1 Strategies for constant Bernoulli factories

By the definition of Keane and O'Brien (1994), any constant function (i.e. independent of p) is *simulable*, because we can ignore the p-coins altogether and simply use an auxiliary random variable. In particular, for any $c \in [0,1]$, we can sample a random variable $u \sim U(0,1)$ and output heads iff $u < c$.

We will see later on that constant functions are also *strongly simulable*; that is, they can be simulated from p-coins without the use of auxiliary random variables. We later reveal constant functions (with $c \in (0,1)$) to be a special case of the class of strongly simulable functions given by the Keane-O'Brien theorem (Keane and O'Brien, 1994). But first we take a detour into how one might construct exact simulations of certain constants, as opposed to approximate simulations via Bernstein polynomials that are usually applied in the literature.

1.1 A first naïve construction

Let us consider first the Bernoulli factory described in Von Neumann (1951), for the constant function $f(p) \equiv 1/2$:

> *To cite a human example, for simplicity, in tossing a coin it is probably easier to make two consecutive tosses independent than to toss heads with probability exactly one-half. If independence of successive tosses is assumed, we can reconstruct a 50-50 chance out of even a badly biased coin by tossing twice. If we get heads-heads or tails-tails, we reject the tosses and try again. If we get head-tails (or tails-heads), we accept the result as heads (or tails). The resulting process is rigorously unbiased, although the amended process is at most 25 percent as efficient as ordinary coin-tossing.*

Most authors now use the slightly more convenient convention with the outcomes inverted, so that once 10 or 01 is observed, the output of the Bernoulli factory is equal to the last coin flip (0 or 1 respectively). It is easy to show that, as claimed, this procedure produces an output heads with probability 1/2:

$$\mathbb{P}(Y=1) = \mathbb{P}(X_\tau = 1) = \mathbb{P}(X_2 = 1 \mid X_2 \neq X_1) = \frac{\mathbb{P}(X_1=0, X_2=1)}{\mathbb{P}(X_1=0, X_2=1) + \mathbb{P}(X_1=1, X_2=0)} = \frac{p(1-p)}{2p(1-p)} = \frac{1}{2}.$$

The running time of this procedure is random, and indeed unbounded. In particular, $\tau \stackrel{d}{=} 2 \times \text{Geom}(2p(1-p))$, with expectation $\mathbb{E}(\tau) = \frac{1}{p(1-p)}$. Figure 1 shows the expected running time for various values of p. The expected running time achieves its minimum value of four when $p = 1/2$ (that is, the coin was already unbiased), hence von Neumann's claim that "the amended process is at most 25 percent as efficient as ordinary coin-tossing" (which has deterministic running time one). The expected running time approaches infinity when p is close to 0 or 1, but is moderate across a large range non-extreme values.

We can generalise von Neumann's procedure to produce Bernoulli factories for $f(p) \equiv 1/k$ with $k \in \{2, 3, \dots\}$. We simply toss the p-coin k times and, if the outcome is a sequence with one head and $k-1$ tails, we output the last coin flip in the sequence. Otherwise we start again. Now the probability of observing a usable sequence is $p(1-p)^{k-1}$, which is (strictly if $p \in (0,1)$) decreasing in k, so the efficiency will be even worse than in the initial case $k = 2$. The running time has distribution $\tau \stackrel{d}{=} k \times \text{Geom}(kp(1-p)^{k-1})$. But we can mitigate this slightly: a sequence with one tail and $k-1$ heads would also work, with the outputs inverted. So we need only wait for whichever of these sequences occurs first. Now the distribution of the running time is $\tau \stackrel{d}{=} k \times \min\{\text{Geom}(kp(1-p)^{k-1}), \text{Geom}(kp^{k-1}(1-p))\} \stackrel{d}{=} k \times \text{Geom}(1 - [1-kp(1-p)^{k-1}][1-kp^{k-1}(1-p)])$.

Figure 2 shows the expected running time when $k = 3$; when considering only sequences with two heads and a tail, or only sequences with two tails and a head, or whichever occurs first. It is clear from the figure that this adjustment considerably improves the efficiency, particularly when p is not close to 1/2. The expected running time is minimised at $p = 1/2$, where it takes the value $\frac{64}{13} \simeq 4.9$. This is not much worse than the $f(p) = 1/2$ Bernoulli factory, and the curves for $f(p) = 1/2$ and $f(p) = 1/3$ are at least this close everywhere.

This is looking promising: here is a reasonably efficient procedure that can produce any function $f(p) \equiv 1/k$; or of course $f(p) \equiv (k-1)/k$ by inverting the outputs. But there is a problem. Figures 3 and 4 show the expected running time for $k = 4$ and $k = 8$ respectively. As k increases, the procedure quickly becomes inefficient for p close to 1/2. At $k = 4$ we see the first non-convex function, with two minima appearing — at $p \simeq 0.30, 0.70$ with $\mathbb{E}(\tau) \simeq 8.77$. And by the time $k = 8$ things have gone badly wrong. The minima are now at $p \simeq 0.125, 0.875$ with value $\mathbb{E}(\tau) \simeq 20.37$, but more importantly, the local maximum at $p = 1/2$ is now a large peak with $\mathbb{E}(\tau) \simeq 130.0$. Alas, this naïve construction is not so efficient after all.

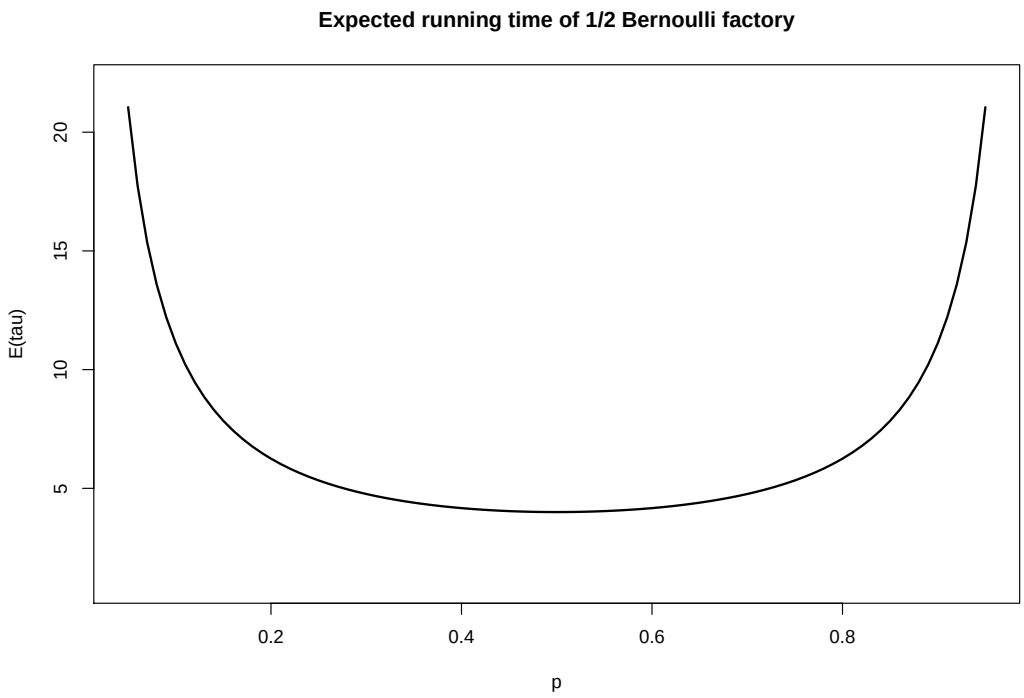

Figure 1: ●

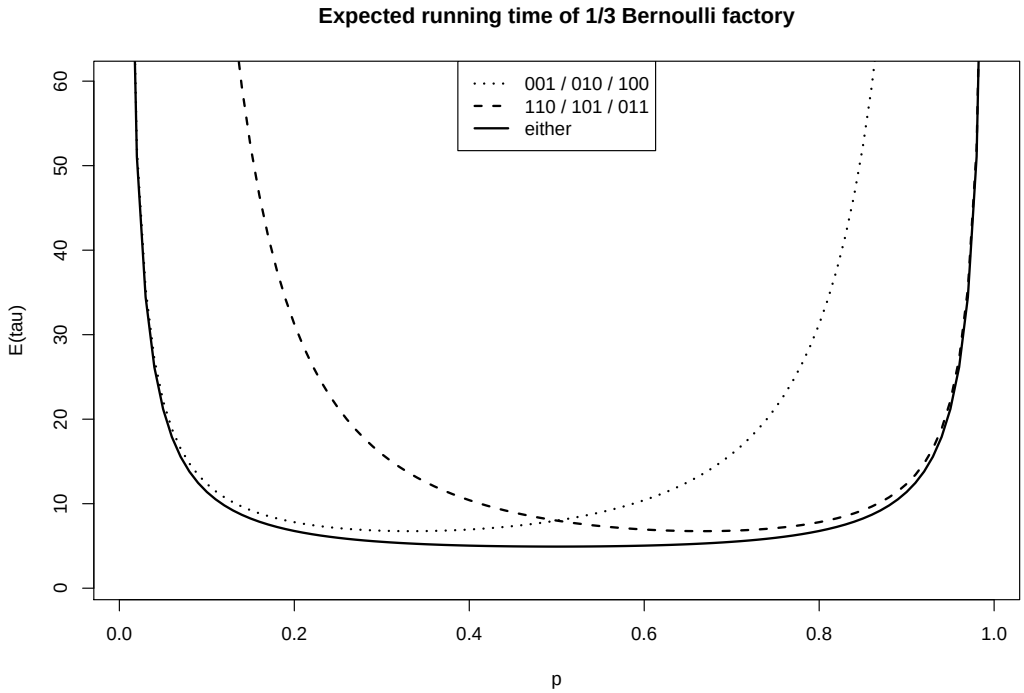

Figure 2: ●

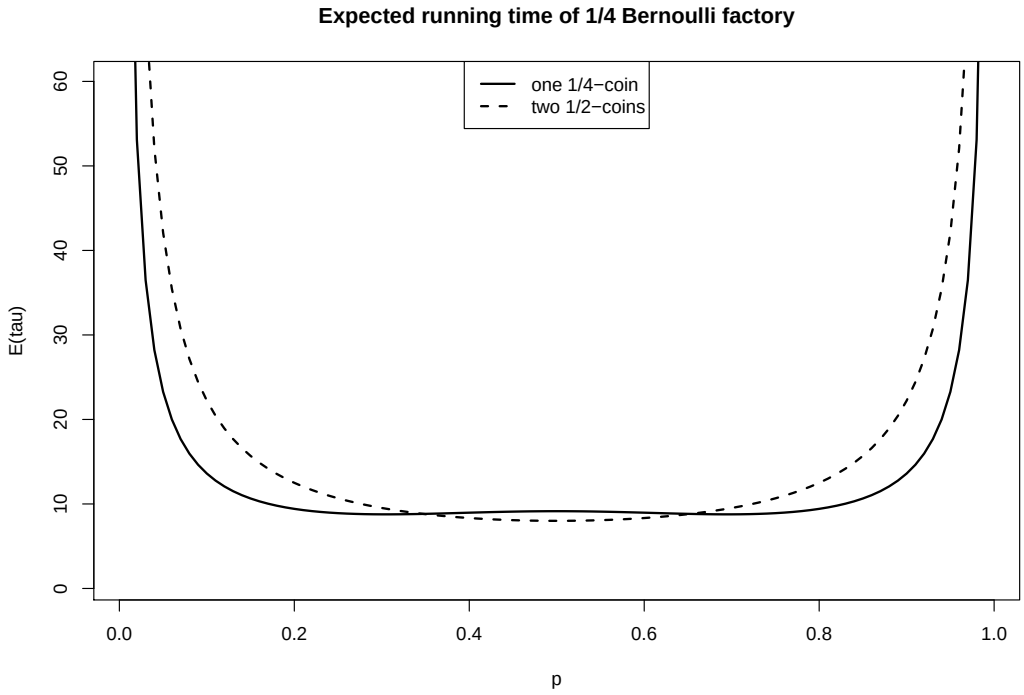

Figure 3: •

It is not surprising that, for larger k, the efficiency is high for moderately extreme values of p but low for p close to $1/2$. This procedure requires us to observe $k-1$ tails (or heads) out of k tosses, which is a rare event with an unbiased coin, but a likely one with a coin strongly biased towards tails or heads respectively.

Figures 3 and 4 also show the expected running time for an alternative procedure. A $(1/4)$-coin can be constructed from two $(1/2)$ coins by outputting heads iff both coins output heads, and a $(1/8)$-coin can be constructed from three $(1/2)$-coins in the same way. Figure 3 shows that with $k = 4$ this alternative procedure is already more efficient for some values of p close to $1/2$, and with $k = 8$ (Figure 4) the comparison is striking, although the alternative still doesn't do so well for extreme values of p.

It is possible to improve the efficiency of this algorithm a bit further using early stopping. Since a usable output must contain either exactly one head or exactly one tail, we can stop early if we have seen two or more of both heads and tails. This alteration will have a greater effect for larger values of k, since we could possibly stop after just four flips as opposed to k in the original procedure. It also has a greater effect when p is close to $1/2$, since this is the region where it is most likely to observe the two-heads-two-tails event that allows us to stop early. Therefore this adjustment is potentially very helpful as it seems to target precisely the cases where the original procedure performs badly. Figure 5 shows the difference it makes in the case $k = 8$, using a Monte Carlo approximation for the expected number of queries with early stopping. We see that, as expected, early stopping drastically increases the efficiency of the procedure when p is close to $1/2$, yielding a factor 2 of efficiency for $p = 1/2$. However, it is also evident that the gain is not sufficient to overcome the problems with this procedure.

1.2 Constants as functions of (1/2)-coins

We have mentioned how a $(1/2^n)$-coin can be constructed by combining the results of n $(1/2)$-coins. We achieved this using an AND operator, i.e. multiplying the outputs. It is easy to see that the set of functions constructible using only AND operators on $(1/2)$-coins is indeed limited to $\{f(p) = 2^{-n} \mid n \in \mathbb{N}\}$. But there are other logical operators at our disposal: it is well known in computer science that any logic circuit can be constructed using only NAND operators, so perhaps we could do much better if we made use of NOT (inversion) operators too.

As an example, imagine we would like to construct a Bernoulli factory for $f(p) \equiv \frac{11}{16}$. Noting that AND is

