Informal Lab Report 9

Warm-Up: Monte Carlo Integration

Slide Type

Exercise 1

Slide Type Sub-Slide

Write a program to evaluate

$$I = \int_0^2 \sin^2\left[\frac{1}{(2-x)x}\right] dx$$

using the "hit-or-miss" method with $N = 10^4$ points.

Also evaluate the error on your method.

Slide Type

```
import numpy as np

def f(x):
    return np.sin(1/((2-x)*x))**2
```

Slide Type -

```
N = 10000 # Number of samples
k = 0
a = 0.
b = 2.

for i in range(N):
    sample_x = a + (b-a)*np.random.random() # x coord of random sample
                    # x-coord needs to be limited to the interval [a, b]
    sample_y = np.random.random() # y coord of random sample
    if sample_y <= f(sample_x):
        k += 1

# compute ratio of points below
A = (b-a)*1.
result = A*k/N # resulting integral
print("Integral = {0:.6e}".format(result))

# compute error
error = np.sqrt(result*(A-result)/N)
print('Error for hit-or-miss method = {0:.6e}'.format(error))
```

```
Integral = 1.440800e+00
Error for hit-or-miss method = 8.976053e-03
```

Slide Type

Exercise 2

Slide Type Sub-Slide

Write a program to evaluate

$$I = \int_0^2 \sin^2\left[\frac{1}{(2-x)x}\right] dx$$

using the mean value method with $N = 10^4$ points.

Also evaluate the error on your method.

Slide Type

```
k = 0
k2 = 0

for i in range(N):
    x = (b-a)*np.random.random() # random x samples
    k += f(x) # f(x) for the random x samples
    k2 += f(x)**2

I = k * (b-a) / N
print(I)

var = k2/N - (k/N)**2
error_meanval = (b-a)*np.sqrt(var/N)
print('error for mean value method = ', error_meanval)
print('In comparison, error for hit-or-miss method = ', error_meanval)
```

```
1.4535737179918171
error =  0.00528115381650881
recall error in hit-or-miss =  0.00528115381650881
```

Monte Carlo for Statistical Mechanics

Slide Type

StatMech review

Slide Type Sub-Slide

If I was your prof for PHY252, this seems familiar... right?

- For a system in equilibrium at temperature T, the probability of finding the system in any particular state i is given by the Boltzmann distribution,
$$P(E_i) = \frac{\exp[-E_i/(k_B T)]}{Z}, \quad Z = \sum_{i=1}^{ALL} \exp[-E_i/(k_B T)]$$
where E_i is the energy of state i, k_B is Boltzmann's constant, Z is partition function
- System at temperature T undergoes transitions between states, with probability of being in a particular state $P(E_i)$

Slide Type Sub-Slide

- To calculate expectation value of a macroscopic property X (e.g. total energy, magnetization, spin): average over the many microstates that the system visits
$$\langle X \rangle = \sum_{i=1}^{ALL} X_i P(E_i)$$
where X_i is the value of the quantity in the i^{th} microstate and P is the probability of finding the system in that microstate.

Slide Type

Computational Issues

Slide Type

- Simple example: single mole of gas has $N_A \approx 6 \times 10^{23}$ molecules. Assume each molecule had only 2 possible spin states (gross underestimation), then the total number of spin microstates of the mole of gas is 2^{N_A}, which is huge.
- If we want to get any useful macroscopic information $\langle X \rangle$ about the system, we need to be more clever about how we count than just brute-force counting everything.

Slide Type Sub-Slide

$$\langle X \rangle = \sum_{i=1}^{ALL} X_i P(E_i), \quad P(E_i) = \frac{\exp[-E_i/(k_B T)]}{Z}, \quad Z = \sum_{i=1}^{ALL} \exp[-E_i/(k_B T)]$$

- Huge number of terms in sum ⇒ often impossible to perform analytically ⇒ use Monte Carlo summation.
- Two difficulties to overcome:
  1. estimating $\langle X \rangle$: properly sampling which terms to sum over (*solution: importance sampling*),
  2. estimating Z (*solution: Markov Chain Monte Carlo*)

Slide Type

Importance sampling for Stat. Mech.

Slide Type Slide

- Randomly sample the terms in the sum and only use those as an estimate. Replace $\langle X \rangle = \sum_{i=1}^{ALL} X_i P(E_i)$ with a sum over N randomly sampled microstates,
$$\langle X \rangle = \frac{\sum_{i=1}^{N} X_i P(E_i)}{\sum_{i=1}^{N} P(E_i)}.$$
  - Denominator ensures total probability over the sampled states is 1.
- To get a good estimate for the sum, it is only worth keeping the big terms
  - A lot of states have $E_i \gg k_B T$, therefore $P(E_i)$ really small since probability is proportional to $\exp[-E_i/(k_B T)]$
- Need to preferentially choose terms where the integrand is non-negligible, but assign them less weight individually (to not bias the final estimate) ... so use importance sampling!

