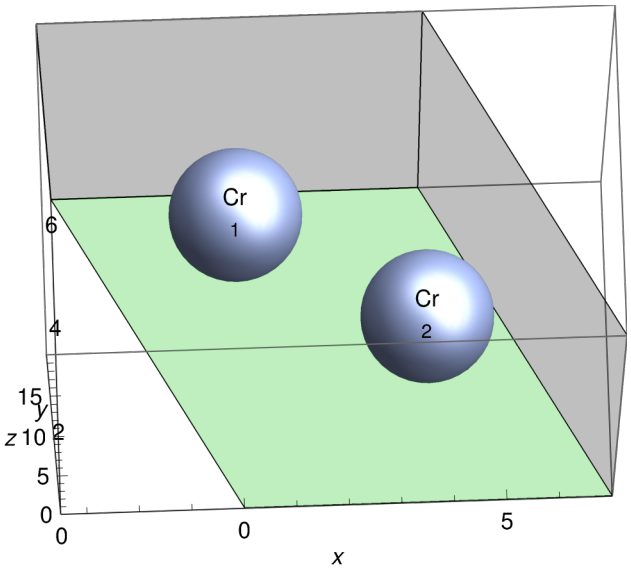


Below is the magnetic "skeleton" consisting of only the Cr atoms. The Cr atoms form a honeycomb lattice, although this is not obvious from a picture of one unit cell only. The main point is that there are

two Cr atoms per unit cell.

In[335]:=
```
{skeleton , map1 , map2 } = filterElements  [unitCell , {"Cr", "Fe", "Mn", "Ni", "Co"}];
drawStructure  [skeleton , BondCutoffDistance   → 1, ImageSize  -> 324 ]
```

Out[336]=

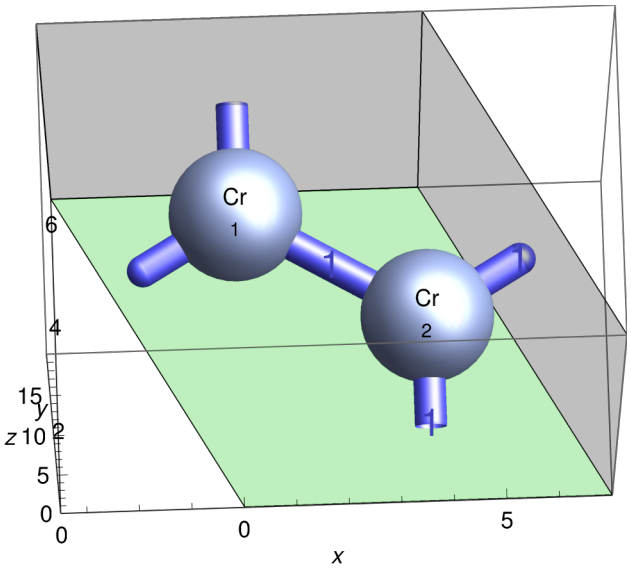


3.2 Design an effective spin model

Let's attempt to model the magnetism of CrI_3 using a nearest-neighbor Ising model, $E(\{S_i\}) = NE_0 - \sum_{\langle ij \rangle} J S_i S_j$. Don't worry about details yet. Basically, there is only one type of bond (bond type "1"), which is assumed to have exchange coupling J, as depicted below.

In[337]:=
```
Export ["bonds .dat", #, "String "] &@"
#i   j    u   v   w   b
#------------------
 1   2    0   0   0   1
 1   2   -1   0   0   1
 1   2    0   1   0   1";
bonds = loadBonds ["bonds .dat"];
drawStructure  [skeleton , Bonds → bonds , BondRadii  → .3] // Show [#, ImageSize  -> 324 ] &
```

Out[339]=

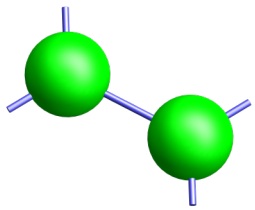


3.3 Calculate energies of FM and AF spin configurations

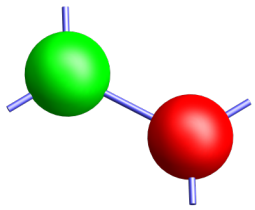
Monolayer CrI_3 contains two inequivalent Cr atoms that form a honeycomb lattice. The formula unit is Cr_2I_6. This makes it easy to set up a ferromagnetic (FM) configuration and an antiferromagnetic (AF) configuration, so that we may attempt to determine the nearest-neighbor exchange coupling. The code below uses my `drawSpins[]` utility to visualize the configurations, with ↑ in green and ↓ in red:

In[340]:=
```
drawSpins [skeleton , {+1, +1}, Bonds → bonds , ImageSize → 200 , Axes → None , Boxed → False , Arrows → False ]
drawSpins [skeleton , {+1, -1}, Bonds → bonds , ImageSize → 200 , Axes → None , Boxed → False , Arrows → False ]
```

Out[340]=



Out[341]=



We will calculate the energies of the FM and AF configurations with the ions fixed in place. (In principle, one should allow the ions to relax, because sometimes there are magnetostriction effects where the spin couples to the lattice. Here I am just doing a basic demo.) INCAR files look like this.

```
-------- 2FM/INCAR  --------
SYSTEM  = CrI3  collinear  magnetic  calculation
ISTART  = 0            # fresh  start
IBRION  = 0            # fix  ions
ISPIN     = 2          # collinear  magnetism
AMIX      = 0.2        # customary  parameters  for  magnetic  calculations
BMIX      = 0.0001
AMIX_MAG  = 0.8
BMIX_MAG  = 0.0001
LMAXMIX   = 4
LORBIT    = 11         # output  charges  and  moments  on  each  atom
MAGMOM    = 2*3 6*0    # initial  magnetic  moments  are  +3 muB  on  each  Cr  ion  and  0 muB  on  each  I  ion
-------- 3AF/INCAR  --------
...
MAGMOM    = +3.0  -3.0  0 0 0 0 0 0      # initial  moments  are  +3 and  -3 on  the  two  Cr  atoms
```

3.4 Verify magnetizations from CHGCAR

After running VASP we should do some sanity checks. I have set up a file that allows me to plot the magnetization by running

```
ovito mag.ovito .
```

Alternatively, we can do this within Mathematica.

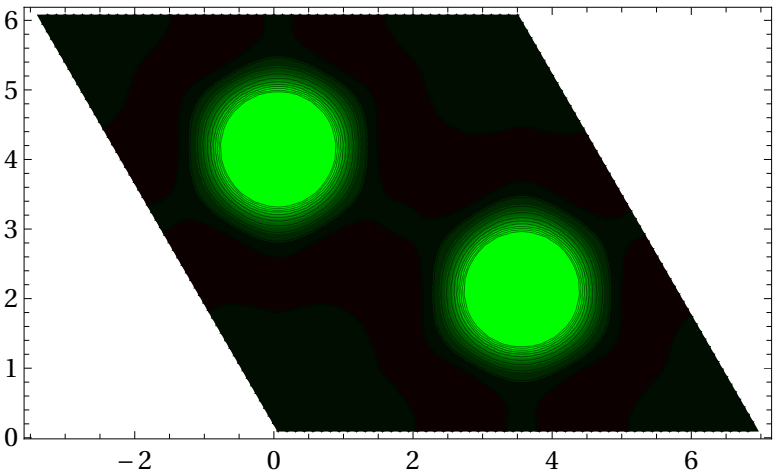
– Load magnetization $M(u, v, w)$ voxel grid data from CHGCAR

– Calculate magnetization projected along the c direction, $M(u, v) = \sum_w M(u, v, w)$

– Make contour plot of projected magnetization $M(x, y)$

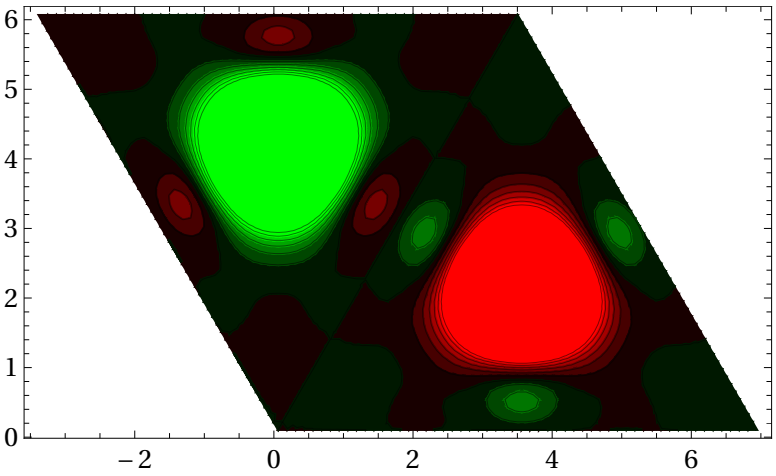
In[457]:=
```
{struc , rhouvw , muvw } = readCHGCAR ["2FM/CHGCAR "];
muv = Total [muvw , {3}]; (* project along c direction *)
plotMuv [{umax , vmax }, struc @ BasisVectors , muv ]
```

Out[459]=



In[460]:=
```
{struc , rhouvw , muvw } = readCHGCAR ["3AFM/CHGCAR "];
muv = Total [muvw , {3}]; (* project along c direction *)
plotMuv [{umax , vmax }, struc @ BasisVectors , muv ]
```

Out[462]=



The first simulation had initial moments $(+3, +3)\,\mu_B$ on the Cr ions. The second simulation had initial moments $(+3, -3)\,\mu_B$. The figures above show the $M(x, y)$ patterns. Indeed the simulations have remained in the FM and AFM states respectively.

Moreover, according to the OUTCAR file, the FM configuration has magnetic moments $(+3.025, +3.025)\,\mu_B$ on the Cr ions, and the AFM configuration has $(+2.952, -2.952)\,\mu_B$. The OUTCAR also tells us that most of this comes from the Cr 3d orbitals. This indicates that the magnetization is well localized on the Cr^{3+} ions.

3.5 Fit energies using nearest-neighbor Ising model on honeycomb lattice

Now that we have confirmed that thee compare the energies of the FM and AFM states:

In[343]:=
```
EFM = extractE0 [NotebookDirectory  [] <> "/2FM/OSZICAR "]
EAF = extractE0 [NotebookDirectory  [] <> "/3AFM/OSZICAR "]
dE = EAF - EFM
```

Out[343]= -31.2423

Out[344]= -31.2349

Out[345]= 0.007338

The energy difference between FM and AFM configurations, per Cr_2I_6 unit cell, is

$\Delta E = 7.338$ meV.

Let us fit this data to an Ising model described by the energy function

$$E(\{\sigma\}) = E_0 - \sum_{\langle ij \rangle} J \sigma_i \sigma_j$$

where $\sigma_i \in \{-1, +1\}$ are variables representing Ising spins, J is the ferromagnetic nearest-neighbor coupling, and $\langle ij \rangle$ represents a sum over all nearest neighbors. Draw a diagram carefully and count bonds. We see that the energy per unit cell, according to this model, is

$E_{\text{FM}} = E_0 - 3\,J$

$E_{\text{AF}} = E_0 + 3\,J$

$\therefore \quad \Delta E = 6\,J.$

Fitting this to the DFT data gives

$J = \frac{1}{6}\,\Delta E \approx 1.223$ meV ≈ 14.2 K.

Calculation using linear regression formalism (overkill)

3.6 Predict finite-temperature magnetic properties: mean-field theory

Quick summary of mean-field theory (MFT) for Ising model:

$$H_{\text{Ising}}(\{S\}) = \text{const} - \sum_{\langle ij \rangle} J_{ij}\, S_j\, S_i$$

$$\therefore \quad H_{\text{MF}}(\{S\}) = \text{const} - \sum_i h_i\, S_i, \qquad h_i = \sum_{j@i} J_{ij} \left\langle S_j \right\rangle \qquad \text{where } j@i \text{ means "}j \text{ is a neighbor of } i\text{"}$$

$$\therefore \quad \langle S_i \rangle = \frac{\sum_{S_i} e^{\beta h_i S_i}\, S_i}{\sum_{S_i} e^{\beta h_i S_i}} = \frac{e^{\beta h_i} - e^{-\beta h_i}}{e^{\beta h_i} + e^{-\beta h_i}} = \tanh \beta h_i \qquad \text{where } \beta = 1 / k_B T.$$

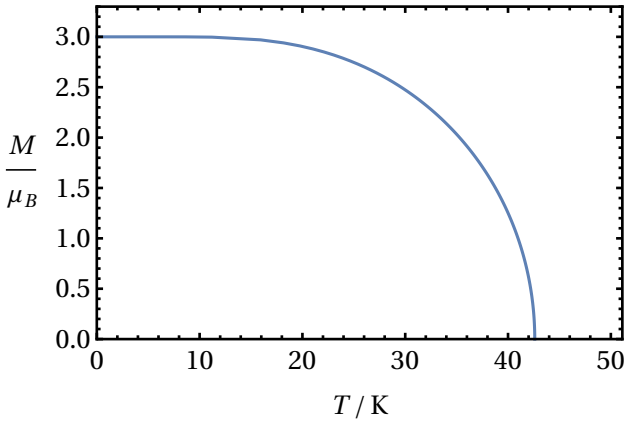
For simple lattices $\langle S_i \rangle$ is the same on all sites. Write $\langle S_i \rangle = M$. Then

$M = \tanh \frac{zJM}{T}$ where z is the coordination number (number of nearest neighbors).

Given T, can solve for M. Or rewrite as $T = zJM \,/\, \text{arctanh}\ M$. Thus $\frac{T}{T_c} = \frac{M}{\text{arctanh}\ M}$ and $T_c = zJ$. For the honeycomb lattice, each atom has three nearest neighbors, so $z = 3$. Put in numbers:

In[346]:=
```
J = 14.2 ;(* nn exchange coupling in Kelvin *)
M0 = 3.00 ; (* assume fixed moment of 3μB *)
M = Range [.00001 , .99999 , .01]~Union ~{.999 , .9999 , .99999 , 1. - 10^-10};
ListLinePlot [{M / ArcTanh [M] * 3 J, M0 * M}^T ~Join ~{{0, M0}}, ImageSize → {324 , 216},
 AspectRatio → Full , FrameLabel → {"T / K", "M / μB"}, PlotRange → {{0, M0 * J * 1.2}, {0, 1.1 M0}}]
Clear [
 M]
```

Out[349]=



The curve above shows the predictions of MFT with our model. The magnetization starts at $M(0) = 3\ \mu_B$ and drops to zero at

$$T_c^{\mathrm{MF}} = 42.6\ \mathrm{K}.$$

However, mean-field theory is not accurate in 2D, especially for lattices with low coordination number ($z = 3$). We should not believe it.

We can get better results using Monte Carlo simulations.

