HeisenbergEntanglementHintsPython

February 8, 2024

Heisenberg Entanglement

(Sethna, "Entropy, Order Parameters, and Complexity", ex. XXX. Exercise developed in collaboration with Jaron Kent-Dobias.)

[ ]:
```
# Sometimes gives interactive new windows
# Must show() after plot, figure() before new plot
# %matplotlib

# Adds static figures to notebook: good for printing
%matplotlib inline

# Interactive windows inside notebook! Must include plt.figure() between plots
# %matplotlib notebook

# Better than from numpy import *, but need np.sin(), np.array(), plt.plot(),␣
  ↪etc.
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
from scipy.linalg import eigh, logm
from scipy.optimize import curve_fit
```

Here we introduce the quantum Heisenberg antiferromagnet, and use it to explore how entropy, temperature, and equilibration can emerge through the entanglement of two portions of a large system – closely related to the eigenstate thermalization hypothesis. We saw in Entanglement of two spins that ignoring part of a system can take a quantum pure state into a mixture of states on the remaining subsystem; this should remind you of our derivation of the canonical ensemble from a microcanonical system divided into subsystem and bath (Section 6.1, Fig. 6.1). This analogy becomes much more powerful with a larger system, a one-dimensional chain of spin 1/2 particles.

The one-dimensional Heisenberg antiferromagnet has Hamiltonian

$$\mathcal{H}_{N_{\text{spins}}} = \sum_{m=1}^{N_{\text{spins}}-1} \mathbf{S}_m \cdot \mathbf{S}_{m+1}, \tag{1}$$

where we have set the strength of the coupling J to 1 – positive, and hence favoring antiparallel spins. Here the quantum spins $\mathbf{S} = (\sigma_X, \sigma_Y, \sigma_Z)$ have spin 1/2, and are written in terms of the

Pauli matrices

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \qquad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2}$$

Let us begin with an analytical calculation of the Hamiltonian and the eigenstates for $N_\text{spins} = 2$, considered already in Entanglement of two spins. We work in the four-dimensional σ_z basis

$$\begin{pmatrix} |\uparrow_1\rangle\langle\uparrow_2| \\ |\uparrow_1\rangle\langle\downarrow_2| \\ |\downarrow_1\rangle\langle\uparrow_2| \\ |\downarrow_1\rangle\langle\downarrow_2| \end{pmatrix}. \tag{3}$$

(a) Show analytically that

$$\mathcal{H}_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 2 & 0 \\ 0 & 2 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{4}$$

Find the eigenvalues and eigenstates for this Hamiltonian. Is the ground state the triplet or the singlet? Does this make sense for an antiferromagnet? (Hint: The spin $\mathbf{S}_1$ commutes with the kets $|\uparrow_2\rangle$ and $|\downarrow_2\rangle$ and vice-versa.)

Your answer here. LaTeX works ($e = mc^2$).

Implementing this calculation elegantly on the computer demands that we understand how the single-spin σ operators and the dot product $\mathbf{S}_m \cdot \mathbf{S}_{m+1}$ act on the entire 2^{N_spins}-dimensional Hilbert space. The fact that they commute with the parts of the wavefunction involving other spins says that they act as identity matrices on those parts of the Hilbert space. That is, $\sigma_x[1]$ for the first spin needs to be promoted to $\sigma_x[1] \otimes I_{2^{N_\text{spins}-1}}$, and $\sigma_x[2]$ for the second needs to be turned into $I_2 \otimes \sigma_x[1] \otimes I_{2^{N_\text{spins}-2}}$, …

(b) Implement this numerically for the two-spin system. Calculate the Heisenberg Hamiltonian, and verify the answer of part (a). (Hint: Many modern programming languages have support for tensor data structures. These efficient routines will be important in later steps, so use them here.)

[ ]:
```
# Pauli matrices
sigmaX = ...
sigmaY = np.array([[0,-1j],[1j,0]])
sigmaZ = ...
```

One subtle point. In combining an operator L_{ij} acting on subsystem A with $M_{\alpha\beta}$ acting on subsystem B, we want an operator O which labels rows using $i\alpha$ and columns with $j\beta$ $O_{i\alpha\, j\beta}$. We can then use O as a two-index matrix to compute eigenvectors and eigenvalues. In some implementations, this demands that we swap the two inner axes of the naive product $L_{ij}M_{\alpha\beta}$.