Slide Type Sub-Slide

- For a dicrete sum (see p. 478),
$$\langle X \rangle = \sum_{i=1}^{N} X_i P(E_i) \approx \frac{1}{N} \sum_{k=1}^{N} \frac{X_k P(E_k)}{w_k} \sum_{i=1}^{ALL} w_i.$$
- For weight w, use $P(E_i)$
$$\langle X \rangle \approx \frac{1}{N} \sum_{k=1}^{N} \underbrace{\frac{X_k P(E_k)}{P(E_k)}}_{=X_k} \underbrace{\sum_{i=1}^{ALL} P(E_i)}_{=1}.$$

Slide Type Sub-Slide

In the end,

$$\langle X \rangle \approx \frac{1}{N} \sum_{k=1}^{N} X_k.$$

- Looks simple and no different from mean-value MC,
- but recall that the X_k 's are drawn from non-uniform distribution: we randomly choose terms in the sum based on their Boltzmann probabilities, $P(E)$. But how? Recall

$$P(E_i) = \frac{\exp[-E_i/(k_B T)]}{Z}, \quad Z = \sum_{i=1}^{ALL} \exp[-E_i/(k_B T)]$$

- To do it this way, we need Z, which is a sum over all states. But if we could do this, we wouldn't need Monte Carlo in the first place!

Slide Type

Markov chain method

Slide Type Slide

Mish-mashing https://en.wikipedia.org/wiki/Markov_chain and https://en.wikipedia.org/wiki/Markov_property,

> A Markov chain is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event. [...] (sometimes characterized as "memorylessness"). In simpler terms, it is a process for which predictions can be made regarding future outcomes based solely on its present state [...]. In other words, conditional on the present state of the system, its future and past states are independent.

- Random walks (Brownian motion) are Markov chains.
- Here: events are jumps in energy states, one after another.

Slide Type Sub-Slide

- Text goes into details on how to implement the Markov Chain method with a Metropolis algorithm.
- Crucial key: Metropolis does not directly compute probability to be in one state, but instead uses probability to transition between two states
  - Z cancels out in the process!
- In this lab, we will summarize it algorithmically first, then briefly outline why it works mathematically.

Slide Type

Algorithm

Slide Type Sub-Slide

1. Choose a random starting state i
2. Calculate the energy of that state E_i
3. Choose a transition to a new state j uniformly at random from allowed set
4. Calculate the energy of this new state, E_j
5. Calculate the acceptance probability for this transition:
   - $P_a = 1$ if $E_j \leq E_i$ (always accept a lower energy state)
   - $P_a = \exp\left(-\frac{E_j - E_i}{k_B T}\right)$ if $E_j > E_i$ (accept a higher energy state sometimes, more often for high T).
6. Accept/reject the move according to the acceptance probability
7. Measure the quantity X you want in its current state (new or old i) & store it
8. Repeat from step 2.

Slide Type Sub-Slide

Why does Metropolis create a system where each microstate has a probability $P(E_i)$, the Boltzmann distribution?

- Let τ_{ij} be transition probability from microstate i to microstate j, such that

$$\frac{\tau_{ij}}{\tau_{ji}} = \frac{P(E_j)}{P(E_i)},$$

  which the Metropolis algorithm satisfies (see p. 482). For example, a small ratio above means
  - RHS: "*j is much less probable than i*" and equivalently,
  - LHS: "*probability of $j \to i$ is much higher than $i \to j$*",

  both statements being true in equal amounts.
- "In equal amounts" is crucial: it means that if you start from any initial state, your system will progressively evolve towards one where all microstates follow Boltzmann.
- Does it always converge? Yes. If you love linear algebra, see proof in Appendix D of Newman (it is actually quite elegant).

Slide Type

Exercise 3

Slide Type Sub-Slide

Write code to implement steps 5-6 of the algorithm. It should be in the form of a function that takes energies and temperature as arguments, and returns something representing true or false (accept or reject). Hints:

- Simplify the problem by setting $k_B = 1$.
- For the probability of the event:
  - $P_a = 1$ if $E_j \leq E_i$ (always accept a lower energy state)
  - $P_a = \exp\left(-\frac{E_j - E_i}{k_B T}\right)$ if $E_j > E_i$ (sometimes accept a higher energy state, more often for higher T).
- Draw a random number in $[0, 1)$ and compare it to P_a`
  - E.g., at very high T, $\exp(\ldots) \approx 1$ and almost all moves are accepted.
  - E.g., at low T, say, $\exp[-\Delta E/(k_B T)] = 1\%$, then `random()` has 1% chance of drawing a number that is $< 1\%$.
  - If $E_j \leq E_i$, then $\exp(\ldots) \geq 1$ and your function should automatically accept.