Calculation using linear regression formalism (overkill)

3.7 Predict finite-temperature magnetic properties: Monte Carlo

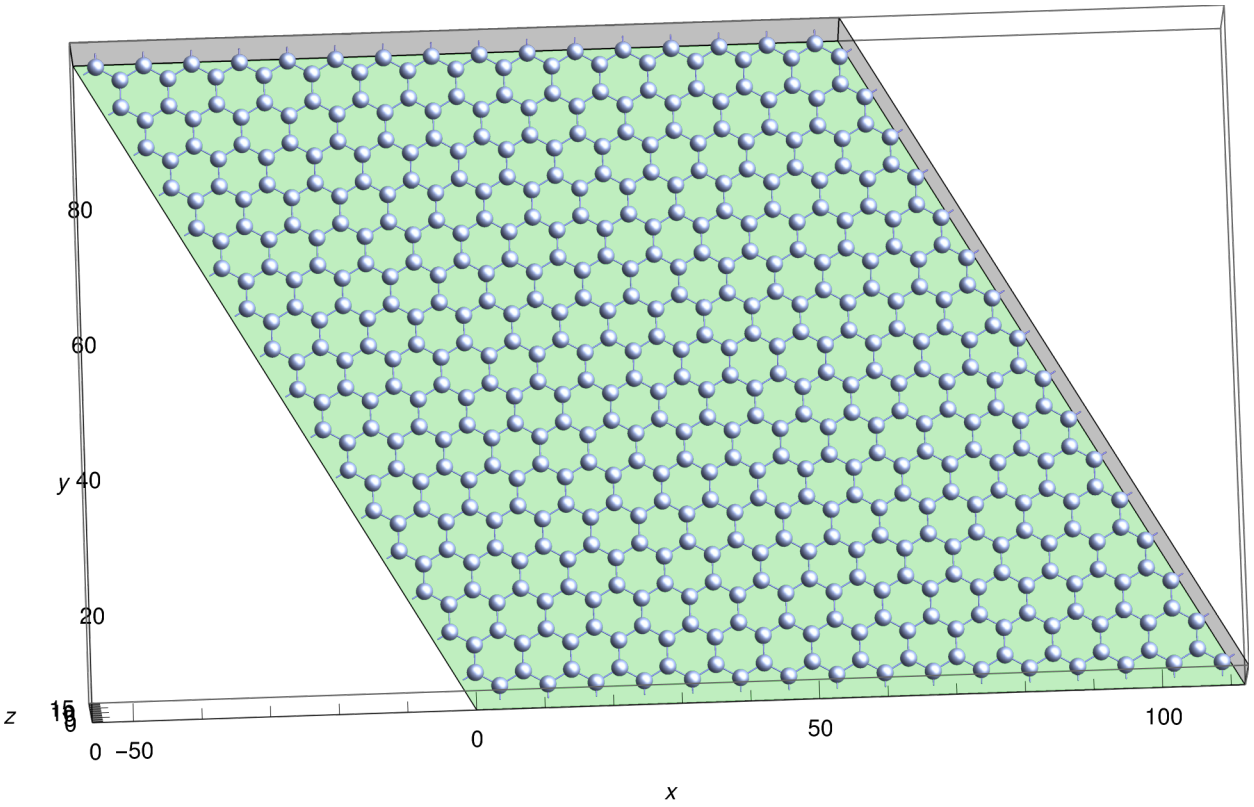
Some details about code

Monte Carlo animation

Let us run MC on a 16×16 lattice. Prepare `mclattice.dat, mcbonds.dat, pars.dat, Jb.dat`:

In[761]:=
```
mcLattice = replicatePOSCAR [skeleton , {16 , 16 , 1}];
mcBonds = replicateBonds [bonds , {16 , 16 , 1}];
drawStructure [mcLattice , Bonds → mcBonds , ElementLabels → None , AtomLabels → None , BondLabels → None ]
writePOSCAR [mcLattice , "MC/mclattice .dat "]
Export ["MC/mcbonds .dat ", mcBonds ᵀ, "Table "];
```

Out[763]=




```
Jb = {14.2 };
Export ["MC/Jb .dat ", Jb , "Table "];

temperature = 60.0 ;
Export ["MC/pars .dat ", #] & @ subst @"
useGUI         = 1                        # show graphics
stepsTuning    = 999999999
stepsCollect   = 0                        # if stepsCollect =0 then run Mickey Mouse animation
timeTuning     = 180                      # run for 3 minutes
timeCollect    = 0
spinType       = 1                        # Ising model
temperature    = ${temperature }
";
```

In[770]:=
```
(*-------- MAKE EXECUTABLE (I.E., COMPILE AND LINK ) --------*)
err = Run["(cd MC; make -B mc > make .log 2>&1)"];
If[err == 0, Print ["Compiled executable mc successfully !"], FilePrint ["MC/make .log "]];
```

```
Compiled executable mc successfully !
```

In[771]:=
```
(*-------- RUN --------*)
err = Run["(cd MC; ./mc > make .log 2>&1)"];
If[err == 0, Print ["Ran executable mc successfully !"], FilePrint ["MC/run .log "]];
```

```
Ran executable mc successfully !
```

Instead of running from Mathematica, it is better to open a terminal, cd to the `MC` directory, and run `./MC` so that you can see the usage info and GUI controls in the terminal.

Serious Monte Carlo calculation for fixed system size L and many temperatures T

Now let us do a serious calculation. Turn off the graphics. Launch multiple runs:

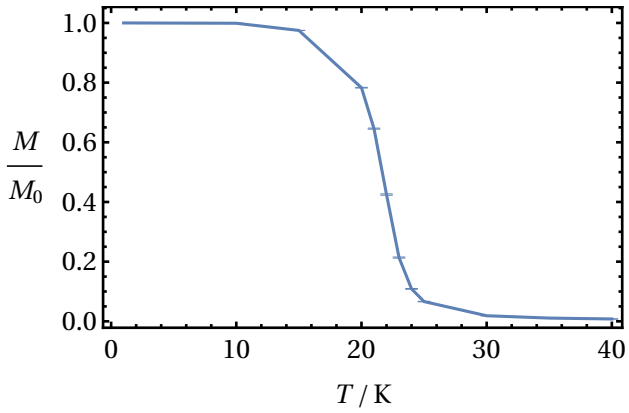
```
L = 16;
mcLattice = replicatePOSCAR [skeleton, {L, L, 1}];
mcBonds = replicateBonds [bonds, {L, L, 1}];

TList = {1, 10, 20, 25, 30, 35, 40}; (*TList ={15,21,22,23,24};*)
(* you can return to this point to do additional runs *)
Jb = {14.2};
Do[
  (*-------- CREATE DATA DIRECTORY --------*)
  codeDir = Directory [] <> "/MC";     (* same code *)
  dataDir = subst @"MC/DATA/T${T}___"; (* different data directories *)
  Quiet @CreateDirectory [dataDir, CreateIntermediateDirectories -> True];
  (*-------- PREPARE THE 4 INPUT FILES --------*)
  Export [subst @"${dataDir}/Jb.dat", Jb, "Table"];
  writePOSCAR [mcLattice, subst @"${dataDir}/mclattice.dat"];
  Export [subst @"${dataDir}/mcbonds.dat", mcBonds ᵀ, "Table"];
  Export [subst @"${dataDir}/pars.dat", #] & @subst @"
useGUI        = 0               # don't show graphics
stepsTuning   = 10000            # Fix number of steps
stepsCollect  = 100000           # Fix number of steps
timeTuning    = 3600             # quit if not done after 1 hour
timeCollect   = 3600             # quit if not done after 1 hour
spinType      = 1                # Ising
temperature   = ${T}           # THIS IS DIFFERENT IN EACH DIRECTORY
";
  (*-------- RUN --------*)
  Print [subst @"Running mc in ${dataDir}..."];
  err = Run [subst @"(cd ${dataDir}; ${codeDir}/mc > run.log 2>&1)"];
  , {T, TList}];
```

```
Running mc in MC/DATA/T15___ ...
Running mc in MC/DATA/T21___ ...
Running mc in MC/DATA/T22___ ...
Running mc in MC/DATA/T23___ ...
Running mc in MC/DATA/T24___ ...
```

Collect the data and plot with error bars:

In[983]:=
```
dataDirs = FileNames ["MC/DATA/*___"]; (* get list of directory names *)
data = Table [
    (*-------- IN EACH DATA DIRECTORY ,
    EXTRACT T PARAMETER FROM pars.dat BY SEARCHING FOR "temperature = ..." --------*)
    T = Import [subst @"${dataDir }/pars.dat", "String"] // StringSplit [#, "temperature "][[-1]] & // StringSplit [#, "="][[2]] & //
        StringSplit [#, "#"][[1]] & // StringToStream // Read [#, Number ] &;
    stats = Import [subst @"${dataDir }/stats.dat"];
    M2 = stats [[2, 1]] ~Around~ stats [[2, 2]]; (* squared magnetization is 2nd row in stats.dat *)
    M4 = stats [[3, 1]] ± stats [[3, 2]]; (* 4th power of magnetization is 3rd row in stats.dat *)
    {T, M2, M4} (* T, M2, etc., are numbers *)
    , {dataDir , dataDirs }];
{T, M2, M4} = data^T;(* now T, M2, etc., are arrays *)
ListPlot [{T, M2}^T, PlotRange → All, Joined → True, FrameLabel → {"T/K", "M/M0"}]
```

Out[984]=



Serious Monte Carlo calculation for many sizes and many temperatures T

Now write a big loop. (You may wish to delete everything in the MC/DATA directory first.)

```
LList = {4, 8, 16};
TList = {1, 10, 15, 20, 25, 30, 35, 40, 50};(*TList ={15,21,22,23,24};*)
(*  you can return to this point to do additional runs *)
Jb = {14.2};
Do[
  (*-------- CREATE DATA DIRECTORY --------*)
  codeDir = Directory [] <> "/MC";     (* same code *)
  dataDir = subst @"MC/DATA/L${L}_T${T}"; (* different data directories *)
  Quiet @CreateDirectory [dataDir, CreateIntermediateDirectories    → True];
  (*-------- PREPARE THE 4 INPUT FILES --------*)
  mcLattice = replicatePOSCAR [skeleton, {L, L, 1}];
  mcBonds = replicateBonds [bonds, {L, L, 1}];
  Export [subst @"${dataDir }/Jb.dat", Jb, "Table "];
  writePOSCAR [mcLattice, subst @"${dataDir }/mclattice .dat"];
  Export [subst @"${dataDir }/mcbonds .dat", mcBonds ᵀ, "Table "];
  Export [subst @"${dataDir }/pars .dat", #] &@subst @"
useGUI        = 0               # don't show graphics
stepsTuning   = 10000            # Fix number of steps
stepsCollect  = 100000           # Fix number of steps
timeTuning    = 3600             # quit if not done after 1 hour
timeCollect   = 3600             # quit if not done after 1 hour
spinType      = 1                # Ising
temperature   = ${T}           # THIS IS DIFFERENT IN EACH DIRECTORY
systemSize    = ${L}           # THIS IS NOT READ BY THE C++ CODE, IT IS JUST FOR IDENTIFICATION
";
  (*-------- RUN --------*)
  Print [subst @"Running mc in ${dataDir }..."];
  err = Run [subst @"(cd ${dataDir }; ${codeDir }/mc > run.log 2>&1)"];
  , {L, LList }, {T, TList }];
```

```
Running mc in MC/DATA/L4_T50 ...
```

```
Running mc in MC/DATA/L8_T50 ...
```

```
Running mc in MC/DATA/L16_T50 ...
```

Collect the data and display in a big table:

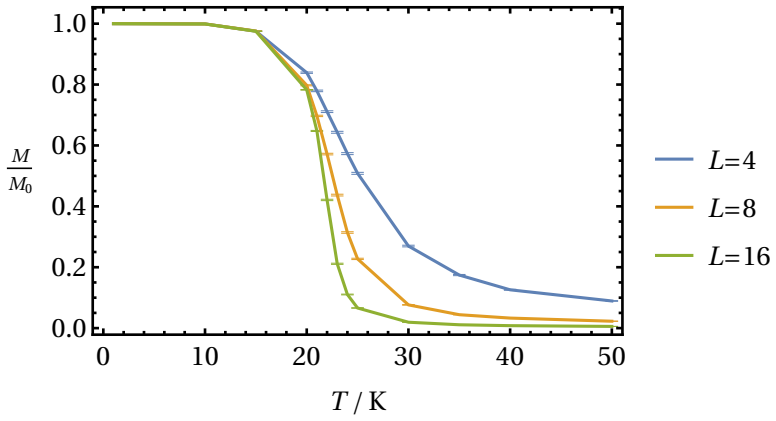
In[150]:=
```
dataDirs = FileNames ["MC/DATA/L*T*"]; (* get list of directory names *)
data = Table [
    (*-------- IN EACH DATA DIRECTORY , EXTRACT L and T by "temperature = ..." and "systemSize = ..." --------*)
    L = Import [subst @"${dataDir }/pars .dat", "String "] // StringSplit [#, "systemSize "][[-1]] & // StringSplit [#, "="][[2]] & //
        StringSplit [#, "#"][[1]] & // StringToStream // Read [#, Number ] &;
    T = Import [subst @"${dataDir }/pars .dat", "String "] // StringSplit [#, "temperature "][[-1]] & // StringSplit [#, "="][[2]] & //
        StringSplit [#, "#"][[1]] & // StringToStream // Read [#, Number ] &;
    stats = Import [subst @"${dataDir }/stats .dat"];
    M2 = stats [[2, 1]] ~Around ~ stats [[2, 2]]; (* squared magnetization is 2nd row in stats .dat *)
    M4 = stats [[3, 1]] ~Around ~ stats [[3, 2]]; (* 4th power of magnetization is 3rd row in stats .dat *)
    {L, T, M2, M4} (* T, M2, etc., are numbers *)
    , {dataDir , dataDirs }];
{L, T, M2, M4} = data ^T;(* now T, M2, etc., are arrays *)
data // TableForm [#, TableHeadings → {None , {"L", "T", "⟨M^2⟩", "⟨M^4⟩"}}] &
```

Out[153]//TableForm=

L	T	$\langle M^2 \rangle$	$\langle M^4 \rangle$
16	1	1	1
16	10	0.999035	0.99808 ±0.00005
16	15	0.97510 ±0.00019	0.9512 ±0.0004
16	20	0.7827 ±0.0011	0.6240 ±0.0015
16	21	0.6476 ±0.0017	0.4498 ±0.0019
16	22	0.4210 ±0.0023	0.2287 ±0.0018
16	23	0.2113 ±0.0019	0.0794 ±0.0012
16	24	0.1104 ±0.0013	0.0276 ±0.0006
16	25	0.0661 ±0.0008	0.01133 ±0.00028
16	30	0.01936 ±0.00027	0.001098 ±0.000033
16	35	0.01103 ±0.00016	0.000364 ±0.000011
16	40	0.00808 ±0.00011	0.00019 (3 ± 6)
16	50	0.00560 ±0.00008	0.00009 (28 ± 31)
4	1	1	1
4	10	0.99918 ±0.00011	0.99847 ±0.00020
4	15	0.9753 ±0.0007	0.9562 ±0.0012
4	20	0.8390 ±0.0022	0.7523 ±0.0029
4	21	0.7792 ±0.0026	0.6743 ±0.0033
4	22	0.7109 ±0.0029	0.5892 ±0.0035
4	23	0.6432 ±0.0031	0.511 ±0.004
4	24	0.5731 ±0.0032	0.4314 ±0.0035
4	25	0.5084 ±0.0032	0.3635 ±0.0034
4	30	0.2696 ±0.0027	0.1410 ±0.0023
4	35	0.1743 ±0.0020	0.0693 ±0.0014
4	40	0.1258 ±0.0016	0.0404 ±0.0009
4	50	0.0892 ±0.0011	0.0211 ±0.0005
8	1	1	1
8	10	0.99902 ±0.00006	0.99807 ±0.00013
8	15	0.9758 ±0.0004	0.9535 ±0.0007
8	20	0.7975 ±0.0017	0.6656 ±0.0023
8	21	0.6970 ±0.0022	0.5362 ±0.0026
8	22	0.5717 ±0.0026	0.3955 ±0.0027
8	23	0.4369 ±0.0029	0.2652 ±0.0025
8	24	0.3140 ±0.0025	0.1606 ±0.0020
8	25	0.2274 ±0.0022	0.0978 ±0.0015
8	30	0.0764 ±0.0010	0.0154 ±0.0004
8	35	0.0441 ±0.0006	0.00547 ±0.00016
8	40	0.0330 ±0.0005	0.00314 ±0.00010
8	50	0.02266 ±0.00033	0.00150 ±0.00005

