Homework 7 Solution
PHZ 5156, Computational Physics
October 6, 2005

PART I

The structure of my code is:

```
from scipy import *
from RandomArray import *
import Gnuplot,Gnuplot.funcutils

def cluster(i,j):
def energy():
def metropolis():
def snapshot():

# Main routine

do12 = 1
testplot = 0
do4 = 0
do5 = 0

if do12:          #test functions cluster and energy
if testplot:      #test my snapshot function
if do4 or do5:    #run  problem 4 or problem 5
```

That is, the layout has the imports and then function definitions followed by the main driving code. The main code has blocks to run parts 1 and 2 (i.e., tests of functions cluster and energy) followed by a block that tests a function testplot that plots "snapshots" of the spin configurations, and finally a block that runs part 4 or 5. The block was chosen by setting one of the variables to 1 and the rest to 0.

PROBLEMS 1 and 2

The following two boxes show the functions cluster(i,j) and energy().

```
def cluster(i,j):
    """Energy of cluster centered on site (i,j)"""
    if i < n-1:
        ip1 = i + 1
    else:
        ip1 = 0
    if i > 0:
        im1 = i - 1
    else:
        im1 = n-1
    if j < n-1:
        jp1 = j + 1
    else:
        jp1 = 0
    if j > 0:
        jm1 = j-1
    else:
        jm1 = n-1
    cluster = -s[i,j]*(s[ip1,j]+s[im1,j]+s[i,jp1]+s[i,jm1])
    cluster = float(cluster)
    return cluster
```

```
def energy():
    n,m = shape(s)
    energy = 0
    for i in arange(n):
        for j in arange(n):
            energy = energy + cluster(i,j)

    energy = energy/2.
    return  energy
```

These are tested by setting the variable do12=1 in the main routine, which causes this block of code to run, with output that is just below:

```
if do12: #test functions cluster and energy
         print "Test functions cluster and energy"
         n = 4
         s = ones((n,n))

         i,j = 2,2
         print "s=\n",s
         print "i,j,s(i,j)=",i,j,s[i,j]
         print "test of cluster with all up = ",cluster(i,j)
         print "test of energy with all up = ",energy()

         s[i,j] = - s[i,j]
         print "s=\n",s
         print "i,j,s(i,j)=",i,j,s[i,j]
         print "test of cluster with middle spin flipped = ",cluster(i,j)
         print "test of energy with middle spin flipped = ",energy()
```

```
Test functions cluster and energy
s=
[[1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]]
i,j,s(i,j)= 2 2 1
test of cluster with all up =  -4.0
test of energy with all up =  -32.0
s=
[[ 1  1  1  1]
 [ 1  1  1  1]
 [ 1  1 -1  1]
 [ 1  1  1  1]]
i,j,s(i,j)= 2 2 -1
test of cluster with middle spin flipped =  4.0
test of energy with middle spin flipped =  -24.0
```

A little thought shows that this is correct. For instance when all spins are up the total energy is –J times the number of pairs. There are two pairs per site, so when all spins are aligned the energy is $-2Jn^2$. With J=1 and n=4 this gives –32.

Flipping one spin changes the energy of each of its surrounding bonds from –1 to +1. This changes the energy by 4 x 2 = 8, so E=-32+8=-24.

PROBLEM 3

My final Metropolis code follows. I tested this by getting part 4 to work with small systems, so I will not show any tests. Notice that this code returns the total energy per spin at each sweep.

```
def metropolis():
        E = energy()
        Energies=zeros(nsweeps+1,Float)
        Energies[0]=E
        spinplot()

        for sweep in range(1,nsweeps+1):
                for step in range(n**2):
                        i,j = randint(0,n,2)
                        delta = -2 * cluster(i,j)
                        if delta <= 0:
                                s[i,j] = -s[i,j]
                                E = E + delta
                        else:
                                w = random()
                                if  exp(-delta/T) >=  w:
                                        s[i,j] = -s[i,j]
                                        E = E + delta
                        # End of spin-flip test
                #End of loop over steps

                Energies[sweep] = E
                if (((sweep+1)/10)*10 == sweep+1): spinplot(retr(sweep))
        return  Energies/(n**2)
```

This calls snapshot() with the initial spins and after every so many sweeps (10 here) to plot a picture of the spin configuration as it evolves. The function snapshot() is shown below.

```
def spinplot(titleplot):
        g = Gnuplot.Gnuplot(debug=1)
        nup = 0
        ndown = 0
        xup = zeros(n*n+1,Float)
        yup = zeros(n*n+1,Float)
        xdown = zeros(n*n+1,Float)
        ydown = zeros(n*n+1,Float)
        for i in arange(0,n):
                for j in arange(0,n):
                        if s[i,j] == 1:
                                nup = nup + 1
                                xup[nup] = i
                                yup[nup] = j
                        else:
                                ndown = ndown + 1
                                xdown[ndown] = i
                                ydown[ndown] = j

        options = 'set terminal aqua 1 title "' + titleplot + '" fsize 32'
        g(options)
        g.title(titleplot)
        g('set xrange [-.5:31.5]')
        g('set yrange [-.5:31.5]')
        g('set pointsize 0.6')
        g('set size square')
        g('set style fill')
        g('set noxtics')
        g('set noytics')
        if nup != 0: g1 = Gnuplot.Data(xup[1:nup+1],yup[1:nup+1],with='points  3')
        if ndown != 0: g2 = Gnuplot.Data(xdown[1:ndown+1],ydown[1:ndown+1],with='points  10')
        if nup != 0:
                if ndown !=0:
                        g.plot(g1,g2)
                else:
                        g.plot(g1)
        else:
                g.plot(g2)
        return
```

For each up spin, i and j go into xup and yup, respectively.

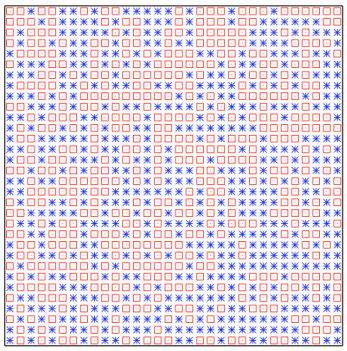
For each down spin, i and j go into xdown and ydown.

I worked a bit to get the titles and sizes that made these plots look OK, using Gnuplot.