[ ]:
```
def TensorProduct(L,M):
    """Gives tensor product of two matrices L_ij M_ab, as matrix O_(ia)(jb).
       Needs to swap axes."""
    totalSize = len(L)*len(M)
    return np.einsum("ij,ab->iajb",L,M).reshape(totalSize,totalSize)
```

```
 
```

```
[ ]: def sigmaOfn(sigma,n):
         """Tensor product Pauli matrix with identity matrix
            to act on spin n=1,2 of two spin Hilbert space"""
         if n==0:
             sigma = TensorProduct(sigma,np.identity(2))
         elif n==1:
             sigma = TensorProduct(np.identity(2),sigma)
         return sigma

     h2 = np.dot(sigmaOfn(sigmaX,0),...) + np.dot(...) + ...

     print("h2 =\n", h2, "\n")

     Heisenberg2 = np.array([[1,0,0,0],[0,-1,2,0],[0,2,-1,0],[0,0,0,1]])

     print("Heisenberg2 =\n", Heisenberg2)
```

In this exercise, we shall discuss how pure energy eigenstates states in a system AB become mixed states when we split the system into a subsystem A and a bath B, and study the properties of these mixed states. We shall index operators acting on the subsystem A with Latin letters i, j, operators on the bath B with Greek letters α, β, and operators on the total system AB with capital letters I, J, or sometimes with pairs of indices $i\alpha$, $j\beta$.

(c) If $\rho_{i\alpha,j\beta}$ is the density matrix for the whole system AB, show analytically that the sum $\sum_\alpha \rho_{i\alpha j\alpha}$ gives the reduced density matrix for the subsystem (e.g., as defined in 'Entanglement of two spins').

Your answer here.

We can use the two-spin problem of part (a) to preview the rest of the exercise, in a context where you know the answer from Entanglement of two spins. Here we view the first spin as the the 'subsystem' A, and the second spin as the 'bath' B.

(d) Select the singlet eigenstate, and normalize it if necessary. Generate the pure-state density matrix, and reshape it into the four index tensor $\rho_{i\alpha,j\beta}$. Trace over the bath as in part (c), and verify that the reduced density matrix ρ^A_{ij} describes an unpolarized spin. Calculate the entropy by taking the suitable matrix trace.

In python, the eigenvalue routine returns the eigenvectors as the columns of the array. We want the rows, so we must take the transpose.

```
[ ]: vals, vecsTranspose = eigh(Heisenberg2)
     vecs = np.transpose(vecsTranspose)
     print(vals)
     print(vecs)
```

```
[ ]: singlet = vecs[...]
     singlet
```

[ ]:
```
print("Singlet state")
rhoPure = np.einsum("i,j->ij",singlet, np.conj(singlet))
print("rhoPure =\n",rhoPure)
rhoDoubleIndex = rhoPure.reshape(2,2,2,2)
print("\n rhoDoubleIndex =\n", rhoDoubleIndex)
rhoA = np.einsum('iaja',rhoDoubleIndex)
print("\n rhoA =\n", rhoA)
SA = -np.trace(np.dot(rhoA,logm(rhoA)))
print("\n SA =", SA, "=?",np.log(2.))
```

To generate the Heisenberg Hamiltonian for multiple spins, we can save steps by noting that we already know the Hamiltonian for two spins from part (a). So the term $\mathbf{S}_m \cdot \mathbf{S}_{m+1}$ in our Heisenberg Hamiltonian becomes

$$I_{2^{m-1}} \otimes \mathcal{H}_2 \otimes I_{2^{N_\text{spins}-(m+1)}} \tag{5}$$

(e) Use this to write a function that returns the Heisenberg Hamiltonian $\mathcal{H}_{N_\text{spins}}$ as a $2^{N_\text{spins}} \times 2^{N_\text{spins}}$ matrix. Check, for $N_\text{spins} = 2$ it returns $\mathcal{H}_2$ from part (a). Check also for $N_\text{spins} = 3$ its eigenvalues are $(-4, -4, 2, 2, 2, 2, 0, 0)$, and for $N_\text{spins} = 4$ that its distinct eigenvalues are $\{-3-2\sqrt{3}, -1-2\sqrt{2}, 3, -1+2\sqrt{2}, -1, -3+2\sqrt{3}\} \approx \{-6.46, -3.8, 3, 1.8, -1, 0.46\}$.

