Homework8

David Mathews

March 26, 2018

1 Observables

$$C_v \equiv \frac{1}{N_s} * \frac{d* < E >}{dT} = \frac{1}{N_s * T^2} * (< E^2 > - < E >^2)$$
$$\frac{1}{N_s} * \frac{d}{dT} \frac{\sum_c E * e^{-B*E}}{\sum_c e^{-BE}} = \frac{1}{N_s * T^2} * (< E^2 > - < E >^2)$$
$$\frac{\sum_c \frac{E^2}{T^2} * e^{-BE}}{\sum_c e^{-BE}} - \sum_c \frac{E}{T^2} * e^{BE} * \sum_c E * e^{-BE} = \frac{1}{N_s * T^2} * (< E^2 > - < E >^2)$$
$$\frac{1}{N_s * T^2} * (< E^2 > - < E >^2) = \frac{1}{N_s * T^2} * (< E^2 > - < E >^2)$$

This proof is just simple differentiation. The main jump came from rewriting the exponential $\sum_c \frac{E}{T^2} * e^{BE} = \sum_c \frac{E}{T^2} * e^{2BE} * e^{-BE}$ which allows for the second term to be rewritten.

$$X \equiv \frac{1}{N_s} \frac{d < E >}{dT}|_{h=0} = \frac{1}{N_s * T} * (< M^2 > - < M >^2)$$
$$< M > = \frac{\sum_c M * e^{-BE} e^{hMB}}{\sum_c e^{-BE+hMB}}$$

We again need to use chain rule to evaluate this

$$\frac{d}{dh} \sum_c M * e^{-BE} e^{hMB} = \sum_c M^2 * B * e^{-BE+hMB}$$
$$\frac{d}{dh} \sum_c e^{-BE+hMB} = \sum_c -MB * e^{-BE+hMB}$$

Combining using chain rule gives

$$\frac{d < E >}{dT}|_{h=0} = \frac{= \sum_c M^2 * B * e^{-BE+hMB}}{\sum_c e^{-BE+hMB}} - \sum_c MBe^{BE-hME}$$

Using the same method as in the first proof, the rightmost term can be rewritten

$$\frac{d < E >}{dT}|_{h=0} = B* < M^2 > -B* < M >^2$$

Bringing in the outside terms again

$$C_v \equiv \frac{1}{N_s} \frac{d < E >}{dT}|_{h=0} = \frac{N_s}{T}(< M^2 > - < M >^2)$$

2 Full Enumeration

ising_enum_hw.py

This code was completed as shown below.

```
def get_ener(spin):
    L=int(np.sqrt(len(spin)))
    spm=np.reshape(spin,(L,L))#create LxL matrix
    spx=np.roll(spm,1,axis=1)
    spy=np.roll(spm,1,axis=0)
    enerx=np.dot(spin,spx.flatten())
    enery=np.dot(spin,spy.flatten())
    return -1*(enerx+enery)
```

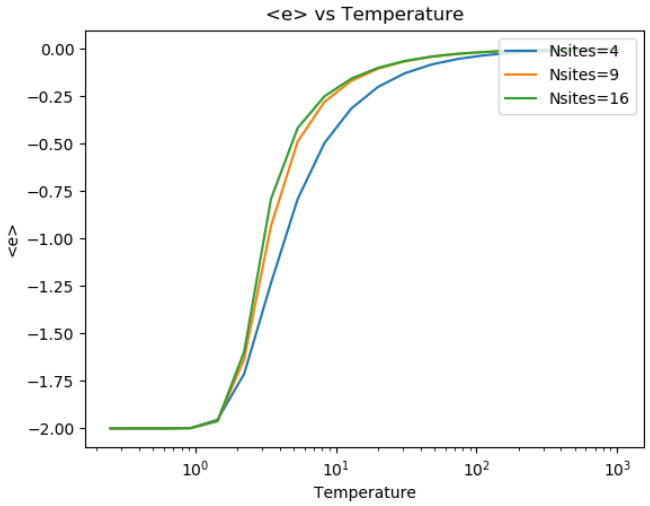
This function uses the reshape and roll functions heavily to manipulate the configuration as desired. As the dot product function is far faster than a for loop, the goal is to make it so each dot product calculates a different part of the configuration energy. By rolling the matrix in a particular direction, x or y, and taking the dot product with the original matrix (after being spread back into a 1d array), the different energies can be calculated very efficiently.

In order to calculate the expectation values of e, e^2 and m^2, the following code was written.