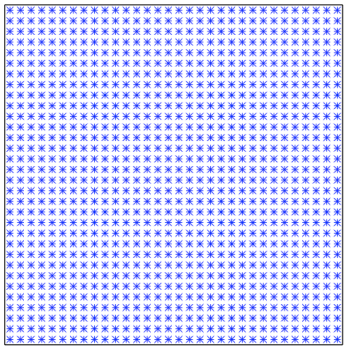
Before using this I tested it with this block in the main routine:

```
if testplot:
        n = 32
        s = randint(0,2,(n,n))*2 - 1
        spinplot("Random")
        s = ones((n,n))
        spinplot("All  Up")
        s = -s
        spinplot("All Down")
```

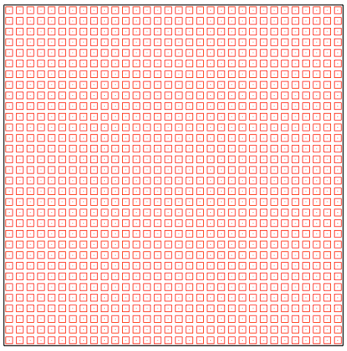
The results are below.



Random



All Up



All Down

These interesting pictures show a random grid of 32 x 32 spins, then all spins up, then all spins down.

PROBLEM 4

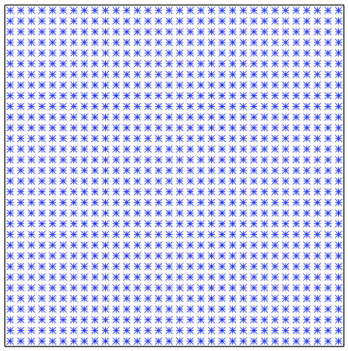
All that is left to show is the corresponding block of main code, which follows. The most interesting piece of this is the construction of a title for each plot of energies vs. sweep.

```
if do4 or do5:
        n = 32
        nsweeps = 50

        T = 3.   # I change T by hand and rerun
        if do4:
                s = ones((n,n))
        else:
                s = randint(0,2,(n,n))*2 - 1

        Energies = metropolis()

        g=Gnuplot.Gnuplot(debug=0)
        g1 = Gnuplot.Data(Energies)
        g('set data style lines')
        title = "Energies per spin with T=" + repr(T)
        g.title(title)
        g.plot(g1)
```

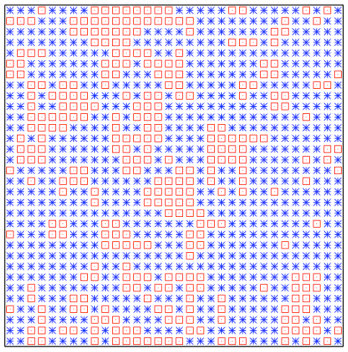
Rather than using a loop over temperatures, I set the value of T by hand and ran this for one temperature at a time. That made it easier for me to keep track of the plots and copy them into this document.

I played with the number of sweeps (nsweeps) to get a find a value that seemed to let the system reach equilibrium.

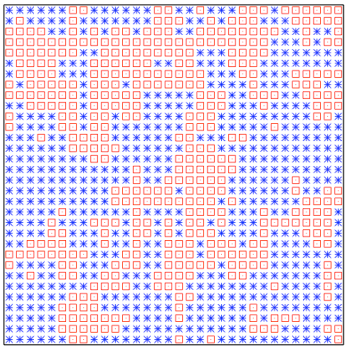
Temperature T=3



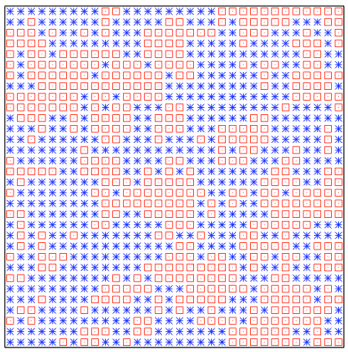
Initial



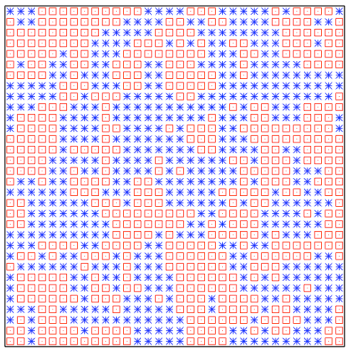
10



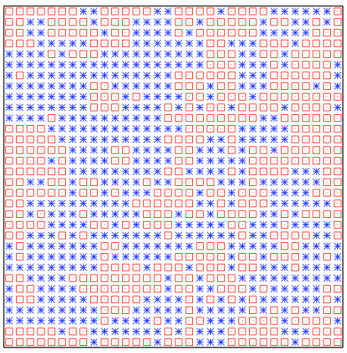
20



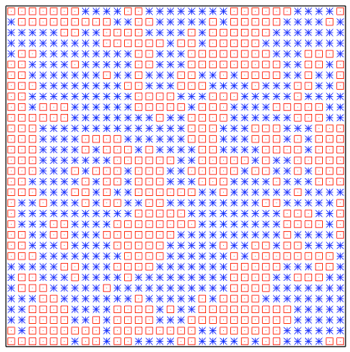
30



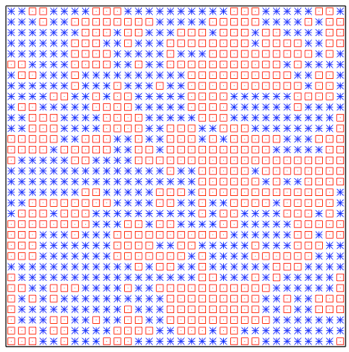
40



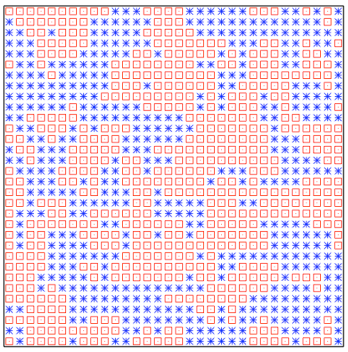
50



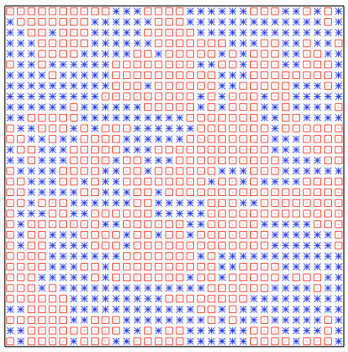
60



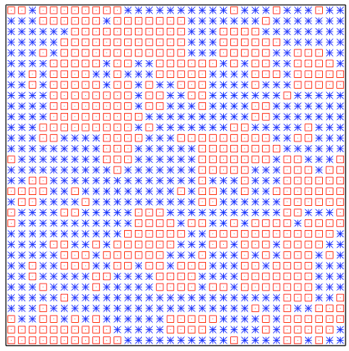
70



80

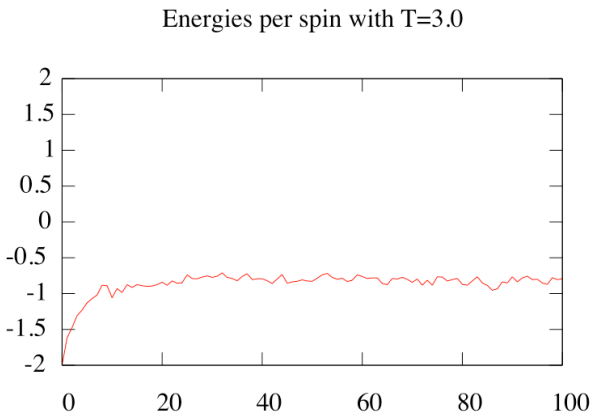


80



90

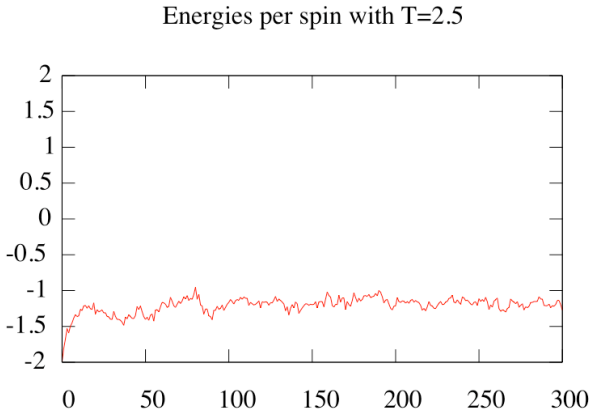
A close look at the sequence of pictures above shows that the spins move quickly to a situation where roughly half are up and half are down. But they are not distributed entirely randomly – instead they form "patches" of up and down. As "time" goes on these patches move and shift, but the system stays more or less half up and half down.



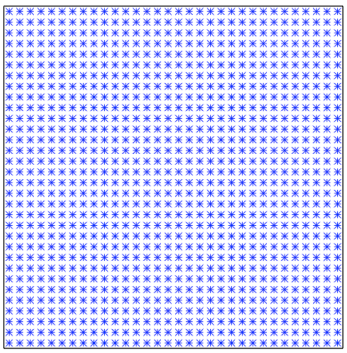
The above plot showing the average energy per spin tells a similar story. After 30 or 40 sweeps the system appears to have equilibrated. The fact that the energy stabilizes around –0.8 or so is related to the size of the patches in the snapshots above.

Temperature T=2.5

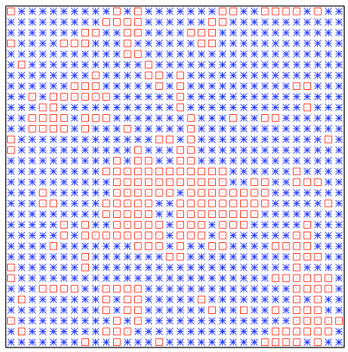
At this temperature the system took longer to equilibrate. I ended up running for 300 sweeps, plotting a snapshot every 50 sweeps. Notice that the energy seems to be about –1.1, lower than at T=3, and that the fluctuations are larger here (the energy seems to wander around a bit more).



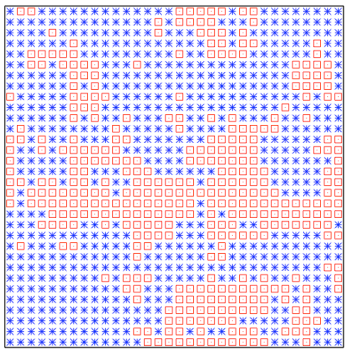
Notice all that in the snapshots below the patches are bigger than at T=3.



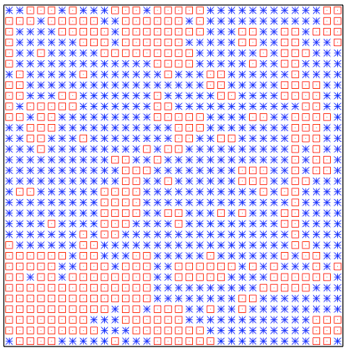
Initial



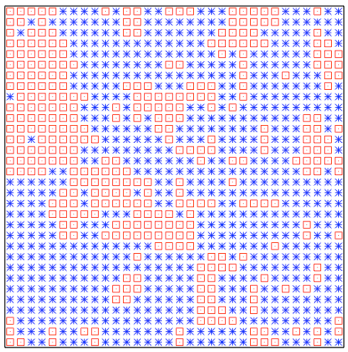
50



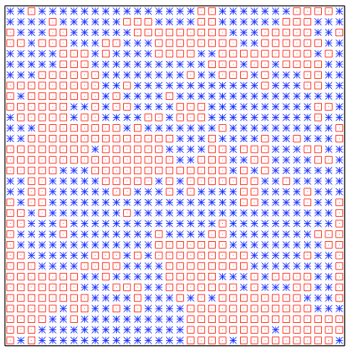
100



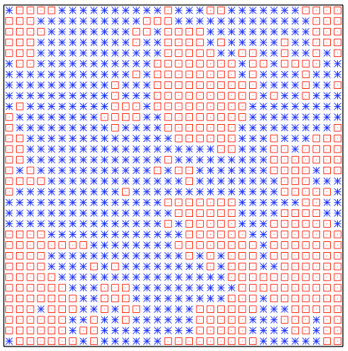
150



200



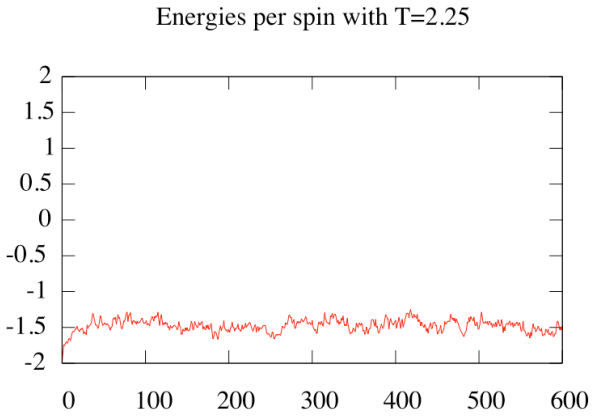
250

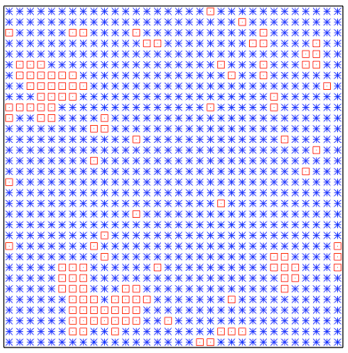


300

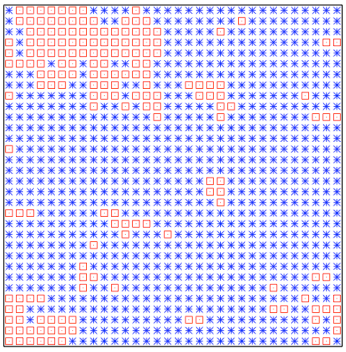
Temperature T=2.25

This took even longer to equilibrate, and the energy plot has even larger fluctuations. The average energy is somewhere around –1.5. The snapshots show similar large fluctuations, with patches of down spins wandering about. I ran for 600 sweeps, plotting every 100. In the following I did not plot the initial up-spin configuration.

