DSECOP 2023 Workshop

Connecting Monte Carlo Methods to Modern AI/ML

Ashley S. Dale
Physics, Purdue School of Science, IUPUI

Outline

1. Module Overview
2. Notebook Walkthrough
3. Future Work
4. Inclusion in a Computational Physics or Statistical Mechanics course
5. Conclusion

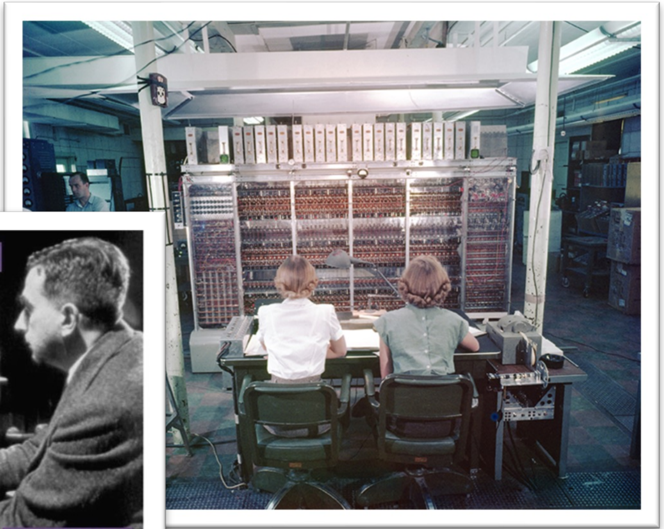
Module Overview

Physicists, Computers, and the Metropolis-Hastings Algorithm

- **MANIAC Computer** c. 1953 at Los Alamos National Lab

- MANIAC is custom built for the ***Metropolis-Hastings Algorithm*** invented by A. Rosenbluth, M. Rosenbluth, M. Teller and E. Teller

- Metropolis-Hastings Algorithm **calculates equations of state** using Monte Carlo integration



THE JOURNAL OF CHEMICAL PHYSICS VOLUME 21, NUMBER 6 JUNE, 1953

Equation of State Calculations by Fast Computing Machines

NICHOLAS METROPOLIS, ARIANNA W. ROSENBLUTH, MARSHALL N. ROSENBLUTH, AND AUGUSTA H. TELLER,
Los Alamos Scientific Laboratory, Los Alamos, New Mexico

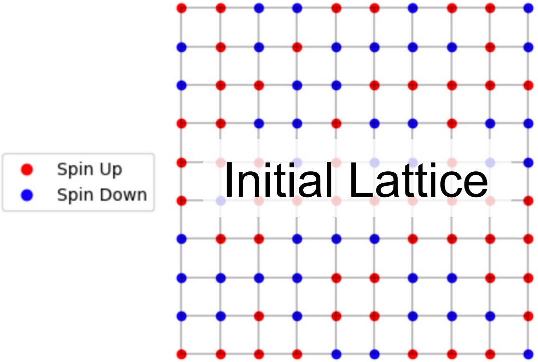
AND

EDWARD TELLER,* *Department of Physics, University of Chicago, Chicago, Illinois*

Images source: https://discover.lanl.gov/news/0412-maniac/
https://twitter.com/XihongLin/status/1369146556637732873

Metropolis-Hastings Algorithm

1. Initialize variables: Lattice *L*, temperature *T*
2. Choose a site in the lattice
3. Calculate the energy
   – Physicists like to use a Hamiltonian
4. IF energy *E* **decreases** when a state change is made at the lattice site
   – Keep the state change and check another lattice site
5. IF energy *E* **increases** with a state change BUT the state is thermodynamically probable according to the Boltzmann distribution
   – Keep the state change
6. Repeat from step 2 until you are happy with the system state



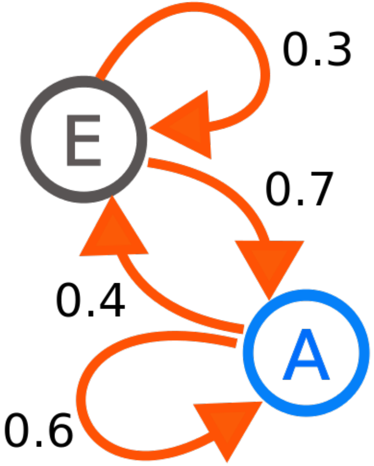
Physics + Monte Carlo + Machine Learning

Physics	Monte Carlo	Machine Learning
Understand physics model vs physics simulation	Markov Chains	The *Universal Approximation Theorem* and when to choose a deep neural network
Familiarity with Ising model for ferromagnetic systems	Monte Carlo Integration	Three kinds of regression models: 1. Polynomial based 2. 1D Latent-feature based 3. 2D image based
Understand when to use analytical, linear, and non-linear computational approaches	Metropolis-Hastings algorithm	Code optimization and efficiency techniques. It won't work if it's too slow!
Familiarity with simulation metrics	The importance of choosing the correct probability distribution	Unsupervised learning approaches compared to supervised learning approaches

