CSE 103: Probability and statistics **Fall 2010**

Topic 5 — Random generation, I

5.1 Simulating simple discrete distributions with a fair coin

The simplest of all distributions is the fair coin. Let's code its two outcomes, heads and tails, by 1 and 0 respectively.

$$\begin{array}{ll} 0 & \text{with probability } 1/2 \\ 1 & \text{with probability } 1/2 \end{array}$$

We call this the Bernoulli(1/2) or $B(1/2)$ distribution.

With just the ability to flip fair coins (or equivalently, to sample from the $B(1/2)$ distribution), we can generate random samples from any discrete distribution with a finite sample space. We will get to this result in several steps.

5.1.1 Uniform distribution over b-bit integers

The uniform distribution over b-bit integers has the following probability space:

$$\begin{aligned} \Omega &= \{0,1\}^b \\ \Pr(\omega) &= 1/2^b \text{ for all } \omega \in \Omega \end{aligned}$$

Call this distribution Unif($\{0,1\}^b$). To sample from it, simply flip a fair coin for each of the b bits.

```
For i = 1 to b:
   Draw X_i from B(1/2)
Output X_1 X_2 ··· X_b
```

5.1.2 Uniform distribution over $\{1, 2, \ldots, n\}$

Now consider a very similar distribution with probability space

$$\begin{aligned} \Omega &= \{1,2,3,\ldots,n\} \\ \Pr(\omega) &= 1/n \text{ for all } \omega \in \Omega \end{aligned}$$

Call this distribution Unif($\{1,\ldots,n\}$). If n is of the form 2^b, then the previous algorithm, called with $b = \log_2 n$, gives us a uniform distribution over $\{0,1,\ldots,n-1\}$. So we can just add 1 to that value and we're done.

More generally, here's a sampling algorithm.

```
Let b = ⌈log_2 n⌉
Repeat:
   Generate a sample X from Unif({1,2,...,2^b}), as described above
   If X ≤ n:  output X and halt
```

First, let's check that this indeed outputs the right distribution, that each of the values $1, 2, \ldots, n$ gets output with probability exactly $1/n$. Specifically, we need to show that if X is a sample from $\text{Unif}(\{1, 2, \ldots, 2^b\})$, then for any $i \in \{1, 2, \ldots, n\}$, we have $\Pr(X = i | X \leq n) = 1/n$. This follows from the formula for conditional probability:

$$\Pr(X = i | X \leq n) \;=\; \frac{\Pr(X = i \text{ AND } X \leq n)}{\Pr(X \leq n)} \;=\; \frac{\Pr(X = i)}{\Pr(X \leq n)} \;=\; \frac{1/2^b}{n/2^b} \;=\; \frac{1}{n}.$$

How many coin flips does this algorithm use? Each time we go through the repeat loop, we use b flips to generate X; but how many times do we loop? First notice that $b = \lceil \log_2 n \rceil$, which means that b is the smallest integer that is greater than or equal to $\log_2 n$. Therefore

$$b - 1 < \log_2 n \leq b \quad \Rightarrow \quad \frac{1}{2} 2^b < n \leq 2^b \quad \Rightarrow \quad \Pr(X \leq n) = \frac{n}{2^b} > \frac{1}{2}.$$

Therefore, on each iteration of the repeat loop, the probability of halting, $\Pr(X \leq n)$, is at least $1/2$. So the expected number of iterations is at most 2, which means that the expected number of coin flips needed is at most $2b$.

5.1.3 Uniform distribution over $[0, 1]$

This time, we want a uniform distribution over real numbers in the interval $[0, 1]$. However, the size of this sample space is uncountably infinite, and for a variety of practical reasons, we will typically want only a finite amount of precision.

Recall the binary representation of fractional values: $0.z_1 z_2 z_3 \cdots$. Here z_1 is the position for $1/2$, z_2 is the position for $1/4$, z_3 is the position for $1/8$, and so on. For instance, $0.101 = 1/2 + 1/8 = 5/8$ whereas $0.0011 = 1/8 + 1/16 = 3/16$.

Let's say that we want b bits of precision.

$$\begin{aligned} \Omega &= \{0.z_1 z_2 \cdots z_b : z_1, \ldots, z_b \in \{0, 1\}\} \\ \Pr(\omega) &= 1/2^b \;\text{ for all } \omega \in \Omega \end{aligned}$$

This is exactly like generating a random b-bit integer: just stick a "0." in front.

5.1.4 A biased coin

The next distribution we want is a coin with bias p, where the outcome is once again coded as 0/1:

$$\begin{array}{ll} 0 & \text{with probability } 1 - p \\ 1 & \text{with probability } p \end{array}$$

We call this the Bernoulli(p) or $B(p)$ distribution. Can we simulate $B(p)$ using $B(1/2)$?

Here's an easy way to do so.

```
Generate X from Unif[0,1]
If X ≤ p:  output 1
else:      output 0
```

Since $\Pr(X \leq p) = p$, this generates the right distribution. But how many fair coin flips does it use? As stated here, it seems to require that X is infinite precision. The way around this is to notice that we can just generate X one bit at a time, until it is clear whether X is less than p or more than p.

For instance, suppose $p = 3/8$. In binary, this is 0.011. Writing $X = 0.X_1X_2X_3\cdots$, we first flip a coin to get X_1, then another coin to get X_2, and so on. Suppose $X_1 = 1$. Then we can stop at once, because we know that X is at least 1/2 and therefore $X \geq p$, no matter what $X_2, X_3, \ldots$ turn out to be. On the other hand, if $X_1 = 0$, then all we know is that $X \leq 1/2$, so we can't be sure whether it is bigger or smaller than p, and we have to continue. Here's the modified algorithm:

```
Let 0.p1 p2 p3 ··· be the binary representation of p
Repeat for i = 1, 2, 3, ... :
   Draw X_i from B(1/2)
   If p_i = 1 and X_i = 0:  halt and output 1
   If p_i = 0 and X_i = 1:  halt and output 0
```

How many bits are needed? That is, how many times does the algorithm loop? Notice that on each iteration, the algorithm halts if $X_i \neq p_i$. This happens with probability exactly 1/2. Therefore, the expected number of iterations is exactly 2.

So we can simulate a biased coin using, on average, two fair coins.

5.1.5 Arbitrary discrete distribution with finite sample space

Let's move to a much more general distribution.

$$\begin{aligned} \Omega &= \{\omega_1, \ldots, \omega_k\} \\ \Pr(\omega_i) &= p_i \end{aligned}$$

where the p_i are nonnegative and sum to 1. An example is the roll of a die, which has $k = 6$ and $p_1 = \cdots = p_k = 1/6$.

To sample from this distribution, we use the same ideas as for a biased coin. Let's start with the infinite precision version.

```
Generate X from Unif[0,1]
For all i = 1 to k:
   If p_1 + ··· + p_{i-1} < X ≤ p_1 + ··· + p_i:  output ω_i
```

In effect, we divide the interval $[0, 1]$ into k bins, where the ith bin stretches from $p_1+\cdots+p_{i-1}$ to $p_1+\cdots+p_i$, and therefore has length exactly p_i. We generate X uniformly from $[0, 1]$ and then output the index of the bin that it falls into. The chance of falling into the ith bin (that is, of outputting ω_i) is therefore exactly p_i.

As before, we can run this process by generating X one bit at a time, and stopping as soon as it is clear which bin X will fall into. It is possible to show that the expected number of bits (coin flips) needed is at most $1 + \log_2 k$.