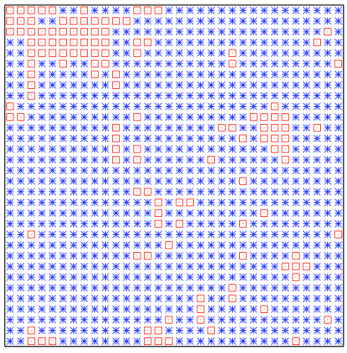




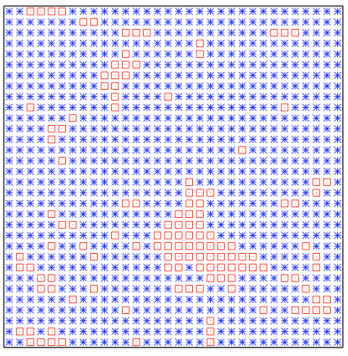
100



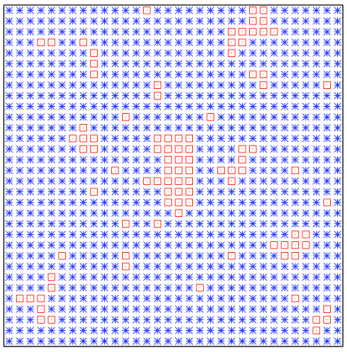
200



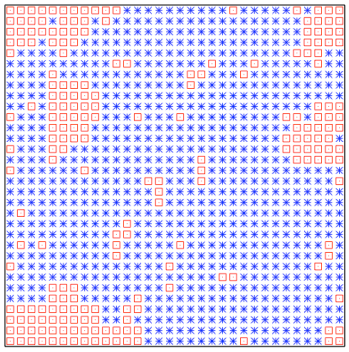
300



400



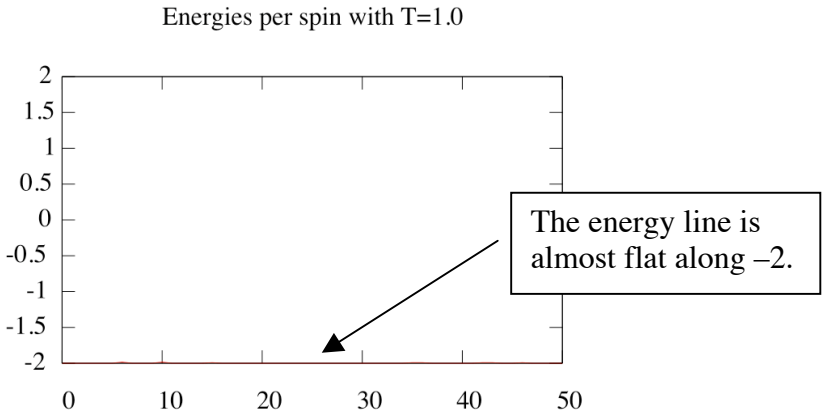
500

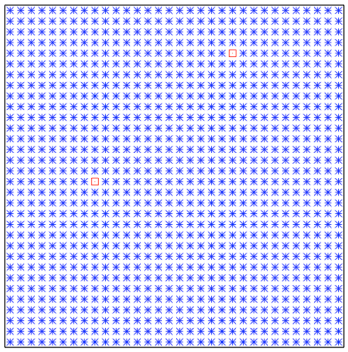


600

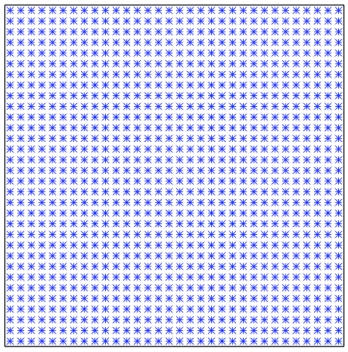
Temperature T=1

It turns out that the initial condition is essentially the equilibrium state: at T=1 the spins tend to be almost completely aligned. As a result it took virtually no sweeps to reach equilibrium. The energy per spin is nearly –2 and there are few fluctuations.

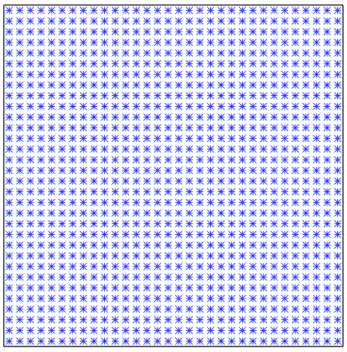




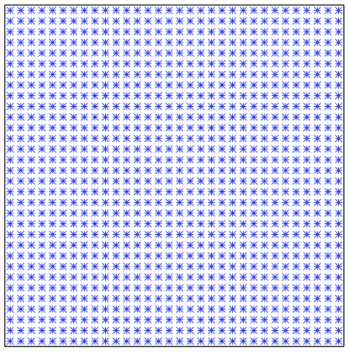
10



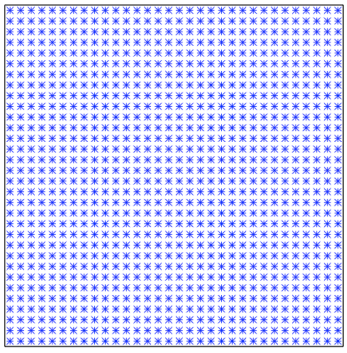
20



30



40



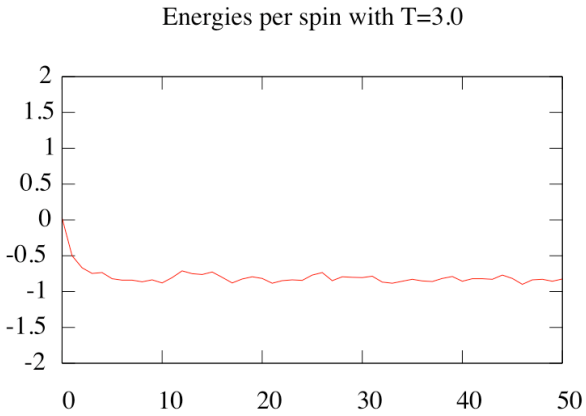
50

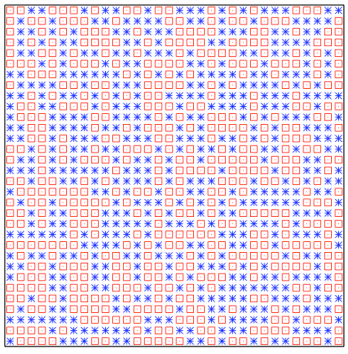
PROBLEM 5

This problem redoes the above but with a different initial spin configuration – random spins instead of aligned spins. It is easy to see what will happen – the system may equilibrate more rapidly at higher temperatures (where the spins are more random), and definitely will take longer at low temperatures (where they want to align).

