metropolis

February 2, 2018

1 Metropolis Solutions.

Homework assignment for implementing Ising Metropolis Algorithm

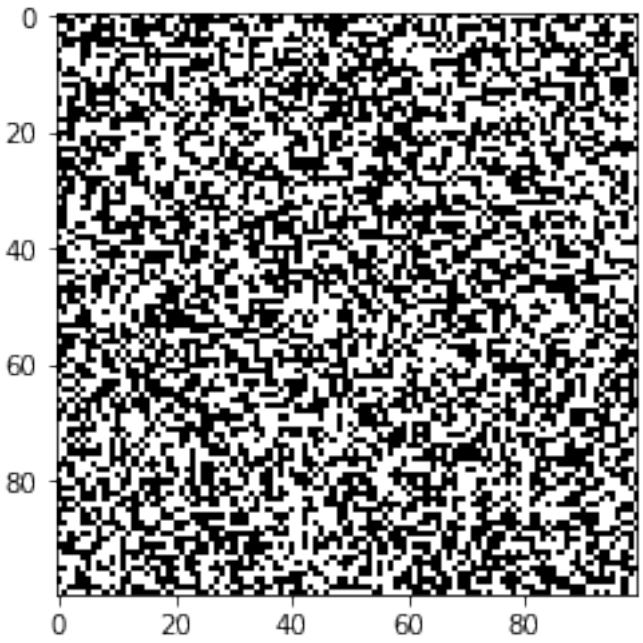
```
In [1]: import numpy as np
        from scipy import stats
        from matplotlib import pyplot as plt
```

1.1 Problem 1

Write a function that accepts an integer n and returns a random spin configuration for an $n \times n$ lattice (as an $n \times n$ NumPy array of 1s and -1s). Test your function with $n = 100$, plotting the spin configuration via `plt.imshow()`.

```
In [2]: def random_lattice(n):
            """Construct a random spin configuration for an nxn lattice."""
            state=2*np.random.randint(2,size=(n,n))-1
            return state
```

```
In [3]: fxn=random_lattice(100)
        plt.imshow(fxn,cmap='gray')
        plt.show()
```



1.2 Problem 2

Write a function that accepts a spin configuration σ for a lattice as a NumPy array. Compute the energy $H(\sigma)$ of the spin configuration. Be careful to not double count site pair interactions!

```
In [4]: def lattice_energy(L):
            """Computer the energy of the spin configuration corresponding to the
            lattice L.
            """
            energy=0
            n=len(L)
            for i in range(n):
                for j in range(n):
                    energy-=L[i,j]*(L[(i+1)%n,j]+L[(i-1)%n,j]+L[i,(j+1)%n]+L[i,(j-1)%n])
            return energy/4#to eliminate double counts
```

1.3 Problem 3

Write a function that accepts an integer n and chooses a pair of indices (i, j) where $0 \leq i, j \leq n-1$. Each possible pair should have an equal probability $\frac{1}{n^2}$ of being chosen.

```
In [5]: def flip_location(n):
            """Choose a random pair of indices 0 <= i,j <= n-1."""
            i=np.random.randint(0,high=n)
            j=np.random.randint(0,high=n)
            return i, j
```

1.4 Problem 4

Write a function that accepts a spin configuration σ, its energy $H(\sigma)$, and integer indices i and j. Compute the energy of the new spin configuration σ^*, which is σ but with the spin flipped at the (i,j)th entry of the corresponding lattice. Do not explicitly construct the new lattice for σ^*.

```
In [6]: def updated_energy(L, L_energy, i, j):
            """Compute the energy of the spin configuration that results
            when the (i,j)th spin of L is flipped.
            """
            n=len(L)
            flip=L[i,j]*(L[(i+1)%n,j]+L[(i-1)%n,j]+L[i,(j+1)%n]+L[i,(j-1)%n])
            return L_energy+2*flip
```

1.5 Problem 5

Write a function that accepts a float β and spin configuration energies $H(\sigma)$ and $H(\sigma^*)$. Calculate whether or not the new spin configuration σ^* should be accepted (return `True` or `False`).

```
In [7]: def accept(beta, energy_old, energy_new):
            """Accept or reject the new spin configuration."""
            if energy_new<energy_old:
                return True
            elif np.random.random()<np.e**(beta*(energy_old-energy_new)):
                return True
            return False
```

1.6 Problem 6

Write a function that accepts a float $\beta > 0$ and integers n, `n_samples`, and `burn_in`. Initialize an $n \times n$ lattice for a spin configuration σ using `random_lattice()`. Use the Metropolis algorithm to (potentially) update the lattice `burn_in` times. 1. Use `flip_location()` to choose a site for possibly flipping the spin, thus defining a potential new configuration σ^*. 2. Use `updated_energy()` to calculate the energy $H(\sigma^*)$ of the proposed configuration. 3. Use `accept()` to accept or reject the proposed configuration. If it is accepted, set $\sigma = \sigma^*$ by flipping the spin at the indicated site. 4. Track $-\beta H(\sigma)$ at each iteration (independent of acceptance).