In[154]:=
```
dataForL [L_] := data // Select [#, #[[1]] == L &] & // Transpose ;
(* this function filters data according to the L parameter *)
ListPlot [{
  {L, T, M2, M4} = dataForL [4]; {T, M2}ᵀ,
  {L, T, M2, M4} = dataForL [8]; {T, M2}ᵀ,
  {L, T, M2, M4} = dataForL [16]; {T, M2}ᵀ
}, PlotRange → All, Joined → True, FrameLabel → {"T/K", "M/M₀"}, PlotLegends → {"L=4", "L=8", "L=16"}]
```

Out[155]=



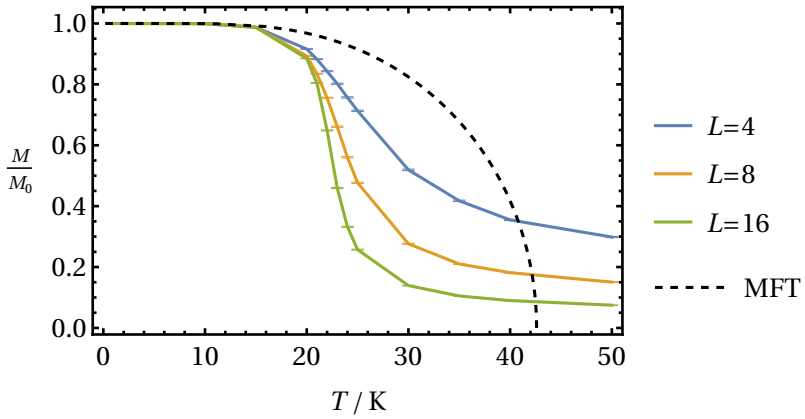
For our nearest-neighbor Ising model for CrI_3 with $J = 14.2$ K, the graph above shows $\langle M^2 \rangle$ as a function of T for different system sizes.

If you do a Monte Carlo simulation correctly, you will have $\langle M \rangle = 0$! You cannot get the spontaneous magnetization that way!

If you really wish, you can estimate the spontaneous magnetization as $\sqrt{\langle M^2 \rangle}$, as below.

In[156]:=
```
grMC = ListPlot [Table [{L, T, M2, M4} = dataForL [Lselected ]; {T, √M2 }ᵀ, {Lselected , {4, 8, 16}}],
    PlotRange → All , Joined → True , FrameLabel → {"T /K", "M/M₀"}, PlotLegends → {"L=4", "L=8", "L=16"}];
J = 14.2 ;(* nn exchange coupling in Kelvin *)
M0 = 1.00 ; (* assume fixed moment of 3μ_B *)
M = Range [.00001 , .99999 , .01]~Union ~{.999 , .9999 , .99999 , 1. - 10^-10};
grMFT = ListLinePlot [{M/ArcTanh [M] *3 J, M0 * M}ᵀ~Join ~{{0, M0}}, PlotStyle -> {Dashed , Black },
  ImageSize → {324 , 216}, AspectRatio → Full , PlotRange → {{0, M0 * J * 1.2}, {0, 1.1 M0}}, PlotLegends → {"MFT "}];
Clear [M];
Show [grMC , grMFT , PlotRange → {{0, 50}, All }]
```

Out[159]=

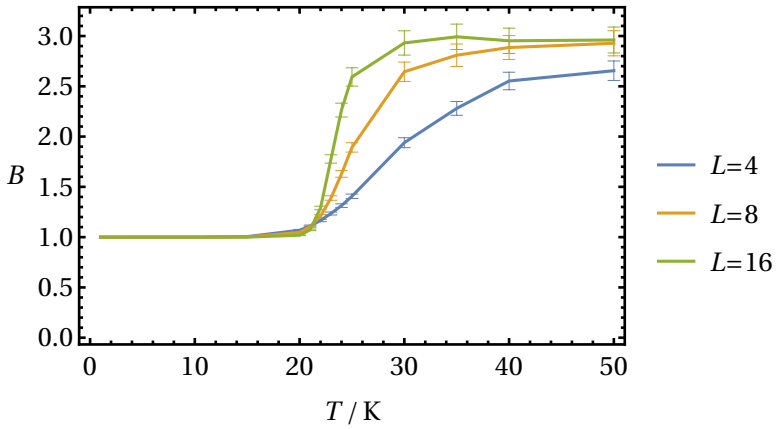


▲ As $L \to \infty$, the magnetization drops to zero near $T_c^{\mathrm{MC}} \approx 21$ K. The true critical temperature is about 50% lower than $T_c^{\mathrm{MFT}} = 42$ K!

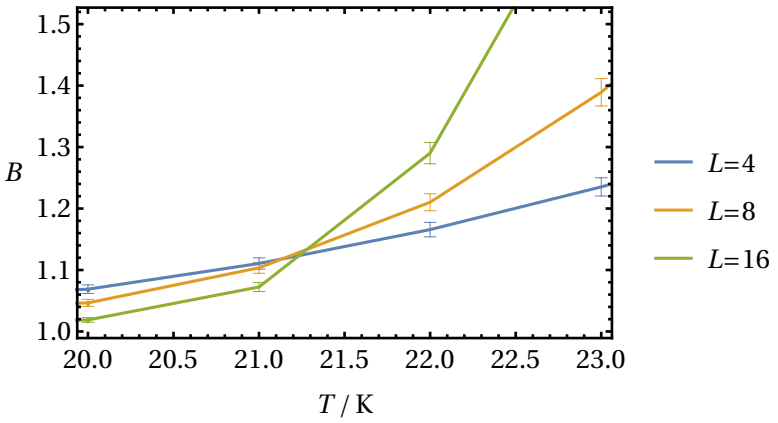
To locate T_c^{MC} accurately, we can do finite-size scaling of the above data. It turns out that a convenient way is to look for the Binder parameter crossing point:

In[160]:=
```
ListPlot [{
  {L, T, M2, M4} = dataForL [4]; B = M4 / M2^2; {T, B}^T,
  {L, T, M2, M4} = dataForL [8]; B = M4 / M2^2; {T, B}^T,
  {L, T, M2, M4} = dataForL [16]; B = M4 / M2^2; {T, B}^T
 }, PlotRange → All, Joined → True, FrameLabel → {"T / K", "B"}, PlotLegends → {"L=4", "L=8", "L=16"}]
% // Show [#, PlotRange → {{20, 23}, {1, 1.5}}, ImageMargins → All] &
```

Out[160]=



Out[161]=



▲ Define the quantity $B = \langle M^2 \rangle^2 / \langle M^4 \rangle$. This is a dimensionless number that varies from $B = 1$ at low T to $B = 3$ at high T. The crossing point determines the critical temperature:

$T_c^{\mathrm{MC}} \approx 21.3\ K$.

In this case we didn't really need Monte Carlo. Thermodynamic properties of many 2D Ising models are known analytically. For the Ising model on a honeycomb lattice, it is known that $T_c / J = 1 / \operatorname{arccoth} \sqrt{3} \approx 1.518651$. Therefore, according to our nearest-neighbor honeycomb lattice Ising model, the Curie temperature is

$T_c \approx 21.6$ K

in agreement with our MC.

In[162]:=
```
(*Quantity ["eV"]/Quantity ["BoltzmannConstant  "]//UnitConvert *)
J = dE/6; J
JinKelvin  = J * 16 021 766 340 / 1 380 649
Tc = 1. / ArcCoth [ √3 ] * JinKelvin
```

Out[162]= 0.001223

Out[163]= 14.1923

Out[164]= 21.5532

3.8 Attempt 1: $T_c \approx 22$ K

Using a nearest-neighbor Ising model, we have predicted $T_c \approx 22$ K.
According to experiments, the Curie temperature of monolayer CrI_3 is 45 K! It is unusual for order to be more stable in experiment than in theory. This suggests that we should try improving the model.

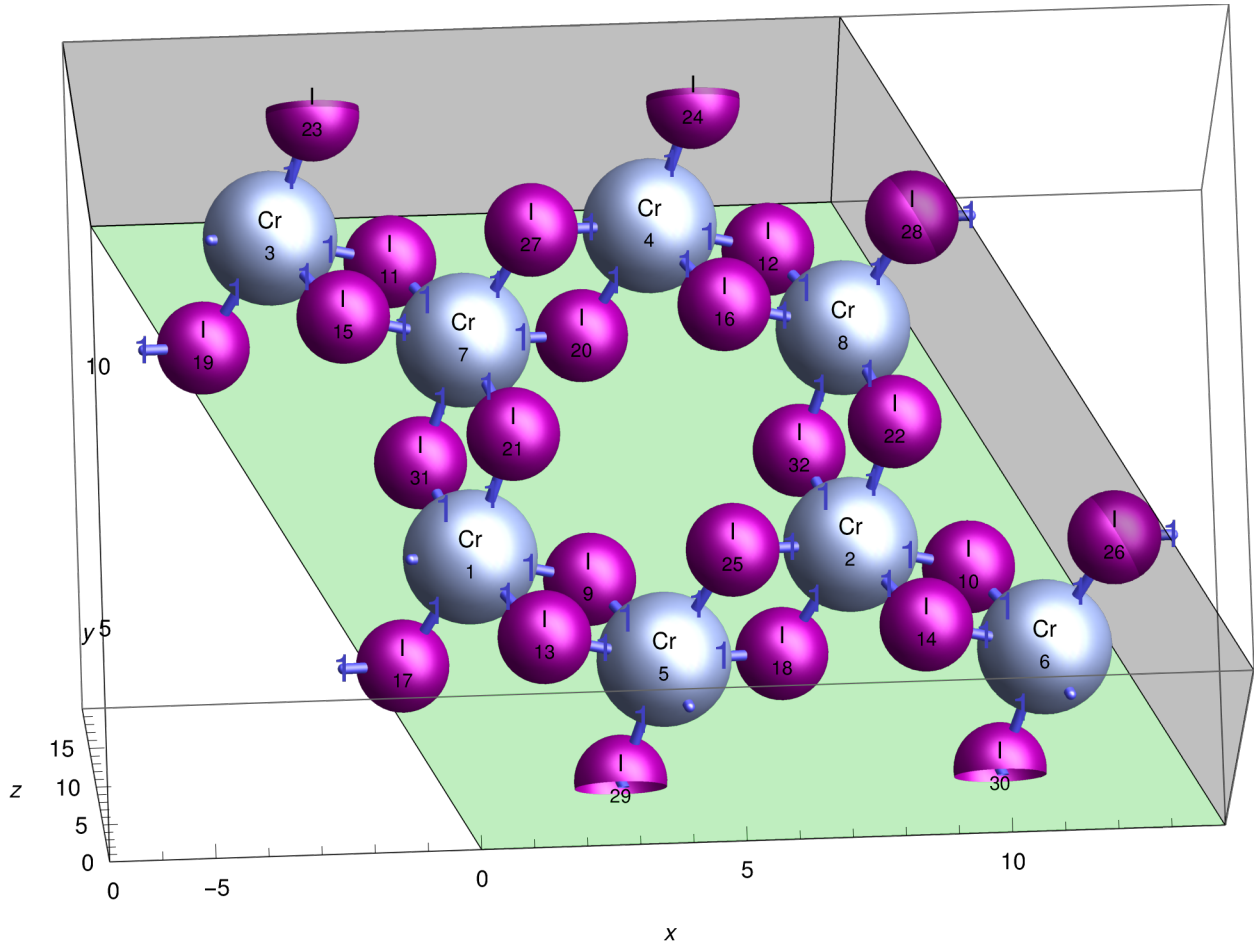
4 Attempt 2: Fit spin configurations of 2×2 supercell with Ising model with (J_1, J_2)

4.1 Replicate structure to make 2×2 supercell

Let us make a supercell so we can try out more spin configurations:
– Copy files from `CrI3/1RELAX` to `CrI3/4MULTICELL`
– Replicate the unit cell
  – Using my Python script: `replicatePOSCAR.py 1RELAX/POSCAR  2 2 1 4MULTICELL/POSCAR`
  – Or using my Mathematica code: `replicatePOSCAR[{2,2,1}]`
– Check that the result looks right
  – Using OVITO: `ovito 4MULTICELL/POSCAR`
  – Or using my Mathematica code: `drawStructure[]`
– Relax ions again (optional)

In[413]:=
```
unitCell = readPOSCAR ["1RELAX /POSCAR "];
superCell = replicatePOSCAR [unitCell , {2, 2, 1}];
writePOSCAR [superCell , "4MULTICELL /POSCAR "]
drawStructure [superCell , BondCutoffDistance → 2.7]
```

Out[416]=



The Cr atoms form a honeycomb lattice. We can visualize this explicitly by filtering.
Below, "skeleton" is a structure containing only the magnetic atoms.
"map1" and "map2" tell us atoms in the *original structure* correspond to which atoms in the *filtered structure*, and vice versa.