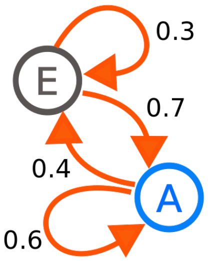
Notebook Walkthrough

N1: Intro to Monte Carlo Methods and Markov Chains

1. The difference between a model and a simulation
2. The definition of a Markov Chain, and how to implement one
3. How to implement Monte Carlo integration
4. How to evaluate simulation results using standardized metrics

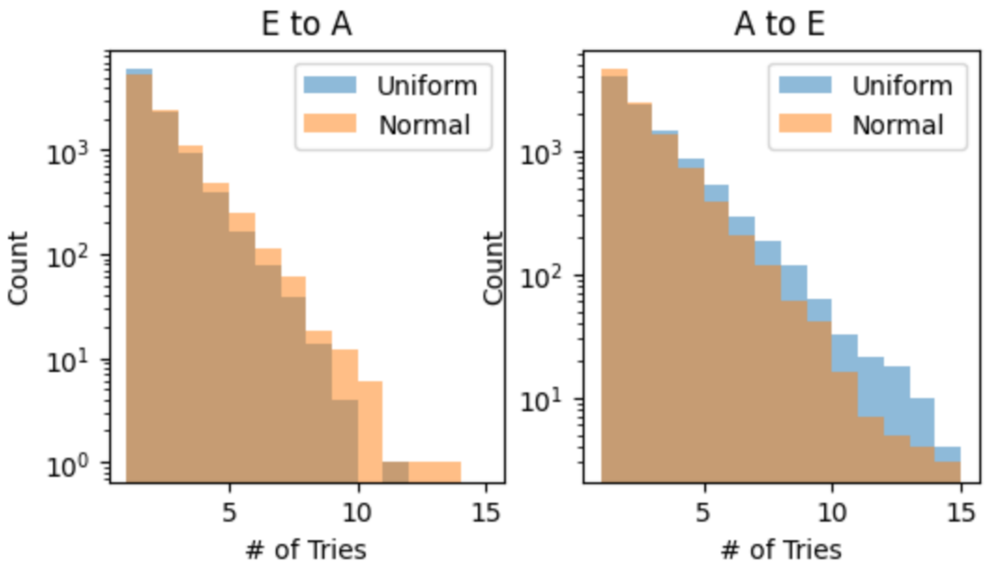


Images source: https://en.wikipedia.org/wiki/Markov_chain#/media/File:Markovkate_01.svg



N1: Programming Exercise 1

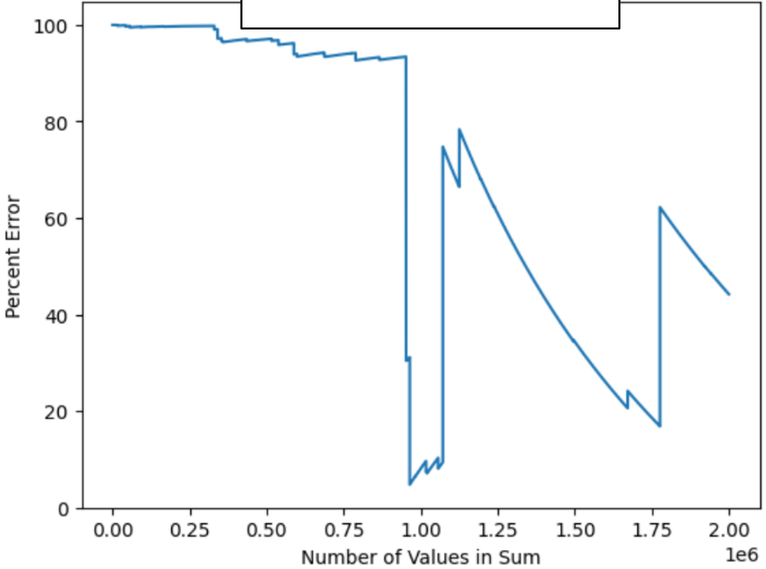
Asks students to program an implementation of the Markov Chain shown, then consider the effect of sampling from different distributions.



N1: Programming Exercise 2

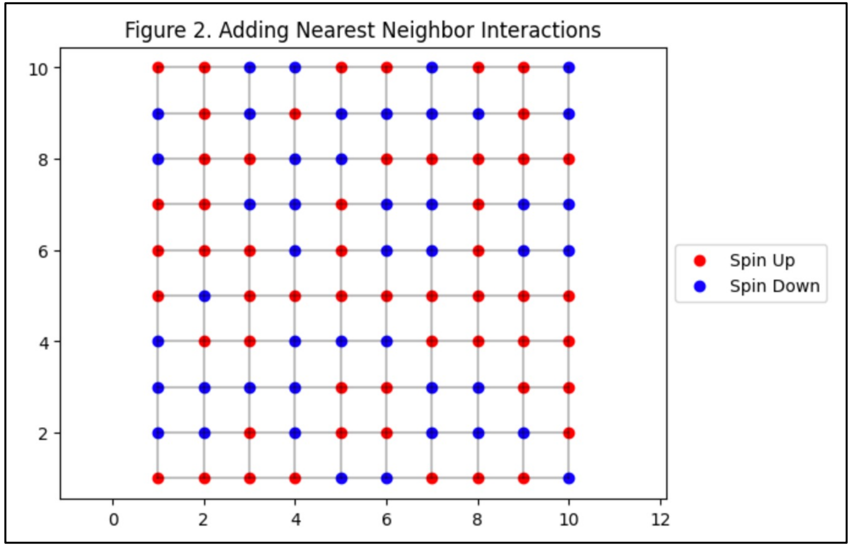
1. Asks students to select an integral from a table in the notebook
2. Students calculate an analytical solution for that integral
3. Students use Monte Carlo integration to evaluate the integral, and compare the results between the numerical and analytical solutions

$$\langle F^N \rangle = \frac{b-a}{N-1} \sum_{i=0}^{N} f(X_i)$$



N2: 2D Ising Model Simulation

1. The Ising model for ferromagnetic systems
2. How to implement the Metropolis-Hastings algorithm for a 2D Ising Lattice
3. How to evaluate the efficiency of your code and time the execution
4. How to speed up your code so that you can run larger simulations using the same computational resources



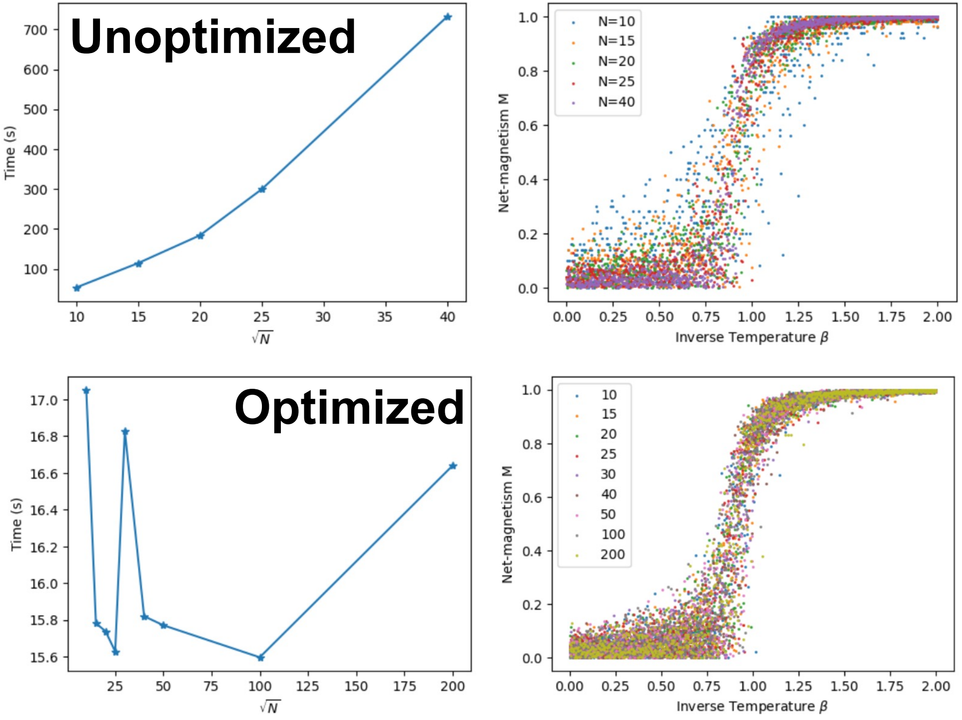
N2: Programming Exercise 1

Students implement the Metropolis-Hastings algorithm for the 2D Ising Model twice:

1. **Unoptimized**: 40x40 lattice takes > 700 seconds

2. **Optimized**: 200x200 lattice takes < 20 seconds

Overall statistics are comparable

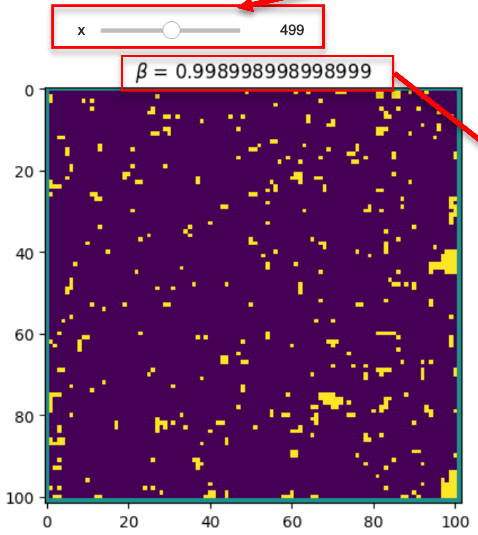


N2: Programming Exercise 2

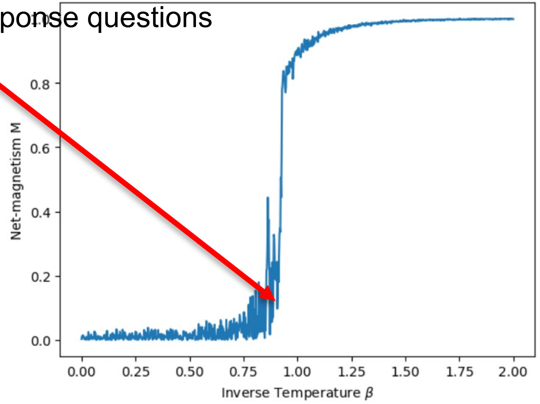
Students use their optimized code to create a set of 500 lattices at a range of temperatures.

Interactive notebook plotting and free-response questions encourage students to match lattice structures to the net-magnetization plot.

Slider lets students scan through 500 lattices sequentially

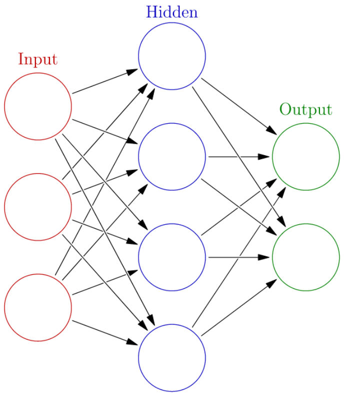


Lattice magnetic domain structure is explicitly connected to net-magnetization plot in free-response questions

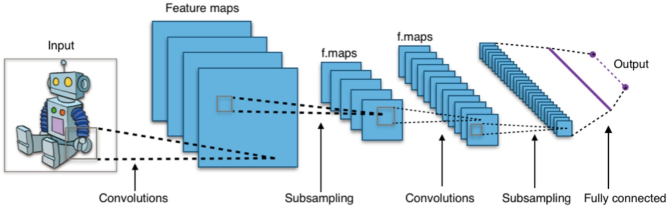


N3: Temperature Prediction of Magnetic Patterns

- The difference between linear/non-linear regression and Deep Neural Networks (DNN)
- The Universal Approximation Theorem AKA "Why a DNN works at all"
- How to implement a Fully-Connected Deep Neural Network (FC-DNN) for regression of 1D data
- How to implement a Convolutional Neural Network (CNN) for regression of 2D data



Fully-Connected Deep Neural Network



Convolutional Neural Network

Images source: https://en.wikipedia.org/wiki/Convolutional_neural_network#/media/File:Typical_cnn.png, https://en.wikipedia.org/wiki/Artificial_neural_network#/media/File:Colored_neural_network.svg

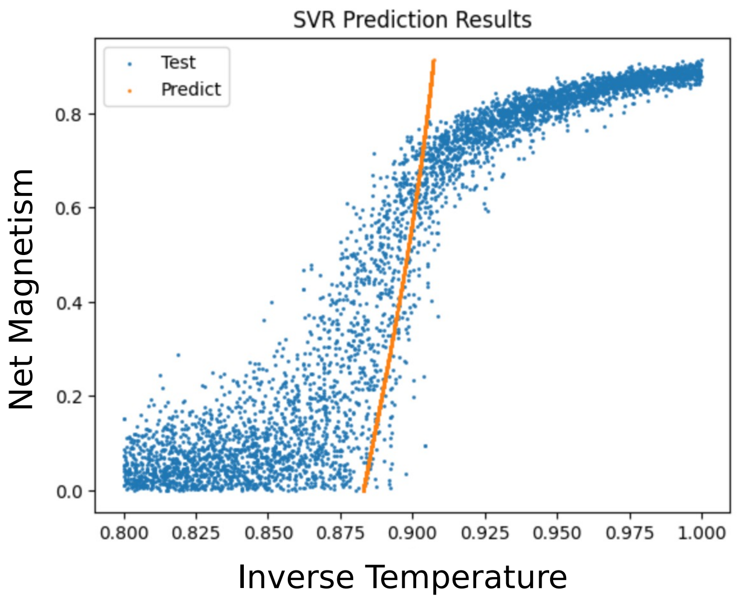
N3: Programming Exercise 1

Implement a Support Vector Regression (SVR) model in two lines using sklearn:

```
svr = SVR(kernel="sigmoid", C=1, gamma="auto", epsilon=1E-12)
y_predict = svr.fit(X_train[:, None], y_train).predict(X_test[:, None])
```

Goal is to create intuition about how the model behaves given the data by trying to optimize it.

Students modify the hyper-parameters and keep a log of the Mean Squared Error for their attempts.

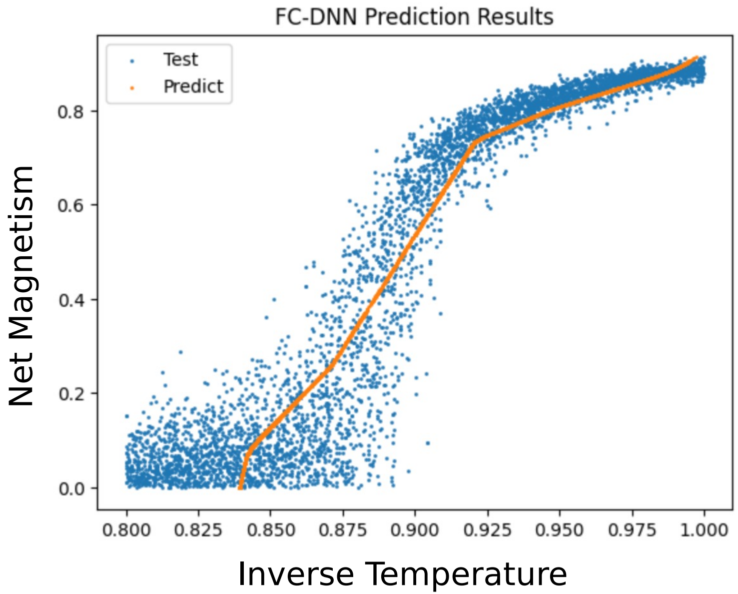


N3: Programming Exercise 2

Students implement a Fully Connected Deep Neural Network (FC-DNN) using `scikit-learn` and 4 lines of code

Dataset is the net-magnetism of the lattices and the inverse temperature

Students modify the hyperparameters for the model and track the Mean Squared Error of their results



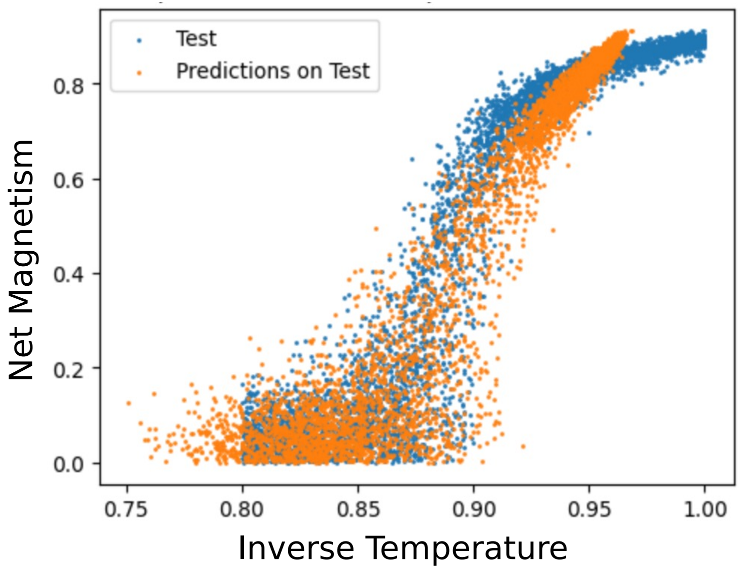
N3: Programming Exercise 3

Students implement a 2D Convolutional Neural Network (CNN) using TensorFlow and a GPU

Dataset is the 2D lattices and inverse temperature

This model is much noisier than either the SVM or FC-DNN model, but it does a better job capturing thermal fluctuations

If students have time and resources, they can further optimize the model. However, notebook questions are limited to conceptual understanding of this work.

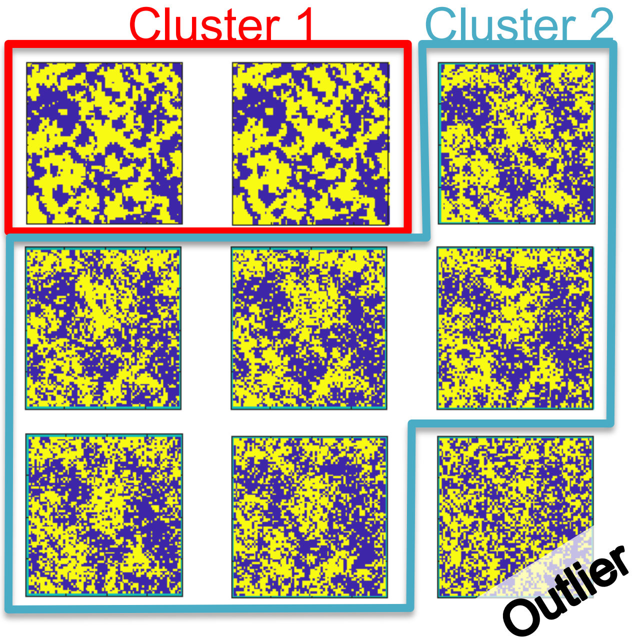


Future Work

N4: Magnetic Domain Clustering

1. Understand the difference between supervised and unsupervised machine learning methods

2. Understand how latent features (like magnetic cluster size) can be used to create meaningful analysis



Magnetic domains simulated during a temperature sweep

N5: Quantum Computing Extension

Google has a *Quantum Virtual Machine* (QVM) available claimed to approximate a hardware system to within experimental error

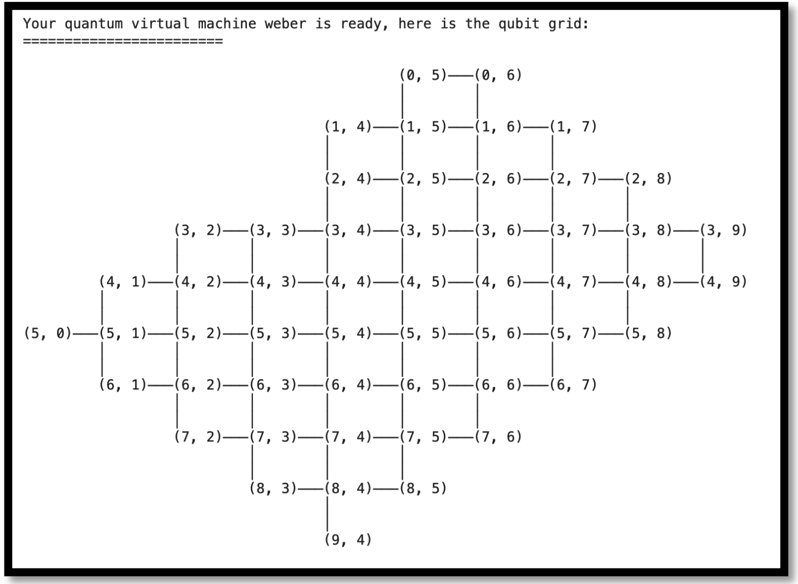
1. Replace atoms with “qubits”

2. Learn to use a QVM by implementing Metropolis-Hastings algorithm and a modified Ising model

3. Introduces basic hardware design and python libraries used for quantum computing applications

4. Allows comparison with purely software methods

Connections to undergraduate quantum mechanics course



Example qubit array from Google Colab tutorial

Inclusion in Physics Course

Notebook Sequence

N1. Introduction to Monte Carlo Methods and Markov Chains

(optional)
100% homework
Time: 1 hr

N2. 2D Ising Model Simulation

50% in-class
50% homework
Time: 1+ hr

N3. Temperature Prediction from Magnetic Patterns

100% homework
Time: 3-5 hrs

N4. Magnetic Domain Clustering (In-progress!)

100% homework
Time: TBD

N5. Quantum Computing (In-progress!)

50% in-class
50% homework
Time: TBD

Designed for 3rd and 4th year physics students

Emphasis on gaining "Hands-on" Intuition over memorizing derivations and algorithms

Encourage students to treat Jupyter Notebooks as "cheat-sheets"

N1. Introduction to Monte Carlo Methods and Markov Chains

- Soft introduction to Random Variables
- Modeling vs Simulation
- Quantifying uncertainty in computations

N2. 2D Ising Model Simulation

- Ising model derivation
- General approach to optimizing code
- Connecting numeric plots to visualizations

N3. Temperature Prediction from Magnetic Patterns

- How to choose a supervised machine learning model for complicated data
- How to compare different models

N4. Magnetic Domain Clustering

- How to identify latent features in the data
- How to use those latent features meaningfully in analysis

N5. Quantum Computing

- Differences between hardware and software implementations of an algorithm
- Even more probability

Scaffolding Approach for Programming: Fill-in-the-blank Coding

- Students are expected to **form good coding habits** by
  - **Reading** code, code comments, and code documentation
  - **Reusing** copy-paste code or lightly-editing existing code based on instructions in the notebook
  - **Reconstructing** code in template functions
- Students are expected to programmatically perform calculations, e.g. calculating the average value of a list of numbers
- Students are not expected to create algorithms from scratch
- Students are not expected to be familiar with Python libraries or functions
- Instructor solutions and rubric provided for all exercises; additional instruction notes provided where potentially useful

What does this look like in a practical sense?

