MONTE PYTHON - OPEN SOURCE MONTE CARLO DEMOS IN PYTHON: MARKOV CHAIN MONTE CARLO

JULIA MILTON

1. Introduction. Monte Carlo simulations are a class of useful tools to predict the outcome of problems where random variables are present, or that are difficult to solve through other approaches. Many papers have been written that explain and expand the mathematical concepts behind Monte Carlo strategies, however, there are fewer accessible implementations and visual explanations of these methods. Much of the existing code online for methods such as importance sampling and Markov Chain Monte Carlo is also produced in MATLAB, which is a powerful language for analysis and visualization, but is proprietary software that users must purchase a license to use.

The motivation behind this project was to make the information that is necessary to understand and implement some foundational Monte Carlo techniques more accessible to people who do not have extensive backgrounds in mathematics and/or statistics, but who may benefit from using Monte Carlo methods in their work. While rigorous math and notation is essential for developing these techniques and explaining them in academic papers, the intent of each of the demos is to build intuition and provide visualizations of the techniques. Therefore, the mathematical formulations of the each of the methods is provided only briefly, with the bulk of the content of each demo consisting of description of the methods and code that the user can change key parameters in and run to see the results. The simulations are also written in Python - an open-source language - to make the information more accessible.

The overall objective of this project was to develop open source, visual demos for some of the key concepts in Monte Carlo simulations. Included in these demos currently are: rejection sampling, importance sampling, and Markov Chain Monte Carlo. All of the code for these demos is written in Python and is available on GitHub at: https://github.com/juliamilton/Monte-Carlo-Demos

2. Markov Chain Monte Carlo. The goal of Markov Chain Monte Carlo (MCMC) is to be able to draw samples from some probability distribution, even if we don't know the exact probability density at any point. One of the methods for doing MCMC is the Metropolis algorithm, which is the one that is used and discussed in this demo. MCMC and the Metropolis algorithm rely on foundations of Markov chains and the principles of rejection sampling (see the demo for rejection sampling if you are curious). A Markov chain is a sequence of possible events where the probability of each event depends only on the previous state (this is called the "memorylessness" property). MCMC uses a Markov chain to "wander around" on the distribution of interest, picking more samples from areas of higher probability density and less from areas of lower probability density.

3. Basic idea of the method (no-to-little math version):. The general idea of the Metropolis algorithm can perhaps be most intuitively explained through an analogy. Imagine you are responsible for picking apples in an extremely large orchard. You want to visit as many parts of the orchard as possible, maximizing your time where the most apples are grown (maybe the soil is best there, or the sun or

water is better) and minimizing your time in places where there aren't as many good apples to pick. Sadly, you know nothing about soil and you've never been to the orchard before so you don't know how many trees there are, or which ones have the most apples.

To start out, you pick a place randomly that you think might have a lot of apples (it seems sunny). From there you have 2 choices: (1) stay where you are, or (2) select a random new tree to go to. To determine whether you should move to the new location, you count the apples on the new tree - if it has more than the tree you are currently at, you move there. If it has less apples than the tree you are currently at, you don't automatically reject it, you calculate the probability that you should move: $p_{move} = p_{proposed}/p_{current}$.

Finally, you make sense of what p_{move} means. For this, you take out your decision-making wheel of fortune. The decision-making wheel of fortune has markings around the edge between 0 and 1, and an indicator point that you can set to what p_{move} turned out to be (this is analogous to selecting uniformly between 0 and 1). You spin the dial. If the value that it lands on is between 0 and p_{move}, you move to the new tree. If it is greater than p_{move}, you stay where you are.

This analogy captures the key points of MCMC: (1) you pick a new proposed location; (2) you figure out how much "higher" or "lower" (in probability density, which is in terms of number of apples in this analogy) that location is compared to your current location; and (3) you calculate some probability to determine whether to stay put or move to that location to achieve your goal of spending time proportional to density of the location. (This example was adapted from an example described in [2].)

4. The algorithm. Let $\pi(x)$ be the target distribution, a.k.a. a function that is proportional to the desired probability distribution $P(x)$.

1. Initialize by choosing a point, x_t to be a starting point.
2. Choose a candidate next point by generating from an arbitrary probability density $Q(x|x_t)$ that suggests the next point, x, given x_t. An easy way to do this is by adding a $N(0,1)$ random number to x_t: $x = x_t + N(0,1)$.
3. Calculate the acceptance ratio, $\alpha = \pi(x)/\pi(x_t)$.
4. Generate a uniform random number $u = U(0,1)$
5. If $u <= \alpha$, accept the candidate point and set $x_{t+1} = x$.
6. If $u > \alpha$, reject the candidate point and set $x_{t+1} = x_t$.

This is the algorithm that is implemented in the code given. **The actual MCMC algoritm is shown in lines 109-151 of the code.** (Much of the rest of the code is for visualization and plotting.)