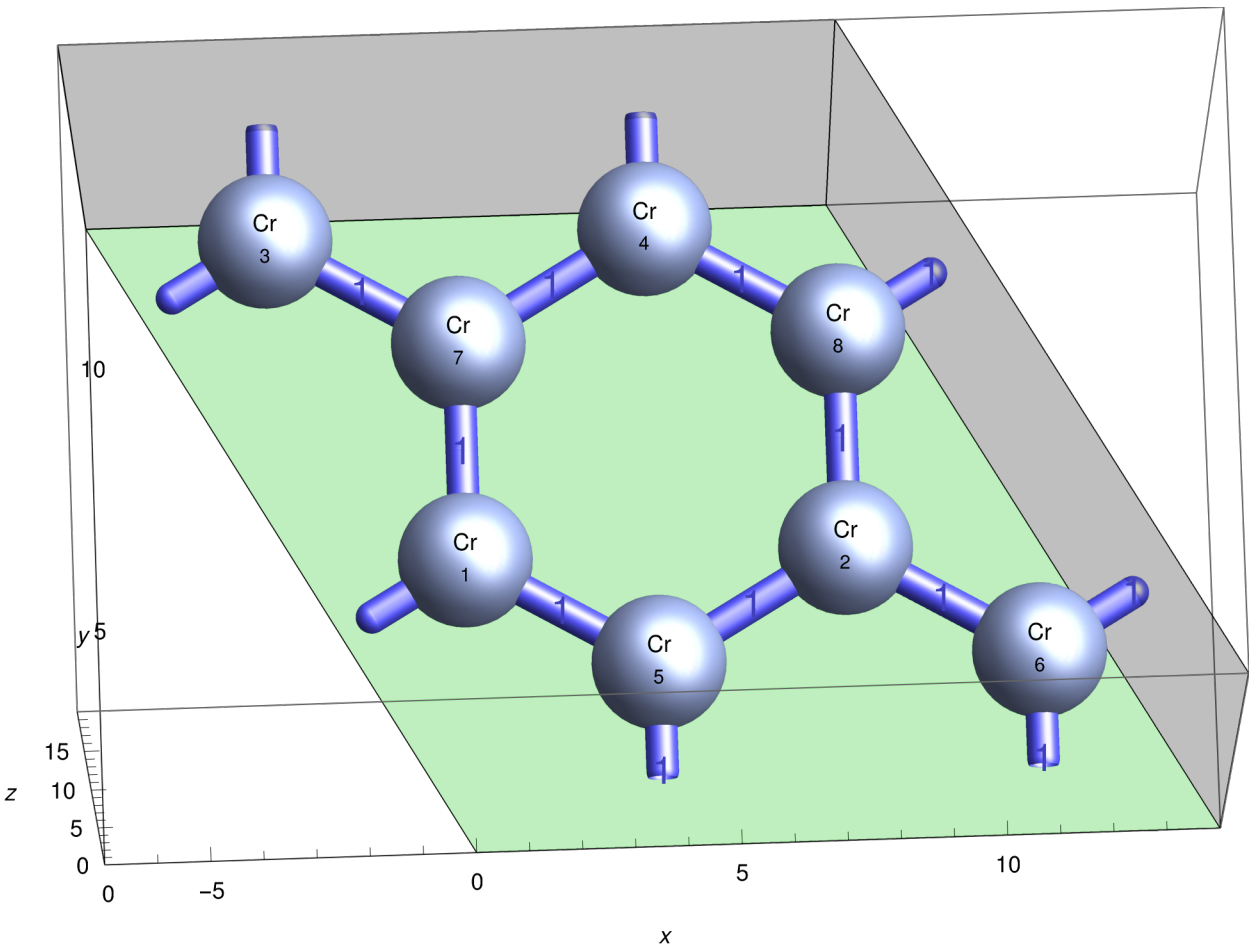
In[417]:=
```
{skeleton , map1 , map2 } = filterElements [superCell , {"Cr", "Fe", "Mn", "Ni", "Co"}];
map1
map2
```

Out[418]= {1, 2, 3, 4, 5, 6, 7, 8}

Out[419]= {1, 2, 3, 4, 5, 6, 7, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

In[420]:=
```
drawStructure [skeleton , BondCutoffDistance  → 5, BondRadii  → .3]
```

Out[420]=



4.2 Design an effective spin model and choose trial spin configurations

Make a file called "bonds. dat" with 6 columns as follows. (I entered the numbers manually. This is the most error-prone step in my workflow.) Then load it using my command, loadBonds[]:

```
b     bond type (for example,  bond type 3 corresponds  to coupling
i     atom index  at start  of bond
j     atom index  at end  of bond
u
v     lattice  vector  connecting  starting  unit  cell  to ending  cell
w
```

In[421]:=
```
Export ["bonds .dat", #, "String "]&@"
#i   j    u   v   w   b
#-------------------
 1   5    0   0   0   1
 2   6    0   0   0   1
 3   7    0   0   0   1
 4   8    0   0   0   1

 1   7    0   0   0   1
 2   8    0   0   0   1
 3   5    0   1   0   1
 4   6    0   1   0   1

 5   2    0   0   0   1
 7   4    0   0   0   1
 6   1    1   0   0   1
 8   3    1   0   0   1

 1   2    0   0   0   2
 2   4    0   0   0   2
 4   1    0   0   0   2
 5   8    0   0   0   2
 8   7    0   0   0   2
 7   5    0   0   0   2

 2   1    1   0   0   2
 1   3    0   0   0   2
 2   3    1   0   0   2
 6   7    1   0   0   2
 7   8   -1   0   0   2
 8   6    0   0   0   2

 3   4    0   0   0   2
 4   2    0   1   0   2
 2   3    0  -1   0   2
 7   5    0   1   0   2
 7   6    0   1   0   2
 5   6    0   0   0   2

 4   3    1   0   0   2
 3   1    0   1   0   2
 4   1    1   1   0   2
 8   6    0   1   0   2
 8   5    1   1   0   2
 6   5    1   0   0   2
";
```

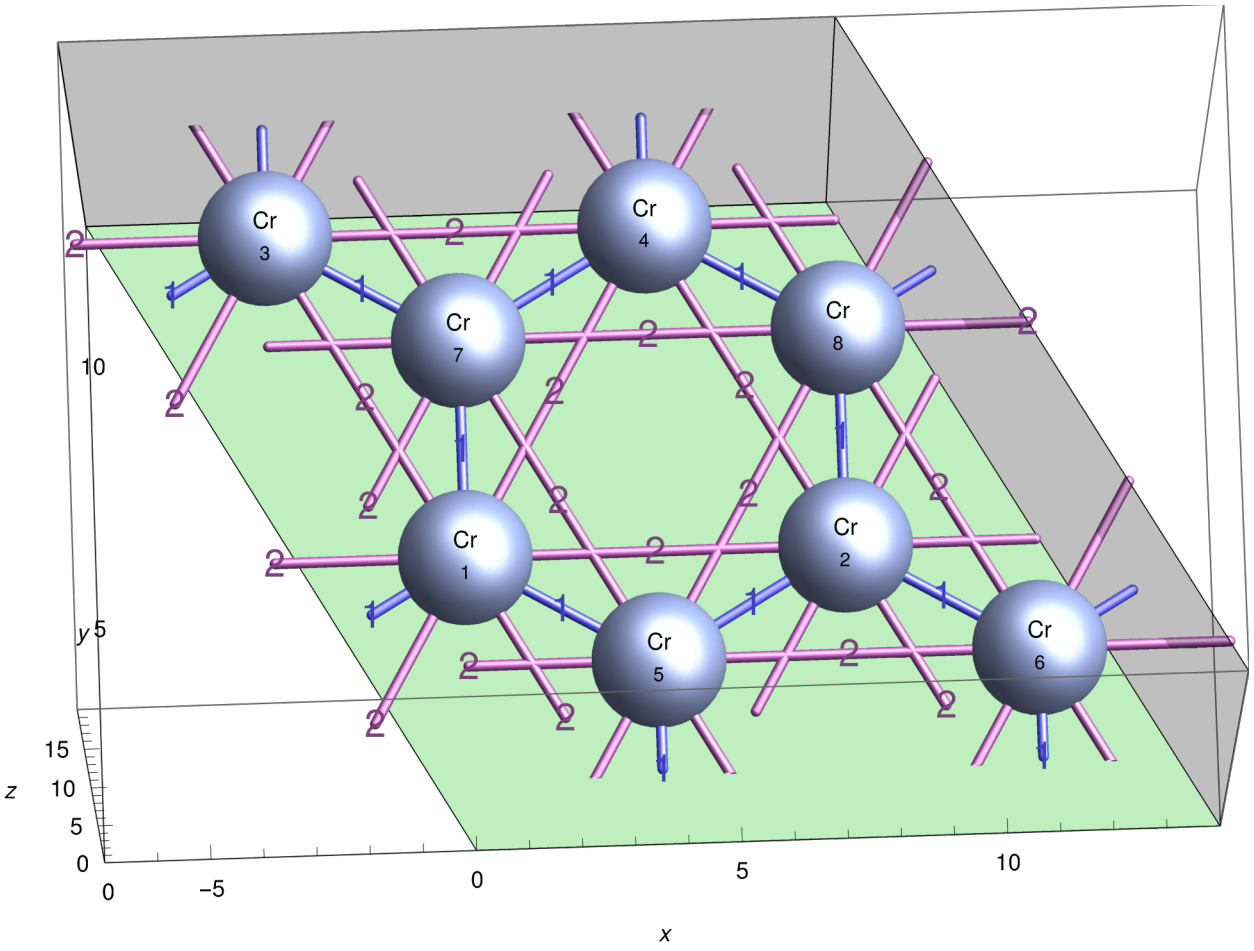
In[422]:=
```
bonds = loadBonds ["bonds .dat"]
```

Out[422]=
```
{{1, 2, 3, 4, 1, 2, 3, 4, 5, 7, 6, 8, 1, 2, 4, 5, 8, 7, 2, 1, 2, 6, 7, 8, 3, 4, 2, 7, 7, 5, 4, 3, 4, 8, 8, 6},
 {5, 6, 7, 8, 7, 8, 5, 6, 2, 4, 1, 3, 2, 4, 1, 8, 7, 5, 1, 3, 3, 7, 8, 6, 4, 2, 3, 5, 6, 6, 3, 1, 1, 6, 5, 5},
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, -1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1},
 {0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, -1, 1, 1, 0, 0, 1, 1, 1, 1, 0},
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
 {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2}}
```

We can now plot the magnetic "skeleton" with the bonds that we have defined:

In[423]:=
```
drawStructure [skeleton , Bonds → bonds , BondRadii → .1]
```

Out[423]=



In[424]:=
```
(*Export [NotebookDirectory []<>/model3Dpic .png",gr]*)
```

The visualization above represents a model of the form

$$E(\{\sigma\}) = N E_0 - \sum_{\beta=1} J_{b_n}\, S_{i_n}\, S_{j_n}$$

where β runs over all bond indices, b_β is the *bond type* of bond β, and i_β and j_β are the atom indices at the start and end of bond β.

Rewrite the model energy function as

$$E(\{\sigma\}) = \sum_{b=0} X_b\, W_b$$

$X_0 = N_{\text{spins}}$ $\qquad$ $W_0 = E_0$ (energy per spin)

$X_1 = -\sum_{\beta | b_\beta = 1} S_{i_\beta}\, S_{j_\beta}$ $\qquad$ $W_1 = J_1$ (coupling along bonds of type 1)

…

$X_B = -\sum_{\beta | b_\beta = B} S_{i_B}\, S_{j_B}$ $\qquad$ $W_B = J_B$ (coupling along bonds of type B).

In machine-learning language, the model is a linear neural net where X_b are the *inputs (features)*, W_b are the *weights*, and E is the *output*.

We will perform DFT calculations on a set of spin configurations $\left\{S_i^{(n)}\right\}$ for $n = 1, 2, 3, \ldots, N$. For each configuration, we can calculate the *inputs* $\left\{X_b^{(n)}\right\}$ and *outputs* $\{E^{(n)}\}$. Split the dataset into a *training*

dataset and a *validation* dataset (each containing inputs and outputs). For the training dataset, minimize a loss function with respect to the model parameters (weights). A typical loss function is

$$\chi^2(\{W\}) = \sum_n [E_{\text{pred}}{}^{(n)} - E^{(n)}]^2$$

where $E_{\text{pred}}{}^{(n)} = \sum_{b=0} X_b^{(n)}\, W_b$ and $X_b^{(n)} = -\sum_{\beta \mid b_\beta = b} S_{i_B}\, S_{j_B}$. This loss function is convenient because it allows us to determine $\{W_b\}$ using linear regression. In this context, the set of feature vectors forms the *design matrix* $\mathbf{K} = X_b^{(n)}$.

After determining the optimal $\{W_b\}$, calculate the loss for the validation dataset. If the training loss is small but the validation loss is large, then we have committed the crime of *overfitting*.

In our example, the number of Cr ions in the supercell is 8. Thus there are $2^8 = 256$ configurations, $\left\{S_i^{(n)}\right\}$, of up and down spins. To determine the parameters of our effective spin model, we only need to run DFT on a subset of these configurations. Here are some criteria for choosing configurations and running simulations:

1. Each configuration, $\mathbf{S}^{(n)}$, should correspond to a distinct feature vector, $\mathbf{X}^{(n)}$. (Otherwise, we are wasting computer time.)
2. The rank of the design matrix, $\mathbf{K}$, should be equal to the number of unknowns. (Otherwise, parameters will be under-determined.)
3. It is a good idea to choose more configurations than necessary, so that some configurations can be used for training and others for validation.
4. The rows of the design matrix should be as orthogonal as possible (i.e., the dot products $\mathbf{X}^{(n)} \cdot \mathbf{X}^{(n)'}$ should be close to zero), so that the linear system is well-conditioned.
5. Symmetrical configurations such as ↑↑↑↑↑↑↑↑ and ↑↓↑↓↑↓↑↓ are more likely to be stable during VASP calculation. If one simulates an unbalanced configuration such as ↑↑↑↓↑↑↑↑ in VASP, unwanted spin flips may occur, so that the final magnetization profile may differ from the initial choice. One may have to take precautions to prevent this. Precautions include:
– reducing the values of AMIX_MAG and BMIX_MAG, so that magnetic moments are only allowed to change a bit at a time
– using I_CONSTRAINED_M to constrain the local moments (this seems to require doing a non-collinear calculation)

Prepare a file `spincfgs.dat` that specifies our choice, and load it. Compute the feature vectors, i.e., the design matrix:

In[425]:=
```
Export ["spincfgs .dat", StringTrim [#] <> "\n", "String "] & @ "
2                 # multiplier , i.e., moment  per  Cr  atom  in  Bohr  magnetons
  1  1  1  1  1  1  1  1
  1  1  1  1 -1 -1 -1 -1
  1 -1 -1 -1  1 -1 -1  1
  1 -1  1  1 -1 -1  1 -1
  1 -1  1 -1  1  1  1  1
";
(*-------- LOAD  --------*)
f = OpenRead ["spincfgs .dat"];
multiplier  = ReadLine [f] // StringToStream  // Read [#, Number ] &;
sna = ReadList [#, Number ] & @ StringToStream  @ # & /@ ReadList [f, "String "];
Close [f];
(*-------- I broke  down  and  used  some  global  variables  here  --------*)
bb = bonds ⟦6⟧;
ib = bonds ⟦1⟧;
jb = bonds ⟦2⟧;
bondTypes  = Max @ bb ;
bondCount  = Length @ bb ;
spinCount  = Length @ sna ;
(*-------- COMPUTE  FEATURES  --------*)
Xnb = Table [
    sa = sna ⟦n⟧;
    X0 = Length @ sa ; (* the  zeroth  "feature " of the spin  configuration  is the number  of spins  *)
    Xb = Table [  -Sum [KroneckerDelta  [bb ⟦b⟧ - btype ] * sa ⟦ib ⟦b⟧⟧ * sa ⟦jb ⟦b⟧⟧, {b, Length @ bb}]  , {btype , bondTypes }];
    Join [{X0}, Xb]
    , {n, Length @ sna }];
Print [Grid [{{"Spin  configurations  S_i^(n)  are  rows  of..." "Feature  vectors  X_b^(n)  are  rows  of..."}
                    {sna // MatrixForm                                  Xnb // MatrixForm}}]];
Print ["Rank  = ", MatrixRank [Xnb], "     Unknowns  = ", bondTypes  + 1]
```

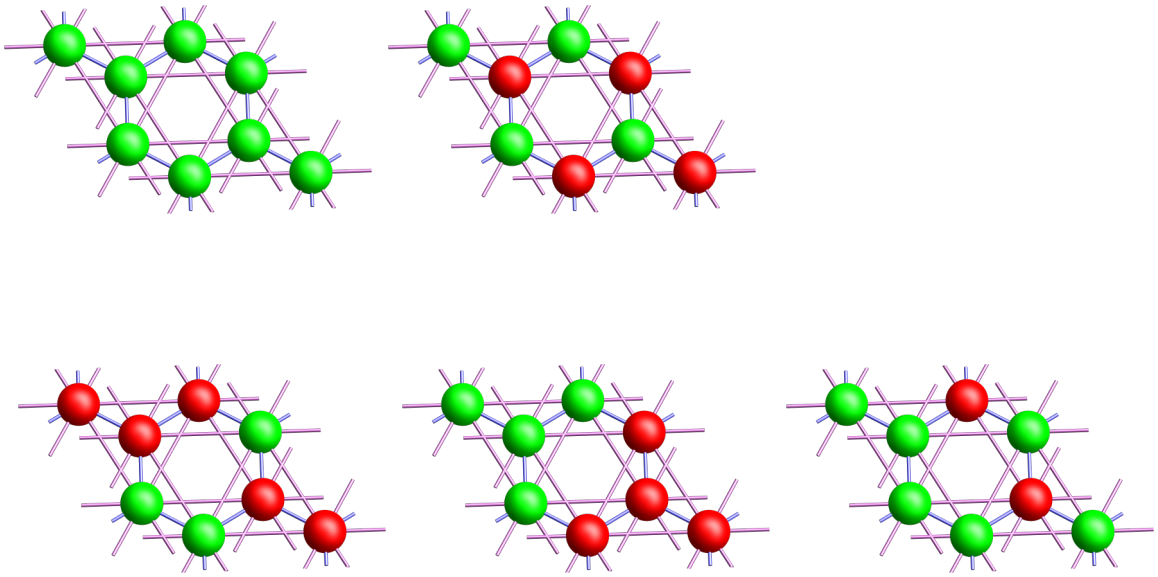
Spin configurations $S_i^{(n)}$ are rows of... Feature vectors $X_b^{(n)}$ are rows of...

