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

Topic 5 — Random generation

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
```

Let's check that each of the values $1, 2, \ldots, n$ is output with probability exactly $1/n$. Specifically, we need to show that if X is a sample from Unif($\{1, \ldots, 2^b\}$), then for any $i \in \{1, \ldots, n\}$, 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

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

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 ≤ 2, which means that the expected number of coin flips needed is $\leq 2b$.

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: $0.z_1z_2z_3\cdots$. 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.

$$\begin{aligned} \Omega &= \{0.z_1z_2\cdots z_b : z_1, \ldots, z_b \in \{0, 1\}\} \\ \Pr(\omega) &= 1/2^b \;\text{ for all } \omega \in \Omega \end{aligned}$$

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.p_1 p_2 p_3 ··· 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.

5.2 From biased coin to fair coin

The previous section shows how to generate arbitrary discrete distributions if we have a fair coin at our disposal. But what if all we have is a biased coin – and we don't even know what the bias is? Can we use it to simulate a fair coin?

Here's how:

```
Repeat:
   Flip (biased) coin twice
   If outcome is HT:  halt and output 0
   If outcome is TH:  halt and output 1
```

Let's say the coin has some bias p. Then in any iteration of the loop,

$$\begin{aligned} \Pr(\text{output } 0) &= \Pr(\text{biased coin yields } HT) &= p(1-p) \\ \Pr(\text{output } 1) &= \Pr(\text{biased coin yields } TH) &= p(1-p) \end{aligned}$$

The two probabilities are equal! Thus we really do get a sample from $B(1/2)$.

How many times do we need to flip the biased coin? On each iteration, we flip it twice, and the chance of halting is $2p(1-p)$. Therefore the expected number of iterations before halting is $1/(2p(1-p))$, and the expected number of coin flips is $1/(p(1-p))$.

5.3 Permutations

How can we pick a random permutation of $(1, 2, \ldots, n)$? Here's a natural algorithm:

```
A = {1, 2, ..., n}
For i = 1 to n:
   Pick an element x ∈ A at random
   Place x in the ith position of the permutation
   A = A − {x}
Output the permutation
```

We need to show that every permutation has exactly a $1/n!$ probability of being generated. To this end, fix any permutation $(a_1, a_2, \ldots, a_n)$ and let E_i be the event that the algorithm places a_i in position i of its

output. Then

$$\begin{aligned}\Pr(\text{algorithm outputs } (a_1, a_2, \ldots, a_n)) &= \Pr(E_1 \cap E_2 \cap \cdots \cap E_n) \\ &= \Pr(E_1)\Pr(E_2 \cap \cdots \cap E_n | E_1) \\ &= \Pr(E_1)\Pr(E_2|E_1)\Pr(E_3 \cap \cdots \cap E_n | E_1, E_2) \\ &= \Pr(E_1)\Pr(E_2|E_1)\Pr(E_3|E_1, E_2)\cdots\Pr(E_n|E_1, E_2, \ldots, E_{n-1}) \\ &= \frac{1}{n} \cdot \frac{1}{n-1} \cdot \frac{1}{n-2} \cdots \frac{1}{1} \quad = \quad \frac{1}{n!},\end{aligned}$$

just as we wanted.

In picking a random permutation, we can picture n slots that need to be filled. Our algorithm picks an element for the first slot, then one for the second slot, then the third slot, and so on. What if we went through the slots in a different order, say right-to-left instead of left-to-right? What if we started with the third slot, then the tenth, and so on, in some arbitrary order? It doesn't make a difference: we still get a random permutation.

Let $(\sigma_1, \ldots, \sigma_n)$ be any permutation: this is the order in which we will fill in the slots. Here's an alternative algorithm for generating a random permutation.

$A = \{1, 2, \ldots, n\}$
`For` $i = 1$ `to` n`:`
  `Pick an element` $x \in A$ `at random`
  `Place` x `in the` σ_i`th position of the permutation`
  $A = A - \{x\}$
`Output the permutation`

To check that this is correct, once again pick any permutation $(a_1, \ldots, a_n)$ and let E_i be the event that the algorithm puts a_{σ_i} in location σ_i. Then the previous derivation holds verbatim.

5.3.1 Implications

The second algorithm for generating random permutations has many interesting implications, for instance:

1. Randomly permute a standard deck of cards.

   What's the chance that the first card is a heart? This is easy: there are 52 possible cards, and 13 of them are hearts. So the answer is $13/52 = 1/4$.

   What's the chance that the *tenth* card is a heart? At first, this seems more complicated, because the first nine cards may or may not be hearts. But remember that we can generate a random permutation by first choosing the card in the tenth position, and then moving on to the remaining positions! When we are choosing this first card, we again have 52 choices, of which 13 are hearts. So the answer is still $1/4$.

2. Deal a ten-card hand from a standard deck of cards.

   What's the chance that the third card is a 7? Well, one way to deal a ten-card hand is to randomly permute the entire deck of cards, and then pick the first ten elements in the permutation. And we can generate the random permutation by first choosing the third position. The chance that it is a 7 is therefore $4/52 = 13$.

   What's the chance that the third card is a 7 *given that the tenth card is a 7*? This time, pick the random permutation by first choosing the tenth position, then the third position, and then all the other positions. So when the third position is being chosen, there are 51 cards remaining, of which three are 7s. Therefore the conditional probability is $3/51 = 1/17$.