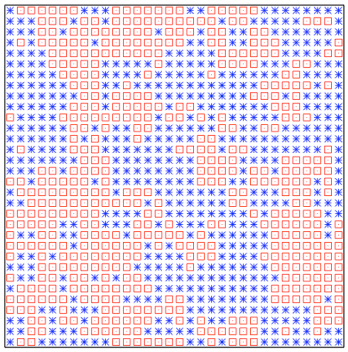
Temperature T=3

Notice that the average energy after equilibration is the same as for the other starting spin configuration. It had better be, or in fact equilibrium has not been reached in one or both. Also notice that the spins change from random to a state where about half continue to be up and half down, but where they are clustered into patches or islands.

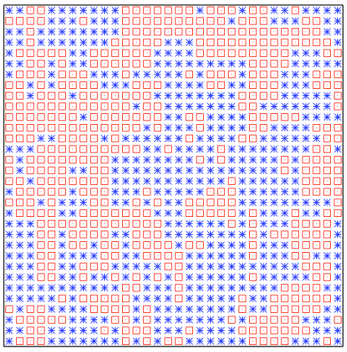




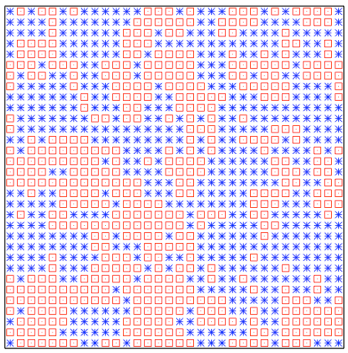
Initial



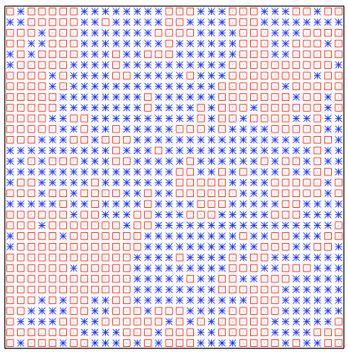
40



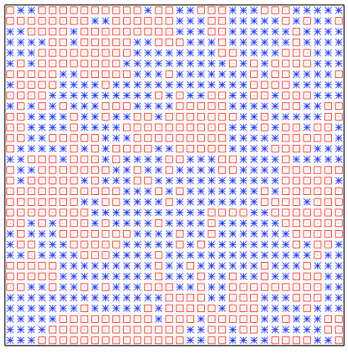
20



30



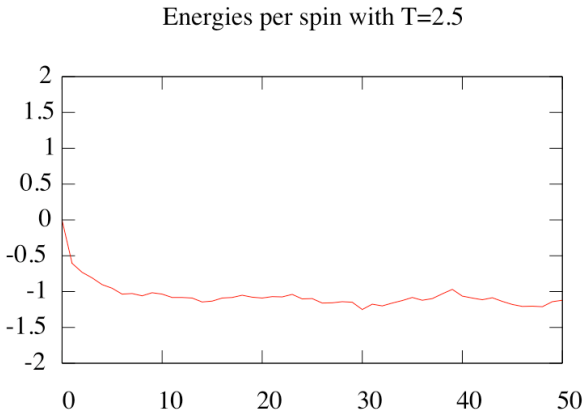
10

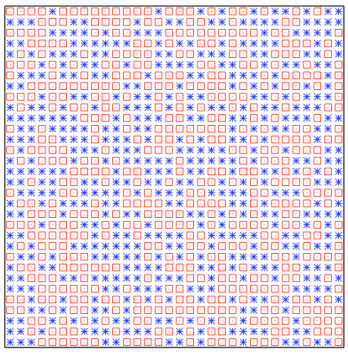


50

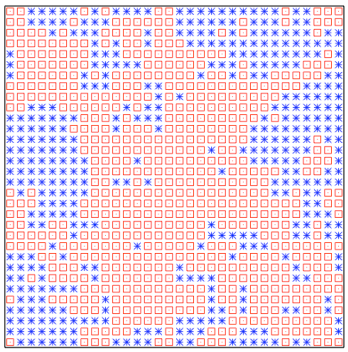
Temperature T=2.5

This seemed to equilibrate quickly, and ended up as necessary at about the same average energy as the other T=2.5 run. The patches here also look like the patches in the previous T=2.5 run, although up and down seem to have reversed.

