Exact sampling

The pinned steppingstone model

Recall the pinned stepping stone model I discussed at the end of the last lecture: We have a 21-by-20 cylinder, where each square is initially colored 0 or 1, all the squares on the upper edge are pinned at 0, all the squares on the lower edge are pinned at 1, and at each step, we choose a non-pinned square at random and change the color of that square to the color of one of the square's 4 neighbors.

The pinned stepping stone model, pruned of its transient states, is ergodic and has a unique stationary measure π.

Here's a typical state, sampled in accordance with the stationary distribution π, or rather a distribution $\pi_{1,000,000}$ that is extremely close but not equal to π:

```
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000010
00000000000000001110
00000000000000000110
00000000000000001110
11110000000000001110
11110000000000000110
10011110000000000100
10111101000000001111
10001101100000000111
10001100100100010111
10001100101101111111
11111100011111111111
11111111011111111111
11111111111111111111
```

Coupling from the past: an example

(graphics courtesy of David Einstein)

How might we sample from π?

One way is to do ordinary pseudorandom simulation for a large, predetermined number of steps (a million, say, as was done for the preceding figure).

Eventually, the color of a square will be determined not by its initial coloring, nor by the initial coloring of any of the non-pinned squares, but solely by our (randomized) choices to change the color of THIS square to the color of THAT neighbor, whose color was earlier changed to the color of one of ITS neighbors, etc., eventually leading back to one of the pinned squares.

Why?

..?..

Look at the Markov chain in which each square *s* is assigned not just a color at time *n*, but a "reason for being that color at time *n*".

This "reason" is just another square, namely, the square *s** (which could be *s* itself) such that our sequence of recolorings forces the color of *s* at time *n* to be the color of *s** at time 0.

If it ever happens that for every non-pinned square *s*, *s** is one of the pinned squares, then this property will hold forever after.

Consider for instance the 1-dimensional pinned steppingstone model from the final problem of the final homework assignment, where site 1 is pinned to color 0 (red) and site 5 is pinned to color 1 (blue). Imagine rolling a 6-sided die with faces are marked "2←1", "2←3", "3←2", "3←4", "4←3", and "4←5", signifying "give cell 2 whatever color cell 1 currently has", "give cell 3 whatever colors cell 2 currently has", etc. Suppose the first 8 die-rolls are

```
{"2←1", "2←3", "3←2", "3←4", "4←3", "4←5"}[[RandomInteger[{1, 6}]]]
```

3←2

```
{"2←1", "2←3", "3←2", "3←4", "4←3", "4←5"}[[RandomInteger[{1, 6}]]]
```

2←1

```
{"2←1", "2←3", "3←2", "3←4", "4←3", "4←5"}[[RandomInteger[{1, 6}]]]
```

3←2

```
{"2←1", "2←3", "3←2", "3←4", "4←3", "4←5"}[[RandomInteger[{1, 6}]]]
```

2←3

```
{"2←1", "2←3", "3←2", "3←4", "4←3", "4←5"}[[RandomInteger[{1, 6}]]]
```

3←4

```
{"2←1", "2←3", "3←2", "3←4", "4←3", "4←5"}[[RandomInteger[{1, 6}]]]
```

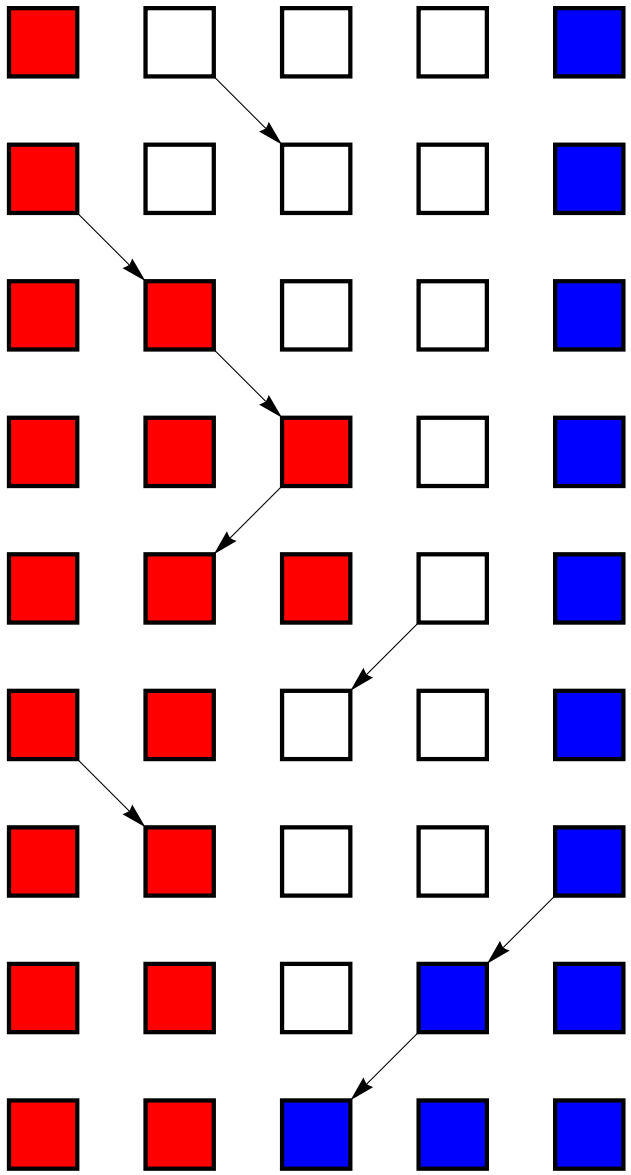
2←1

```
{"2←1", "2←3", "3←2", "3←4", "4←3", "4←5"}[[RandomInteger[{1, 6}]]]
```

4←5

```
{"2←1", "2←3", "3←2", "3←4", "4←3", "4←5"}[[RandomInteger[{1, 6}]]]
```

3←4



Check that at this stage, the colors of all five cells are determined (even if we don't know what the colors of the three middle cells were at the start), and that subsequent randomizations may change the coloring but won't affect the fact that **the coloring can be deduced from the randomizations alone** (without knowledge of the initial coloring).

Most of the time, we don't have to do too many randomizations for this effect to kick in.

More importantly, if we're doing a long run of N recolorings, and it happens that the last n of these recolorings (with $n < N$) have the property that they force every square to have the color of one of the pinned squares, then the result of applying the N recolorings to any initial coloring is the same as the result of applying just the n recolorings to any initial coloring; earlier recolorings get "washed out".

So one sneaky way to find out the outcome of an N-step pseudorandom simulation with N huge, without having to do anywhere close to N steps of actual simulation, is to **guess** an n and just simulate the last n steps of the simulation.

If n is large enough, then the last n recolorings will wash out everything that came before, so applying these n recolorings to the initial coloring (or indeed ANY coloring) will give the same result as applying all N recolorings to the initial coloring.

If n was not large enough, try again with a larger n, or rather, keep going. Eventually, you'll find an n that works, though if you're really unlucky it might be N itself.

For instance, suppose the last 12 dice-rolls looks like this:

```
{"2←1", "2←3", "3←2", "3←4", "4←3", "4←5"}[[RandomInteger[{1, 6}]]]
```

3←4

```
{"2←1", "2←3", "3←2", "3←4", "4←3", "4←5"}[[RandomInteger[{1, 6}]]]
```

4←5

```
{"2←1", "2←3", "3←2", "3←4", "4←3", "4←5"}[[RandomInteger[{1, 6}]]]
```

4←5

```
{"2←1", "2←3", "3←2", "3←4", "4←3", "4←5"}[[RandomInteger[{1, 6}]]]
```

2←3

```
{"2←1", "2←3", "3←2", "3←4", "4←3", "4←5"}[[RandomInteger[{1, 6}]]]
```

2←1

```
{"2←1", "2←3", "3←2", "3←4", "4←3", "4←5"}[[RandomInteger[{1, 6}]]]
```

4←5

```
{"2←1", "2←3", "3←2", "3←4", "4←3", "4←5"}[[RandomInteger[{1, 6}]]]
```

2←1

```
{"2←1", "2←3", "3←2", "3←4", "4←3", "4←5"}[[RandomInteger[{1, 6}]]]
```

2←3

```
{"2←1", "2←3", "3←2", "3←4", "4←3", "4←5"}[[RandomInteger[{1, 6}]]]
```

4←3

```
{"2←1", "2←3", "3←2", "3←4", "4←3", "4←5"}[[RandomInteger[{1, 6}]]]
```

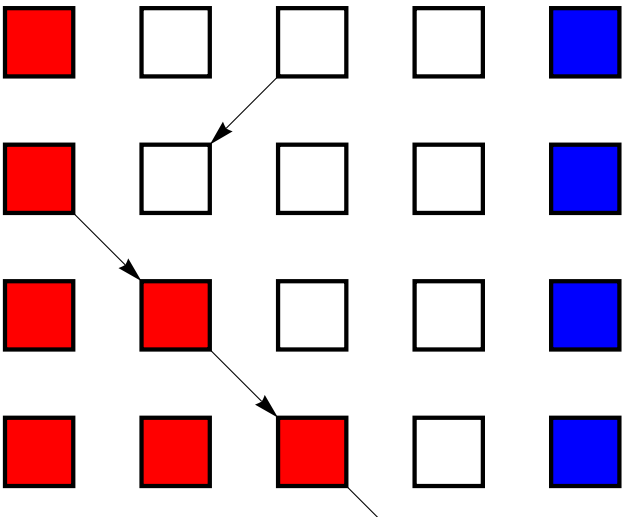
3←2

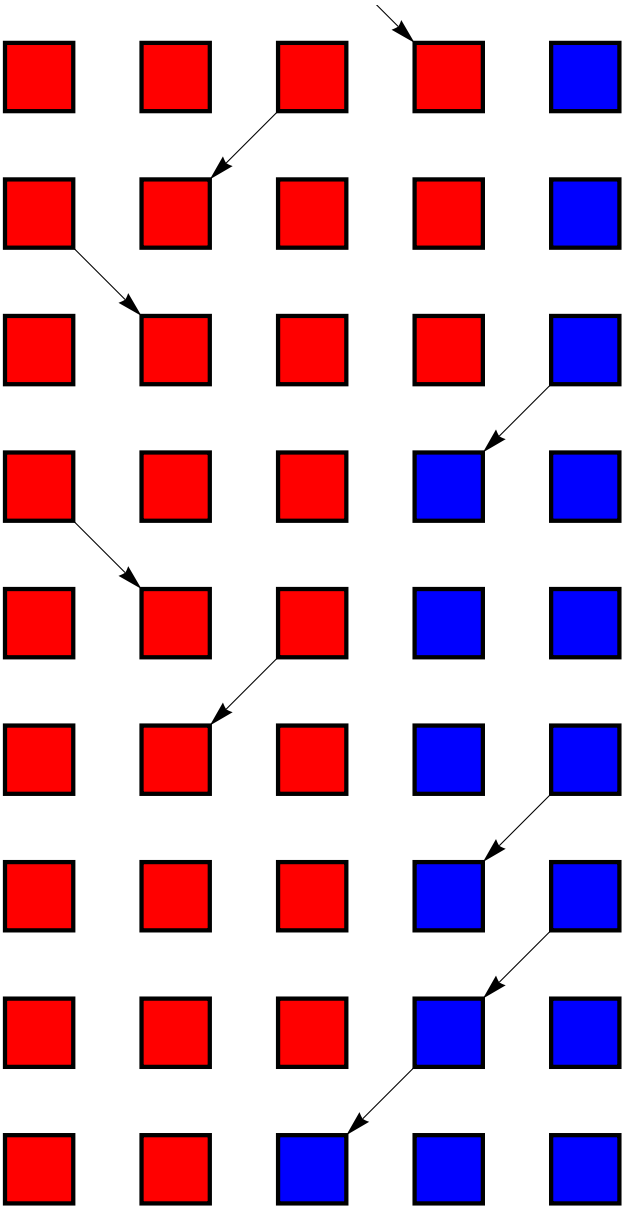
```
{"2←1", "2←3", "3←2", "3←4", "4←3", "4←5"}[[RandomInteger[{1, 6}]]]
```

2←1

```
{"2←1", "2←3", "3←2", "3←4", "4←3", "4←5"}[[RandomInteger[{1, 6}]]]
```

2←3





Then we don't need to know what the preceding $N-12$ dice-rolls were to know what the final coloring is.
Note that as N grows, the typical size of the n that works does not grow; so using this trick to sneakily simulate a trillion recolorings doesn't take much more work on average than using the trick to sneakily simulate a billion recolorings (it only requires a willingness to do more work in very, very, very rare cases).
In fact, there's a way to do this sneaky trick so that N is effectively infinite, and instead of getting an N-step approximation to π, you get π itself.

This is the method of simulation from the past, more commonly called "coupling from the past" or CFTP; see Chapter 22 of Levin, Peres, and Wilmer.

Importance sampling

Knuth's trick

(this description is taken from chapter 17 of Engel)

General problem: How can we count the leaves of a finite tree?

Sample application: How can we count the self-avoiding paths from (0,0) to (10,10) in $\{(i,j): 1 \le i,j \le 10\}$?

Let the nodes at level n be the self-avoiding lattice paths of length n that start at (0,0) and either end at (10,10) or can be extended to give self-avoiding lattice paths that end at (10,10). The node at level n-1 associated with path P is the parent of the node at level n associated with path P' if P is obtained from P' by dropping the last link in the path. Then the leaves of this tree are precisely the self-avoiding paths from (0,0) to (10,10).

Knuth's trick: Take an unbiased directed walk in the tree, starting from the root v_0, taking v_1 to be a uniform random child of v_0, v_2 to be a uniform random child of v_1, etc., until you hit a leaf $v^* = v_n$. Let m_i be the number of children of v_i $(0 \le i \le n\text{-}1)$, so that the prob·

ability of our walk ending in that particular leaf v^* is $1/(m_0 m_1 ... m_{n-1})$. Call $m_0 m_1 ... m_{n-1}$ the weight of the leaf v^*. Estimate the number of leaves in the tree as the weight of the leaf v^* you ended at. Repeat this many times, and average the resulting estimates.

Why it works: Write $M = m_0 m_1 ... m_{n-1}$. Think of v^* and M as random variables. Let L be the (unknown) true number of leaves in the tree. The expected value of M is a sum of L terms $\text{Prob}(v^* = v)\, M(v)$, one for each leaf v in the tree, and each such summand equals 1, since it can be written as $(1/M(v))(M(v)) = 1$, so the expected value of M is exactly L.

Example: Take a tree whose root v has 2 children, w and x ; w is a leaf, and x has 2 children, y and z, both of which are leaves. Then $M(w) = 2$, $M(y) = 2 \times 2 = 4$, and $M(z) = 2 \times 2 = 4$; so if we generate a leaf non-uniformly by taking a random path through the tree (picking x, y, and z with respective probabilities, $\frac{1}{2}$, $\frac{1}{4}$, and $\frac{1}{4}$), we find that the expected value of M is

$$\frac{1}{2} \times 2 + \frac{1}{4} \times 4 + \frac{1}{4} \times 4 = 3 = L.$$

Applying this method to self-avoiding lattice paths, Knuth was able to show that the number of such paths from (0,0) to (10,10) is probably 1.06 ± 0.3 times 10^{24}.

Although the expected value of M is L (the quantity we are trying to estimate), the variance of M is quite large, so many samples are required.

One could partially quasirandomize this method of estimation using rotors: when you come to a new node, choose a random child, but on subsequent visits to the parent, choose the next child in succession. (Note that this scheme is not fully deterministic.) This would ensure that subtrees get visited roughly equally, so more of the full tree gets explored. (E.g., you won't happen to visit the same leaf twice.) Unfortunately, the main source of variance in M comes from the disparate sizes of the subtrees themselves, so this method of reducing variance probably won't achieve much.

A more general view

We have two probability distributions on a set Ω: one of them (p) is the one we're interested in, and one of them (q) is the one that's convenient or helpful to sample from. (This should remind you of acceptance/rejection sampling and Metropolis chains.)

In the context of Knuth's trick, p is the uniform distribution on the leaves, and q is the non-uniform distribution on the leaves that comes from repeatedly taking a uniform random child of each successive node we visit.

We want to compute the expected value of some random variable f; that is, we want to compute $\text{Exp}_p(f) = \sum_{\omega \text{ in } \Omega} f(\omega)\, p(\omega)$.

The obvious way to estimate this quantity is to use the ordinary Monte Carlo estimator

$$\frac{1}{n} \sum_{i=1}^{n} f(W_i)$$

where $W_1, W_2, ..., W_n$ are i.i.d. elements of Ω distributed according to $p(.)$. Instead, we write $\text{Exp}_p(f)$ as

$$\sum_{\omega \text{ in } \Omega} [f(\omega) p(\omega) / q(\omega)]\ q(\omega),$$

and estimate it as $\frac{1}{n} \sum_{i=1}^{n} f(W_i) p(W_i) / q(W_i)$ where $W_1, W_2, ..., W_n$ are i.i.d. elements of Ω distributed according to $q(.)$. This will give us an unbiased estimate of $\text{Exp}_p(f)$.

If the ratio $p(\omega)/q(\omega)$ is never too much bigger than 1, then the variance of $f(W)p(W)/q(W)$ isn't much bigger than the variance of $f(W)$, so the method will give an unbiased estimate of $\text{Exp}_p(f)$ fairly efficiently (i.e., without requiring n to be prohibitively large).

One way to think of

$$\frac{1}{n} \sum_{i=1}^{n} f(W_i) p(W_i) / q(W_i)$$

is as

$$\frac{1}{n}\sum_{i=1}^{n} f(W_i)[p(W_i)/q(W_i)];$$

we are weighing the sample-point $f(W)$ by a ratio $p(W)/q(W)$ that gives $f(W)$ more weight when $p(W)$ (the probability of choosing W under the "true" distribution $p(.)$) is large in comparison with $q(W)$ (the probability of choosing W from the sampling distribution $q(.)$). The importance sampling estimator

$$\frac{1}{n}\sum_{i=1}^{n} f(W_i)[p(W_i)/q(W_i)]$$

is not a true weighted average of $f(W_1),...,f(W_n)$, since the ratios $p(W_i)/q(W_i)$ typically do not add up to n. However, the intuition of a weighted average is not too far off, since the **expected** sum of the ratios is indeed n. Indeed, $\text{Exp}_q(p(W)/q(W)) = \sum_{\omega \text{ in } \Omega} (p(\omega)/q(\omega))\ q(\omega) = \sum_{\omega \text{ in } \Omega} p(\omega) = 1$.

Knuth's trick revisited

In the case of Knuth's trick, where p is the uniform distribution on the leaves and q is the non-uniform distribution on the leaves that comes from walking through the tree, the "random variable" f whose expected value we want to compute relative to the probability distribution p is the unknown constant L. With importance sampling, we sample from the random variable $f(W)p(W)/q(W)$, where W is distributed according to $q()$. In this case $f(W)$ and $p(W)$ cancel (their product of L and $1/L$ is the constant 1), so we're just sampling from $1/q(W)$, which we called M.

Another example

(taken from Sheldon Ross' *Introduction to Probability Models*, Ninth Edition, pp. 714-719)

Consider a list of n entries, some of which are repeats of others; we want to estimate d, the number of distinct elements in the list.

If $m(i)$, also written as m_i $(1 \le i \le n)$, is the number of times that the element in position i appears on the list, then $d = \sum_{i=1}^{n} \frac{1}{m_i}$. So if we take X uniformly at random between 1 and n, the expected value of $1/m(X)$ is

$$\sum_{i=1}^{n} \frac{1}{n} \frac{1}{m_i} = \frac{d}{n}$$

and the expected value of $n/m(X)$ is d. Hence if we have an efficient way to determine $m(X)$, we can generate k i.i.d. samples $X_1,...,X_k$ from $\{1,...,n\}$ and estimate d by the average $\frac{1}{k} \sum_{i=1}^{k} n/m(X_i)$.

Now suppose each item in the list has some value associated with it; say v_i is the value of the ith element. We want to estimate the sum v of the values of the d distinct elements. (If $v_i = 1$ for all i, this reduces to the preceding problem.) We have $v = \sum_{i=1}^{n} \frac{v_i}{m_i}$, so

$\mathrm{Exp}(v(X) / m(X)) = \sum_{i=1}^{n} \frac{1}{n} \frac{v_i}{m_i} = \frac{v}{n}$ and

$\mathrm{Exp}(n\, v(X) / m(X)) = v$, and we can estimate v by

$\frac{1}{k} \sum_{i=1}^{k} n\, v(X_i) / m(X_i)$.

Poisson processes

Bernoulli trials with *p* small and *n* large

Say we want to do $n = 10^7$ independent random trials each of which is successful with probability $p = 10^{-6}$, so that the expected number of successes is 10. (Think of *p* as the probability that a tossed coin lands on its edge!) Mathematically, we want ten million independent 0,1-valued random variables $X_1, ..., X_{10,000,000}$ each of which is 1 with probability *p* and 0 with probability 1-*p*, so that $E(X_1+...+X_{10,000,000}) = np = 10$.

The obvious way to simulate this on a computer involves generating ten million "independent" pseudorandom numbers U_k between 0 and 1; set $X_k = 1$ if U_k is less than 10^{-6} and $X_k = 0$ otherwise.

(I put the word "independent" in quotation marks because the notion of independence presumes randomness, whereas pseudorandom numbers are deterministic!)

```
Timing[Total[Table[If[RandomReal[] < 10^(-6), 1, 0], {10^7}]]]
```

{24.1584, 10}

```
(* 10 of the 10^7 Bernoulli random variables were equal to 1;
all the rest were equal to 0.
 It took 24.1584 seconds for Mathematica to do this. *)
```

A more efficient way is to use a geometric random variable describing the number of trials required until success occurs.

Specifically, let L_1 be a random variable with distribution Geom(p) (think: "L" is for "lag"), so that

$\text{Prob}(L_1=1) = p$,

$\text{Prob}(L_1=2) = (1-p)\, p$,

$\text{Prob}(L_1=3) = (1-p)^2\, p$,

...

Set $X_k = 0$ for $k = 1, 2, \ldots, L_1-1$ and $X_{L_1} = 1$.

Then let L_2 be another random variable with distribution Geom(p), independent of L_1, and set $X_k = 0$ for $k = L_1+1, L_1+2, \ldots, L_1+L_2-1$ and $X_{L_1+L_2} = 1$.

And so on. That is, let $L_1, L_2, L_3, \ldots$ be independent random variables with distribution Geom(p), let $X_k = 1$ for $k = L_1, L_1+L_2, L_1+L_2+L_3, \ldots$ and $X_k = 0$ for all other k.

How many L_i's do we need on average? ...

Just ten.

How many L_i's do we need in best case? ...

Just one.

How many L_i's do we need in worst case? ...

Let's try it (keeping in mind that *Mathematica*'s definition of GeometricDistribution differs from the standard one, i.e. the one that I am using in this course, by an offset of 1):

```
L = Table[1 + RandomInteger[GeometricDistribution[10^(-6)]], {n, 20}]
```

```
{887596, 488412, 872236, 877393, 86840, 174800, 1129440, 457490, 687959,
  1352323, 267013, 61457, 3092, 550670, 408720, 78639, 44965, 542105, 2396885, 2081129}
```

```
Table[Sum[L[[k]], {k, 1, n}], {n, 1, 20}]
```

```
{887596, 1376008, 2248244, 3125637, 3212477, 3387277, 4516717, 4974207, 5662166, 7014489,
  7281502, 7342959, 7346051, 7896721, 8305441, 8384080, 8429045, 8971150, 11368035, 13449164}
```

In this case we get eighteen successes in our 10^7 trials.

This is a good approach when p is small, n is large, and pn is not too large (i.e., not too much bigger than 1).

Recall from homework assignment #2 how we can efficiently generate a Geom(p) random variable: pick a random number U in [0,1], and output

1 if U lies in (1-p, 1],

2 if U lies in ((1-p)^2, 1-p],

3 if U lies in ((1-p)^3, (1-p)^2],

...

That is, take the log of U to the base 1-p and round up to the next higher integer.

Taking the limit as $p \to 0$, $n \to \infty$: Poisson processes

Assume that we carry out kn trials of an event that occurs independently on each trial with probability λ / n. (In the preceding section, k was 10, λ was 1, and n was 10^6.) Then we expect $(kn)(\lambda / n) = k\lambda$ successes (on average).

Suppose that the trials take place at a rate of n per second, so that all kn trials are completed in k seconds. During any given second, the expected number of successes is $(n)(\lambda / n) = \lambda$, independent of n.

The probability that, during any given second, the number of successes is 0 is

$$(1 - \lambda/n)^n,$$

which converges to $e^{-\lambda}$ as $n \to \infty$.

The probability that, during any given second, the number of successes is 1 is

$$n\,(\lambda/n)\,(1 - \lambda/n)^{n-1},$$

which converges to $\lambda e^{-\lambda}$ as $n \to \infty$.

The probability that, during any given second, the number of successes is 2 is

$$(n(n\text{-}1)/2)\;(\lambda/n)^2\,(1 - \lambda/n)^{n-2},$$

which converges to $(\lambda^2/2)e^{-\lambda}$ as $n \to \infty$.

It can be shown that the probability that, during any given second, the number of successes is i converges to

(*) $$\frac{\lambda^i}{i!}e^{-\lambda}$$

as $n \to \infty$.

(Check that if we sum (*) over all i, we get 1. So (*) determines a probability distribution on the non-negative integers.)

More generally the probability that, during any given time interval of width t, the number of successes is i converges to

(**) $$\frac{(\lambda t)^i}{i!} e^{-\lambda t}$$

as $n \to \infty$.

(You can also check that (**) sums to 1.)

You may recognize this distribution as ...

... the Poisson distribution with parameter λt.

Now suppose that, rather than doing a fixed number of trials, we just do trials (at a rate of n per second) until we get our first success. Then the time T until the first success occurs, under our earlier "log U " simulation scheme, will be equal to $1/n$ times the log of U to the base $1 - \lambda/n$ (the rounding up to the nearest integer doesn't matter in the limit), which in the limit goes to

$$\lim_{n\to\infty} \frac{\log U}{n \log(1-\lambda/n)} = \frac{\log U}{n(-\lambda/n)} = \frac{\log U}{-\lambda}$$

(or $\frac{\log(1/U)}{\lambda}$ if you prefer), so that

$$\text{Prob}[T \le t] = \text{Prob}[(\log U)/(-\lambda) \le t]$$
$$= \text{Prob}[\log U \ge -\lambda t] = \text{Prob}[U \ge e^{-\lambda t}]$$
$$= 1 - e^{-\lambda t},$$

and $\text{Prob}[T \ge t] = e^{-\lambda t}$.

You may recognize this distribution as ...

... the exponential distribution with parameter λ.

The time from the first success to the second is governed by the same probability distribution.

So if we take Bernoulli trials (with each trial having a probability of λ/n of success) occurring at a rate of n trials per second, there's a sensible way of taking a continuous-time limit by sending $n\to\infty$.

Instead of referring to "successes", we refer to "events". (Don't confuse this with the earlier way we used the word "event".) The parameter λ is sometimes called "rate" or "intensity", and measures the expected number of events per time-unit.

The first event occurs at time T_1, the second event occurs at time T_1+T_2, the third occurs at time $T_1+T_2+T_3$, etc., where T_1, T_2, T_3, ... are independent positive real-valued random variables governed by the exponential distribution with parameter λ.

The number of events during any time-interval of width t is a non-negative integer-valued random variable governed by the Poisson distribution with parameter λt, with expected value λt.

Example of a Poisson process: events correspond to clicks of a Geiger counter that detects λ decays per second on average. We might get a lot of clicks in the first second, or we might have to wait a century before we hear the first click (though that's extremely unlikely). But for $\lambda \approx 1$, the number of clicks in the first ten seconds, divided by 10, will be fairly close to λ, and the number of clicks in the first hundred seconds, divided by 100, will be even closer to λ, etc.

There is an important analogy between Bernoulli trials (in the discrete probability realm) and Poisson processes (in the continuous probability realm), with

Bernoulli trials : Geometric r.v.'s : Binomial r.v.'s
:: Poisson process : Exponential r.v.'s : Poisson r.v.'s

If you're doing Bernoulli trials, then the time until the next success is governed by a geometric distribution, and the number of successes in any time interval is governed by a binomial distribution.

If you're running a Poisson process, then the time until the next event is governed by an exponential distribution, and the number of events in any time interval is governed by a Poisson distribution.

Simulation

One way to simulate a Poisson process is to sum the inter-event times, just as we simulated a sparse Bernoulli process; in the discrete (Bernoulli) case we summed geometric random variables (with small p), whereas in the continuous (Poisson) case we must sum exponential random variables.

```
L = Table[RandomReal[ExponentialDistribution[1]], {n, 20}]
```

```
{0.573687, 0.368184, 0.186157, 1.12351, 2.1707, 0.272957, 0.135981, 0.334544, 2.18378, 0.0454035,
  0.0601858, 0.855446, 0.568263, 0.699012, 4.38447, 0.0375255, 1.54105, 0.20601, 0.921528, 1.76704}
```

```
Table[Sum[L[[k]], {k, 1, n}], {n, 1, 20}]
```

```
{0.573687, 0.941871, 1.12803, 2.25154, 4.42224, 4.6952, 4.83118, 5.16573, 7.34951,
  7.39491, 7.4551, 8.31055, 8.87881, 9.57782, 13.9623, 13.9998, 15.5409, 15.7469, 16.6684, 18.4354}
```

In this example, there are 14 events that occur in the time-interval [0,10] (i.e., between time 0 and time 10).

```
UpTillTen[] := Module[{k = 0, PartialSum = 0}, While[PartialSum < 10,
    k++; PartialSum += RandomReal[ExponentialDistribution[1]]]; Return[k - 1]]
```

```
N[Mean[Table[UpTillTen[], {100}]]]
```

```
10.08
```

```
N[Mean[Table[UpTillTen[], {1000}]]]
```

```
10.027
```

```
N[Variance[Table[UpTillTen[], {1000}]]]
```

```
10.2753
```

```
N[Variance[Table[UpTillTen[], {10 000}]]]
```

```
10.3972
```

Another way is to first condition on the number of events that occur in the time-interval [*a*,*b*], and then figure out where they are, using three facts (the first is just the definition of the Poisson distribution, and the second and third are easy to derive from our defini -

tion of the Poisson process as a limit of Bernoulli processes):

(a) The number of events that occurred in $[a,b]$ is a Poisson random variable with intensity parameter $\lambda(b-a)$.

(b) The conditional probability that k events occurred in time-interval J, given that n events occurred in time-interval I (with J a subinterval of I), is $n!/k!(n-k)!$ times $(|J|/|I|)^n$.

That is, the conditional distribution on the number of events that occurred in J, given that n events occurred in I, is Binomial$(n, |J|/|I|)$.

(c) Given that 1 event occurred in I, the conditional distribution of the time at which the event occurred is uniform on I.

So, first figure out how many events occurred in $I=[a,b]$ (using the Poisson distribution).

Split I in half and figure out how many events occurred in each half-interval (using the binomial distribution).

Split each half in half again and figure out how many events occurred in each quarter-interval (again using the binomial distribution).

Etc., splitting each number into approximate halves, following a binary tree.

(This should remind you a little bit of what we did last time with card shuffling!).

When we reach the situation where a sub-sub-...-sub-interval contains only 1 event, we don't keep splitting; instead, we choose a (uniform) random point in that sub-sub-...-sub-interval to be the instant at which an event occurred.

Yet another way, like the previous way but simpler, is: first figure out how many events occurred in $I=[a,b]$ (using the Poisson distribution); then choose them independently and uniformly in I.

CADLAG functions

For mathematical simplicity (not the same as realism!), we'll focus on Poisson processes that run forever, starting from time 0.

Let $N(t)$ denote the number of events that have occurred up to and including time t, so that

$N(t) = 0$ for $0 \le t < T_1$,

$N(t) = 1$ for $T_1 \le t < T_1+T_2$,

$N(t) = 2$ for $T_1+T_2 \le t < T_1+T_2+T_3$,

...

So for all t, $N(t)$ equals the limit of $N(t')$ as t' approaches t from the right; as t' approaches t from the left, $N(t')$ does approach some limit, but it need not equal $N(t)$.

Such an integer-valued function of a real variable is called CADLAG ("continue a droit, limite a gauche") or RCLL("right continuous with left limits").

A Poisson counting process started from time 0 is a random variable whose values aren't real numbers but rather CADLAG functions $N(\,)$ from $[0,\infty)$ to {0,1,2,3,...}.

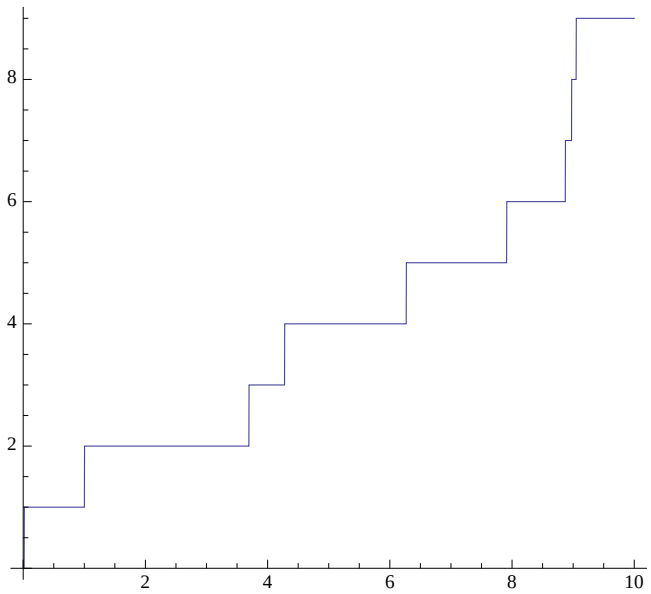
```
(* Here are the lags between one event and the next. *)

L = Table[RandomReal[ExponentialDistribution[1]], {n, 20}]
```

{0.017245, 0.984541, 2.69391, 0.582312, 1.99226, 1.64209, 0.959098, 0.104667, 0.0733868,
1.803, 0.124619, 1.25705, 1.41894, 1.3682, 1.17099, 0.770198, 0.631614, 2.17124, 0.0826515, 0.585606}

```
(* Here are the times at which events occur. *)

Events = Table[Sum[L[[k]], {k, 1, n}], {n, 1, 20}]
```

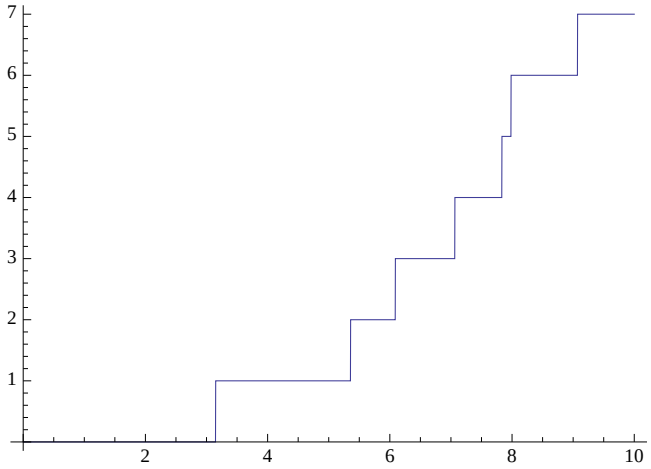
{0.017245, 1.00179, 3.6957, 4.27801, 6.27027, 7.91236, 8.87145, 8.97612, 9.04951,
10.8525, 10.9771, 12.2342, 13.6531, 15.0213, 16.1923, 16.9625, 17.5941, 19.7654, 19.848, 20.4336}

```
(* We write the counting process as a sum of
  shifted Heavisde Theta functions where HeavisideTheta[x] is
  1 if x > 0 and 0 if x < 0. *)

Plot[Sum[HeavisideTheta[x - Events[[n]]], {n, 1, 20}], {x, 0, 10}, AspectRatio → Automatic]
```



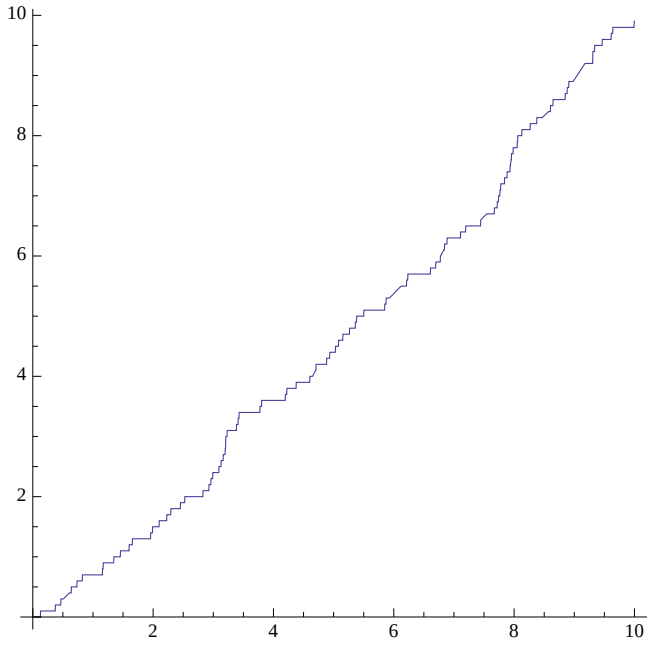

```
cadlag[] := Module[{L, Events}, L = Table[RandomReal[ExponentialDistribution[1]], {n, 100}];
  Events = Table[Sum[L[[k]], {k, 1, n}], {n, 1, 20}];
  Return[Sum[HeavisideTheta[x - Events[[n]]], {n, 1, 20}]]]

(* Here's what one Poisson CADLAG counting function looks like: *)
```

```
Plot[Evaluate[cadlag[]], {x, 0, 10}, AspectRatio → Automatic]
```



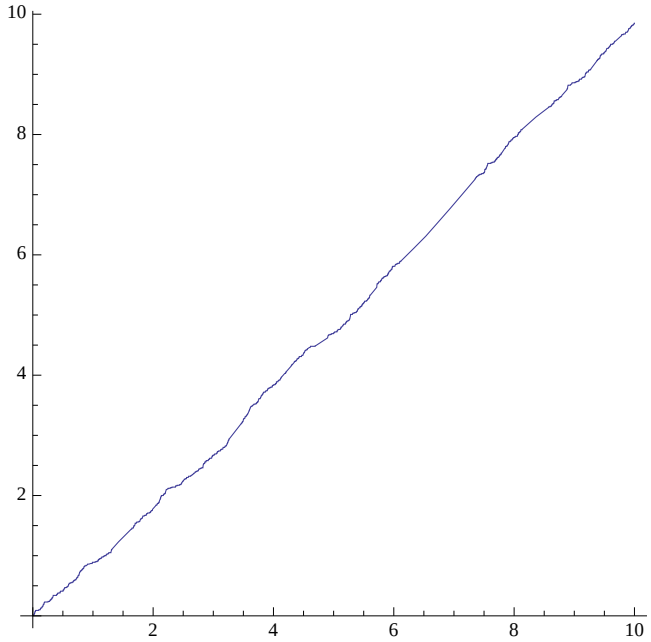

```
(* Here's an average of 10 of them: *)
```

```
Plot[Evaluate[Mean[Table[cadlag[], {10}]]], {x, 0, 10}, AspectRatio → Automatic]
```




```
(* And here's an average of 100 of them: *)
```

```
Plot[Evaluate[Mean[Table[cadlag[], {100}]]], {x, 0, 10}, AspectRatio → Automatic]
```




```
(* As you can see the average is approaching the linear function
  f(t) = t. *)
```

CADLAGs aren't differentiable, but the expected value of a CADLAG-valued random variable can be a nice differentiable function, amenable to calculus methods!

Suppose $N()$ is a random CADLAG given by the Poisson process. Then, for all t, the expected value of $N(t)$ is ...

... λt.

So, thinking now of averaging functions instead of just numbers, the expected "value" of the random function N is ...

... the linear function $f(t) = \lambda t$.

This function satisfies the differential equation $f'(t) = \lambda$. To see this directly (albeit a bit informally), note that

$$\begin{aligned} f(t+\Delta t) - f(t) &= \text{Exp}[N(t+\Delta t)] - \text{Exp}[N(t)] \\ &= \text{Exp}[N(t+\Delta t) - N(t)] \\ &= \lambda\, \Delta t \end{aligned}$$

whence $(f(t+\Delta t) - f(t))/\Delta t = \lambda$, which "goes to" 0 as $\Delta t \to 0$.