In the C and Python convention where indices start with zero, we should use $I_{2^m} \otimes \mathcal{H}_2 \otimes I_{2^{N_\text{spins}-(m+2)}}$.

[ ]:
```
def Heisenberg(nS):
    Ham = TensorProduct(Heisenberg2,np.identity(2**(nS-2)))
    for m in range(1,nS-1):
        Ham = Ham + TensorProduct(TensorProduct(np.identity(...,Heisenberg2),
                                                np.identity(2**(...)))
    return Ham
print(Heisenberg(2))
```

[ ]:
```
np.set_printoptions(precision=3,suppress=True)
print(eigh(Heisenberg(3))[0])
print(eigh(Heisenberg(4))[0])
```

We shall work with a system of $N_\text{spins} = N_{AB} = 10$ spins in the chain; we shall primarily study a subsystem with $N_A = 4$ spins, so the bath has $N_B = N_{AB} - N_A = 6$ spins. We shall use an eigenstate ψ of $\mathcal{H}_{N_{AB}}$ to calculate the reduced density matrix ρ_A for N_A, to investigate the entanglement between A and the bath B, to calculate the entanglement entropy, and to illustrate eigenstate thermalization. For the latter, we want an energy that is lower than average, but not near zero.

(f) Create $\mathcal{H}_{AB} = \mathcal{H}_{10}$. Find its energy eigenvalues and eigenstates, and (if necessary) sort them in increasing order of their energy. Pick the energy eigenstate ψ of the full system that is 1/4 the way from the bottom (the $K = 2^{N_{AB}-3}$ entry). Calculate the pure density matrix ρ^pure, reshape it into the four index tensor $\rho^\text{AB}_{i\alpha,j\beta}$, and trace over the bath to give the reduced density matrix ρ^A_{ij}. Check that ρ^A has trace one (as it must), and calculate $Tr[(\rho^A)^2]$. Is it is a mixed state?

In python, the eigenvalue routine returns the eigenvectors as the columns of the array. We want the rows, so we must take the transpose.

```
[ ]: nAB = 10;
     HamAB = Heisenberg(nAB);
     EABs, eigvecsTranspose = eigh(HamAB)
     psiABs = eigvecsTranspose.transpose()
     K=...
     psiK = psiABs[K]
     EK = EABs[...]

     nA = 4
     rhoPure = np.einsum("i,j->ij",...,np.conj(psiK))
     rhoAB = rhoPure.reshape(2**nA,2**(nAB-nA),2**nA,2**(nAB-nA))
     rhoA = np.einsum(...,rhoAB)

     print("Trace rhoA", np.trace(rhoA))
     print("Trace rhoA^2", np.trace(np.dot(rhoA,rhoA)))
```

The entanglement entropy between A and B for a pure state ψ of AB is the entropy of the reduced density matrix of A.

(g) Calculate the entanglement entropy $S = -Tr\rho_A \log \rho_A$. Check that it has the same entropy as subsystem B. Write a loop over N_A ranging through all values from zero to N_{AB}, and plot S as a function of N_A for our particular eigenstate ψ. Where is the entanglement entropy largest? Explain why it goes to zero for the two endpoints.

```
[ ]: SA = ...
     print("SA = ", SA)
     rhoB = np.einsum(...,rhoAB)
     SB = ...
     print("SB = ", SB)
```

```
[ ]: rhos = [rhoPure.reshape(2**nA,2**(nAB-nA),2**nA,2**(nAB-nA)) for nA in␣
      ↪range(0,nAB+1)];
     rhoAs = [np.einsum('iaja', rho) for rho in rhos]
     SAs = [... for rhoA in rhoAs]
     plt.plot(np.arange(0,nAB+1),np.real(SAs),"bo-")
```

The term ‘entanglement’ is mutual; A and B are entangled, rather than B has somehow perturbed A. This is not an accident. As you checked numerically, the entanglement entropies of the two subsystems should be the same. (This can be shown using the Schmidt decomposition – an application of singular value decomposition to density matrices in quantum mechanics).