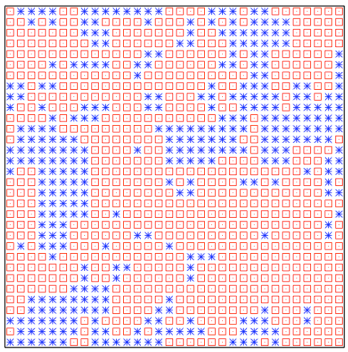




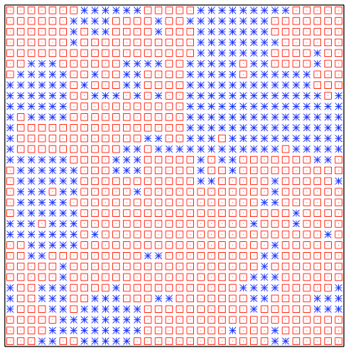
Initial



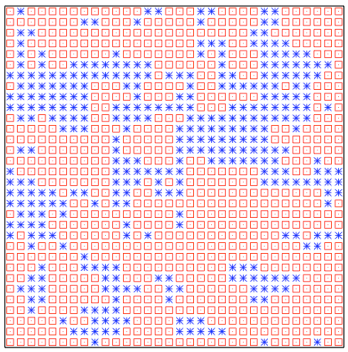
10



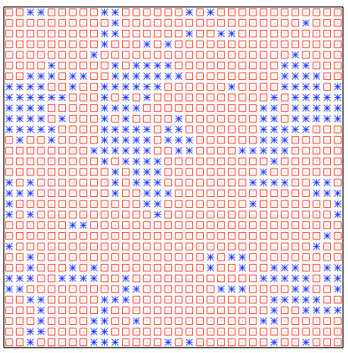
20



30



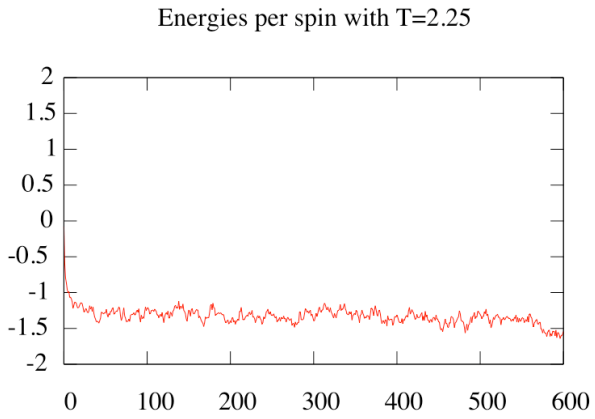
40

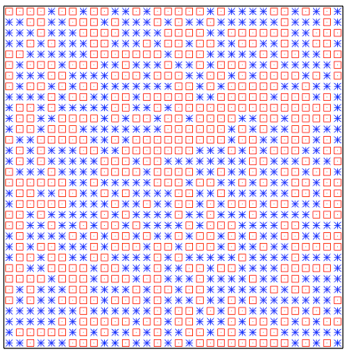


50

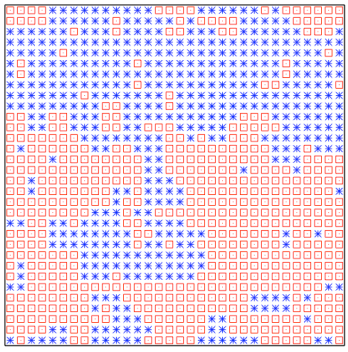
Temperature T=2.25

I ran 600 sweeps so it would be easy to compare with the earlier T=2.25 run. This run illustrates the large fluctuations at T=2.25 – the last sweep shown has fluctuated to almost all down.

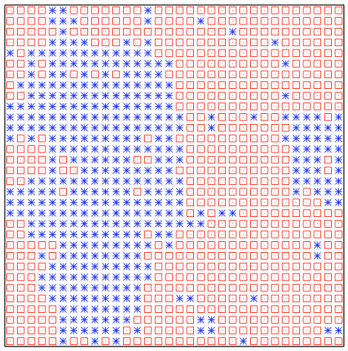




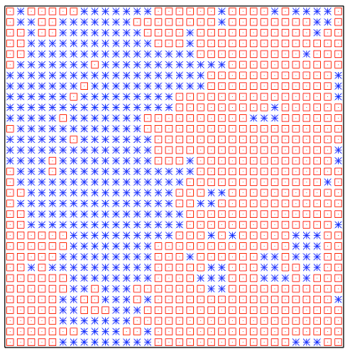
Initial



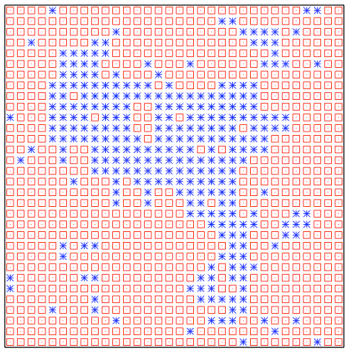
100



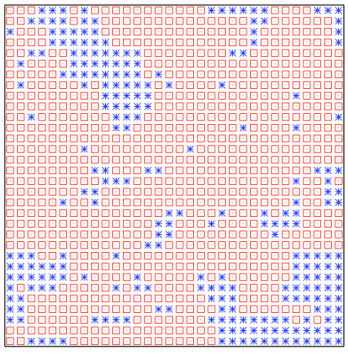
200



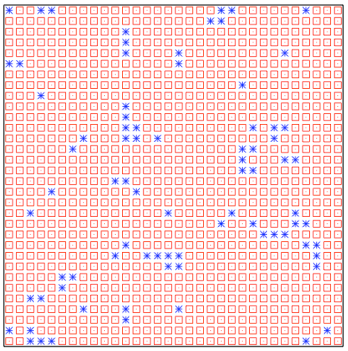
300



400



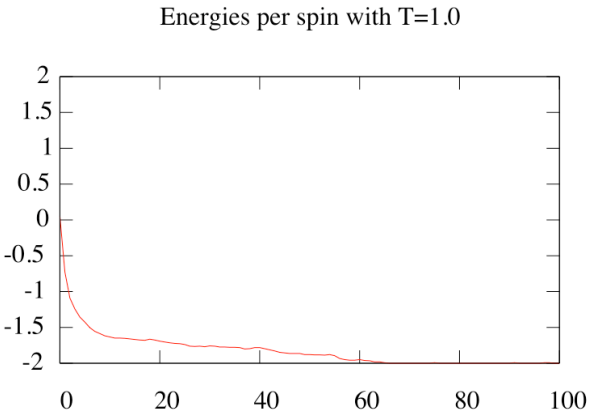
500

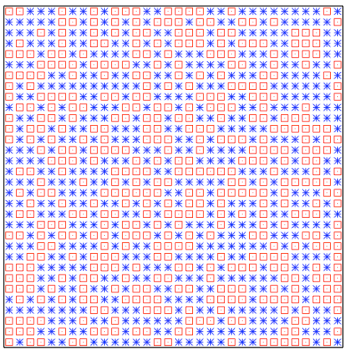


600

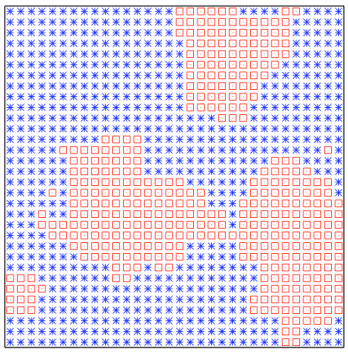
Temperature T=1

At low temperatures the spins want to align, so the random starting configuration must change quite a bit before it equilibrates. There is some tendency to develop big clusters of all-up and all-down that get stuck. Here is a run that managed to shrink the down-spin clusters to size zero in fewer than 100 sweeps.

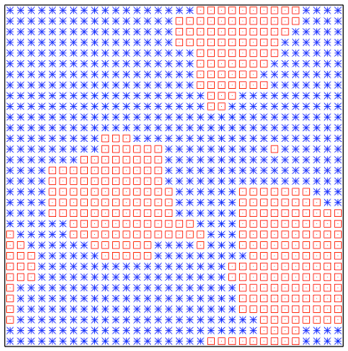




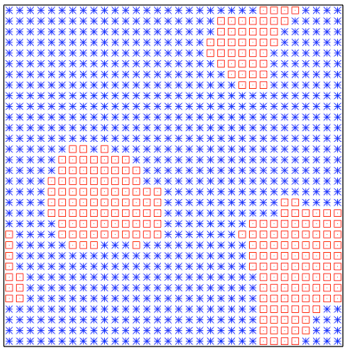
Initial



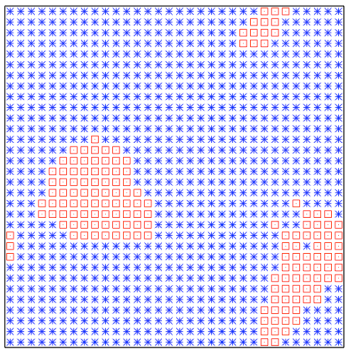
10



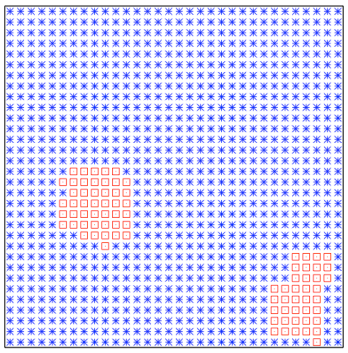
20



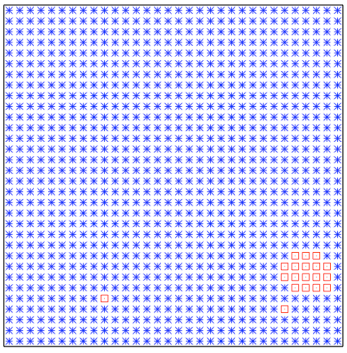
30



40



50



60

PART II

The necessary code modification is quite small. The top few lines initialized an array m and stored the initial magnetization there:

```
def metropolis():
        E = energy()
        Energies=zeros(nsweeps+1,Float)
        Energies[0]=E
        m = zeros(nsweeps+1,Float)
        m[0] = float(sum(sum(s)))/(n**2)
```

The bottom few lines of metropolis() changed to:

```
                Energies[sweep] = E
                m[sweep] = float(sum(sum(s)))/(n**2)
                #if ((sweep/10)*10 == sweep): spinplot(repr(sweep))
        return  Energies/(n**2),m
```

I commented out all the snapshots and other output.

The main code included this new block:

```
if partII:
        n=32

        T = 2.25
        nsweeps = 1000
        nequil = 200

        s = ones((n,n))
        #s=randint(0,2,(n,n))*2 - 1
        Energies,m = metropolis()
        maverage = sum(m[nequil:])/len(m[nequil:])
        print "T=",T,"maverage=",maverage

        g=Gnuplot.Gnuplot(debug=0)
        options = 'set terminal aqua 1 title "m vs. sweep" fsize 32'
        g(options)
        g('set data style lines')
        title = "Magnetization vs. sweep with T=" + repr(T)
        g.title(title)
        g('set yrange [-1.:1.]')
        g1 = Gnuplot.Data(m)
        g.plot(g1)
```

I used this for T<= 2.25
and
this for T>=2.50

I ran for each temperature, deciding by trial and error what to use for nsweeps and nequil.

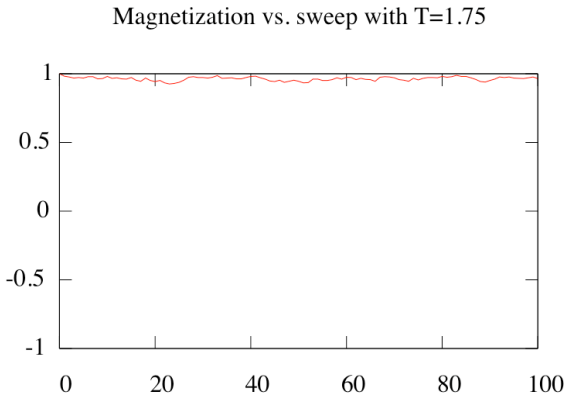
In these runs I used trial and error to decide on the following:

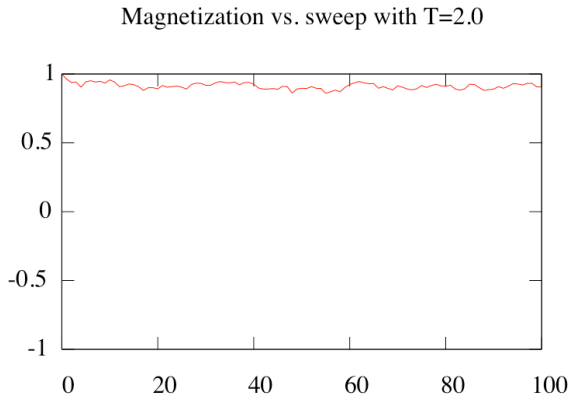
T	1.75	2.00	2.25	2.50	2.75	3.00	3.25	3.50	3.75
nsweep	100	100	2000	300	100	100	100	100	100
nequil	20	20	100	50	20	20	20	20	20

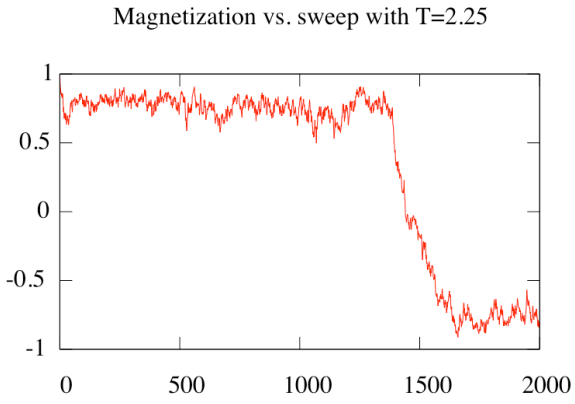
I collected this data one temperature at a time:

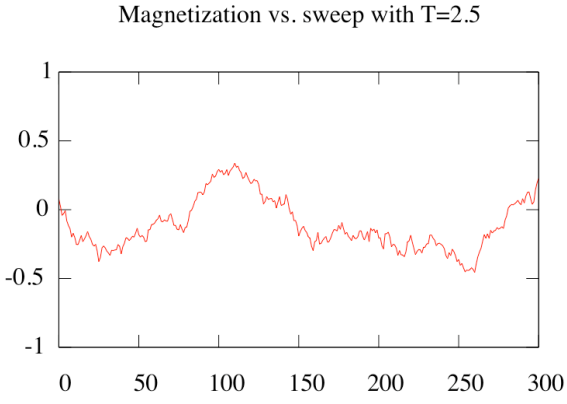
T	1.75	2.00	2.25	2.50	2.75	3.00	3.25	3.50	3.75
maverage	0.9623	0.9086	0.3404	-0.0919	0.0261	-0.0158	-0.0085	-0.0319	0.0322

Following are plots of all the magnetizations vs. sweep for each of these temperatures. Notice a key point: fluctuations get very large near Tc (which is around 2.27). For instance, at T=2.25 the system moves from nearly all up to nearly all down. The average of this over hundreds of thousands of sweeps would be close to zero.