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & -1 & -1 & 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 & 1 & 1 & 1 & 1 \end{pmatrix} \qquad \begin{pmatrix} 8 & -12 & -24 \\ 8 & 12 & -24 \\ 8 & 2 & 4 \\ 8 & 0 & 0 \\ 8 & 0 & -8 \end{pmatrix}$$

Rank = 3 Unknowns = 3

In[439]:=
```
Table [
  drawSpins [skeleton , sna[[n]], Bonds → bonds , ImageSize → 200 , Axes → None , Boxed → False , Arrows → False ],
  {n, Length @ sna }] // Row
```

Out[439]=



4.3 Run VASP for each spin configuration

First prepare a master INCAR file, which contains everything necessary for a collinear magnetic calculation except the MAGMOM line:

```
SYSTEM  = CrI3  collinear  magnetic  calculation
ISTART  = 0            # fresh  start
IBRION  = 0            # fix  ions
ISPIN     = 2          # collinear  magnetism
AMIX      = 0.2        # customary  parameters  for  magnetic  calculations
BMIX      = 0.0001
AMIX_MAG  = 0.8
BMIX_MAG  = 0.0001
LMAXMIX   = 4
LORBIT    = 11         # output  charges  and  moment
```

Then run the following code to create subdirectories for each spin configuration, with the appropriate MAGMOM line appended to each INCAR file. (Warning: this code relies on UN*X commands, so it needs to be adapted in order to work in Windows.)

In[440]:=
```
configCount = Length[sna];
Do[
  (*-------- Prepare MAGMOM string --------*)
  moments = ConstantArray[0, Length@map2]; (* number of atoms in the original POSCAR *)
  moments[[map1]] = sna[[n]]*multiplier ;
  (* transfer magnetic moment info from reduced atom-set to full atom-set *)
  cfgName = ToString@StringForm["CONFIG`` ", n];
  dirName = ToString@StringForm["``/4MULTICELL /``", NotebookDirectory[], cfgName];
  magmomLine = ToString@StringForm["MAGMOM =``", ExportString[{moments}, "Table", "FieldSeparators" -> " "]];
  Print[dirName, ": ", magmomLine];
  (*-------- Create directory --------*)
  (*CreateDirectory[NotebookDirectory[]<>"/4MULTICELL /"<>cfgName,CreateIntermediateDirectories->True];*)
  Run["mkdir -p -m u=rwx,g=rx,o=rx " <> dirName];
  (*-------- Populate directory --------*)
  CopyFile[NotebookDirectory[] <> "/4MULTICELL /POSCAR", dirName <> "/POSCAR", OverwriteTarget -> True];
  CopyFile[NotebookDirectory[] <> "/4MULTICELL /POTCAR", dirName <> "/POTCAR", OverwriteTarget -> True];
  CopyFile[NotebookDirectory[] <> "/4MULTICELL /KPOINTS", dirName <> "/KPOINTS", OverwriteTarget -> True];
  CopyFile[NotebookDirectory[] <> "/4MULTICELL /script", dirName <> "/script", OverwriteTarget -> True];
  CopyFile[NotebookDirectory[] <> "/4MULTICELL /INCAR", dirName <> "/INCAR", OverwriteTarget -> True];
  f = OpenAppend[dirName <> "/INCAR"];
  WriteString[f, magmomLine <> "\n"];
  Close[f];
  , {n, configCount}];
```

```
/home/yloh/CrI3//4MULTICELL /CONFIG1 :
 MAGMOM =2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

/home/yloh/CrI3//4MULTICELL /CONFIG2 :
 MAGMOM =2 2 2 2 -2 -2 -2 -2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

/home/yloh/CrI3//4MULTICELL /CONFIG3 :
 MAGMOM =2 -2 -2 -2 2 -2 -2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

/home/yloh/CrI3//4MULTICELL /CONFIG4 :
 MAGMOM =2 -2 2 2 -2 -2 2 -2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

/home/yloh/CrI3//4MULTICELL /CONFIG5 :
 MAGMOM =2 -2 2 -2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Now:
– rsync the data directories to the cluster
– go into each directory and submit script to launch VASP
– when runs are done, rsync the output files back from the cluster.
(I have “put” and “get” scripts to help me with this.)

4.4 Verify magnetizations

One should ALWAYS inspect the VASP output files to make sure that the final magnetization pattern has not changed dramatically from the initial configuration. As stated earlier, we can use `ovito mag.ovito`. Alternatively, we can use my plotting routines in Mathematica:

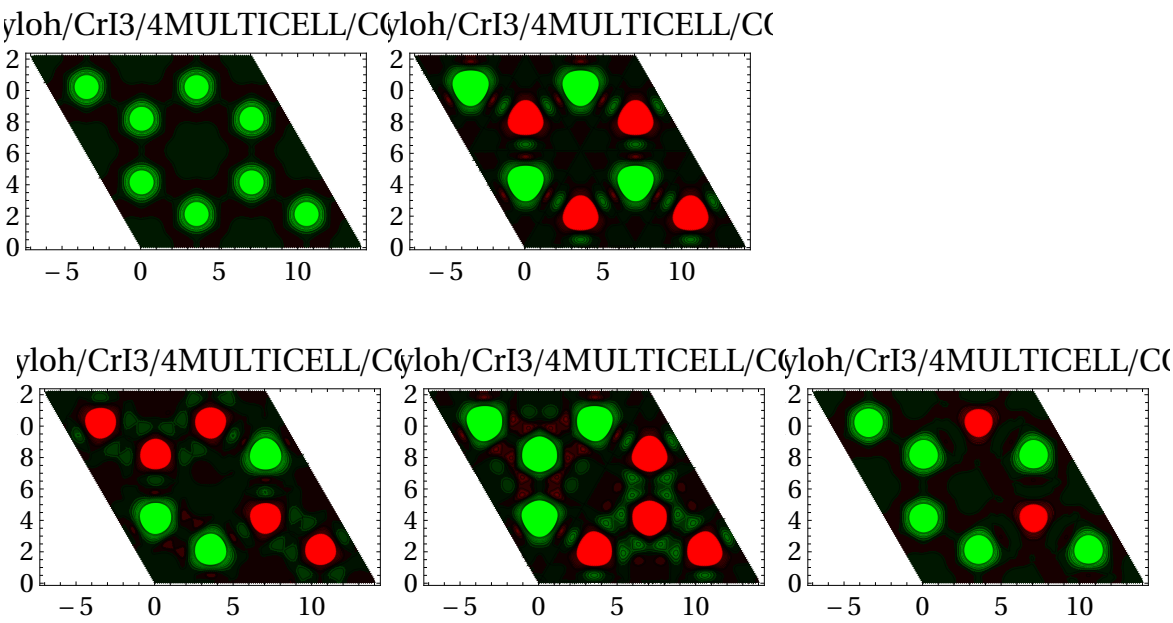
In[211]:=
```
dirs = FileNames[NotebookDirectory[] <> "/4MULTICELL /CONFIG *"]
```

Out[211]=
```
{/home/yloh/CrI3/4MULTICELL /CONFIG1 , /home/yloh/CrI3/4MULTICELL /CONFIG2 ,
 /home/yloh/CrI3/4MULTICELL /CONFIG3 , /home/yloh/CrI3/4MULTICELL /CONFIG4 , /home/yloh/CrI3/4MULTICELL /CONFIG5 }
```

In[219]:=
```
grList = Table [
    {struc , rhouvw , muvw } = readCHGCAR [dir <> "/CHGCAR "];
    muv = Total [muvw , {3}];  (* project  along  c direction  *)
    plotMuv [{umax , vmax }, struc @ BasisVectors  , muv ] // Show [#, PlotLabel  -> dir , ImageSize  -> 200 ] &
    , {dir , dirs }];
```

In[220]:=
```
Row @ grList
```

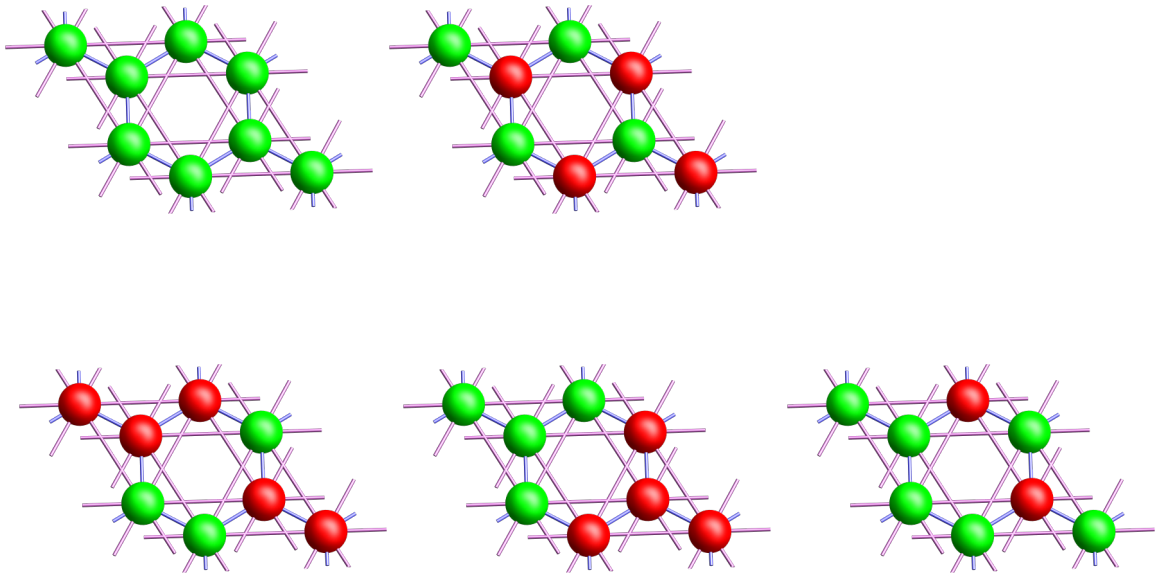
Out[220]=



I emphasize: Above is VASP magnetization from CHGCAR (the output). Below were the input structures. In this case the output followed the input.

In[221]:=
```
Table [
  drawSpins [skeleton , sna[[n]], Bonds → bonds , ImageSize  → 200 , Axes → None , Boxed → False , Arrows  → False ],
  {n, Length @ sna }] // Row
```

Out[221]=



You should also check the magnetizations in OUTCAR. Open each OUTCAR file, search for "`magnetization (x)`" including the space, and examine the table.

Some times VASP will converge to a non-magnetic state! Such results should be discarded (or redone carefully).

4.5 Fit energies using (J_1, J_2) honeycomb Ising model

Extract the energies $E^{(n)}$:

In[222]:=
```
En = Table [extractE0 [dir <> "/OSZICAR "], {dir , dirs }]
```

Out[222]=
```
{-124.71 , -124.693 , -124.63 , -124.629 , -124.671 }
```

In[223]:=
```
(*(*-------- MANUALLY  SELECT  --------*)
{sna ,Xnb ,En}={sna ,Xnb ,En}[[All ,{1 ,2 ,3}]];*)
```

Use least-squares linear regression to obtain the weights W_b (and hence the couplings J_b):

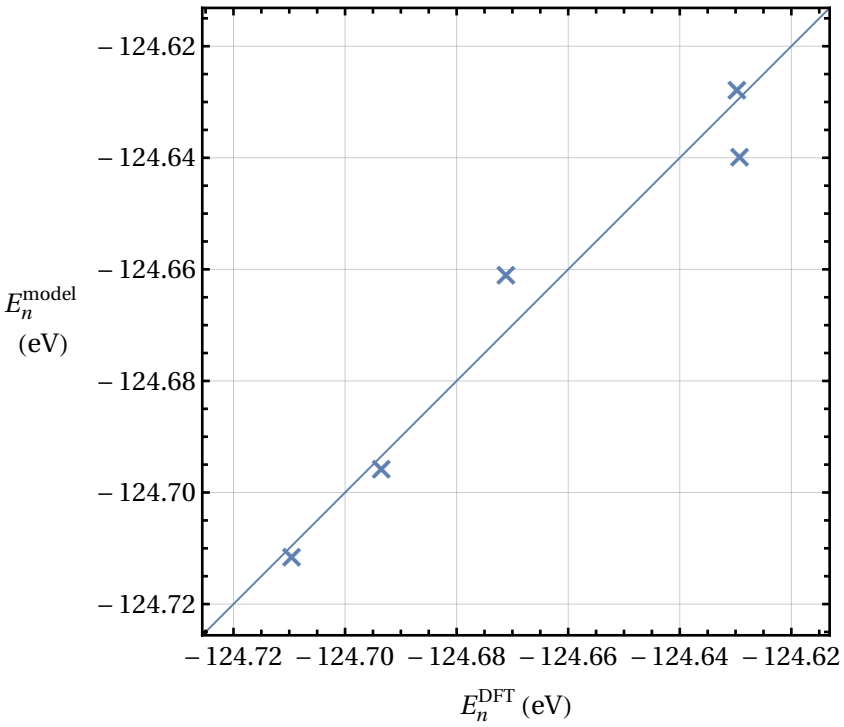
In[289]:=
```
Wb = LeastSquares [Xnb , En];(* weights  = LeastSquares [features , targets ] *)
EnPred = Xnb . Wb ;
Jb = Rest @ Wb ;
(*-------- RAW INPUT VECTORS (SPIN CONFIGS ), FEATURE VECTORS , AND TARGETS (ENERGIES ) --------*)
(* Each row corresponds to one spin configuration *)
MatrixForm /@ {sna , Xnb , En , EnPred , {Wb}} // NumberForm [#, {∞, 3}] &
```

Out[292]//NumberForm=
$$\left\{\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & -1 & -1 & 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 & 1 & 1 & 1 & 1 \end{pmatrix}, \begin{pmatrix} 8 & -12 & -24 \\ 8 & 12 & -24 \\ 8 & 2 & 4 \\ 8 & 0 & 0 \\ 8 & 0 & -8 \end{pmatrix}, \begin{pmatrix} -124.710 \\ -124.693 \\ -124.630 \\ -124.629 \\ -124.671 \end{pmatrix}, \begin{pmatrix} -124.711 \\ -124.695 \\ -124.627 \\ -124.639 \\ -124.660 \end{pmatrix}, \begin{pmatrix} -15.580 & 0.001 & 0.003 \end{pmatrix}\right\}$$

Visualize the prediction accuracy:

In[293]:=
```
Show [
 ListPlot [{En , EnPred }ᵀ, AspectRatio → Automatic , GridLines → Automatic , PlotMarkers → {"×", 24}],
 Plot [x, {x, 1.2 Min @ En - .2 Max @ En , 1.2 Max @ En - .2 Min @ En}, PlotStyle → AbsoluteThickness @ 1],
 FrameLabel -> {"E_n^DFT (eV)", "E_n^model \n(eV)"}, PlotRange -> All , PlotRangePadding -> 0, Axes -> False , ImageSize -> 436
]
```

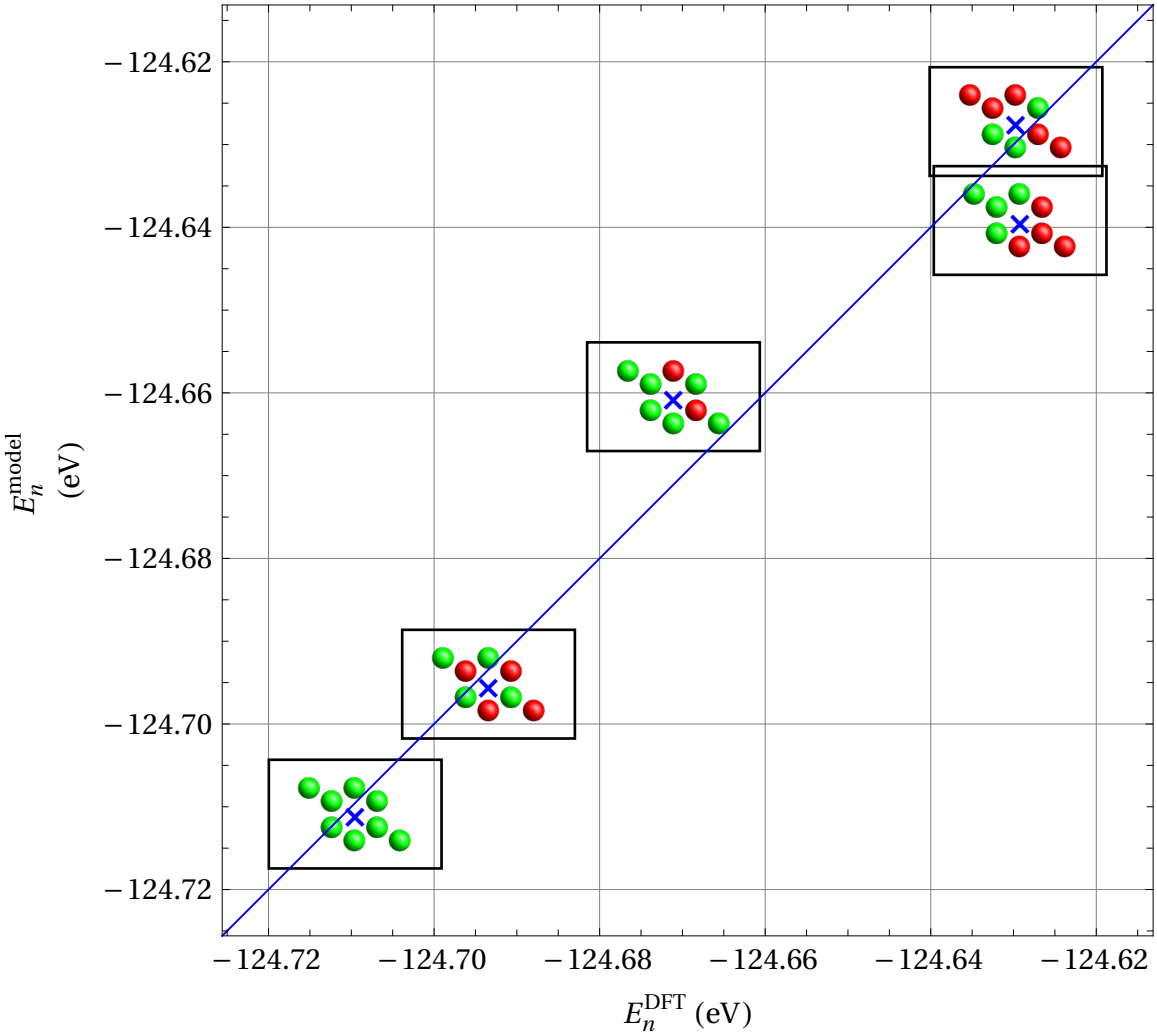
Out[293]=



Visualize the prediction accuracy, showing each configuration explicitly to get more insight:

In[294]:=
```
cfgPlots = Table [
    drawSpins [skeleton , sna[[n]], Bonds → Automatic , RadiusScalingFactor → 1,
         SphericalRegion → False , ViewPoint → {0, 0, 4}, ImageSize → 80 , Axes → None , Boxed → False , Arrows → False ] //
       Rasterize [#, ImageResolution → 300 ] & // RemoveBackground // Framed
    , {n, Length @sna }];
Show [
 Graphics [{ Table [Inset [cfgPlots [[n]], {En[[n]], EnPred [[n]]}, Scaled @{.5 , .5}], {n, Length @En}] }],
 ListPlot [{En , EnPred }ᵀ, PlotStyle → {{AbsoluteThickness @1, Blue }}, PlotMarkers → {"×", 24}],
 Plot [x, {x, 1.2 Min @En - .2 Max @En , 1.2 Max @En - .2 Min @En}, PlotStyle → {{AbsoluteThickness @1, Blue }}],
 AspectRatio → Automatic , GridLines → Automatic ,
 BaseStyle -> {16 , Black }, LabelStyle -> {16 , Black , FontFamily -> "Times "},
 Frame -> True , FrameLabel -> {"E_n^DFT (eV)", "E_n^model \n(eV)"}, ImageSize -> 600 , PlotRangePadding -> 0
]
```

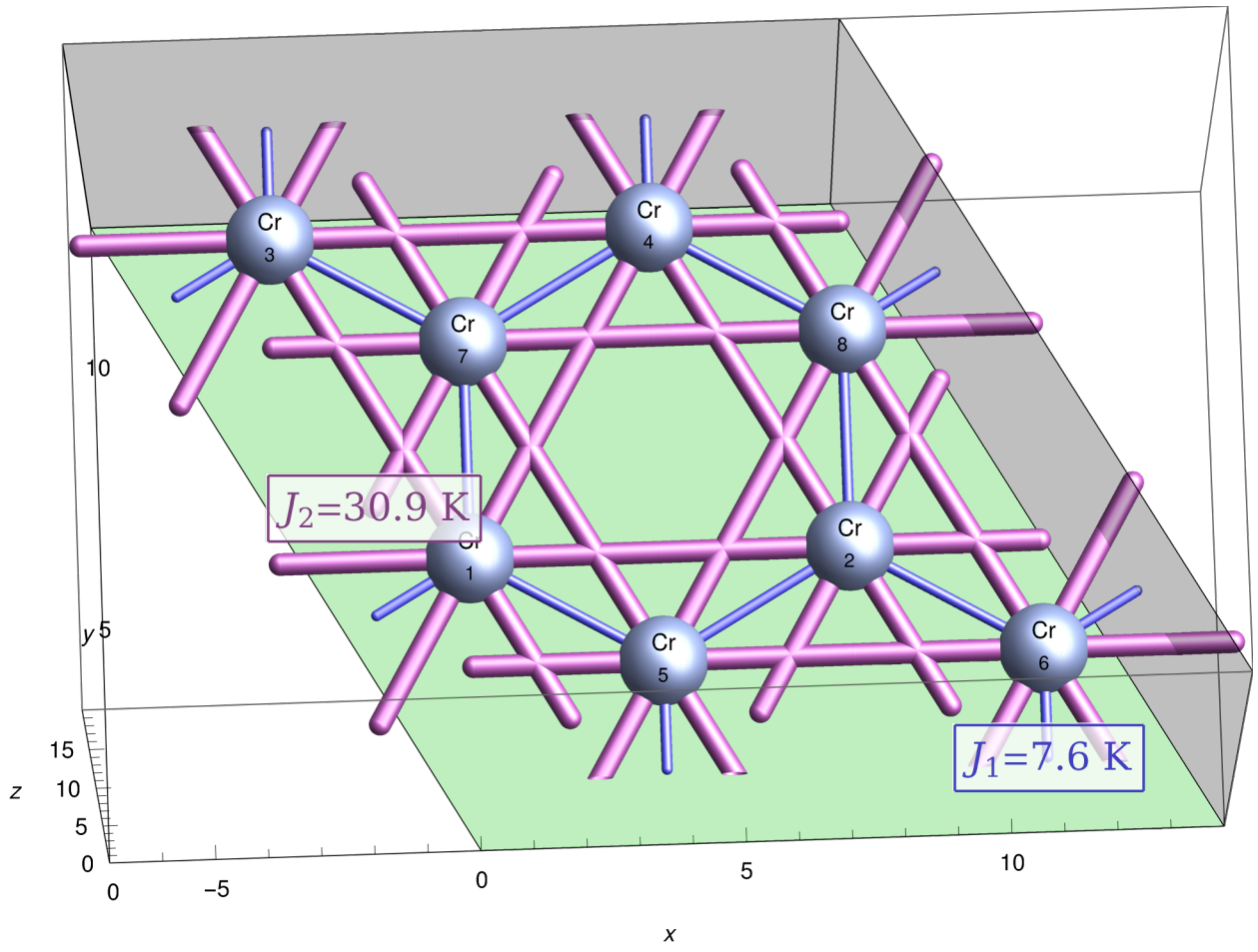
Out[295]=



Visualize the pattern of exchange couplings:

In[296]:=
```
SeedRandom @123; (* change this number to place the labels elsewhere *)
bondsToLabel = RandomChoice @Position [bb, #] & /@ Union @bb // Flatten ;
bondLabels = ConstantArray ["", Length @bb];
Do[
  bondLabels [[b]] =
   Style [#, 20, FontFamily → "Serif "] & @
     StringForm [ "J``=`` K",
      bb[[b]],
      Jb[[bb[[b]]]] * 16 021 766 340 / 1 380 649 // DecimalForm [#, {∞, 1}] & (* convert from eV to Kelvin *)
     ] // Framed [#, Background -> Opacity [.75, White ]] &
  , {b, bondsToLabel }];
drawStructure [skeleton ,
 RadiusScalingFactor → .5,
 Bonds → bonds ,
 BondRadii → 4 √Abs [Jb[[bb]]] ,
 BondLabels → bondLabels
]
```

Out[300]=



Earlier, we fitted a model including only 1st-neighbor couplings, and we found $J_1 \approx 14$ K to 3 nearest neighbors. Thus $T_c^{\mathrm{MFT}} \approx 3 \times 14$ K.

Now, we fitted a model with 1st and 2nd neighbor couplings, and we find $J_1 \approx 8$ K and $J_2 \approx 31$ K to 6 second neighbors! Thus $T_c^{\mathrm{MFT}} \approx 6 \times 31$ K.

In[301]:=
```
JbinKelvin = Jb * 16 021 766 340 / 1 380 649 (* convert from eV to K *)
```

Out[301]=
```
{7.58969 , 30.8804 }
```

4.6 MFT

We have fitted the DFT data for CrI_3 using an Ising model on a honeycomb lattice. Although this is a 2D lattice, there are bonds that cross each other. This makes it impossible to solve this Ising model using the usual analytical techniques. Ultimately we will need Monte Carlo simulations. However, it is very useful to have a quick-and-dirty estimate from mean-field theory, to help us estimate T_c with a few lines of code.

We have a spin model governed by the Hamiltonian

$$E[S] = N E_0 - \sum_{b=1}^{B} J_b \, S_{i_b} \, S_{j_b}$$

where $S_i = \pm 1$ for all i and J_b is the exchange coupling along bond b. The Boltzmann distribution is $P[S] \propto \exp\, -\beta E[S]$ where $\beta = 1 / k_B T$. In the mean-field approximation, each spin experiences an effective magnetic field due to its neighbors,

$$h_i = \sum_j J_{ij} \left\langle S_j \right\rangle .$$

The thermal average of spin i is given by

$$\langle S_i \rangle = \frac{\sum_{S_i = \pm 1} S_i \exp \beta h_i S_i}{\sum_{S_i = \pm 1} \exp \beta h_i S_i} = \frac{e^{\beta h_i} - e^{-\beta h_i}}{e^{\beta h_i} + e^{-\beta h_i}} = \tanh \beta h_i .$$

The mean-field solution can be found by repeatedly plugging in $M_j = \left\langle S_j \right\rangle$ to the RHS of the equation below and iterating to self-consistency:

$$M_i^{(n+1)} = \tanh \left(\beta \sum_j J_{ij} \, M_j^{(n)} \right)$$

where n is the iteration index. In order to find T_c, we may assume that $M_i \ll 1$ for all i. Then the argument of the tanh function is small, and we may approximate the tanh. So the fixed-point iteration reads

$$M_i^{(n+1)} = \frac{1}{T} \sum_j J_{ij} \, M_j^{(n)} .$$

Note that J_{ij} is a real symmetric matrix.

The above equation is a linear recursion relation for the vector $\mathbf{M}^{(n)}$. It may be solved using an eigenexpansion:

$$M_i^{(n)} = \sum_\alpha c_\alpha \left(\frac{\lambda_\alpha}{T} \right)^n u_{\alpha i}$$

Here λ_α are eigenvalues of J_{ij}, and $u_{\alpha i}$ are eigenvectors. In order for the fixed-point iteration to converge to a nontrivial solution, we must have $\frac{\lambda_\alpha}{T} > 1$. Thus T_c is determined by the largest eigenvalue of the J matrix:

$$T_c = \lambda_{\max} .$$

Thus we have derived a useful formula:

For the classical Ising model $P[S] \propto \exp \frac{1}{2T} \sum_{ij} J_{ij}\, S_i\, S_j$, the Curie temperature for magnetic order according to mean-field theory is

$$T_c^{\mathrm{MFT}} = \max \text{eigval}\, \{J_{ij}\}.$$

There are 8 magnetic (Cr) atoms in the unit cell. There are 2 types of couplings. J_{ij} is a 8×8 matrix. Find its eigenvalues:

Out[181]=
```
<|System → Cr2 I6 replicated by 2x2x1, BasisVectors →
  {{14.0033, -0.0000737511, 0.000371361}, {-7.00173, 12.1272, 0.000371361}, {0.00212339, 0.00367781, 19.9906}},
 ElementNames → {Cr}, ElementCounts → {8}, CoordType → coordType,
 Cad → {{0.166655, 0.333345, 0.5}, {0.666655, 0.333345, 0.5}, {0.166655, 0.833345, 0.5}, {0.666655, 0.833345, 0.5},
   {0.333321, 0.166679, 0.5}, {0.833321, 0.166679, 0.5}, {0.333321, 0.666679, 0.5}, {0.833321, 0.666679, 0.5}}|>
```

In[323]:=
```
atomCount = Total @ skeleton @ ElementCounts ;
Jaa = Table [0, {atomCount }, {atomCount }];
Do[ Jaa〚ib〚b〛, jb〚b〛〛 += Jb〚bb〚b〛〛;
  Jaa〚jb〚b〛, ib〚b〛〛 += Jb〚bb〚b〛〛 , {b, bondCount }]; (* set up the Jij matrix *)
Jaa // Dimensions
Jaa // MatrixForm
```

Out[325]= {8, 8}

Out[326]//MatrixForm=

$$\begin{pmatrix}
0 & 0.00532214 & 0.00532214 & 0.00532214 & 0.000654029 & 0.000654029 & 0.000654029 & 0 \\
0.00532214 & 0 & 0.00532214 & 0.00532214 & 0.000654029 & 0.000654029 & 0 & 0.000654029 \\
0.00532214 & 0.00532214 & 0 & 0.00532214 & 0.000654029 & 0 & 0.000654029 & 0.000654029 \\
0.00532214 & 0.00532214 & 0.00532214 & 0 & 0 & 0.000654029 & 0.000654029 & 0.000654029 \\
0.000654029 & 0.000654029 & 0.000654029 & 0 & 0 & 0.00532214 & 0.00532214 & 0.00532214 \\
0.000654029 & 0.000654029 & 0 & 0.000654029 & 0.00532214 & 0 & 0.00532214 & 0.00532214 \\
0.000654029 & 0 & 0.000654029 & 0.000654029 & 0.00532214 & 0.00532214 & 0 & 0.00532214 \\
0 & 0.000654029 & 0.000654029 & 0.000654029 & 0.00532214 & 0.00532214 & 0.00532214 & 0
\end{pmatrix}$$

In[327]:=
```
Print ["Eigvals and corresponding eigvecs (rows):"]
MatrixForm /@ Transpose @ Reverse @ Sort @ Transpose @ Eigensystem [Jaa] // DecimalForm [#, {5, 4}] &
```

Eigvals and corresponding eigvecs (rows):

Out[328]//DecimalForm=

$$\left\{
\begin{pmatrix}
0.0179 \\ 0.0140 \\ -0.0047 \\ -0.0047 \\ -0.0047 \\ -0.0060 \\ -0.0060 \\ -0.0060
\end{pmatrix},
\begin{pmatrix}
-0.3536 & -0.3536 & -0.3536 & -0.3536 & -0.3536 & -0.3536 & -0.3536 & -0.3536 \\
0.3536 & 0.3536 & 0.3536 & 0.3536 & -0.3536 & -0.3536 & -0.3536 & -0.3536 \\
-0.6124 & 0.2041 & 0.2041 & 0.2041 & -0.2041 & -0.2041 & -0.2041 & 0.6124 \\
-0.0000 & 0.5170 & -0.4811 & -0.0359 & 0.0359 & 0.4811 & -0.5170 & 0.0000 \\
0.0000 & -0.2570 & -0.3192 & 0.5762 & -0.5762 & 0.3192 & 0.2570 & 0.0000 \\
0.6124 & -0.2041 & -0.2041 & -0.2041 & -0.2041 & -0.2041 & -0.2041 & 0.6124 \\
0.0000 & 0.2672 & -0.5768 & 0.3096 & 0.3096 & -0.5768 & 0.2672 & 0.0000 \\
-0.0000 & -0.5118 & 0.0245 & 0.4873 & 0.4873 & 0.0245 & -0.5118 & 0.0000
\end{pmatrix}
\right\}$$

In[329]:=
```
λmax = Max @ Eigenvalues @ Jaa
TcMFT = λmax * 16 021 766 340 / 1 380 649
```

Out[329]= 0.0179285

Out[330]= 208.052

The largest eigenvalue is $\lambda_{\max} = 0.00956$ eV. The corresponding eigenvector is uniform, indicating a ferromagnetic configuration where all spins have the same value.

The mean-field critical temperature is

$$T_c^{\mathrm{MFT}} = \lambda_{\max} \approx 208 \text{ K}.$$

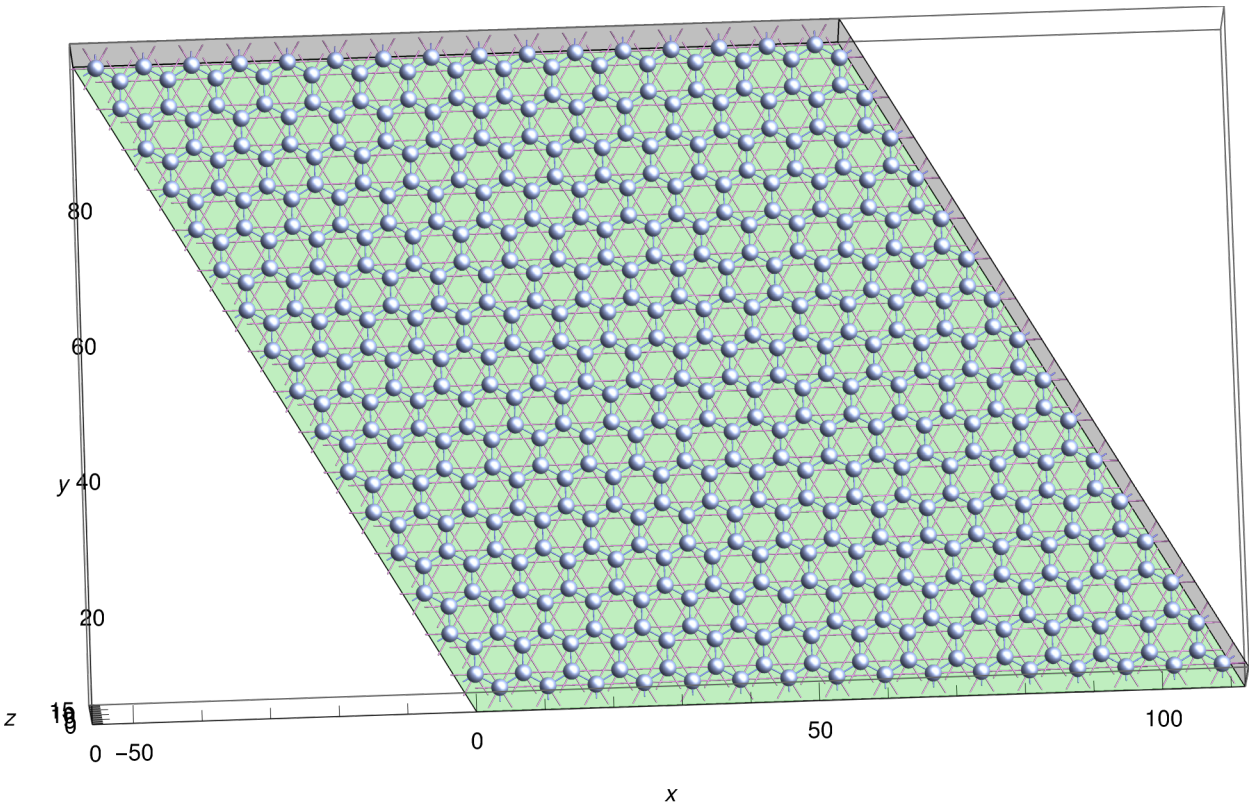
This is much larger than experiment!

4.7 MC (animation)

Let us run MC on a 8×8 lattice. Prepare `mclattice.dat, mcbonds.dat, pars.dat, Jb.dat`:

In[311]:=
```
mcLattice = replicatePOSCAR [skeleton , {8, 8, 1}];
mcBonds = replicateBonds [bonds , {8, 8, 1}];
drawStructure [mcLattice , Bonds → mcBonds , ElementLabels → None , AtomLabels → None , BondLabels → None ]
writePOSCAR [mcLattice , "MC/mclattice .dat "]
Export ["MC/mcbonds .dat ", mcBonds ᵀ, "Table "];
```

Out[313]=



In[316]:=
```
JbInKelvin = {7.589688314632001` , 30.880441558736017` };
Export ["MC/Jb.dat ", JbInKelvin , "Table "];