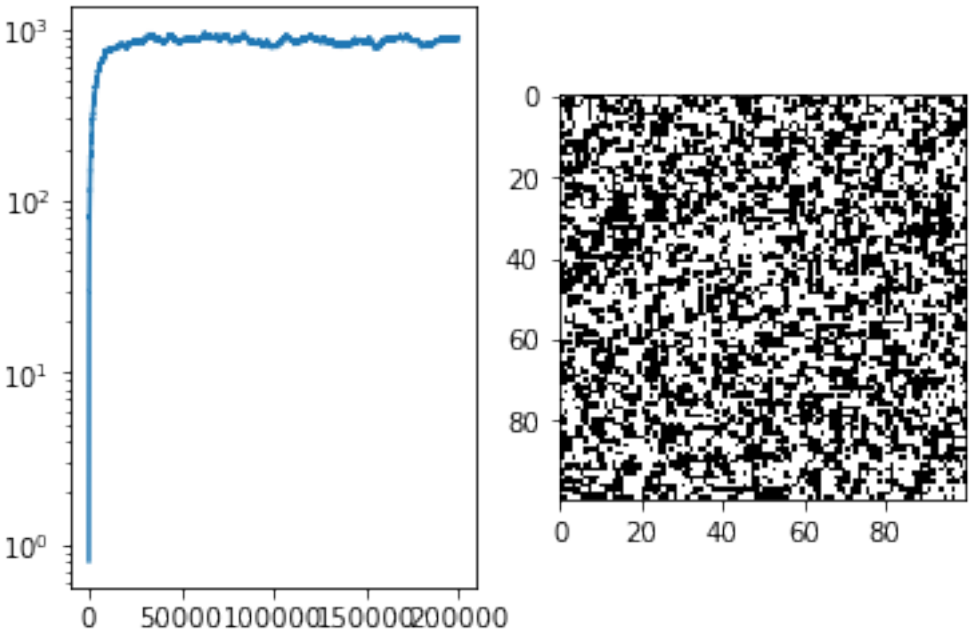
After the burn-in period, continue the iteration `n_samples` times, also recording every 100th sample (to prevent memory failure). Return the samples, the sequence of weighted energies $-\beta H(\sigma)$, and the acceptance rate.

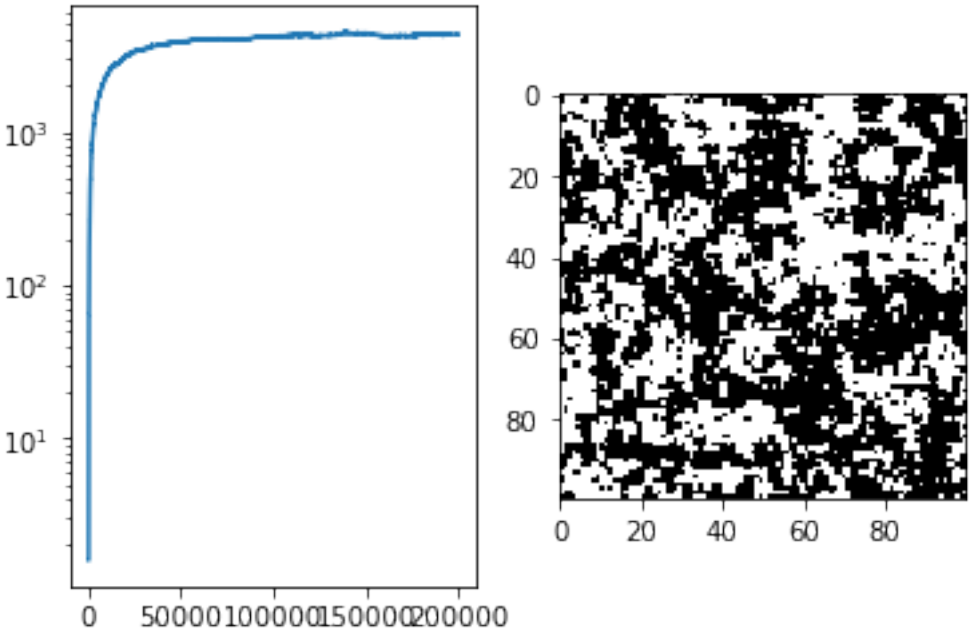
Test your sampler on a 100×100 grid with 200000 total iterations, with `n_samples` large enough so that you will keep 50 samples, for $\beta = 0.2, 0.4, 1$. Plot the proportional log probabilities, as well as a late sample from each test.