In statistical mechanics, a large system AB in the microcanonical ensemble at energy E will, when restricted to a relatively small subsystem A, generate an equilibrium thermal ensemble at the corresponding temperature. The eigenstate thermalization hypothesis argues that many quantum systems this to an extreme: for any eigenstate ψ with energy E, the reduced density matrix ρ_A of the subsystem will converge to a Boltzmann equilibrium thermal ensemble

$$\rho_{jk} = \delta_{jk} e^{-\beta E_k^A} / \sum_\ell e^{-\beta E_\ell^A} \tag{6}$$

as the system size goes to infinity.

Let us calculate the probability p_k that our subsystem is in eigenstate ψ_k^A, $p_k = Tr(|\psi_k^A\rangle\langle\psi_k^A|\,\rho_A)$. We are simulating a rather small system, so fluctuations will be large.

(h) Make a log plot of p_k vs. E_k^A. Do a nonlinear fit to the predicted form above to find β, and plot the result with your data.

```
[ ]: HamA=Heisenberg(nA)
     EAs, psiAsTranspose =eigh(HamA)
     psiAs = np.transpose(psiAsTranspose)
     psiKetBras = [np.einsum("i,j->ij",...,np.conj(psiA)) for psiA in psiAs]
     ps = [np.trace(np.dot(psiKetBra,...)) for psiKetBra in psiKetBras]
     fig = plt.plot(EAs,ps,'bo')
     plt.yscale('log')
     def BoltzmannProb(E,A,beta):
         return A*np.exp(-beta*E)
     A0, beta0 = curve_fit(BoltzmannProb,EAs,ps)[0]
     plt.plot(EAs, BoltzmannProb(EAs, A0, beta0))
     print(beta0)
```

In particular, the reduced density matrix is predicted to be at the temperature of the microcanonical ensemble at the energy E of the original pure state ψ.

(i) Write a function $EbarAB(\beta)$ returning the average energy of the entire system as a function of β. Take a sampling of eigenstates ψ_K of the total system, fit p_k vs E_k^A as in part(h), and plot β vs. E along with your prediction $\beta(EbarAB)$. Do you achieve a rough agreement?

We suggest starting with only a few points, spread over the interval. For example, points starting at $K = 32$ and separated by $\Delta K = 64$ will span the range avoiding the endpoints.

```
[ ]: def beta(psi):
         """
         Returns best fit beta for reduced density matrix
         corresponding to the Kth eigenstate of the system
         """
         rho = np.einsum("i,j->ij",psi,np.conj(psi)).
      ↪reshape(2**nA,2**(nAB-nA),2**nA,2**(nAB-nA))
         rhoA = np.einsum('iaja',rho)
         ps = [np.trace(np.dot(psiKetBra,rhoA)) for psiKetBra in psiKetBras]
         A0, beta0 = curve_fit(BoltzmannProb,EAs,ps, maxfev=50000)[0]
         return beta0

     Kmin = 2**(nAB-5)
     Ks = np.arange(Kmin,2**nAB,2*Kmin)
     betas = [beta(psiABs[K]) for K in Ks]
```

```
Es = EABs[Ks]
```

[ ]:
```
plt.plot(Es,betas,"bo")
plt.ylim(-2,2)
```

[ ]:
```
def EbarAB(beta):
    Z = sum(np.exp(-beta*EABs))
    return sum(... * ...)/Z

plt.plot(Es, betas, "bo")
betasTheory = np.arange(-2,1.5,0.01)
EbarsTheory = [EbarAB(beta) for beta in betasTheory]
plt.plot(EbarsTheory,betasTheory,"r")
plt.ylim(-2,2)
```