temperature = 60.0 ;
Export ["MC/pars .dat ", #] & @ subst @ "
useGUI         = 1                        # show graphics
stepsTuning    = 999999999
stepsCollect   = 0                        # if stepsCollect =0 then run Mickey Mouse animation
timeTuning     = 180                      # run for 3 minutes
timeCollect    = 0
spinType       = 1                        # Ising model
temperature    = ${temperature }
";
```

In[320]:=
```
(*-------- MAKE EXECUTABLE (I.E., COMPILE AND LINK) --------*)
err = Run["(cd MC; make -B mc > make.log 2>&1)"];
If[err == 0, Print["Compiled executable mc successfully !"], FilePrint ["MC/make.log "]];
```

```
Compiled executable mc successfully !
```

Now open a terminal, cd to the `MC` directory, and run `./MC` so that you can see the usage info and GUI controls in the terminal.

By exploring this simulation you can see that $T_c \approx 140$ K.

4.8 Attempt 2: $T_c \approx 140$ K

We tried to improve our model by including 2nd-neighbor couplings in the Ising model. But now we predict $T_c \approx 140$ K.
According to experiments, the Curie temperature of monolayer CrI_3 is 45 K! We are now way off in the other direction! What went wrong?

5 Attempt 3: Classical Heisenberg model

5.1 Heisenberg spins

In collinear-spin DFT calculations, we assume that all electrons are either ↑ or ↓. Therefore the spins end up being ↑ or ↓.

We have been fitting to a model containing Ising spins, which are classical spins that are either ↑ or ↓. However, if the spins are large and classical, they can point in any direction in 3D space. Then we should be using a classical Heisenberg model of the form

$$E[\mathbf{S}] = E_0 - \sum_{b=1}^{B} J_b \, \mathbf{S}_{i_b} \cdot \mathbf{S}_{j_b}$$

where $|\mathbf{S}_i| = 1$ for all i. Fitting this model leads to the same J_b as before.

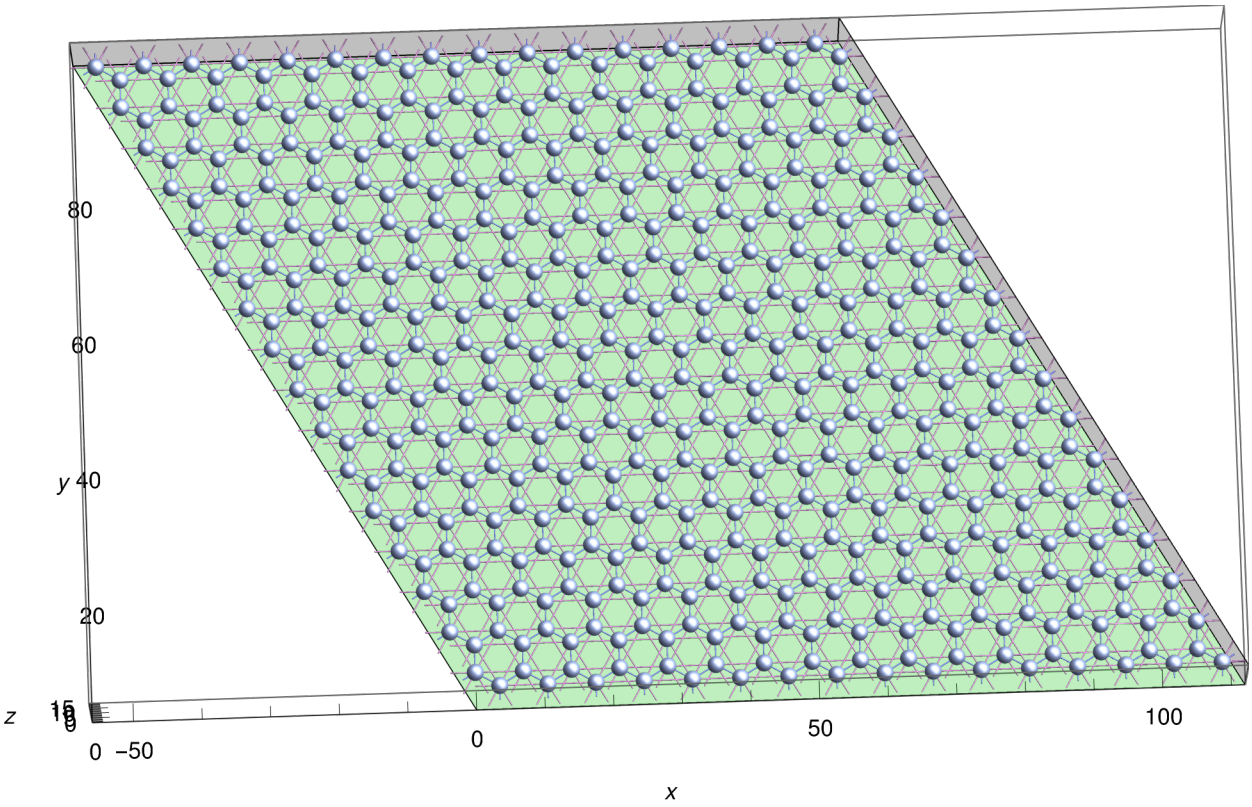
Now, one can do Monte Carlo simulations to see what happens.

5.2 MC (animation)

Let us run MC on a 8×8 lattice. Prepare `mclattice.dat, mcbonds.dat, pars.dat, Jb.dat`:

In[442]:=
```
mcLattice = replicatePOSCAR [skeleton , {8, 8, 1}];
mcBonds = replicateBonds [bonds , {8, 8, 1}];
drawStructure [mcLattice , Bonds → mcBonds , ElementLabels → None , AtomLabels → None , BondLabels → None ]
writePOSCAR [mcLattice , "MC/mclattice .dat "]
Export ["MC/mcbonds .dat ", mcBonds ᵀ, "Table "];
```

Out[444]=




```
JbInKelvin = {7.589688314632001` , 30.880441558736017` };
Export ["MC/Jb.dat ", JbInKelvin , "Table "];

temperature = 60.0 ;
Export ["MC/pars .dat ", #] & @ subst @"
useGUI         = 1                        # show graphics
stepsTuning    = 999999999
stepsCollect   = 0                        # if stepsCollect =0 then run Mickey Mouse animation
timeTuning     = 180                      # run for 3 minutes
timeCollect    = 0
spinType       = 3                        # Heisenberg spins
metropStepsize   = 0.04                    # Metropolis stepsize
temperature    = ${temperature }
";
```

In[451]:=
```
(*-------- MAKE EXECUTABLE (I.E., COMPILE AND LINK) --------*)
err = Run["(cd MC; make -B mc > make.log 2>&1)"];
If[err == 0, Print["Compiled executable mc successfully !"], FilePrint ["MC/make.log "]];
```