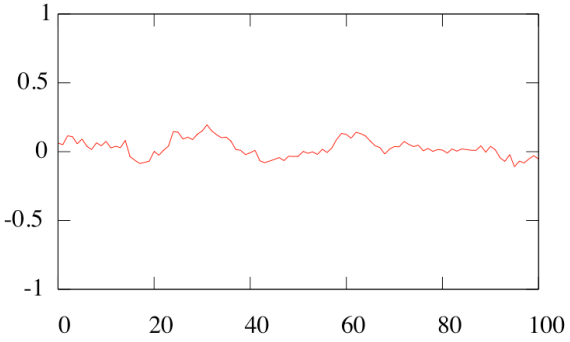




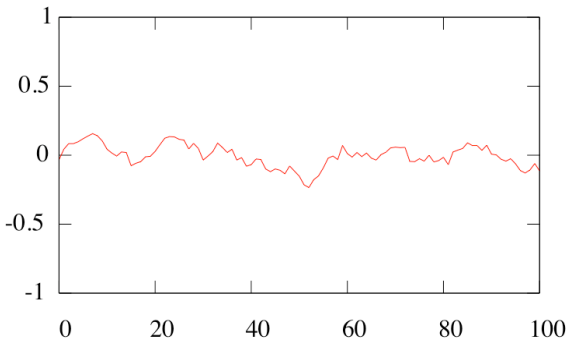




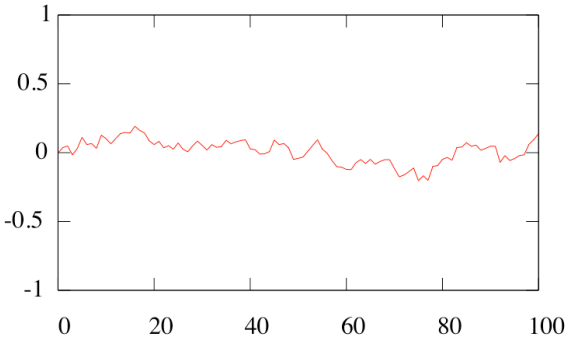
Magnetization vs. sweep with T=2.75



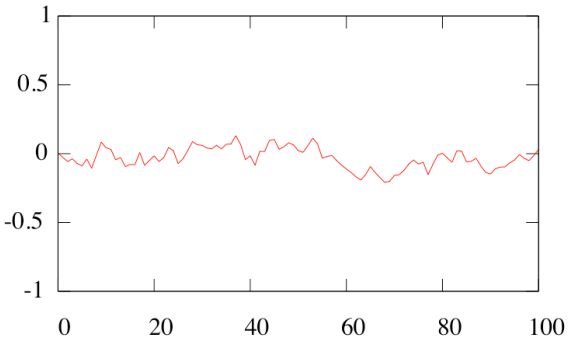
Magnetization vs. sweep with T=3.0



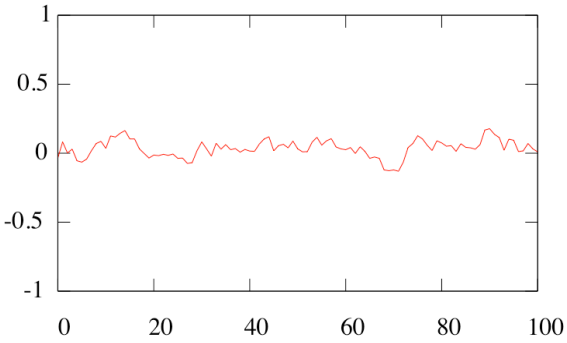
Magnetization vs. sweep with T=3.25



Magnetization vs. sweep with T=3.5

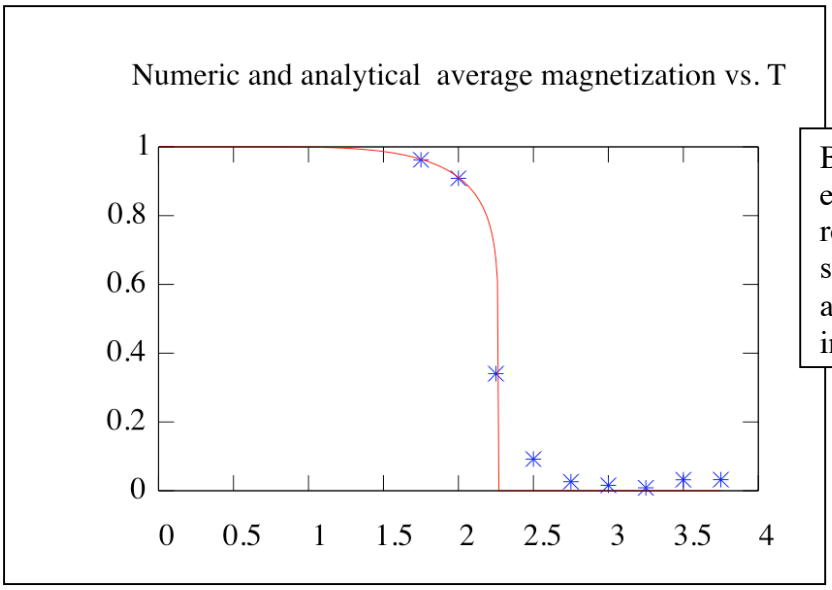


Magnetization vs. sweep with T=3.75



Once all this was done I wrote the following little Python code to plot the above numeric results for the magnetization and also the Yang analytic results.

```
from scipy import *
import Gnuplot,Gnuplot.funcutils

Tnum = arange(1.75,4.00,.25)

#Numeric – read in from previous table.
Mavenum = array( (0.9623, 0.9086, 0.3404, -0.0919, 0.0261, -0.0158,
-0.0085, -0.0319, 0.0322))
Mavenum = abs(Mavenum)

Tc = 2./log(1.+sqrt(2.))
print "Tc=",Tc

Ta = arange(0.01,3.76,.01)
Maveanal = zeros(len(Ta),Float)

for j in arange(len(Ta)):
        if Ta[j] < Tc:
                Maveanal[j] = (1-(sinh(2./Ta[j]))**(-4))**(1./8.)

g=Gnuplot.Gnuplot(debug=0)
options = 'set terminal aqua 1 title "m vs. T" fsize 32'
g(options)
g('set data style lines')
g.title('Numeric and analytical  average magnetization vs. T')
g('set yrange [0.:1.]')
g1 = Gnuplot.Data(Tnum,Mavenum,with='points  3')
g2 = Gnuplot.Data(Ta,Maveanal,with='lines  1')
g.plot(g1,g2)
```



Beautiful! The rounding off evident in my numerical results is because my system was finite – the analytical results are for an infinite system.