Monte Carlo Simulations

A Max Franklin Production

Metropolis MC Simulation

- Generates N configurations of a system, C1,C2...CN so that
  - lim(N-> infinity) NC/N = P(C), a probability distribution. NC is the number of configurations
- This is useful because it allows us to obtain many random samples from a distribution that is difficult to sample directly.
- This allows us to approximate the distribution or find an expected value.

- The Metropolis algorithm is most useful because it generates microstates with highest probabilities more.

Steps to the Simulation

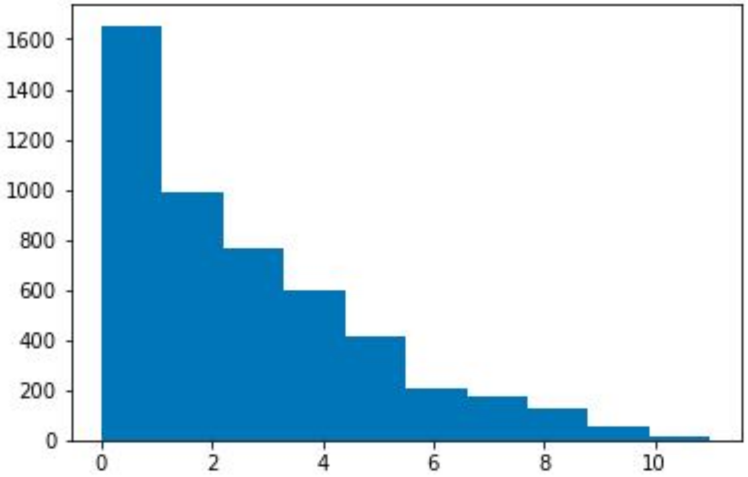
1) Start with the simulation in some state. In our example, the energy of each particle is 0.
2) Make a change in the microstate by choosing a particle at random and changing its energy by ±1 in an Einstein solid.
3) Compute the change in energy of the system, ΔE.
4) If $\Delta E<0$, the change goes through. If $\Delta E>0$, accept change with probability w=e^(-BΔE)
   a) Generate a random number r in the unit interval. If $r \leq w$, accept the change. Otherwise, reject the change.
5) Repeat many times, compute averages once the system has reached equilibrium

References

“Jason Blevins.” *The Metropolis-Hastings Algorithm*, jblevins.org/notes/metropolis-hastings.

Metropolis Monte Carlo Method, xbeams.chem.yale.edu/~batista/vaa/node42.html.

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import random as ran
def MonteCarlo(N, Steps, B):
    T = []                          # creates a blank list of total energy
    energies = np.zeros(N)          #creates array of N particles, each with zero initial energy
    for i in range(0,Steps):        #repeats process over given step number
        r = ran.randint(0,N-1)      #this random number is to pick a random entry in the particle array
        w = ran.uniform(0,1)        #random number between 0 and 1
        e = np.exp(-B)              # this is e^(-beta*ΔE), which we compare against the random number w
        x = ran.uniform(0,1)        #this random number determines if +1 or -1 is done to the energy
        if x>0.5:
            if w < e:               #if x>.5 and e>w, the change is accepted and we add one to the energy
                energies[r] = energies[r]+1
            else:                   #if e<W, the change is rejected
                energies[r] = energies[r]
        if x<0.5:                   #if x<.5, we subtract one from the energy
            if energies[r]>0:       #this change is accepted if the energy would not become negative (energy>0)
                energies[r]=energies[r]-1
            else:                   #if the particle energy is 0, the change is rejected
                energies[r]=energies[r]
        T.append(np.sum(energies))  #we add the sum of particle energyes to the total energy list
    plt.hist(T[:5000])              #eventually, we can plot the total energy at high steps. It should be exponential
    plt.show()
```



1600
1400
1200
1000
800
600
400
200
0
0
2
4
6
8
10