Stefan Walzer, Stefan Hermann
ITI Prof. Sanders
winter term 2025/2026

Exercise Sheet 3 – Important Random Variables and How to Sample Them

Probability and Computing

Exercise 1 – Ber(1/3) from Ber(1/2)

Design an algorithm that, given a sequence $B_1, B_2, \ldots \sim \text{Ber}(1/2)$ of random bits, computes a sample $B \sim \text{Ber}(1/3)$ in expected time $\mathcal{O}(1)$.

Solution 1

We interpret $B_1, B_2, B_3, \ldots$ as the binary expansion of a number $U = (0.B_1B_2B_3\ldots)_2$. Then $U \sim \mathcal{U}([0,1])$. We define $B := \mathbb{1}_{U<1/3}$. This immediately implies $B \sim \text{Ber}(1/3)$ as desired. The binary expansion of 1/3 is $1/3 = (0.01010101\ldots)_2$. Thus, the following algorithm results, which always takes the next two digits of U's binary representation and checks whether they allow a decision:

```
for i = 1 to ∞ do
    (x, y) ← (B_{2i−1}, B_{2i})
    if (x, y) = (0, 0) then
        return 1
    else if (x, y) = (1, 0) or (x, y) = (1, 1) then
        return 0
```

Each round leads to a decision with probability 3/4. If R is the number of rounds, then

$$\mathbb{E}[R] = \sum_{i \in \mathbb{N}_0} \Pr[R > i] = \sum_{i \in \mathbb{N}_0} \frac{1}{4^i} = \frac{1}{1 - \frac{1}{4}} = \frac{4}{3}.$$

Remark: In practice, one wouldn't actually do it this way. Instead, as in the next exercise, one assumes that one can directly sample $U \sim \mathcal{U}([0,1])$ (as accurately as floating-point numbers allow).

Remark: The runtime is unbounded — and this is unavoidable. We can show this by contradiction. Suppose there exists an algorithm that always terminates after reading only a fixed prefix $B_1, \ldots, B_C$ of the random bit sequence for some $C \in \mathbb{N}_0$. Then its output B is a random variable $B : \Omega \longrightarrow \{0,1\}$ on the probability space $\Omega = \{0,1\}^C$ (with uniform distribution). Each outcome has probability 2^{-C}. Hence, any event (and thus also the event $\{B = 1\}$) must have probability that is an integer multiple of 2^{-C}. This contradicts the requirement that $\Pr[B = 1] = 1/3$.

Exercise 2 – $\mathrm{Ber}(p)$ and $\mathcal{U}(\{1,\ldots,n\})$ from $\mathcal{U}([0,1])$

We now assume a machine model that can handle real numbers and allows us to sample $U \sim \mathcal{U}([0,1])$. Show that we can also sample $B \sim \mathrm{Ber}(p)$ for $p \in [0,1]$ and $X \sim \mathcal{U}(\{1,\ldots,n\})$ for $n \in \mathbb{N}$.

Hint: For the rest of this sheet and the course, we take this result as given.

Solution 2

Given $U \sim \mathcal{U}([0,1])$, define $B := \mathbb{1}_{U<p}$ and $X := \lceil U \cdot n \rceil$. Then indeed:

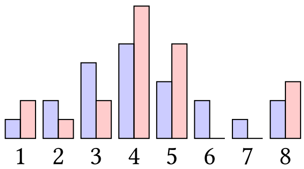
$$\Pr[B=1] = \Pr[U<p] = p, \text{ and}$$
$$\text{for } 1 \le i \le n\text{: } \Pr[X=i] = \Pr\left[U \cdot n \in (i-1, i]\right] = \Pr\left[U \in \left(\tfrac{i-1}{n}, \tfrac{i}{n}\right]\right] = \tfrac{1}{n}.$$

Remark: Strictly speaking, since $U \sim \mathcal{U}([0,1])$, the value $U = 0$ is possible, which would yield $X = 0$, even though we want $X \in \{1,\ldots,n\}$. However, this happens with probability 0. One can fix this by defining that 0 rounds up to 1, or simply ignore such minor edge cases.

Exercise 3 – Rejection Sampling in General

Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be distributions over a finite set D. Assume:

1. We can sample $X \sim \mathcal{D}_1$ in time $\mathcal{O}(1)$.
2. For any $x \in D$, $p_1(x) := \Pr_{X\sim\mathcal{D}_2}[X=x]$ as well as $p_2(x) := \Pr_{X\sim\mathcal{D}_1}[X=x]$ can be computed in $\mathcal{O}(1)$.
3. There exists $C > 0$ such that for all $x \in D$,
$$p_2(x) \le C \cdot p_1(x).$$



Possible histogram for $\mathcal{D}_1$ (blue, left) and $\mathcal{D}_2$ (red, right). It always holds that "red $\le 2 \cdot$ blue", so condition (3) holds with $C = 2$.

Design an algorithm that samples $Y \sim \mathcal{D}_2$ in expected time $\mathcal{O}(C)$.

Solution 3

The algorithm works as follows:

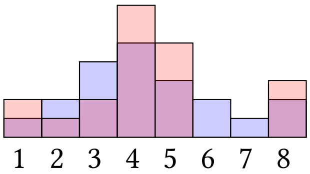
while True **do**
  sample $X \sim \mathcal{D}_1$ // $\mathcal{O}(1)$
  sample $U \sim \mathcal{U}([0,1])$ // $\mathcal{O}(1)$
  if $U < \frac{p_2(X)}{C \cdot p_1(X)}$ **then** // $\mathcal{O}(1)$
    return X

To verify correctness, note that $\frac{p_2(X)}{C\cdot p_1(X)} \in [0,1]$ by assumption (3). Let Y be the outcome of a single iteration: $Y = X$ if X is accepted, and $Y = \perp$ otherwise. Then, for $x \in D$:

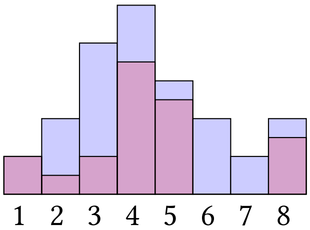
$$\Pr[Y=x] = \Pr[X=x] \cdot \Pr\left[U < \frac{p_2(x)}{C \cdot p_1(x)}\right] = p_1(x) \cdot \frac{p_2(x)}{C \cdot p_1(x)} = \frac{p_2(x)}{C}.$$

Thus, $\Pr[Y = x]$ is *proportional* to $p_2(x)$, and so $\Pr[Y = x \mid Y \neq \bot] = p_2(x)$. In other words: whenever a sample is returned, it follows the distribution $\mathcal{D}_2$ as desired. The success probability per iteration is $\Pr[Y \neq \bot] = \sum_{x \in D} \Pr[Y = x] = 1/C$. Hence, the expected number of rounds until success is C.

Intuition: You can visualize this process. If we draw the histograms of the two distributions on top of each other:



If we scale up the blue bars by a factor of C, the red bars are always below the blue ones.



To sample from the red distribution, it suffices to draw a random red point and return the index of the bar in which it lies. To achieve this, we draw a random blue point (in the illustration: a point that is blue or purple) and keep it if it is red.

In the algorithm, a random blue point is drawn by first choosing a bar X, and then selecting a random height $U \cdot C \cdot p_1(X)$ along that bar. This height is then compared with the height of the corresponding red bar.

Exercise 4 – $G \sim \text{Geom}_1(p)$ with Inverse Transform Sampling

Design an algorithm that, for a given $p \in (0, 1]$, samples a random variable $G \sim \text{Geom}_1(p)$ in time $\mathcal{O}(1)$.