Example: Reading Code Comments

This type of exercise asks students to

1. Read the existing code and comments
2. Decide what should be returned as the result
3. Uncomment a command to make this happen

```
# Decide on a probability distribution function for the simulation
# (uncomment one of the probability lines in the function):

def get_probability():
    # This function returns a probability value when called

    # --> This is a sample from a uniform distribution from numpy

    #prob = np.random.uniform(0, 1, 1)

    # --> This is a sample from a normal distribution from numpy
    # --> It is centered at 0.5 to match the default uniform distribution

    #prob = np.random.normal(loc=0.5, size=1)

    return prob
```

Important word

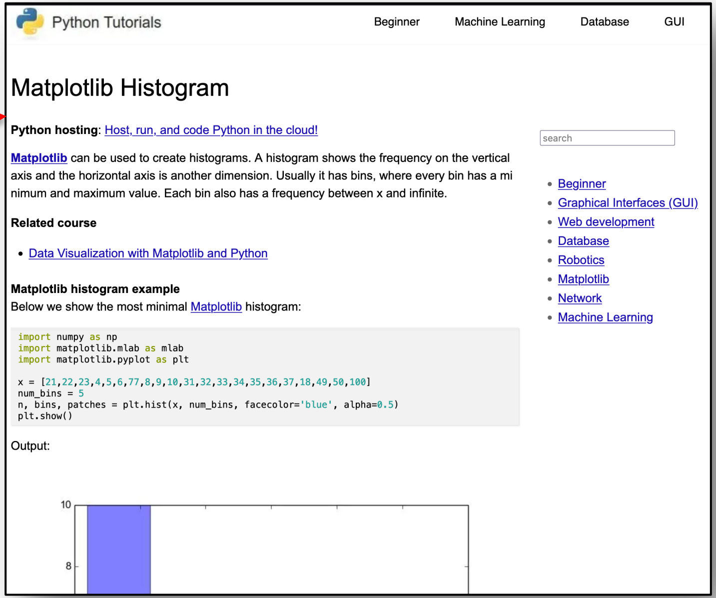
Example: Reading Documentation

Notebook Prompt

Create a figure showing the histogram distribution of the number of steps it took to change states for both `data_EtoA` and `data_AtoE`.

- How to make histograms with Matplotlib.pyplot

```
[ ]  1  # Put your plotting code here
```


Python Tutorials

Beginner Machine Learning Database GUI

Matplotlib Histogram

Python hosting: Host, run, and code Python in the cloud!

Matplotlib can be used to create histograms. A histogram shows the frequency on the vertical axis and the horizontal axis is another dimension. Usually it has bins, where every bin has a minimum and maximum value. Each bin also has a frequency between x and infinite.

Related course

- Data Visualization with Matplotlib and Python

Matplotlib histogram example

Below we show the most minimal Matplotlib histogram:

```
import numpy as np
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

x = [21,22,23,4,5,6,77,8,9,10,31,32,33,34,35,36,37,18,49,50,100]
num_bins = 5
n, bins, patches = plt.hist(x, num_bins, facecolor='blue', alpha=0.5)
plt.show()
```

Output:

search

- Beginner
- Graphical Interfaces (GUI)
- Web development
- Database
- Robotics
- Matplotlib
- Network
- Machine Learning

Students are expected to become familiar with resources available outside of the notebooks to help them finish assignments. *Many* hyperlinks are included throughout the assignments.

Example: Reusing and Reconstructing code in template functions

Completed code provided earlier in the notebook

```python
# Making the lattice a little larger
sqrt_N = 25

# Create a new lattice
init_lattice = np.random.uniform(size=(sqrt_N,sqrt_N))

#mask lattice
init_lattice[init_lattice>0.5]=1
init_lattice[init_lattice !=1]=-1

# A new step here to create non-interacting atoms around the edge
lattice = np.zeros((sqrt_N+2, sqrt_N+2))
lattice[1:sqrt_N+1, 1:sqrt_N+1] = init_lattice

# Define a range of temperatures to test
beta = np.linspace(0, 2, 1000)

# Empty variable to hold the magnetism calculations
M = []

# For each temperature
for temp in tqdm(beta):

    # Repeat the MCMC step 100 times to make sure the system is stable
    for n in range(100):

        [rows, cols] = lattice.shape # Figure out the size of the lattice

        for r in range(1,rows-1): # keep the neighbors inside the region
            for c in range(1,cols-1):

                # sum over the nearest neighbors
                sum_NN = (lattice[r-1,c]+lattice[r+1, c]+lattice[r,c+1]+lattice[r,c-1])

                # calculate the energy
                E_a = -0.5*lattice[r,c]*sum_NN

                # re-calculate the energy for a spin state change
                E_b = -1*E_a

                # choose whether to keep the new state or not
                if E_b < E_a or (np.exp(-(E_b - E_a)*temp) > np.random.rand()):
                    lattice[r, c] *= -1

    # After the system is stable, calculate the net magnetism by summing over
    # all of the spin values and averaging them
    M.append(np.abs(np.sum(np.sum(lattice)))/(sqrt_N*sqrt_N))
```

