Kyle Briggs

12/03/2018

Intro to Monte Carlo Simulations: 2D Ising Ferromagnets

Administrative Stuff

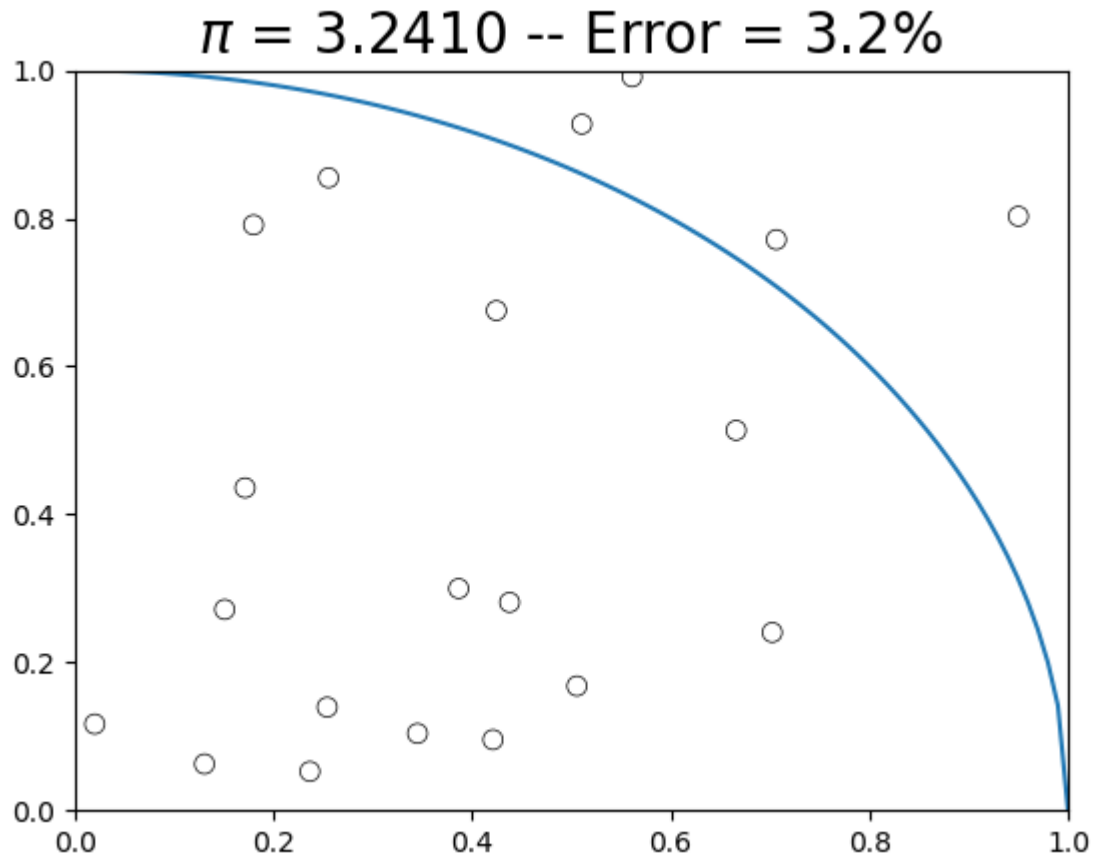
- Problem Set #3 has been extended to Tuesday, March 20

Monte Carlo

- General class of algorithms which use random numbers to sample a system of interest.
  - Statistical physics and thermodynamics
  - Stock market dynamics
  - Climate change
  - Artificial intelligence
  - Search and Rescue
  - … and many more!

- Can even be used to obtain statistical estimates of deterministic systems.

Monte Carlo Example – Numerical Integration



The fraction of the total area under the curve is equal to the fraction of randomly generated points which fall below it. For example:

$$y = \sqrt{1 - x^2}$$

Since

$$\int_0^1 \sqrt{1 - x^2}\,dx = \frac{\pi}{4}$$

$$\pi = 4\frac{N_{in}}{N}$$

Why use Monte Carlo?

- Consider a numerical integral in d dimensions. We could integrate this numerically with, say, Simpson's rule with N points per dimension, or we could do it with Monte Carlo using N random numbers:

	Simpson's Rule	Monte Carlo
Error Scaling	$N^{-\frac{2}{d}}$	$N^{-\frac{1}{2}}$
Time Complexity	N^d	N

- Now consider trying to evaluate a partition function for a mole of gas particles where $d \approx 10^{23}$. Even if the integrand is known analytically, Monte Carlo wins both in speed and accuracy, and it's not close.

Why use Monte Carlo?

- Many physical problems have stochasticity built into the dynamics. (i.e. anything with thermal noise). Such problems are very difficult or impossible to solve analytically
  - Sometimes you can get analytical expressions for average steady state solutions (i.e. thermodynamics).

- Generally, analyzing a system with randomness built into the dynamics requires a numerical algorithm which samples that randomness.

- Systems with many coupled degrees of freedom can behave chaotically, which resembles randomness in many ways and is difficult to handle deterministically.

The Metropolis Algorithm

- Given a probability distribution $P(\vec{x}_t)$, we wish to generate microstates $\vec{x}_t$ which properly sample P.
  - Starting from state $\vec{x}_n$:
  - **Uniformly** randomly generate a small trial step $\vec{\delta}$ and calculate the relative probability $R = \frac{P\left(\vec{x}+\vec{\delta}\right)}{P(\vec{x})}$
    - If $R \geq 1$, set $\vec{x}_{n+1} = \vec{x}_n + \vec{\delta}$
    - If $R < 1$, choose a random number $r \sim U(0,1)$
      - If $r \leq R$, set $\vec{x}_{n+1} = \vec{x}_n + \vec{\delta}$
      - If $r > R$, set $\vec{x}_{n+1} = \vec{x}_n$

Detailed Balance

- In equilibrium, we require:
  - net flux between every pair of possible microstates is 0 (dynamic local equilibrium).
  - The occupation number of each microstate is proportional to its probability: $N_n(\vec{x}) \propto P(\vec{x})$
- Let $N_n(\vec{x})$ denote the occupation number of state $\vec{x}$ at time n, and let $p(\vec{x} \to \vec{y})$ be the probability of a transition from state $\vec{x}$ to state $\vec{y}$.

$$0 = \Delta N_n(\vec{x} \leftrightarrow \vec{y}) = N_n(\vec{x})p(\vec{x} \to \vec{y}) - N_n(\vec{y})p(\vec{y} \to \vec{x})$$

$$\frac{N_n(\vec{x})}{N_n(\vec{y})} = \frac{p(\vec{y} \to \vec{x})}{p(\vec{x} \to \vec{y})}$$

Detailed Balance

- Transition probabilities are the product of the probability $T(\vec{x} \to \vec{y})$ of choosing a trial step which moves between those states, and the probability $A(\vec{x} \to \vec{y})$ of actually accepting the trial step:

$$p(\vec{x} \to \vec{y}) = T(\vec{x} \to \vec{y})A(\vec{x} \to \vec{y})$$

- T is sampled from a uniform distribution, so it goes away:

$$\frac{N_n(\vec{x})}{N_n(\vec{y})} = \frac{p(\vec{y} \to \vec{x})}{p(\vec{x} \to \vec{y})} = \frac{A(\vec{y} \to \vec{x})}{A(\vec{x} \to \vec{y})}$$

Detailed Balance

- From the definition of our algorithm, we have two cases:
  - Case 1: $P(\vec{x}) > P(\vec{y})$
    - $A(\vec{y} \to \vec{x}) = 1$
    - $A(\vec{x} \to \vec{y}) = \frac{P(\vec{y})}{P(\vec{x})}$
  - Case 2: $P(\vec{x}) < P(\vec{y})$
    - $A(\vec{y} \to \vec{x}) = \frac{P(\vec{x})}{P(\vec{y})}$
    - $A(\vec{x} \to \vec{y}) = 1$
- In both cases:

$$\frac{N_n(\vec{x})}{N_n(\vec{y})} = \frac{P(\vec{x})}{P(\vec{y})}$$

Example – Gaussian Random Numbers

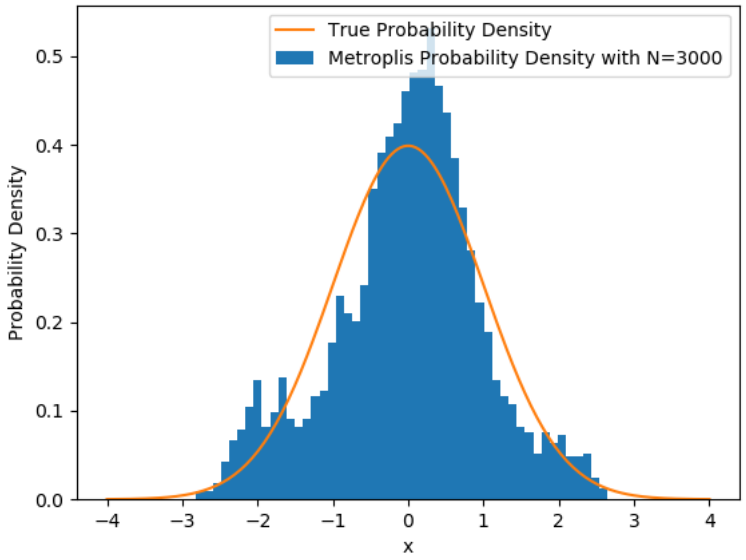
- Suppose we want to sample random numbers from a Gaussian distribution using the Metropolis algorithm. We have