Solution 4

The cumulative distribution function of G is:

$$F_G(i) = \Pr[G \leq i] = 1 - (1 - p)^i.$$

For the (generalized) inverse, it follows for $u \in (0, 1]$:

$$\begin{aligned} F_G^{-1}(u) &:= \min\{i \in \mathbb{N}_0 \mid F_G(i) \geq u\} = \min\{i \in \mathbb{N}_0 \mid 1 - (1-p)^i \geq u\} \\ &= \min\left\{i \in \mathbb{N}_0 \mid i \geq \frac{\log(1-u)}{\log(1-p)}\right\} = \left\lceil \frac{\log(1-u)}{\log(1-p)} \right\rceil. \end{aligned}$$

According to the method, the following should work:

sample $U \sim \mathcal{U}([0,1])$
return $G = \left\lceil \frac{\log(1-U)}{\log(1-p)} \right\rceil$

We can also verify that everything worked by checking that the G produced by the algorithm has the desired distribution function:

$$\begin{aligned}\Pr[G \le i] &= \Pr\left[\left\lceil \frac{\log(1-U)}{\log(1-p)} \right\rceil \le i\right] = \Pr\left[\frac{\log(1-U)}{\log(1-p)} \le i\right] = \Pr[\log(1-U) \ge i \log(1-p)] \\ &= \Pr[1-U \ge (1-p)^i] = \Pr[U \le 1-(1-p)^i] = 1-(1-p)^i.\end{aligned}$$

Exercise 5 – Sampling without Replacement

We consider algorithms that, for $k, n \in \mathbb{N}$ with $0 \le k \le n/2$, compute a set $S \subseteq [n]$ of size k, chosen uniformly at random among all subsets of $[n]$ of size k.

(a) Why can we assume $k \le n/2$ without loss of generality?

(b) Describe an algorithm that has an expected runtime of $O(k \log k)$.
Hint: Rejection sampling and search tree.

(c) **Bonus:** Design an algorithm that has a worst-case runtime of $O(k \log k)$.

(d) **Bonus:** Research how to achieve a worst-case runtime of $O(k)$:

`https://stackoverflow.com/a/67850443`

Solution 5

(a) $S \subseteq [n]$ is a random set of size k if and only if $[n] \setminus S$ is a random set of size $n-k$.

(b) Conceptually, the algorithm samples *with* replacement, stores the results in a search tree, and ignores any samples that have already occurred. It continues until k distinct results have been obtained. This is a form of rejection sampling, and it is quite clear that it is correct.

```
Algorithm SampleWithoutReplacement(n, k):
    S ← ∅ // as search tree
    while |S| < k do
        sample X ~ U({1, ..., n})
        if X ∉ S then
            S ← S ∪ {X}
    return S
```

By the assumption from (a) and the loop condition, at the beginning of each iteration we have $|S| < k \le n/2$. Thus, the probability of drawing something we already have is always at most $1/2$. It follows that the number F of unsuccessful iterations is expected to be at most the number of successful ones, i.e. $\mathbb{E}[F] \leq k$.

The total runtime is $T = (k + F) \cdot O(\log k)$ because there are $k + F$ iterations, each performing search tree operations in $O(\log k)$. Hence $\mathbb{E}[T] = O(k \log k)$.

(c) The idea is to explicitly manage the set of elements that can still be drawn. In the following, Array$[1..n]$ is used, which always contains a permutation of the set $\{1, \ldots, n\}$. At the beginning of iteration i, Array$[1..i-1]$ contains the elements already drawn, and Array$[i..n]$ contains those still available.

```
Algorithm SampleWithoutReplacement(n, k):
    Array = [1, 2, ..., n]  // everything still drawable
    for i = 1 to k do
        sample j ~ U({i, ..., n})
        swap Array[j] and Array[i]  // does nothing if j = i
    return Array[1..k]
```

Unfortunately, this results in a runtime of $O(n + k)$ because of the array initialization. However, this can be fixed. Clearly, at most $2k$ indices i can satisfy Array$[i] \neq i$. It therefore suffices to store only these exceptional positions in a search tree. This yields a runtime of $O(k \log k)$.

(d) See `https://github.com/ciphergoth/sansreplace/blob/master/cardchoose.md`