```
Compiled executable mc successfully !
```

Now open a terminal, cd to the `MC` directory, and run `./MC` so that you can see the usage info and GUI controls in the terminal. Are you surprised by the results?

5.3 Attempt 3: $T_c \approx 0$ K

For a 2D material such as CrI_3, if there is SU(2) symmetry in spin space, then the Mermin-Wagner

theorem tells us that Goldstone modes destroy long-range order at any finite temperature. Therefore, with this "improved" theory, our new prediction is $T_c = 0$ K!

6 Attempt 4: Calculating the magnetic anisotropy energy (MAE) function

6.1 General theory

In reality, due to spin-orbit coupling, the spins are coupled to the lattice, and prefer to align along certain directions. In the case of CrI_3, we need to calculate the MAE. We do this by doing a non-collinear magnetic calculation ("vasp_ncl"), where the spins can point in any direction. For several values of (θ, ϕ) in spherical polars:

– Set up a ferromagnetic state with $\mathbf{S} = (g\mu_B S)\,(\sin\theta\cos\phi,\ \sin\theta\sin\phi,\ \cos\theta)$, by setting either SAXIS or MAGMOM in the INCAR file

– Turn on spin-orbit coupling (LSORBIT)

– Run VASP to find $E(\theta, \phi)$.

Then fit the results using a function of the form

$$E(\theta, \phi) = N\,[K_{20}\,Y_{20}(\theta, \phi) + K_{40}\,Y_{40}(\theta, \phi) + K_{60}\,Y_{60}(\theta, \phi) + K_{66}\,Y_{66}(\theta, \phi) + \ldots]$$

where N is the number of spins and $Y_{lm} \equiv Y_l^m$ are the spherical harmonics. The details of which spherical harmonics to include depend on the symmetry of the lattice. I think monolayer CrI_3 has C_{3v} symmetry, which rules out things like Y_{44}.

From this point on, we make a leap of faith, and assume that the energy of the system can be parametrized as a Heisenberg model with magnetocrystalline anisotropy:

$$E(\{\mathbf{S}_i\}) = \sum_i \sum_{lm} K_{lm}\,Y_{lm}(\theta_i, \phi_i) - \sum_{b=1}^{B} J_b\,\mathbf{S}_{i_b} \cdot \mathbf{S}_{j_b}$$

where (θ_i, ϕ_i) are the angles describing spin i, and $\mathbf{S}_{i_b} \cdot \mathbf{S}_{j_b}$ is related to the angle between spin i and spin j. With the above notation, the "non-magnetic" baseline energy E_0 can be absorbed into K_{00}.

There is no guarantee that the energy is separable like this. Nevertheless, in the spirit of Ginzburg-Landau-type expansion, we may hope that we have at least captured the leading orders.

6.2 Extracting the lowest-order MAE parameter

Strictly speaking, one should follow the "recommended procedure for the calculation of magnetic anisotropies" on the VASP wiki:
https://www.vasp.at/wiki/index.php/LSORBIT.
Here I have not bothered to use the WAVECAR and CHGCAR from a collinear calculation. See files and scripts in `5MAE/`. In principle, we should calculate $E(\theta, \phi)$ for uniform ferromagnetic configurations in all directions, and fit it. In practice, it is often enough just to take 5-6 points along a meridian.

In[535]:=
```
dirs = FileNames ["5MAE/THETA*"]
```

Out[535]=
```
{5MAE/THETA000_PHI000 , 5MAE/THETA030_PHI000 , 5MAE/THETA060_PHI000 ,
 5MAE/THETA090_PHI000 , 5MAE/THETA120_PHI000 , 5MAE/THETA150_PHI000 , 5MAE/THETA180_PHI000 }
```

```
θdeg = {0, 30, 60, 90, 120, 150, 180};
En = Table[extractE0[dir <> "/OSZICAR"], {dir, dirs}]
```

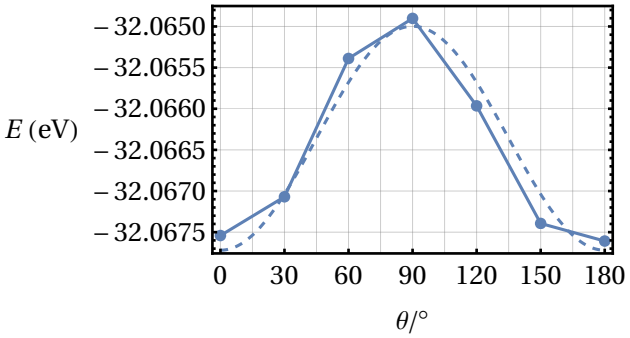
Out[537]= {-32.0675, -32.0671, -32.0654, -32.0649, -32.066, -32.0674, -32.0676}

Out[538]= 0.002706

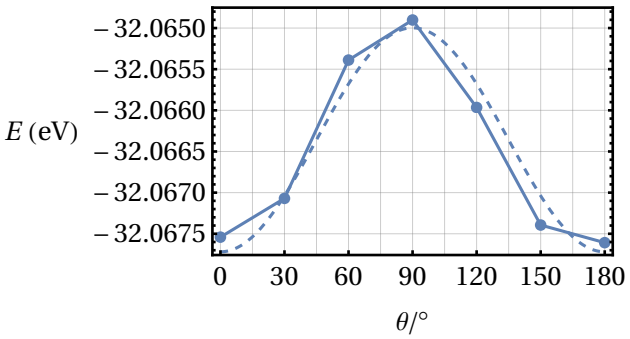
In[542]:=
```
data = {θdeg, En}^T;
Clear[θd]; myFit = Fit[data, {1, Sin[θd*π/180]^2}, θd]; Print["Fitted E_MAE(θ) = ", myFit];
Show[
 ListLinePlot[data, FrameLabel → {"θ/°", "E (eV)"}, FrameTicks → {{Automatic, Automatic}, {Range[0, 180, 30], None}},
  GridLines → {Range[0, 180, 15], Automatic}, PlotMarkers → Automatic],
 Plot[myFit, {θd, 0, 180}, PlotStyle → Dashed]
]
```

Fitted $E_{MAE}(\theta) = -32.0677 + 0.00272185\ \mathrm{Sin}\left[\frac{\pi\,\theta d}{180}\right]^2$

Out[544]=

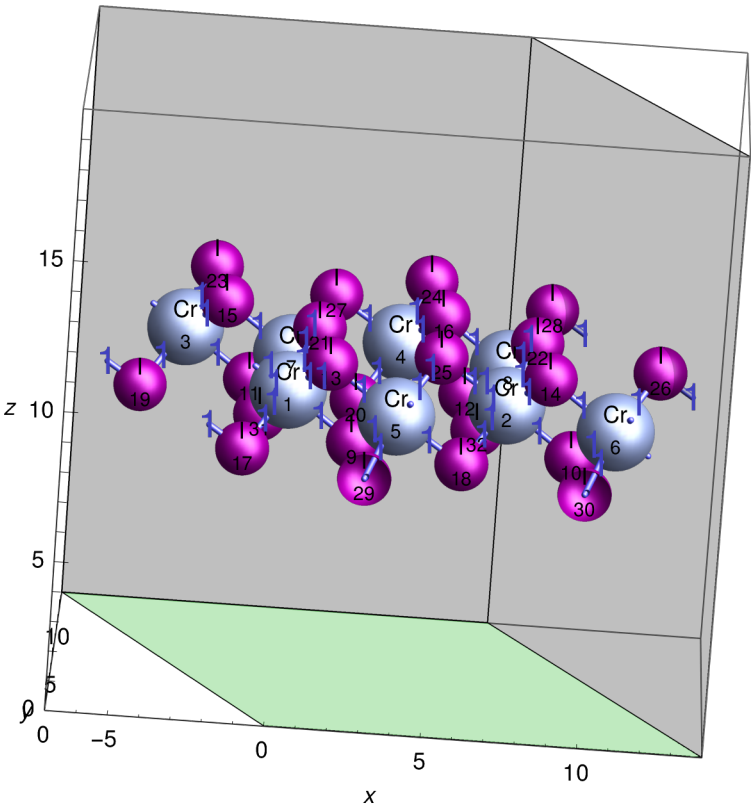


Out[541]=



The above plot shows the energy as a function of magnetization angle, from $\theta = 0°$ (spin up) to $\theta = 90°$ (spin in the XY plane) to $\theta = 180°$ (spin down). Note:

1. The plot is almost symmetrical. (If I did a more careful job, it should be symmetrical to within DFT precision.)
2. The plot is fit quite well by the form $E_{\mathrm{MAE}}(\theta) = E_0 + K \sin^2\theta$, or, equivalently, $E_{\mathrm{MAE}}(\theta) = E_0 + K_{20}\, Y_{20}(\theta, \phi)$.
3. The magnetic anisotropy energy (MAE) is $K = E(90°) - E(0°) \approx 2.7$ meV ≈ 31 K. This is very small! One needs to do the DFT accurately!
4. The energy minima occur at $\theta = 0°$ and $\theta = 180°$. This means that monolayer CrI_3 is an **easy-axis ferromagnet** whose spins prefer to **point out of the plane** of the 2D material.



In[545]:=
```
Max @ En - Min @ En
%* 16 021 766 340 / 1 380 649
```

Out[545]= 0.002706

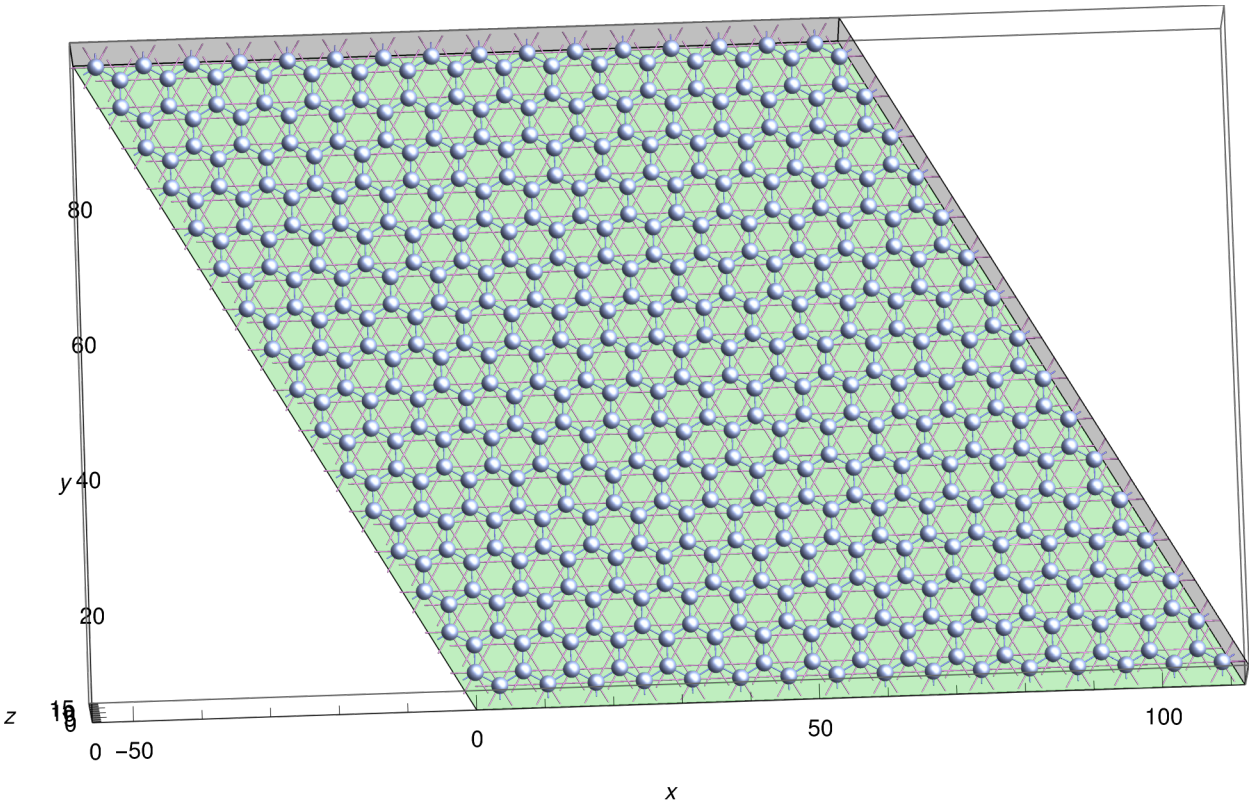
Out[546]= 31.4018

6.3 MC simulation of anisotropic Heisenberg model (animation)

Let us run MC on a 8×8 lattice. Prepare `mclattice.dat, mcbonds.dat, pars. dat, Jb.dat`:

In[311]:=
```
mcLattice = replicatePOSCAR [skeleton , {8, 8, 1}];
mcBonds = replicateBonds [bonds , {8, 8, 1}];
drawStructure [mcLattice , Bonds → mcBonds , ElementLabels → None , AtomLabels → None , BondLabels → None ]
writePOSCAR [mcLattice , "MC/mclattice .dat "]
Export ["MC/mcbonds .dat ", mcBonds ᵀ, "Table "];
```

Out[313]=



In[552]:=
```
JbInKelvin = {7.589688314632001` , 30.880441558736017` };
Export ["MC/Jb.dat ", JbInKelvin , "Table "];

KInKelvin = 31.401826036954304` ;
temperature = 60.0 ;
Export ["MC/pars .dat ", #] & @ subst @ "
useGUI         = 1                        # show graphics
stepsTuning    = 999999999
stepsCollect   = 0                        # if stepsCollect =0 then run Mickey Mouse animation
timeTuning     = 180                      # run for 3 minutes
timeCollect    = 0
spinType       = 3                        # Heisenberg model
temperature    = ${temperature }
easyAxisAniso  = ${KInKelvin }          # THIS IS NEW
";
```

In[320]:=
```
(*-------- MAKE EXECUTABLE (I.E., COMPILE AND LINK) --------*)
err = Run["(cd MC; make -B mc > make.log 2>&1)"];
If[err == 0, Print ["Compiled executable mc successfully !"], FilePrint ["MC/make.log "]];
```

```
Compiled executable mc successfully !
```

Now open a terminal, cd to the `MC` directory, and run `./MC` so that you can see the usage info and GUI controls in the terminal.

By exploring this simulation you can see that $T_c \approx 50$ K.

6.4 Attempt 4: $T_c \approx 50$ K

This is almost right! It is only slightly higher than the experimental Curie temperature (45 K)!

7 Attempt 5: Quantum-mechanical treatment

7.1 Basic idea

So far we have dealt with classical Ising and Heisenberg spins. In reality, the quantum nature of spin may be important. Each Cr^{3+} ion is a $S = \frac{3}{2}$ spin, which is not small, but not large either. In principle, this might be best described by a quantum Hamiltonian

$$\hat{H} = \sum_i E_{\text{MAE}}(\hat{\mathbf{S}}_i) - \sum_{b=1}^{B} J_b \, \hat{\mathbf{S}}_{i_b} \cdot \hat{\mathbf{S}}_{j_b}.$$

A major difference between this quantum Hamiltonian and the previous classical Hamiltonians is that here spins can undergo correlated quantum fluctuations, so that instead of being "classically antiparallel" with a dot product of $-S^2 \hbar^2$, they can go towards a quantum singlet state in which they are "even more antiparallel" and the dot product is $-S(S+1)\hbar^2$ (??)

Note that DFT captures *at least some* of these quantum fluctuations! Therefore, the correct way to fit a model is:
– Calculate the energy of FM and AFM configurations according to the quantum Hamiltonian, using analytical approximations or quantum Monte Carlo, as a function of model parameters (e.g., $\{J_b\}$)
– Calculate the energy of FM and AFM configurations according to DFT
– Choose the model parameters so that the model predictions match the DFT "data" as well as possible.

Obviously, analytic approximations are highly desirable, otherwise one has to perform a large number of QMC simulations, which is not fun.

We might expect that quantum fluctuations might reduce T_c slightly. One really has to do the above fitting carefully, though. There are papers on this.

Initialization cells (best version as of 2021-5-26)