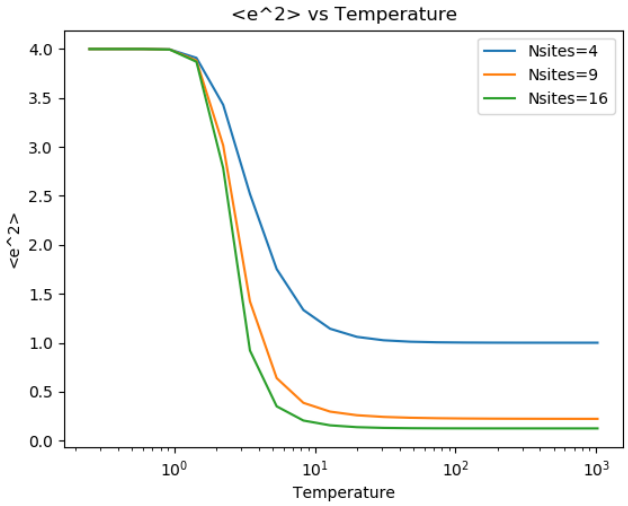
```
def enermags(Nstate,Nsite,L):
  mags=np.zeros(Nstate)
  eners=np.zeros(Nstate)
  for state in range(Nstate):
    svec=np.array(list(np.binary_repr(state).zfill(Nsite))).astype(np.int8)
    spin=2*svec-1
    mags[state]=float(get_mag(spin))
    eners[state]=get_ener(spin)
  return eners,mags

numNsite=len(Ls)
expecteners=np.ndarray(shape=(numNsite,len(temps)))
expecteners2=np.empty_like(expecteners)
expectmags=np.empty_like(expecteners)
expectmags2=np.empty_like(expecteners)
specval=np.empty_like(expecteners)
susceptval=np.empty_like(expecteners)

for j in range(len(Ls)):
  Nsite=Ls[j]*Ls[j]
  Nstate=int(pow(2,Nsite))
  eners,mags=enermags(Nstate,Nsite,Ls[j])
  for i in range(len(temps)):
    T=temps[i]
    #normalization values
    normal=np.sum(np.exp(-1.0/float(T)*eners))
    topener=np.sum(eners/float(Nsite)*np.exp(-1.0/float(T)*eners))
    topmag=np.sum(mags/float(Nsite)*np.exp(-1.0/float(T)*eners))
    topener2=np.sum(eners*eners/float(Nsite)/float(Nsite)*np.exp(-1.0/float(T)*eners))
    topmag2=np.sum(mags*mags/float(Nsite)/float(Nsite)*np.exp(-1.0/float(T)*eners))
    expecteners[j][i]=topener/normal
    expecteners2[j][i]=topener2/normal
    expectmags[j][i]=topmag/normal
    expectmags2[j][i]=topmag2/normal
    specval[j][i]=float(Nsite)/(float(T))*(expecteners2[j][i]-expecteners[j][i]**2.0)
    susceptval[j][i]=float(Nsite)/(float(T))*(expectmags2[j][i]-expectmags[j][i]**2.0)
```

This is not the entire code, that has been attached to my submission, but the most important parts. This section of the code computes the various expectation values and stores them into Numpy arrays for plotting later on in the code. The general idea here is that the energies and magnetizations of a configuration don't actually change with temperature. Their expectation values do, but the energies of each configuration don't. I use this to avoid calculating extra values unnecessarily.

The following plots were made with this code.