]:

Slide Type

```
import numpy as np
from random import random, randrange


def acceptance(E_now, E_test, T):
    kb = 1.0
    deltaE = E_test-E_now
    #calculate acceptance probability
    #accept or reject new state?
    if deltaE <= 0.0:
        return 1
    beta=1.0/(kb*T)
    p = np.exp(-beta * deltaE)
    if random() <= p:
        return 1
    return 0
```

Slide Type

Ising model

Slide Type Slide

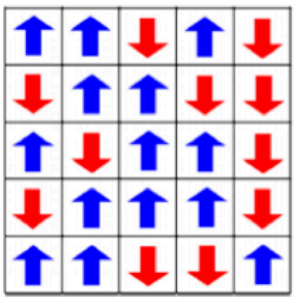
- Simple model of ferromagnetism, demonstrating many physical characteristics of fancier models.
- Assume an object is made up of a collection of dipoles (e.g., electron spins) and the net magnetization is the sum of the magnetization of all the spins.
  - the spins can only point up or down.
  - the spins interact and favour parallel alignment of pairs of spins
  - the interactions are non-zero only between nearest neighbours (i.e., distance dependent).
- Macroscopic energy E and magnetization M (with no external field) are given by

$$E = -J \sum_{\langle ij \rangle} s_i s_j \quad \text{and} \quad M = \sum_i s_i$$

where $s = +1$ if spin is up & $s = -1$ if spin is down.
  - Lowest energy occurs if the spins all line up.

Slide Type Sub-Slide

- Spins can randomly flip as the system visits a set of allowable states given its temperature. At any particular moment the system may look like:



Exercise 4

Slide Type Sub-Slide

For the 1-dimensional Ising model, with N dipoles, write code to: create an array representing the dipole spins, and initialize the state of the system such that there's random spin at each location. (The code should take N as an argument, and return the initialized array).

Then, write a function to calculate the energy and magnetization (these should be the return values) for a given array of dipoles (this should be the argument). Use $J = 1$ milliJoule.

In [71]:

Slide Type

```
def initSpins(N):
    state = np.zeros(N, int)
    for i in range(N):
        if random() < 0.5:
            state[i] = 1
        else:
            state[i] = -1
    return state

def energyfunction(dipoles):
    J= 1.0
    energy = -1.0*J*np.sum(dipoles[0:-1]*dipoles[1:])
    return energy

def magnetization(dipoles):
    return sum(dipoles)
```

Slide Type

Exercise 5

Slide Type

Implement the Metropolis algorithm to model the behaviour of the array of dipoles.

- create new state: flip 1 spin randomly
- calculate new total energy
- calculate acceptance probability
- decide whether to accept or reject new state
- store 'new' energy & magnetization
- repeat M times

Your function should keep track of energy and magnetization throughout. Return an array representing the energy at each flip attempt, and another array representing the magnetization at each flip attempt.

Use your code from the previous 2 exercises to help. You should have T and M as parameters that can be easily set.

In [69]:

Slide Type

```
from random import random, randrange

def metropolis_dipoles(state, T, M):
    kb = 1
    beta = 1.0/(kb*T)
    Elist = []
    Mlist = []
    E = energyfunction(state)
    Mag = magnetization(state)
    Elist.append(E)
    Mlist.append(Mag)

    for i in range(M):
        test=np.copy(state)
        picked = randrange(0, len(state))
        test[picked] = -test[picked]

        #new energy
        E_test = energyfunction(test)

        if acceptance(E,E_test, T) == 1:
            state = test
            E = E_test
            Mag = magnetization(test)

        Elist.append(E)
        Mlist.append(Mag)
    return Elist, Mlist
```

Exercise 6

Slide Type

Using your code from the previous exercise, plot energy and magnetization as a function of time (assuming 1 flipping attempt per millisecond) for $N = 100$, $M = 100000$, for each of 3 different temperatures: $T = 0.1, 1.0, 10.0$

72]:

Slide Type

```
import matplotlib.pyplot as plt

# define constants
T = [0.1, 1, 10]
num_dipoles = 100
num_flips = 100000 # number of flips
# generate array of dipoles and initialize diagnostic quantities

dipoles = initSpins(num_dipoles)
for t in T:
    energy, magnet = metropolis_dipoles(dipoles, t, num_flips)

    # plot energy, magnetisation
    fg, ax = plt.subplots(2, 1, sharex=True)
    ax[0].plot(magnet)
    ax[0].set_ylabel('Magnetization')
    ax[0].grid()
    ax[1].plot(energy)
    ax[1].set_xlabel('Times in ms')
    ax[1].set_ylabel('Energy in mJ')
    ax[1].grid()

    plt.tight_layout()
    plt.show()
```