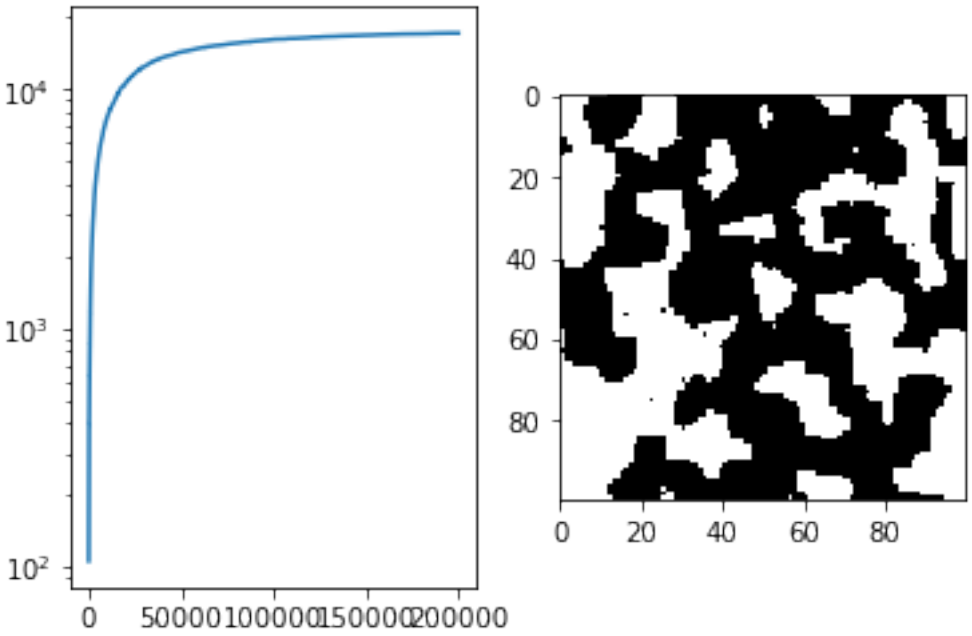
```
In [10]: def ising_metropolis(beta, n=100, n_samples=5000, burn_in=195000):
             """Use the Metropolis algorithm to choose new spin configurations.

             Parameters:
                 beta (float > 0): Constant inversely proportional to the temperature.
                 N (int > 0): The size of the lattice.
```

```
            n_samples (int): The number of samples to generate.
            burnin (int): The number of iterations to burn before sampling.

        Returns:
            ((n_samples//100, n,n) ndarray): The sampled spin configurations.
            (n_samples) ndarray: The weighted energies of each configuration.
            (float): The proportion of proposed samples that were accepted.
        """
        #Initialize lattice
        L=random_lattice(n)
        #list to sample lattices
        sample=[]
        #list to track beta energy
        beta_energy=[]
        accepted=0
        L_energy=lattice_energy(L)
        #burn in
        for _ in range(burn_in):
            i,j=flip_location(n)
            L_star_energy=updated_energy(L, L_energy, i, j)
            acc=accept(beta, L_energy, L_star_energy)
            if acc==True:
                L[i,j]=-1*L[i,j]
                L_energy=L_star_energy
                accepted+=1
            beta_energy.append(-beta*L_energy)
        #actual stuff
        for i in range(n_samples):
            i,j=flip_location(n)
            L_star_energy=updated_energy(L, L_energy, i, j)
            acc=accept(beta, L_energy, L_star_energy)
            if acc==True:
                L[i,j]=-1*L[i,j]
                L_energy=L_star_energy
                accepted+=1
            beta_energy.append(-beta*L_energy)
            if i%100==0:
                sample.append(L)
        return L, beta_energy, accepted/(n_samples+burn_in)
```

In [12]:
```
for b in [.2,.4,1]:
    L, beta, acc = ising_metropolis(b)
    #print(acc)
    plt.subplot(121)
    plt.semilogy(beta)
    plt.subplot(122)
    plt.imshow(L,cmap="gray")
    plt.show()
```


10^3
10^2
10^1
10^0
0
50000
100000
150000
200000
0
20
40
60
80


10^3
10^2
10^1
0
50000
100000
150000
200000
0
20
40
60
80


10^4
10^3
10^2
0
50000
100000
150000
200000
0
20
40
60
80
0
20
40
60
80

In [ ]: