Lecture Notes on Monte Carlo Methods

Andrew Larkoski

November 7, 2016

1 Lecture 1

This week we deviate from the text and discuss the important topic of Monte Carlo methods. Monte Carlos are named after the famous casino in Monaco, where chance and probability rule. This week we will discuss how to numerically simulate outcomes of an experiment ("data"). This is a huge and important subject into which we will just dip our feet.

Every experiment in physics corresponds to sampling a probability distribution for the quantity that you measure.

Example 1: What is the distribution of the times between cars passing a point on the road?

> We must measure the time it takes for another car to pass the point. As more and more cars are measured, we have more and more data points on which to determine the distribution. We will refer to each instance or data point of an experiment as an event. Any experiment performed by humans contains only a finite number of events. How can we model this?

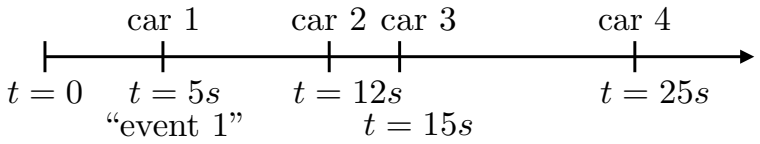

Figure 1: Illustration of four events in the measurement of the distribution in time of cars passing a point.

Example 2: What is the distribution of the positions of a particle in an infinite square well?

> We must prepare n identical infinite square well systems and measure the position of the particle in each system. Each measurement of the position is an event. Note that to determine the distribution requires repeatability: the measurements

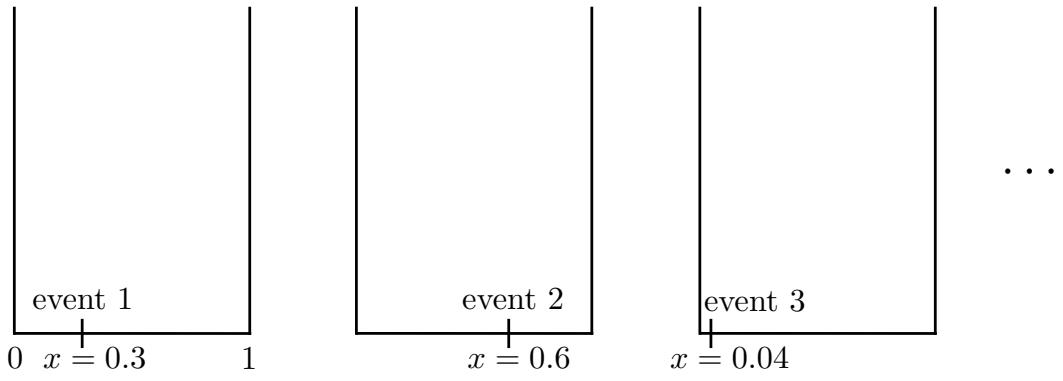

Figure 2: Illustration of an example of three events in which the position of the particle in the infinite square well was measured.

> must be done on identical systems. Otherwise, we do not know how to interpret our results. (The possibility of repeatability is intimately related to energy and momentum conservation, but that's a topic for another class.) How do we sample a probability distribution numerous times?

For any finite number of measurements, we cannot reproduce the probability distribution exactly, but we can approximate it arbitrarily precisely. To do this, take our set of events:

$$I = \{\text{events of experiment}\}\,, \tag{1}$$

and divide them into sets according to their values; these are called bins. For example, for the infinite square well on the domain $x \in [0, 1]$, we will divide the possible measured positions into N_{bins} bins. If an event has a measured position

$$x \in \left[\frac{i-1}{N_{\text{bins}}}, \frac{i}{N_{\text{bins}}}\right]\,, \tag{2}$$

for an integer $i \in [1, N_{\text{bins}}]$, then we add 1 to this bin.

Doing this for all of the N_{ev} events in an experiment generates a histogram, which is a finite approximation to a smooth distribution. This will produce a histogram whose entries sum to the total number of events. If instead we want a histogram that integrates to 1, so that it can be directly interpreted as a probability distribution, instead of adding 1 to the ith bin we add

$$\frac{1}{\Delta x_i\, N_{\text{ev}}}\,. \tag{3}$$

To see that this results in a histogram that integrates to 1, we will denote the number of events in the ith bin by N_i. Then the integral of the histogram is just the sum:

$$\sum_{i=1}^{N_{\text{bins}}} \Delta x_i \cdot \frac{N_i}{\Delta x_i\, N_{\text{ev}}} = \frac{1}{N_{\text{ev}}} \sum_{i=1}^{N_{\text{bins}}} N_i = 1\,. \tag{4}$$

To do this integral, we have used that the integration measure $dx = \Delta x_i$ and that the sum of the events in each bin is just the total number of events.

To interpret and compare our data to a prediction, we want to generate a histogram of events numerically. That is, we want to generate pseudodata, which are a finite number of events drawn from a given probability distribution of our choice. We can then compare directly between the histogram of our real data to our pseudodata, and attempt to draw conclusions. There are numerous methods to quantitatively judge the goodness-of-fit of a histogram by a probability distribution (things like the Kolmogorov-Smirnov test, Student's t-test,[1] etc.), but we won't discuss them here.

While this procedure sounds relatively simple, there are many challenges that must be overcome to produce pseudodata:

- To sample a probability distribution requires the generation of random numbers. We will typically work with uniform, real random numbers on the domain of $[0, 1]$. We will show that this is sufficient to sample an arbitrary probabiltiy distribution. How is this accomplished? How does Mathematica generate random real numbers?
  - Well, Mathematica actually doesn't produce truly random numbers. The numbers generated by Mathematica (or any other programming language) is pseudorandom: they appear random, but are actually defined by a deterministic algorithm. Good pseudorandom number generators repeat on scales of 2^{10000} calls; this is called the period of the generator. This will be sufficient, though later in this class we will discuss other choices. One of the most common pseudorandom number generator is the "Mersenne Twister", which has a period equal to a Mersenne prime (see the poster outside Joel's office).
  - There are some examples of truly random numbers and random number generators. The most famous is probably the collection in the book "A Million Random Digits with 100,000 Normal Deviates". This book is copyrighted, and if you are in need of amusement, check out the Amazon reviews for this book. Now, there are companies producing true random number generators which exploit the randomness of quantum noise, though they aren't found in every household yet.
- More than likely, however, you actually don't know the probability distribution that you should be sampling. For the infinite square well example, we know the probability distribution and can sample it efficiently. What do you do if you don't know it? For example, what if you want to determine the distribution of spins of a material in 2D at finite temperature? How do you make pseudodata of an event (a measurement of the spins in the system)?
  - We do know what the configuration of spins should satisfy physically. Energy is minimized if the spins align, and so we can check spin by spin if they are

[1] Student was the pseudonym of William Sealy Gosset, who worked for the Guinness brewery. Gosset applied statistics to quality control of the beer produced at Guinness but was forbidden to publish, so he created the name "Student" to evade the restriction.

aligned. At finite temperature, energy is shared between the system of spins and the outside world, so the spins do not have to be all aligned. However, it is most likely that the spins are aligned. We will discuss this example in more detail later in the course.

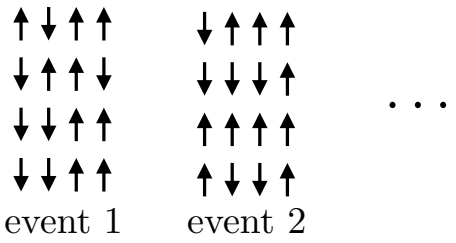

Figure 3: Events in the measurement of a system of 16 spins at finite temperature.

For the rest of this week, we will just assume that we have a random number generator which can produce uniformly distributed real numbers in $[0, 1]$. Briefly later in the course, we will question if this is actually the best strategy. Also, we will assume there that we know the probability distribution that we want to sample and generate pseudodata. So, the problem we will address this week is how to generate random events distributed according to a probability distribution $p(x)$, utilizing a random number generator uniform on $[0, 1]$.

In the next lecture, we'll get into how to do this. For the rest of this lecture, I want to spend some time discussing one of the most extensive applications of Monte Carlo methods: simulation of proton-proton collision events at the Large Hadron Collider experiment in Geneva, Switzerland.

The Large Hadron Collider (LHC) is the largest physics experiment ever created. It consists of a 27 km (18 mile) ring in which protons are accelerated to kinetic energies of 6.5 tera-electron volts (TeV), which is about 7000 times the rest mass energy of the proton! This is roughly the kinetic energy of a flying mosquito, but contained in a single proton. The two oppositely circulating beams are made to collide head-on at so-called interaction points on the ring. At these interaction points, enormous experiments have been constructed to detect and measure all particles that are produced in the collision. The two largest experiments are ATLAS (A Toroidal LHC ApparatuS) and CMS (Compact Muon Solenoid). Each of these experiments are roughly the size and weight of 5 story buildings! The ATLAS and CMS experimental collaborations each have over 3000 members, by far the largest experimental groups ever. And all of this to attempt to understand the structure of the proton, and by extension nature, at the highest energies and shortest distances ever probed.

So what's going on there? I will discuss a greatly simplified picture of the collisions of protons, but I hope it will illustrate the complexity and necessity of Monte Carlo tools. Each collision of a pair of protons is an event: one instance of some probability distribution that governs how protons interact. Protons, to good approximation, are bags of more fundamental quarks and gluons which directly interact in collision. We look inside the protons using

Figure 4: A view of Geneva, Switzerland, from the Jura Mountain range to the west. The city of Geneva is located at the end of Lake Geneva, the ring of the Large Hadron Collider is illustrated in yellow, the runway of the Geneva airport is visible as a size reference, and Mont Blanc is present off in the distance. Photo Credit: CERN.

the "Neanderthal method": we smash things together and determine what was inside by what comes out (cf. to opening presents on your birthday). A huge number of particles are produced in these collisions; things called pions, kaons, muons, and familiar protons, photons, and electrons. Schematically, we might illustrate one such event as in figure 6.

Arrows pointing toward the center represent the colliding protons while arrows pointing away represent the particles that are produced. I have just illustrated one of a nearly infinite number of possible collections of produced particles (called "final states"). Also, these particles can have any relativistic momentum, only subject to momentum conservation. Additionally, it is the ATLAS and CMS experiments that measure properties of the final state particles (like their charge, energy, etc.). They do not have little labels on them. Any experiment is imperfect: energies are not measured arbitrarily accurately; angles are smeared by finite resolution; there are cables that are in the way of equipment; etc. With all of these challenges, how are we able to make predictions for the final state of LHC collisions?

This problem is nearly ideal for a Monte Carlo solution for the following reasons:

1. In proton-proton collisions, there are essentially an infinite number of possible outcomes. While there is a fundamental theory underlying proton collision physics (the Standard Model), it is amazingly complex and there is no known way (and perhaps

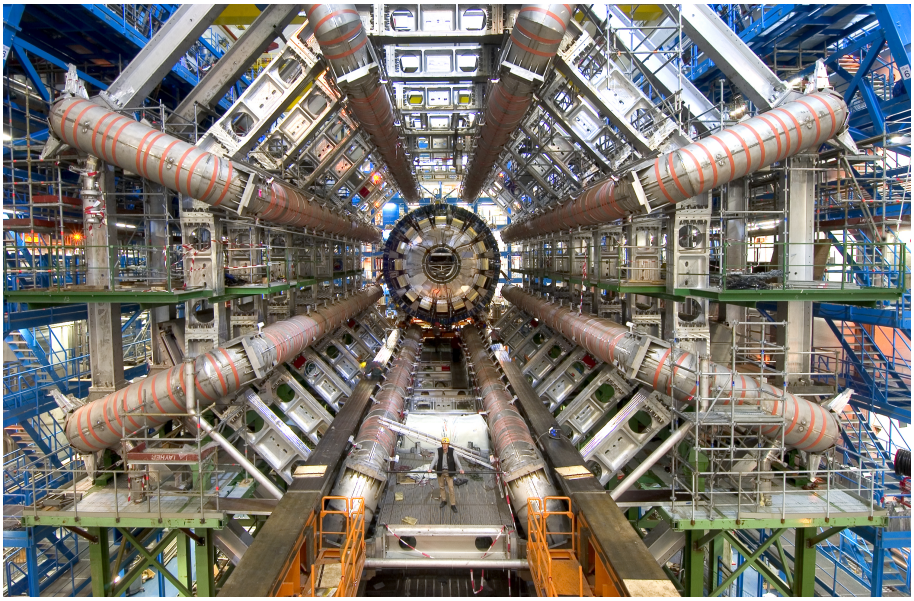

Figure 5: An image of the ATLAS experiment during its construction. Notice the human for scale; the large piping located around the experiment houses currents that source toroidal electromagnets. Photo Credit: CERN.

no way whatsoever) to achieve exact, analytic solutions for probability distributions. By incorporating the physics of the Standard Model, we are able to construct Monte Carlo simulations that produce final states faithfully according to their probability to occur.

2. Monte Carlo also enables for realistic outcomes to be simulated. We not only need the types of particles produced in the final state, but also their momentum, which is a series of four real numbers. This is essentially impossible with a large number of particles in the final state. However, to good approximation, we are able to assume that the production of particles is a Markov process: that is, the production probability of additional particles only depends on what particles currently exist. This enables a simple recursive, iterative algorithm to be able to generate arbitrary final states.

3. Detectors and experiments are imperfect, so we also need to model the response of the equipment to particular particles with certain momentum to make realistic pseudo-data. Because the Monte Carlo produces individual pseudoevents, it is straightforward to "measure" the final state with a detector simulation. The results of the pseudoexperiments can then be compared to real data to test hypotheses or to discover new particles and forces of nature!

The most widely used Markov Chain Monte Carlo simulators go by names like "Pythia" and "Herwig" and are used ubiquitously to understand the data from ATLAS and CMS.

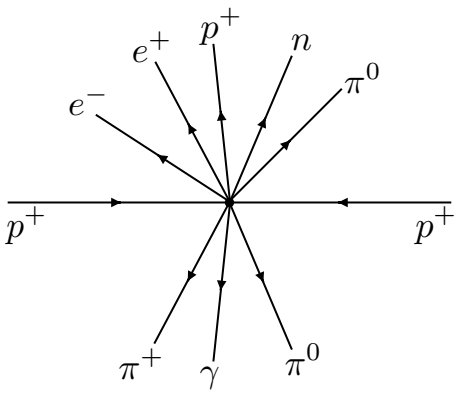

Figure 6: Schematic representation of a proton-proton collision event. Note charge conservation.

2 Lecture 2

Last lecture, we introduced the idea of Monte Carlo methods and the need for simulation of pseudodata. This becomes important for modeling effects of finite data samples, finite experimental resolution, or for understanding systems for which the analytic probability distribution is unknown or even unknowable. In this lecture, we discuss in detail the mathematical foundation for Monte Carlo methods.

With the basis from previous lecture, we assume that we have a (known) probability distribution $p(x)$ and can efficiently sample (pseudo)random real numbers on $[0, 1]$. How do we generate a set of numbers $\{x\}$ distributed according to $p(x)$?

For simplicity, let's assume that $p(x)$ is defined on $x \in [0, 1]$ so that

$$\int_0^1 dx\, p(x) = 1\,. \tag{5}$$

The results that follow are true for arbitrary probability distributions; just confining ourselves to $x \in [0, 1]$ is for compactness. The probability for a value to lie within dx of x is $p(x)\, dx$. Assuming the inverse exists, we can make the change of variables to y as:

$$p(x)\, dx = p(x(y))\frac{dx}{dy}dy\,, \tag{6}$$

where $x(y)$ is the change of variables. This then defines a new probability distribution:

$$\bar{p}(y) = p(x(y))\frac{dx}{dy}\,. \tag{7}$$

Now, if we assume that this change of variables renders $\bar{p}(y)$ uniform on $y \in [0, 1]$, then

$$p(x(y)) = \frac{dy}{dx} \quad \Rightarrow \quad y = \int_0^x dx'\, p(x') + c\,. \tag{8}$$

c is an integration constant which is 0 because $p(x)$ is normalized.

The integral

$$\int_0^x dx'\, p(x') \tag{9}$$

is just the cumulative distribution, or the probability that the random variable is less than x. We will denote it by

$$\Sigma(x) = \int_0^x dx'\, p(x')\,. \tag{10}$$

Note that $\Sigma(0) = 1$, $\Sigma(1) = 1$, and $\Sigma(x)$ is monotonic on $x \in [0, 1]$. Because it is monotonic, $\Sigma(x)$ can always be inverted. Now, we can solve for x:

$$x = \Sigma^{-1}(y)\,. \tag{11}$$

That is, the random variable x distributed as $p(x)$ can be generated by inverting the cumulative distribution and using the uniformly distributed $y \in [0, 1]$ as the independent variable.

The fundamental step for this method of sampling a distribution is generating random numbers distributed on $[0, 1]$. The following Mathematica function `rand` outputs an array of random numbers distributed on $[0, 1]$ using its internal random number generator:

```
rand[Nev_]:=Module[{randtab,r},
randtab={};
For[i = 1, i ≤ Nev, i++,
r = RandomReal[{0,1}];
AppendTo[randtab,r];
];
Return[randtab];
];
```

This is the central result for Monte Carlo simulation. "Any" (invertible, closed form) probability distribution can be sampled from a uniform distribution. Let's see how this works in a couple of examples.

Example.

Let's consider a quantum particle in the infinite square well. The probability distribution for a the position of a particle in the ground state of a square well on $x \in [0, 1]$ is

$$p(x) = 2\sin^2(\pi x)\,. \tag{12}$$

Note that this is normalized. The cumulative distribution is

$$\Sigma(x) = \int_0^x dx'\, p(x') = x - \frac{\sin(2\pi x)}{2\pi}\,. \tag{13}$$

Then, to sample this distribution, we set the cumulative distribution equal to y, which is uniformly distributed on $[0, 1]$:

$$y = x - \frac{\sin(2\pi x)}{2\pi}\,. \tag{14}$$

This isn't solvable for x in closed form, but for a given $y \in [0, 1]$, there exists a unique value of x that can be determined numerically (using Newton's method, for example, to find roots).

Example.

Let's consider the probability distribution of the time it takes a radioactive element to decay. Radioactive decay of an atom is governed by its half-life, λ. At time $t = \lambda$, there is a 50% chance that the atom has decayed. Equivalently, given a sample of material composed of an unstable element, 50% of it will have decayed by $t = \lambda$. The probability distribution of the time t it takes one of these unstable atoms to decay is

$$p(t) = \frac{\log 2}{\lambda} e^{-\frac{t}{\lambda}\log 2}\,, \tag{15}$$

where the time $t \in [0, \infty)$. That is, the atom can decay any time after we start watching (at time $t = 0$). Note that the ratio of the probability at time $t = \lambda$ to $t = 0$ is $1/2$:

$$\frac{p(t = \lambda)}{p(t = 0)} = e^{-\log 2} = \frac{1}{2}\,. \tag{16}$$

That is, the likelihood that the atom is still there at time $t = \lambda$ is 50%.

Now, we would like to be able to sample this distribution via a Monte Carlo. So, we first determine the cumulative distribution:

$$\Sigma(t) = \int_0^t dt'\, p(t') = 1 - e^{-\frac{t}{\lambda}\log 2}\,, \tag{17}$$

which is 0 at $t = 0$ and 1 at $t = \infty$, and is monotonic. Next, we set this equal to a uniformly distributed $y \in [0, 1]$ and invert, solving for t. We find

$$t = -\frac{\lambda}{\log 2}\log(1 - y)\,. \tag{18}$$

Note that for $y \in [0, 1]$, this time ranges over $t \in [0, \infty)$.

The following Mathematica function `expdist` samples an exponential distribution with half-life λ and outputs a histogram of the resulting distribution. Its arguments are λ, the number of events `Nev`, the end-point of the histogram `maxpoint`, and the number of bins in the histogram `Nbins`:

```
expdist[λ_,Nev_,maxpoint_,Nbins_]:=Module[{randtab,r,t,binno},
randtab=Table[{maxpoint*i/Nbins,0.0},{i,0,Nbins-1}];
For[i = 1, i ≤ Nev, i++,
r = RandomReal[{0,1}];
t = -λ*Log[1-r]/Log[2];
If[t < maxpoint,
binno=Ceiling[Nbins*t/maxpoint];
randtab[[binno,2]]+=Nbins/(maxpoint*Nev);
];
];
Return[randtab];
];
```

A plot of the output histogram is compared to the analytic function expression for the distribution in figure 7. Here, we use $\lambda = 1$s, `Nev` $= 10000$, `maxpoint` $=$ 7s, and `Nbins` $= 40$.

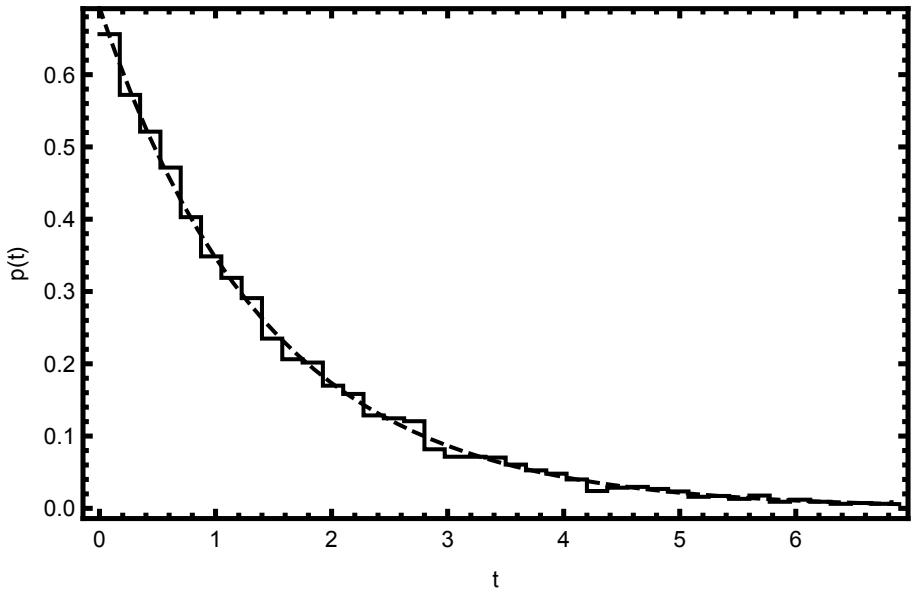

Figure 7: A plot comparing the output of the program `expdist` to the analytic expression for the exponential probability distribution with half-life $\lambda = 1$s, $p(t) = \frac{\log 2}{\lambda} \exp[-\frac{t}{\lambda} \log 2]$. To make the plot of the histogram, we set `InterpolationOrder` $\rightarrow$ `0`, which gives the histogram the stair step-like shape. The histogram and the plot of $p(t)$ lie on top of one another.

So, we know how to sample a given probability distribution. What if, however, we instead want to simulate some process, where each step in the process is some probability distribution itself? In the case of the unstable element, let's imagine that we have a large sample of it,

and we want to determine the number of atoms that decay in a given time interval $t \in [0, T]$. How do we do this? For this problem and many others in physics, it can be expressed as a Markov Chain, which we will explore now.

To a good approximation, radioactive decays in a sub-critical sample are independent of one another. So, the probability of any atom decaying is just the exponential distribution. The exponential distribution is also "memoryless": it is independent of when you start the clock (and appropriately normalize).[2] So, to determine how many atoms decay in a time T, we have the simple procedure:

1. Determine the time that the first atom decayed t_1 from

$$t_1 = -\frac{\lambda}{\log 2} \log(1 - y) \,, \tag{19}$$

   for y uniform on $[0, 1]$. If $t_1 > T$, then stop; no atoms decayed in time T. If $t_1 < T$, then continue.

2. Determine the time (from $t = 0$) that the second atom decayed t_2 from

$$t_2 - t_1 = -\frac{\lambda}{\log 2} \log(1 - y) \,, \tag{20}$$

   again with y uniform on $[0, 1]$. If $t_2 > T$, then stop; 1 atom decayed in time T. If $t_2 < T$, then continue.

3. Determine $t_3, t_4, \ldots$, and continue until $t_n > T$. Then, $n - 1$ atoms decayed.

This procedure required a couple of things to work as simply as it did. First, at every step the probability distribution that we sampled was identical (just the exponential distribution). Also, the time of the next decay only depended on the time of the immediate previous decay (and not on all previous decays).

Such an iterative procedure that samples a distribution recursively and each step only depends on the immediate previous step is a Markov Chain. What we constructed above would properly be called a Markov Chain Monte Carlo (MCMC). Oh, by the way, this was really overkill for this problem. The distribution for the number of decays in time T is just the Poisson distribution.

Okay, we have a nice theory for how to sample a 1D probability distribution. How do we do this for higher dimensional distributions? Let's consider 2D for concreteness; the method we develop there will easily generalize. Let's consider a 2D probability distribution $p(x, y)$ for $(x, y) \in [0, 1]$. To do a similar thing as in 1D, we need to calculate the cumulative distribution and then invert. The cumulative distribution is

$$\Sigma(x, y) = \int_0^x dx' \int_0^y dy' \, p(x', y') \,, \tag{21}$$

[2] We're simplifying things here a bit to describe the method. We're making the somewhat silly assumption that the atoms decay 1 at a time.

but it is not clear at all how to invert this. How do you uniquely invert a function of two variables into two other variables? To proceed, we need another tactic.

Instead, if we are able to express $p(x, y)$ as the product of two 1D probability distributions, then we can apply our methods from 1D to 2D distributions. We might think that we could write

$$p(x, y) = p(x)p(y)\,, \tag{22}$$

but this is not true in general. Instead, we utilize the definition of a conditional probability:

$$p(x|y) = \frac{p(x, y)}{p(y)}\,, \tag{23}$$

where we define

$$p(y) = \int_0^1 dx\, p(x, y)\,. \tag{24}$$

We read $p(x|y)$ as "probability of x given y", and the conditional probability is a 1D probability distribution itself. For $p(x|y)$, y is a fixed parameter, and not a random variable. To verify that it is normalized we have:

$$\int_0^1 dx\, p(x|y) = \int_0^1 dx \frac{p(x, y)}{p(y)} = \frac{1}{p(y)} \int_0^1 dx\, p(x, y) = 1\,. \tag{25}$$

Using the conditional probability, we can then express a 2D distribution as a product of two 1D distributions:

$$p(x, y) = p(y)p(x|y) = p(x)p(y|x)\,. \tag{26}$$

Note that this representation is symmetric in x and y.

Then, to sample the 2D distribution $p(x, y)$, we first choose a random y according to $p(y)$ and a Monte Carlo for it. Then, given that value of y, we choose x according to $p(x|y)$. This produces a pair (x, y) chosen according to $p(x, y)$, and we can repeat to generate a large sample of pseudodata. Let's see how this works in an example.

Example.

Let's consider the 2D distribution

$$p(x,y) = x + y\,, \tag{27}$$

where $(x,y) \in [0,1]$. Note that this is normalized:

$$\int_0^1 dx \int_0^1 dy\,(x+y) = 1\,, \tag{28}$$

and that the probability cannot be split up into a product: $p(x,y) \neq p(x)p(y)$. To sample this distribution via a Monte Carlo, we therefore must first find conditional probabilities. Let's find $p(y)$. We have

$$p(y) = \int_0^1 dx\,(x+y) = y + \frac{1}{2}\,. \tag{29}$$

Then, the probability of x given y is

$$p(x|y) = \frac{p(x,y)}{p(y)} = \frac{x+y}{y+\frac{1}{2}}\,. \tag{30}$$

So, we break up the 2D distribution as

$$p(x,y) = x + y = \left(y + \frac{1}{2}\right)\left(\frac{x+y}{y+\frac{1}{2}}\right)\,. \tag{31}$$

To then Monte Carlo this problem, we need to find the corresponding 1D cumulative distributions:

$$\Sigma(y) = \int_0^y dy'\left(y' + \frac{1}{2}\right) = \frac{y(1+y)}{2}\,. \tag{32}$$

$$\Sigma(x|y) = \int_0^x dx'\left(\frac{x+y}{y+\frac{1}{2}}\right) = \frac{x(2y+x)}{2y+1}\,. \tag{33}$$

Let's then generate two random numbers r_1, r_2 uniform on $[0,1]$, and set these equal to the cumulative distributions and invert. That is,

$$\Sigma(y) = r_1 \quad \Rightarrow \quad y = -\frac{1}{2} + \sqrt{\frac{1}{4} + 2r_1}\,, \tag{34}$$

$$\Sigma(x|y) = r_2 \quad \Rightarrow \quad x = -y + \sqrt{y^2 + (2y+1)r_2}\,. \tag{35}$$

A Mathematica program `xygen` that generates pairs of random numbers (x,y) distributed according to $p(x,y) = x + y$ is provided below. The argument of `xygen` is the number of events that you want to generate.

```
xygen[Nev_]:=Module[{outtab,r1,r2,x,y},
outtab={};
For[i = 1, i ≤ Nev, i++,
r1 = RandomReal[{0,1}];
r2 = RandomReal[{0,1}];
y = -1/2 + √(1/4 + 2*r1);
x = -y + √(y*y + (2*y+1)*r2);
AppendTo[outtab,{x,y}];
];
Return[outtab];
];
```

3 Lecture 3

Last lecture, we developed the mathematical foundation for Monte Carlos, Markov Chains and their utility, and how to sample multi-dimensional distributions. Everything developed last lecture, however, required analytic calculations to invert cumulative probability distributions. At the very least, efficient numerical methods were required to invert cumulative probability distributions. In this lecture, we will discuss how to relax this assumption and efficiently sample generic distributions.

Let's just assume that the probability distribution $p(x)$ exists on the domain $x \in [0,1]$ and we want to sample it. It might not even have an analytic expression, but just be expressed as a collection of points. For this general case, there's no way to invert $p(x)$ as we did before, so we need another technique. What we will do is bound the probability distribution $p(x)$ that we want by an integrable, normalizable function $q(x)$ that we can sample efficiently and simply. If we find a $q(x)$ such that

$$q(x) \geq p(x)\,, \tag{36}$$

for all $x \in [0,1]$, then we can use the probability distribution derived from $q(x)$ to sample $p(x)$. Let's see how this works.

To be concrete, let's take $q(x)$ constant on $x \in [0,1]$. With the maximum value of $p(x)$

$$p_{\max} = \max_x p(x) < \infty\,, \tag{37}$$

we can set $q(x) = p_{\max}$ so that $q(x) \geq p(x)$ for all $x \in [0,1]$. We can turn $q(x)$ into a probability distribution by normalizing; we will call this probability distribution $\tilde{q}(x)$:

$$\tilde{q}(x) = 1\,. \tag{38}$$

Note then that

$$p_{\max} \cdot \tilde{q}(x) \geq p(x)\,. \tag{39}$$

Because we are able to easily generate uniform random numbers on $x \in [0,1]$, we are able to generate a function that bounds $p(x)$ everywhere. So, how to we sample $p(x)$? We can

generate additional uniform random numbers to veto those points that are inconsistent with $p(x)$. This introduces a loss of efficiency, but we will discuss later how to improve this.

The procedure for sampling $p(x)$ given the bounding function $q(x) = p_{\max}$ is the following:

1. Choose a random number uniformly on $x \in [0, 1]$.

2. Determine the maximum value of $p(x)$ on $x \in [0, 1]$; call it $p_{\max}$.

3. At the chosen random x, keep the event with probability

$$p_{\text{keep}} = \frac{p(x)}{p_{\max}} \in [0, 1] \,. \tag{40}$$

   This can be accomplished by choosing another uniform random number $y \in [0, 1]$ and vetoing (=removing) the event if

$$y > \frac{p(x)}{p_{\max}} \,. \tag{41}$$

4. The events that remain will be distributed according to $p(x)$. To normalize the distribution to integrate to 1, we multiply by $p_{\max}$.

Let's see how this works in an example.

Example.

Consider again the probability distribution of the position of the particle in the ground state of the infinite square well:

$$p(x) = 2 \sin^2(\pi x) \tag{42}$$

for $x \in [0, 1]$. Note that the maximum value of $p(x)$ is $p_{\max} = 2$. So, we find x by choosing a random number on $[0, 1]$ and then keep that x with probability

$$p_{\text{keep}} = \frac{p(x)}{p_{\max}} = \sin^2(\pi x) \in [0, 1] \,. \tag{43}$$

This never required inverting cumulative probability distributions.

A Mathematica function `vetowell` that implements this veto method for sampling probability distributions is as follows. `vetowell` has two arguments: the number of events to generate `Nev`, and the number of bins to put in the histogram `Nbins`.

```
vetowell[Nev_,Nbins_]:=Module[{outtab,r1,r2,binno},
outtab=Table[{i/Nbins,0.0},{i,0,Nbins-1}];
For[ i = 1, i ≤ Nev, i++,
r1=RandomReal[{0,1}];
r2=RandomReal[{0,1}];
binno=Ceiling[r1*Nbins];
If[r2 < Sin[π*r1]^2, outtab[[binno,2]] += 2.0*Nbins/Nev];
];
Return[outtab];
];
```

A plot of the output histogram from `vetowell` is plotted in figure 8. To make the plot, we have generated 1000000 events and used 100 bins on $x \in [0, 1]$.

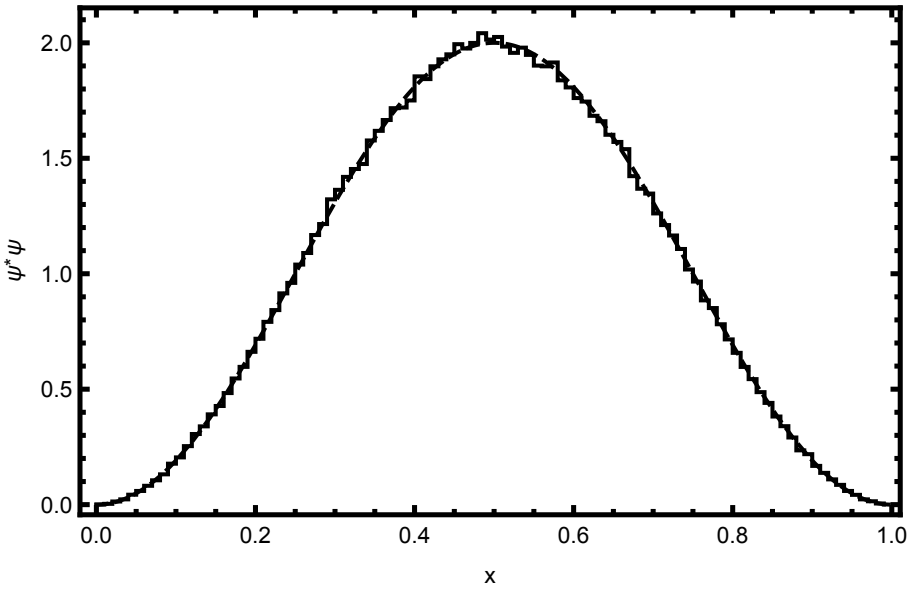

Figure 8: A plot comparing the output of the program `vetowell` to the analytic expression for the probability distribution of the position of the particle in the ground state of the infinite square well. To make the plot of the histogram, we set `InterpolationOrder` $\rightarrow$ `0`, which gives the histogram the stair step-like shape. The histogram and the plot of $p(t)$ lie on top of one another.

Note that $\sin^2(\pi x) \rightarrow 0$ if $x \rightarrow 0$ or $x \rightarrow 1$, and so step 3 of this algorithm becomes very inefficient (most events are vetoed) in these regions. Efficiency can be regained by bounding the probability distribution $p(x)$ more strongly by a non-uniform function $q(x)$. Let's see this in an example.

Example.

Again, let's consider the ground state of the infinite square well, where

$$p(x) = 2\sin^2(\pi x)\,. \tag{44}$$

Now, consider the function

$$q(x) = x(1-x)\,, \tag{45}$$

on $x \in [0,1]$. Note that with $p_{\max} = 2$ and $q_{\max} = 1/4$, we have

$$8q(x) \geq p(x)\,, \tag{46}$$

for all $x \in [0,1]$. The probability distribution corresponding to the function $q(x)$ is

$$\tilde{q}(x) = 6x(1-x)\,. \tag{47}$$

Then, given Eq. 46, we can generate events according to the probability distribution $\tilde{q}(x)$ and then veto appropriately. The cumulative distribution of $\tilde{q}(x)$ is

$$\Sigma^q(x) = 3x^2 - 2x^3\,. \tag{48}$$

By inverting this cumulative distribution, we can sample $\tilde{q}(x)$ via Monte Carlo. Then, given some x from the $\tilde{q}(x)$ distribution, we keep that event with probability equal to

$$p_{\text{keep}} = \frac{p(x)}{8q(x)} = \frac{\sin^2(\pi x)}{4x(1-x)} \in [0,1]\,. \tag{49}$$

The events that remain after this veto step are distributed according to $p(x)$. To properly normalize the distribution, at the end we must multiply by $p_{\max}$ and divide the histogram entries by the maximum value of $\tilde{q}(x)$.

A very reasonable question to ask, both for these pseudodata applications as well as for evaluating integrals, is what the accuracy of Monte Carlo is. To be concrete, we will just answer the following question: what is the uncertainty on the probability to be near x, as the number of Monte Carlo events $N \to \infty$? That is, we want to study the Monte Carlo approximation to

$$P = p(x)\,dx\,. \tag{50}$$

In Monte Carlo, this is approximated by the ratio of the number of events in the bin near the point x, N_x, divided by the total number of events in the distribution, N. By the law of large numbers, this converges to P as $N \to \infty$:

$$\lim_{N\to\infty} N_x = PN\,. \tag{51}$$

What is the variance of this? By our Monte Carlo, all events in the bin are independent of one another and each contributes 1 to the bin. The rate at which any event populates this bin is fixed and equal to P. These criteria uniquely define the Poisson distribution for the number of events in the bin. This is actually straightforward to derive, which we will do now.

Derivation of Poisson Distribution

We want to determine the probability distribution for the number of events in a bin of a histogram that was filled via a Monte Carlo. The probability distribution that we are sampling is $p(x)$ and the probability P for an event to be in the bin near x is

$$P = p(x)\,dx\,. \tag{52}$$

The probability for an event to not be in the bin is then $1 - P$. All events in our pseudodata sample are independent. We therefore determine the probability that there are k events in the bin near x is:

$$p_k = P^k(1-P)^{N-k}\binom{N}{k}\,. \tag{53}$$

This probability is called the binomial distribution for the following reason. The factor on the right,

$$\binom{N}{k} = \frac{N!}{k!(N-k)!}\,, \tag{54}$$

is read as "N choose k" and is the number of ways to pick k objects out of a set of N total objects. The probability in Eq. 53 is called the binomial distribution because the sum over all k corresponds to the binomial expansion:

$$\sum_{k=0}^{N} p_k = \sum_{k=0}^{N} P^k(1-P)^{N-k}\binom{N}{k} = \left(P + (1-P)\right)^N = 1\,. \tag{55}$$

Now, we could stop here, but we can simplify the probability by making a few more assumptions. We will assume that the width of the bin is very small so that $P \ll 1$ and $k \ll N$. In this limit, note that

$$\binom{N}{k} = \frac{N!}{k!(N-k)!} \simeq \frac{N^k}{k!}\,, \tag{56}$$

and $(1-P)^k \simeq 1$, for finite k. With these simplifications, the probability for k events in a bin becomes

$$p_k \to \frac{(NP)^k}{k!}\left(1 - \frac{NP}{N}\right)^N = \frac{(NP)^k}{k!}e^{-NP}\,, \tag{57}$$

where the equality holds in the $N \to \infty$ limit. This probability distribution is called the Poisson distribution. Note that it is normalized:

$$\sum_{k=0}^{\infty} \frac{(NP)^k}{k!}e^{-NP} = 1\,, \tag{58}$$

has mean $\langle k\rangle = NP$ and variance $\sigma^2 = \langle k^2\rangle - \langle k\rangle^2 = NP$.

Therefore, with N events in a sample and probability P for any of those events to be in a particular bin, the average number of events in the bin is NP and the variance is also NP. Therefore, the standard deviation (the square-root of the variance) on the number of events in the bin scales like $\sqrt{N}$. The relative error is defined by the ratio of the standard deviation to the mean. This scales like $1/\sqrt{N}$, and so for N events, we expect an error that scales like $1/\sqrt{N}$. Note that this is independent of bin size and the number of dimensions in which the probability is defined. Monte Carlo and related methods are among the most efficient way to sample high dimensional distributions.

Later in the class, we will discuss ways to even improve this $1/\sqrt{N}$ relative error by changing the way that random numbers are generated.

Finally, I want to discuss Monte Carlo generation of non-integrable distributions. Such distributions are not probability distributions as they cannot be normalized. For example, the function

$$f(x) = \frac{1}{x}\,, \tag{59}$$

on $x \in [0,1]$ has undefined (infinite) integral. Such infinite distributions are unphysical; no measurement would ever yield ∞. However, depending on assumptions and your predictive ability, one might find infinite distributions in intermediate steps of a calculation.

For example, one might attempt to calculate in degenerate perturbation theory of quantum mechanics and find at order n in perturbation theory:

$$f_n(x) = \frac{(-1)^n}{x}\frac{\log^{2n} x}{n!}\,. \tag{60}$$

The full distribution is a sum over n:

$$\sum_{n=0}^{\infty} f_n(x) = \frac{e^{-\log^2 x}}{x}\,, \tag{61}$$

which is finite and well-defined.

There are two simple ways to sample non-integrable distributions. The first is to just artificially cut off the domain. For example, for $1/x$, just stop at $\epsilon > 0$:

$$f(x) = \frac{1}{x}\,, \tag{62}$$

for $x \in [\epsilon, 1]$, which is normalizable. One can then do any of the techniques that we developed.

Another way is to throw out the data/event interpretation of the Monte Carlo. For $1/x$, we can generate a uniform random variable $x \in [0,1]$. Then, we can fill the bin near x by an amount equal (or proportional to) $1/x$. This will then reproduce the $f(x) = 1/x$ distribution, but because each event does not contribute just 1 to the distribution, we can't easily give it a probabilistic data interpretation.

This is but a tiny introduction to Monte Carlo methods.