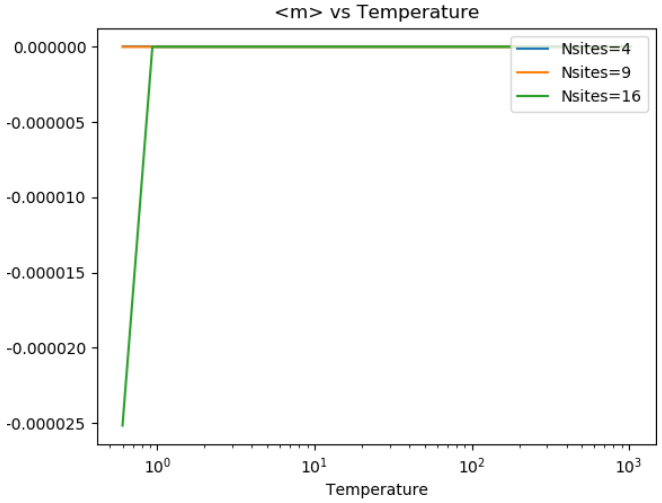
(a) Expectation values of e



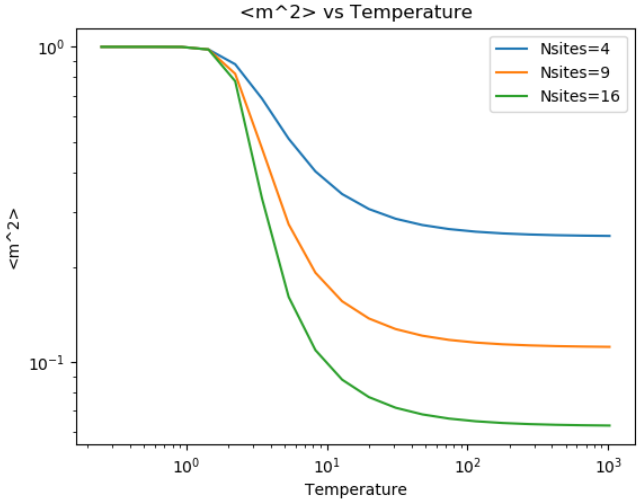
(b) Expectation values of e^2

Figure 1: Both Energy expectation values

As we can see from these plots, the expectation value of energy behaves asymptotically as temperature rises. There is some energy threshold, in the case of $< e >$ this is 0, that the energy will go to as the temperature increases.



(a) Expectation values of m

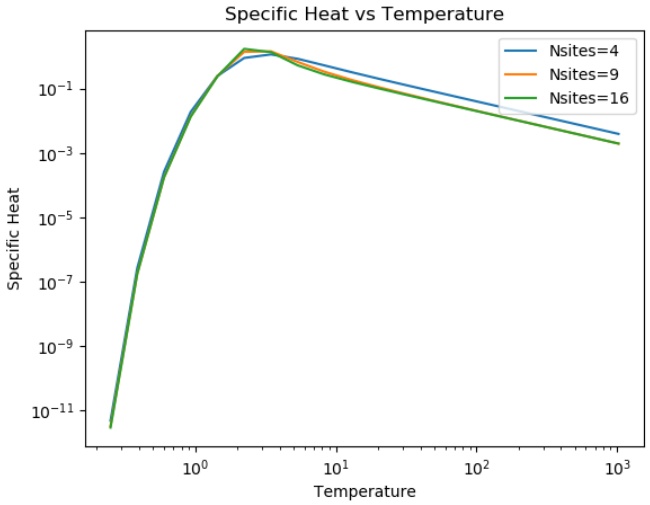


(b) Expectation values of m^2

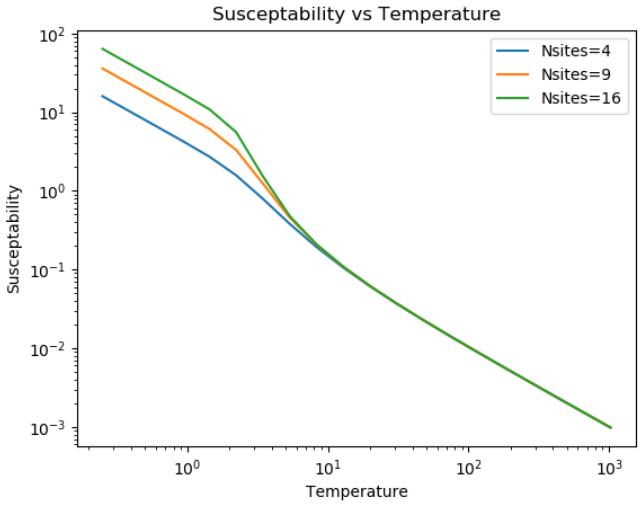
Figure 2: Both magnetization expectation values

For magnetization, as expected $< m >$ is 0 except for some issues with computer precision. $< m^2 >$ seems to behave very similarly to $< e^2 >$ as it starts off at some maximal value and then proceeds to decrease until reaching its asymptotic limit.

For the final 3 plots, I calculate the susceptibility,specific heat, and energy at a range of temperatures. These are shown below.



(a) Specific Heat



(b) Susceptibility

Figure 3: Specific Heat and Susceptibility

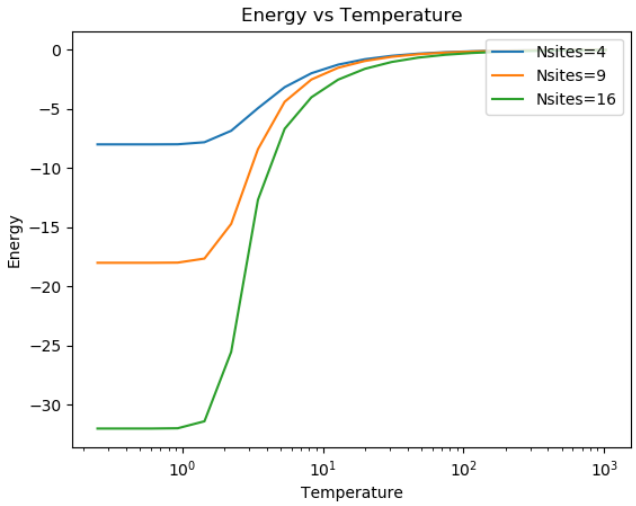


Figure 4: Expectation value of the energy changing with temperature