5. Uses and drawbacks. MCMC is a very useful tool to draw samples from a distribution when all you know about the distribution is how to calculate its likelihood. You do not necessarily need to know the shape of the PDF: for instance, you can calculate how much more likely it is that an apple tree will have 50 apples given a mean population number of 50 apples per tree than given a mean population number of 10 apples per tree.

There are a few conditions that must be met for the Metropolis algorithm to work. One is that the proposal distribution should be symmetric - if it is not, a slightly modified version of this algorithm called the Metropolis-Hastings algorithm can be used. Second, because the first step location chosen at random might be very inaccurate a certain number of samples from the very beginning of the run should be discarded. This is called the "burn in" of the MCMC run.

Additionally, one disadvantage of the Metropolis algorithm is that it is sensitive to the step size chosen for each new proposed location. A small step size may cause the sampler to either get stuck in local maxima (areas with higher probability than their close neighbors, but lower probability than other areas further away) or be very inefficient, but a big step size may cause a high rejection rate.

6. The code and visualizations:. In this demo, we will look at the Metropolis algorithm for MCMC. We are using MCMC to evaluate some parameter of interest about our distribution, the mean of x (aka the expectation, E[X]). The default target distribution $p(x)$ that is included in this demo is given by:

$$p(x) = \frac{0.2}{0.3\sqrt{2\pi}} * e^{(-\frac{1}{2}*(\frac{x-1}{0.3})^2)} + \frac{0.7}{0.4\sqrt{2\pi}} * e^{(-\frac{1}{2}*(\frac{x+0.5}{0.4})^2)}$$

Which is a Gaussian mixture model which we are just using for illustration purposes, since E[X] can be calculated analytically: $E[X] = -0.15$. To see how well the MCMC method is working, we will compare the analytical value of E[X] to the approximation of E[X] from the Monte Carlo draws:

$$\bar{E}[X] = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Where the x_is are the accepted points.

In this demo, the user can change the target distribution, as well as how many runs to do. **The user defined parameters can be changed in lines 204-221.**

Shown on the screen is the estimated value for $E[X]$, which should get closer to -0.15 as the number of runs grows larger, as well as a value for σ^2/n, which is a measure of variance. Lower values of σ^2/n mean that the value for $E[X]$ is not changing as much.

Fig. 1 shows the final screen that is shown after a run of 1000 samples. The screen for this demo tries to show the sampling as it occurs in real time from a "random walk" along the number line (x axis). When a point is selected as a proposal, a blue arrow appears along the x axis showing the step direction that will be taken. If the point is selected, it turns yellow and is added to the histogram above. If it is not selected, it turns blue and is not added to the histogram. The histogram then builds up over time. Also included on the screen is a dotted line outline of the target distribution so users can see how the histogram compares to the true target distribution.

REFERENCES

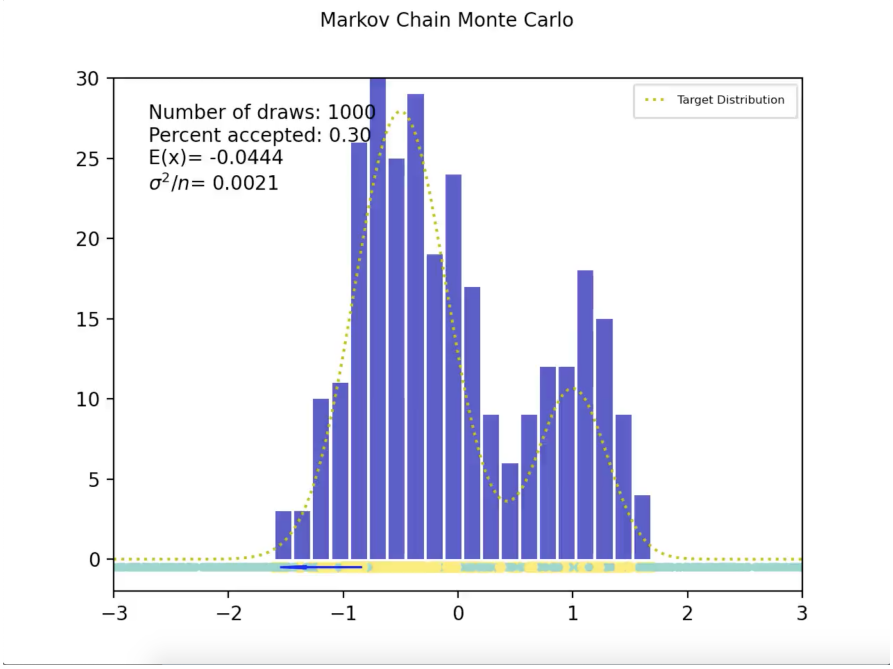

FIG. 1. *Screen shown at the end of the MCMC run*

[1] J. K. KRUSCHKE, *Doing Bayesian data analysis : a tutorial with R and BUGS*, Academic Press, Burlington, MA, 2011, http://www.amazon.com/Doing-Bayesian-Data-Analysis-Tutorial/dp/0123814855.

[2] A. B. OWEN, *Monte Carlo theory, methods and examples*, 2013.

[2] [1]