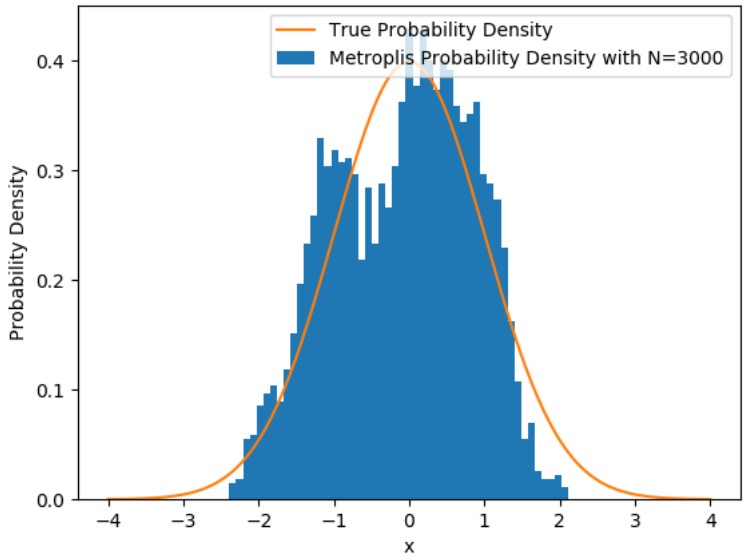
$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

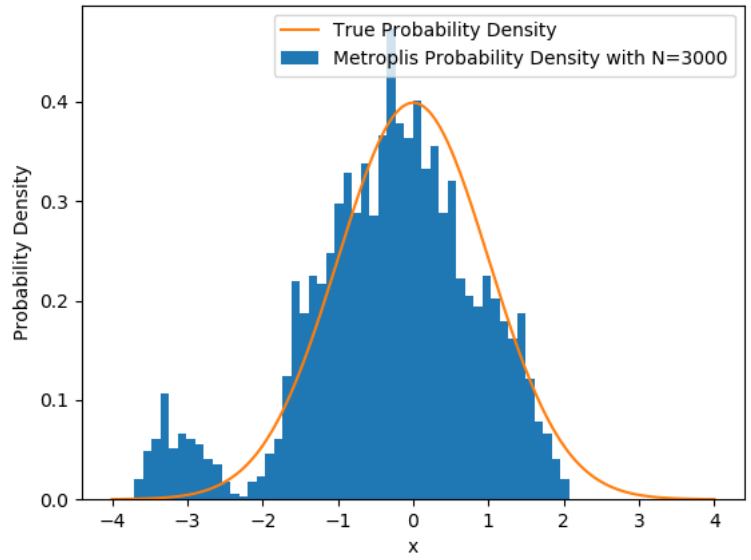
  - Choose a starting point and a max trial step size Δ
  - For each timestep n, choose $\delta \sim U(x_{n-1} - \Delta, x_{n-1} + \Delta)$ and calculate the relative probability $R = \frac{P(x_{n-1}+\delta)}{P(x_{n-1})}$
    - If $R \geq 1$, set $x_n = x_{n-1} + \delta$
    - If $R < 1$, choose a random number $r \sim U(0,1)$
      - If $r \leq R$, set $x_n = x_{n-1} + \delta$
      - If $r > R$, set $x_n = x_{n-1}$

Example – Gaussian Random Numbers

- For these simulations I chose $\Delta = 0.25$ and generate 3000 "microstates". What's gone wrong?



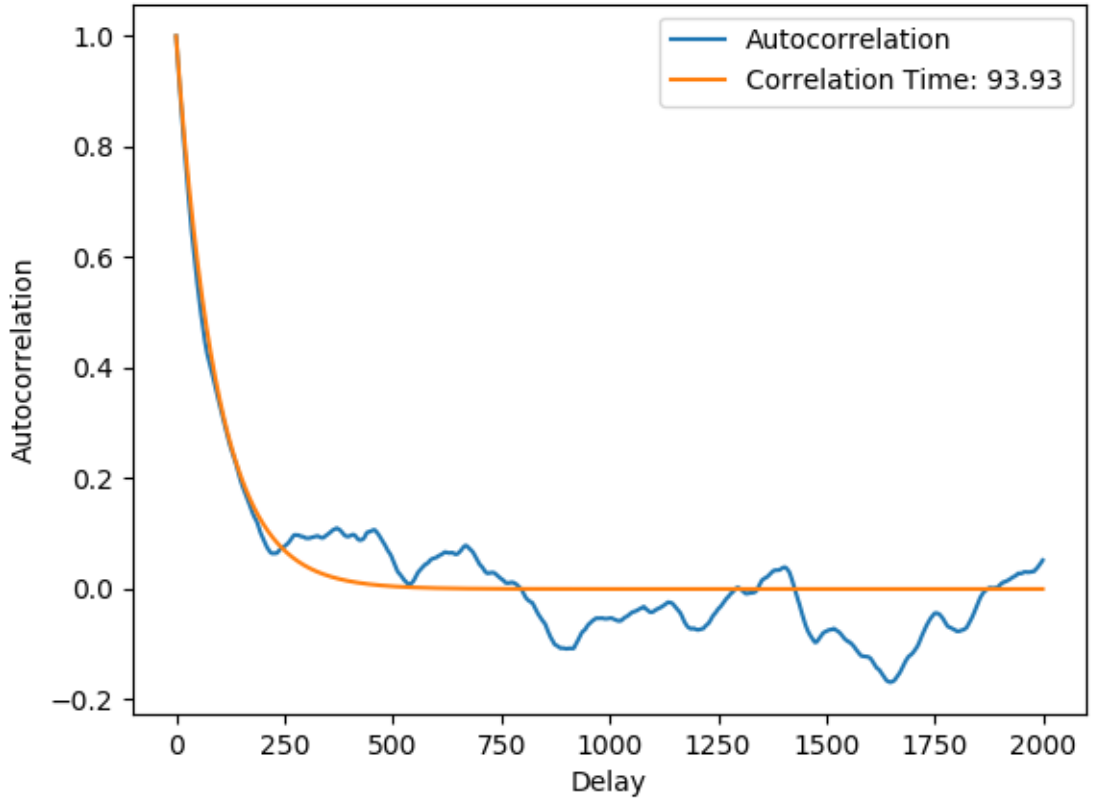




Temporal Correlations

- Because we are only taking small steps, adjacent microstates are highly correlated! We need to discard correlated microstates when obtaining statistics for our ensemble.
- Define the autocorrelation function

$$R(\tau) = \frac{\langle (x_t - \mu)(x_{t+\tau} - \mu) \rangle}{\sigma^2} \approx \exp\left(-\frac{\tau}{\tau_C}\right)$$

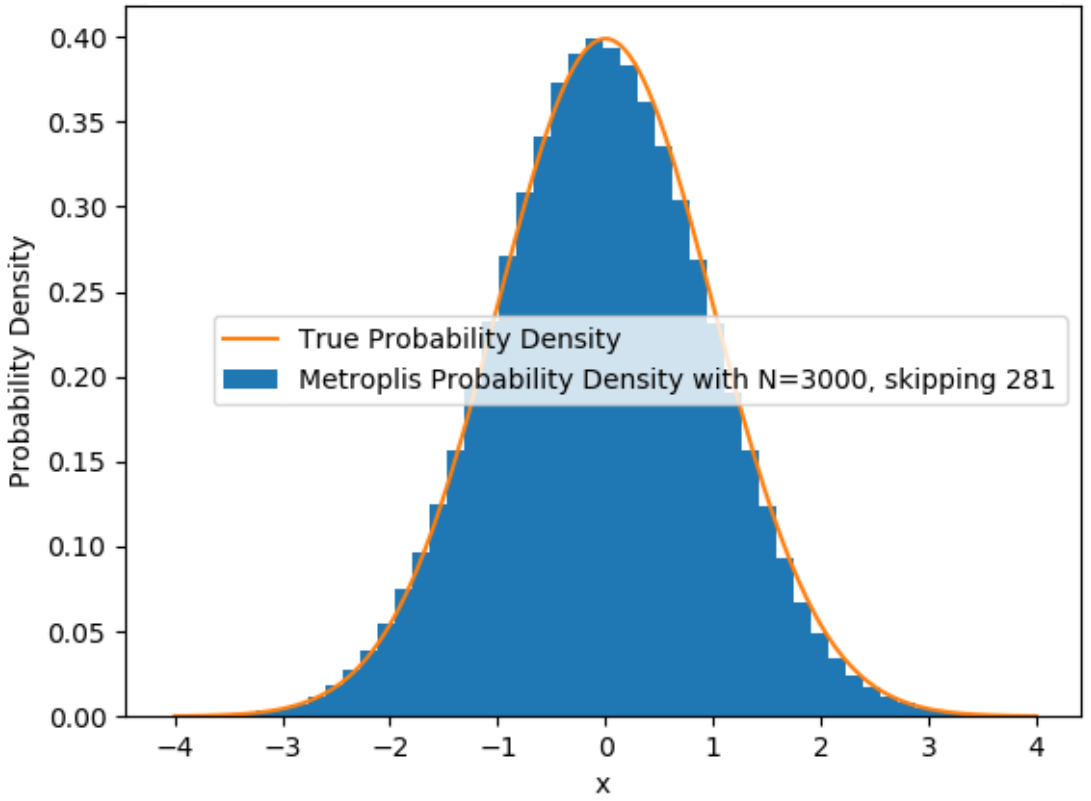


uOttawa

Temporal Correlations Corrected

- Since the autocorrelation is exponentially decaying with time constant τ_C, we should probably only take one microstate in every, $\sim 3\tau_C$ steps.