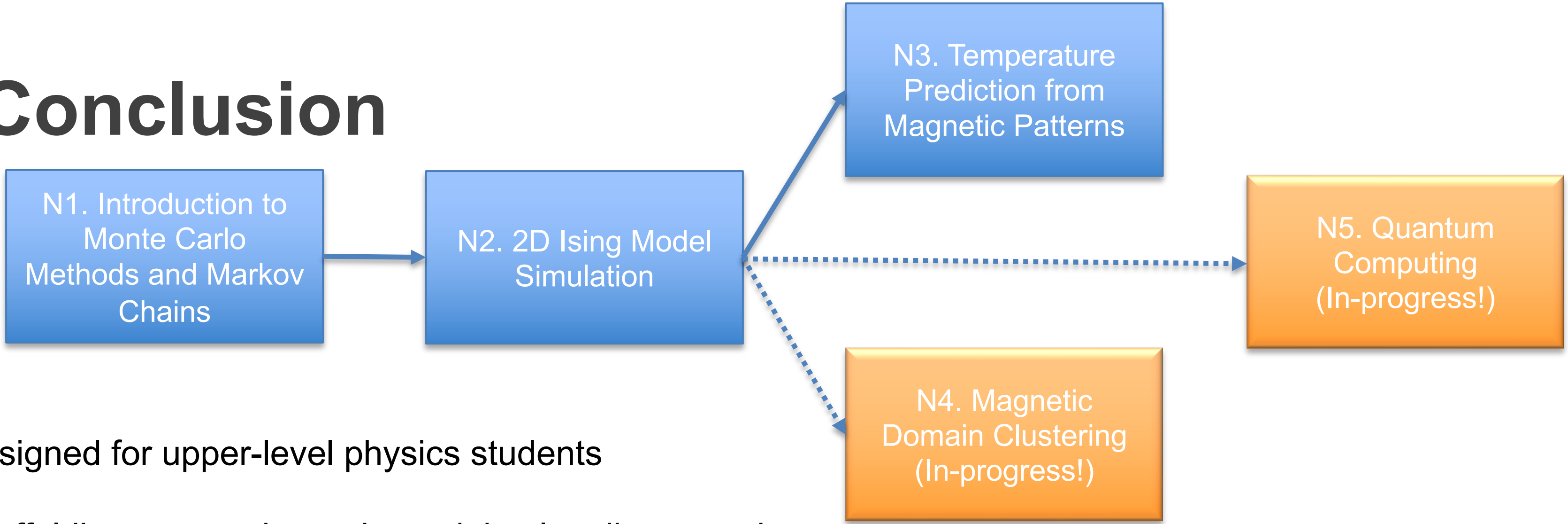
Students are asked to fill in the new function. All that is needed is comprehension and copy-paste.

Input argument names and returned argument names are typically predefined to ease debugging

```python
# Complete the MCMC_step function below

def MCMC_step(beta: float, lattice: np.array):
    """
    Function to repeat the Monte Carlo Markov Chain for this system.
    beta: the inverse temperature value for the MCMC step
    lattice: the system of spins that will be simulated
    returns: an updated version of the input lattice
    """

    # Figure out the size of the lattice
    [rows, cols] = lattice.shape

    # keep the neighbors inside the region
    for r in range(1,rows-1):
        for c in range(1,cols-1):

            # sum over the nearest neighbors
            sum_NN =

            # calculate the energy
            E_a =

            # re-calculate the energy for a spin state change
            E_b = -1*E_a

            # choose whether to keep the new state or not
            if #<ENTER LOGIC STATEMENT HERE>
                lattice[r, c] *= -1

    return lattice
```

Conclusion

Conclusion



- Designed for upper-level physics students
- Scaffolding approach requires minimal coding experience
- Use Jupyter Notebooks as “cheat-sheets” for the future
- Emphasize intuition over how the algorithms behave over memorizing implementations

Let’s chat! *daleas@iupui.edu*, *https://daleas0120.github.io/*, **aps-gds.slack.com**