Markov chain Monte Carlo (MCMC) sampling, part 1: the basics

Markov chain Monte Carlo (MCMC) (https://en.wikipedia.org/wiki/Markov_chain_Monte_Carlo) is a powerful class of methods to sample from probability distributions known only up to an (unknown) normalization constant. But before we dive into MCMC, let's consider why you might want to do sampling in the first place.

The answer to that is: whenever you're either interested in the samples themselves (for example, inferring unknown parameters in Bayesian inference) or you need them to approximate expected values of functions w.r.t. to a probability distribution (for example, calculating thermodynamic quantities from the distribution of microstates in statistical physics). Sometimes, only the mode of a probability distribution is of primary interest. In this case, it's obtained by numerical optimization so full sampling is not necessary.

It turns out that sampling from any but the most basic probability distributions is a difficult task. Inverse transform sampling (https://en.wikipedia.org/wiki/Inverse_transform_sampling) is an elementary method to sample from probability distributions, but requires the cumulative distribution function, which in turn requires knowledge of the, generally unknown, normalization constant. Now in principle, you could just obtain the normalization constant by numerical integration, but this quickly gets infeasible with an increasing number of dimensions. Rejection sampling (https://en.wikipedia.org/wiki/Rejection_sampling) does not require a normalized distribution, but efficiently implementing it requires a good deal of knowledge about the distribution of interest, and it suffers strongly from the curse of dimension, meaning that its efficiency decreases rapidly with an increasing number of variables. That's when you need a smart way to obtain representative samples from your distribution which doesn't require knowledge of the normalization constant.

MCMC algorithms are a class of methods which do exactly that. These methods date back to a seminal paper by Metropolis et al. (https://pdfs.semanticscholar.org/7b3d/c9438227f747e770a6fb6d7d7c01d98725d6.pdf), who developed the first MCMC algorithm, correspondingly called Metropolis algorithm (https://en.wikipedia.org/wiki/Metropolis%E2%80%93Hastings_algorithm), to calculate the equation of state of a two-dimensional system of hard spheres. In reality, they were looking for a general method to calculate expected values occurring in statistical physics.

In this blog post, I introduce the basics of MCMC sampling; in subsequent posts I'll cover several important, increasingly complex and powerful MCMC algorithms, which all address different difficulties one frequently faces when using the Metropolis-Hastings algorithm. Along the way, you will gain a solid understanding of these challenges and how to address them. Also, this serves as a reference for MCMC methods in the context of the monad-bayes (https://www.tweag.io/posts/2019-09-20-monad-bayes-1.html) series. Furthermore, I hope the provided notebooks will not only spark your interest in exploring the behavior of MCMC algorithms for various parameters/probability distributions, but also serve as a basis for implementing and understanding useful extensions of the basic versions of the algorithms I present.

Markov chains

In [ ]:
```
Now that we know why we want to sample, let's get to the heart of MCMC &mdash; Markov chains.
What is a Markov chain?
Without all the technical details, a Markov chain is a random sequence of states in some state space in which the probability of picking a certain state next depends only on the current state in the chain and not on the previous history: it is memory-less.
Under certain conditions, a Markov chain has a unique stationary distribution of states to which it will converge after a certain number of states.
From that number on, states in the Markov chain will be distributed according to the invariant distribution.
MCMC algorithms work by constructing a Markov chain with the probability distribution you want to sample from as the stationary distribution.
In order to sample from a distribution $\pi(x)$, a MCMC algorithm constructs and simulates a Markov chain whose stationary distribution is $\pi(x)$, meaning that, after an initial "burn-in" phase, the states of that Markov chain are distributed according to $\pi(x)$. We thus just have to store the states to obtain samples from $\pi(x)$.
```

For didactic purposes, let's for now consider both a discrete state space and discrete "time". The key quantity characterizing a Markov chain is the transition operator $T(x_{i+1}|x_i)$ which gives you the probability of being in state x_{i+1} at time $i+1$ given that the chain is in state x_i at time i.

Now just for fun (and for illustration), let's quickly whip up a Markov chain which has a unique stationary distribution. We'll start with some imports and settings for the plots:

In [1]:
```
%matplotlib notebook
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [10, 6]
np.random.seed(42)
```

The Markov chain will hop around on a discrete state space which is made up from three weather states:

In [2]:
```
state_space = ("sunny", "cloudy", "rainy")
```

In a discrete state space, the transition operator is just a matrix. Columns and rows correspond, in our case, to sunny, cloudy, and rainy weather. We pick more or less sensible values for all transition probabilities:

In [3]:
```
transition_matrix = np.array(((0.6, 0.3, 0.1),
                              (0.3, 0.4, 0.3),
                              (0.2, 0.3, 0.5)))
```