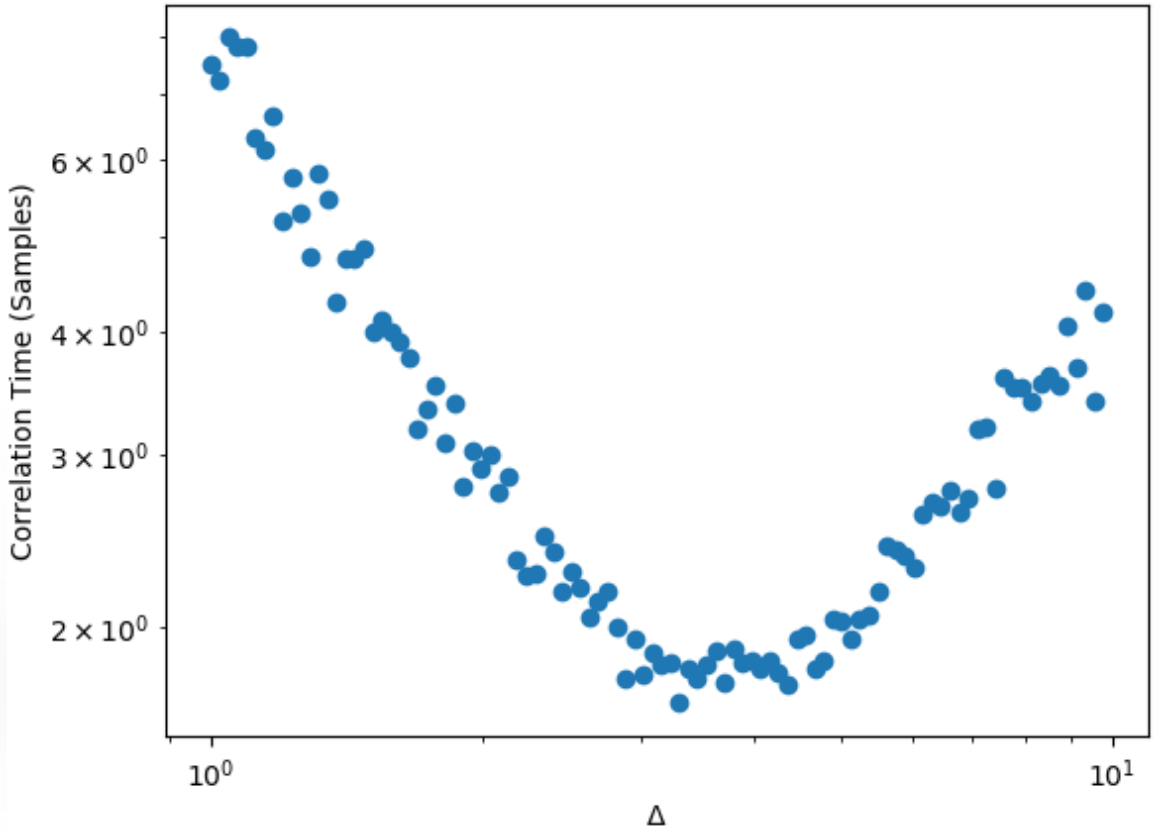
- The downside is that to get N "good" microstates, we have to take $3\tau_C N$ metropolis steps.



Temporal Correlations

- It turns out there is an optimal value of the maximal step size Δ which minimizes the correlation time.

- $\Delta_{best} \approx 3.5\sigma$
  - With this choice, the root mean square value of the step size is $\sqrt{\langle \delta^2 \rangle} = \frac{3.5}{\sqrt{3}}\sigma \approx 2\sigma$
  - Big enough to sample most of the distribution in a single step, but small enough to guarantee good acceptance probability.



Spatial Correlations

- Correlations are not just temporal. We can also consider spatial correlation functions. The 1D spatial autocorrelation for a function $f(x)$ is

$$R_{ff}(X) = \int_{-\infty}^{\infty} f(x)f(x+X)dx$$

- For the Gaussian function $p(x) \propto \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$, the correlation function is also Gaussian:

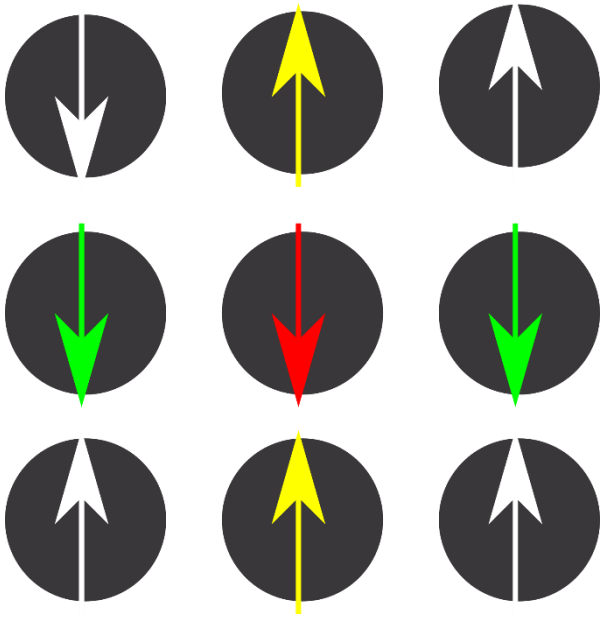
$$R_{pp}(X) \propto \exp\left(-\frac{X^2}{4\sigma^2}\right)$$

- The correlation length (distance over which R_{pp} decays to $\exp(-1)$ of its max value) is equal to 2σ, which is the optimal average step size we found earlier!
  - We want our step size to be equal to the spatial correlation length, on average.

The Ising Model

- Consider a 2D square lattice of N spins $\{\sigma_i\}$ which interact with nearest neighbours with an interaction strength J and with an external magnetic field B via their magnetic moment μ. This system has the Hamiltonian:

$$E(\{\sigma_{ij}\}) = -J\sum_{\{i,j\}}^{N}\sigma_i\sigma_j - \mu B\sum_{i=1}^{N}\sigma_i$$



- A positive value of J indicates a ferromagnet and $\{i,j\}$ denotes nearest neighbours.

Units have no place in simulations

- Floating point arithmetic gets weird when numbers have very large or very small magnitudes. You have to normalize everything before it goes into a simulation. You can also think of this as setting $J = k_B = 1$.

$$E' = EJ^{-1}$$
$$B' = \mu BJ^{-1}$$
$$T' = k_B TJ^{-1}$$

- Dropping primes, we have:

$$E\{\sigma_{ij}\} = -\sum_{\{i,j\}}^{N} \sigma_i \sigma_j - B \sum_{i=1}^{N} \sigma_i$$

- This formulation also means we don't care what the actual value of J is. We can simulate universal results valid for any J.

The Metropolis Algorithm for spin systems

- We want to sample microstates from a canonical ensemble of spins in a 2D lattice. Each microstate $\{\sigma_{ij}\}$ is Boltzmann weighted:

$$P(\{\sigma_{ij}\}) \propto \exp\left(-\frac{E\{\sigma_{ij}\}}{T}\right)$$

  - Start with spins randomly assigned.
  - For each timestep, choose a spin at random and calculate the relative probability $R = \frac{P(\{\sigma_{ij}\}')}{P(\{\sigma_{ij}\})} = \exp\left(-\frac{\Delta E}{T}\right)$ of flipping it.
    - If $R \geq 1$, flip the spin
    - If $R < 1$, choose a random number $r \sim U(0,1)$
      - If $r \leq R$, flip the spin
      - If $r > R$, do not flip the spin

Just kidding, Metropolis is actually terrible

- Recall what happened before when we used a step size that was too small in the Metropolis algorithm example with Gaussian random numbers:
  - Long correlation times!

- The single spin flip is the smallest possible step size in the Ising model, and the correlations times are extremely long as a result.

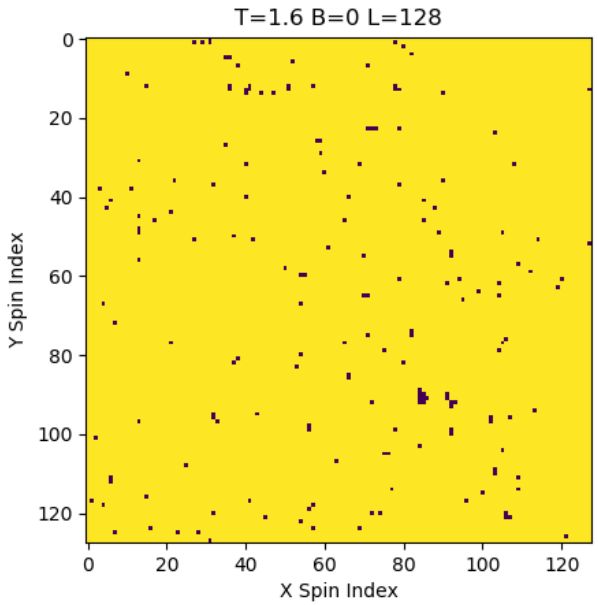
- Just like before, we would like our step size to be on average equal to the "correlation length" for our spin system. What does that look like?

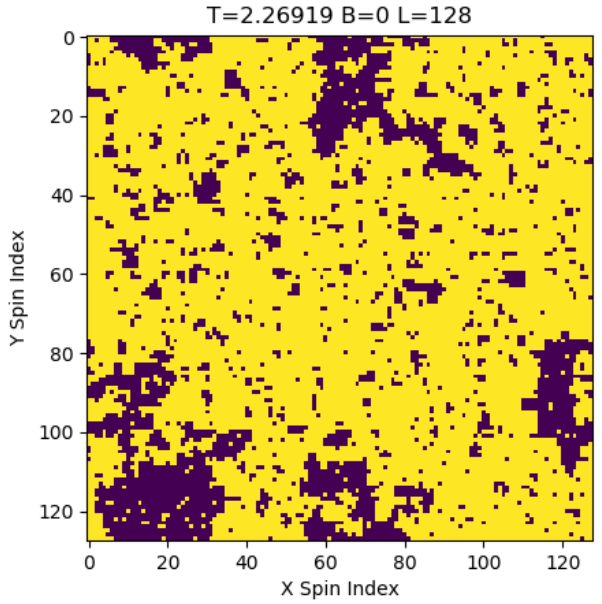
Local Spin Correlations

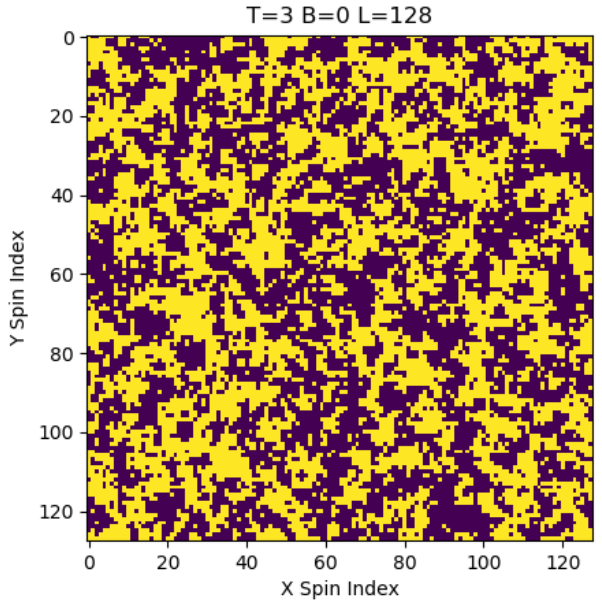
- Consider a few typical microstates, and notice that spins have strong local correlations, which depend on temperature.

- 2D spin-spin correlation function:

$$\Psi(R) = \langle \sigma_r \sigma_{r+R} \rangle = \frac{1}{r^{\eta}} \exp\left(-\frac{R}{\xi}\right)$$







uOttawa

The Wolff algorithm

- This algorithm is only effective when there is no external magnetic field. When that's the case, it works very well.
  - Choose a spin at random. This is the seed for a spin cluster.
  - For each nearest neighbour with the same spin as the seed, add it to the cluster with probability

$$P_{add} = 1 - \exp\left(-\frac{1}{T}\right)$$

  - Repeat this for all of the newly added spins until the cluster stops growing.
  - Flip the entire cluster.

- Cluster sizes on the order of the correlation length guarantee minimal correlation times!

The 2D Ferromagnetic Transition

- The 2D Ising ferromagnet is one of the simplest systems to exhibit a phase transition.
  - At the Curie temperature $T_C = \frac{2}{\ln(1+\sqrt{2})}$, the magnet changes from unmagnetized to magnetized, and many physical quantities diverge, leading to the so-called critical exponents.
  - Define the reduced temperature $\tau = \frac{T-T_C}{T_c}$

$$|M| \sim \tau^{\beta}$$

$$C_V \sim \tau^{-\alpha}$$

$$\chi_M \sim \tau^{-\gamma}$$

$$\xi \sim \tau^{-\nu}$$

$$|M| \sim H^{\delta}$$

$$\Psi(r) \sim \frac{\exp\left(-\frac{r}{\xi}\right)}{r^{\eta}}$$

Finite Size Scaling

- Our system is finite in size, and therefore cannot have an infinite divergence.
  - We can actually take advantage of this to get accurate results anyway, if we're clever.

$$|M| \sim L^{-\frac{\beta}{\nu}}$$

$$C_V \sim L^{\frac{\alpha}{\nu}}$$

- At the critical point the correlation length diverges. However, it cannot actually be larger than the system size, so at $T = T_C$, $\xi \sim L$.
  - Since $\xi \sim \tau^{-\nu}$, we then have $\tau \sim L^{-\frac{1}{\nu}}$, and so:

$$\chi_M \sim L^{\frac{\gamma}{\nu}}$$

Running your simulations

- Give a list of temperatures, external fields, and system sizes
  - Note that size is the side length of the square lattice, so $N = L^2$.
  - The script will generate an ensemble of microstates for every possible permutation of the three lists.
  - Note that all input variables must be normalized as discussed.

```
from ising2d import ising2d
import numpy as np

def main():
    temperatures = np.array([1,2,3])
    fields= np.array([0])
    sizes = [64, 32, 16, 8]
    microstates = 1000
    save_states = 5
    magnet = ising2d(temperatures, fields, sizes, microstates,
                     output_folder = 'example_folder', save_states=save_states)
    magnet.run()

if __name__=='__main__':
    main()
```

uOttawa

Output formats

- The program will create an output folder. Inside are two csv files and one folder
  - correlations.csv
  - observables.csv
  - states
    - .csv files and .png images for microstates you chose to output

- All of your data analysis will be done using observables.csv

Introduction to pandas

- This file contains microstates labelled by L, N, T, and B. The energy and magnetization for each state are given. Every row is a new independent microstate.

- When analyzing .csv files, use the python module pandas. It imports .csv files into data structures that are very easy to chop up as you please.

- Your assignment will be to simulate several ensembles and thoroughly analyze the ferromagnetic phase transition using simulation code which I will provide. You are responsible for generating and analyzing the data.

	A	B	C	D	E	F
1	B	E	L	M	N	T
2	0	-6136	64	-3040	4096	2.246493
3	0	-5960	64	-2984	4096	2.246493
4	0	-6072	64	3152	4096	2.246493
5	0	-6040	64	2664	4096	2.246493
6	0	-6076	64	2598	4096	2.246493
7	0	-6076	64	-3066	4096	2.246493
8	0	-6260	64	-3154	4096	2.246493
9	0	-6292	64	-3010	4096	2.246493
10	0	-5896	64	-3062	4096	2.246493
11	0	-6252	64	3182	4096	2.246493
12	0	-6288	64	2978	4096	2.246493
13	0	-6072	64	-3088	4096	2.246493
14	0	-5884	64	2950	4096	2.246493
15	0	-5696	64	2114	4096	2.246493
16	0	-6160	64	-2988	4096	2.246493
17	0	-6044	64	-3234	4096	2.246493
18	0	-6332	64	-3296	4096	2.246493
19	0	-5960	64	2924	4096	2.246493
20	0	-5832	64	-2666	4096	2.246493
21	0	-6144	64	2992	4096	2.246493
22	0	-6100	64	-2946	4096	2.246493
23	0	-6008	64	2804	4096	2.246493
24	0	-6032	64	-3024	4096	2.246493
25	0	-6176	64	-3124	4096	2.246493
26	0	-6076	64	3146	4096	2.246493
27	0	-6124	64	2866	4096	2.246493
28	0	-5968	64	1500	4096	2.246493
29	0	-6352	64	3300	4096	2.246493
30	0	-5960	64	-3022	4096	2.246493
31	0	-6036	64	2852	4096	2.246493
32	0	-5944	64	-2820	4096	2.246493
33	0	-5720	64	2606	4096	2.246493
34	0	-5996	64	-2822	4096	2.246493
35	0	-5956	64	-3066	4096	2.246493
36	0	-6032	64	3142	4096	2.246493

Pandas cheat sheet

- Load data:

```
import pandas as pd
observables = pd.read_csv('observables.csv')
```

- Get the full energy column as a numpy array:

```
E = observables['E'].values
```

- Get a list of unique temperatures:

```
temperatures = observables['T'].unique()
```

- Get all rows where L=16:

```
subset = observables[observables['L'] == 16]
```

- Get all magnetization values for microstates with T=2 as a numpy array:

```
magnetization = observables[np.absolute(observables['T']-2) < 1e-9]['M'].values
```

Notes on data analysis

- When fitting a power law, it is always better to linearize it and do a linear fit of $\ln y$ against $\ln x$ instead:

$$y = Ax^p$$
$$\ln y = \ln A + p \ln x$$

- You will be asked to do several power law fits in your assignment. You MUST linearize the model before you fit.
  - When you do, you can use numpy.polyfit() to do a linear fit. It will even give you error estimates if you ask for them!

- You must propagate errors through your calculations. At the end you will have estimates for all 6 critical exponents, complete with error bars. Standard error propagation applies: for a function $y = f(\vec{x} \pm \Delta\vec{x})$ of n variables x_i the uncertainty Δy is given by

$$\Delta y = \sqrt{\sum_{i=1}^{n} \left(\frac{\partial f}{\partial x_i}\right)^2 \Delta x_i^2}$$

Assignment

- I will be available via e-mail for questions:

kbrig035@uottawa.ca

- Today is programming help day, but if more time is needed we may be able to book a computer lab when Dr. Harden is back. No promises.

- I wrote ising2d.py, so there may be bugs or unhandled exceptions. I have verified that everything in the assignment works, but if you have problems please report them to me promptly.