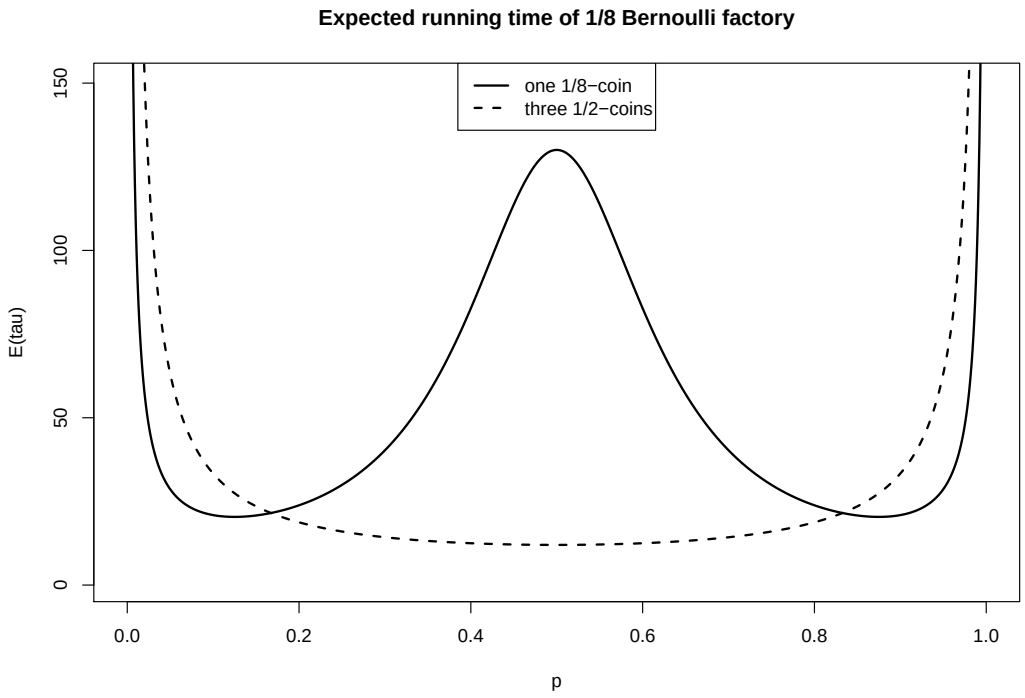

Figure 4: ●

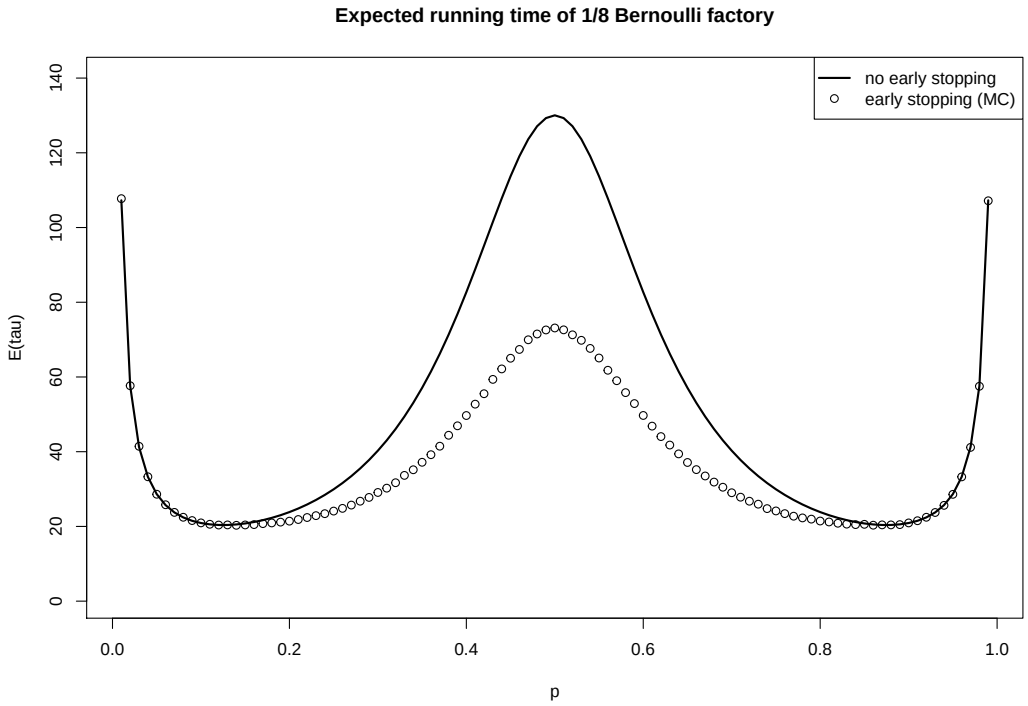

Figure 5: ●

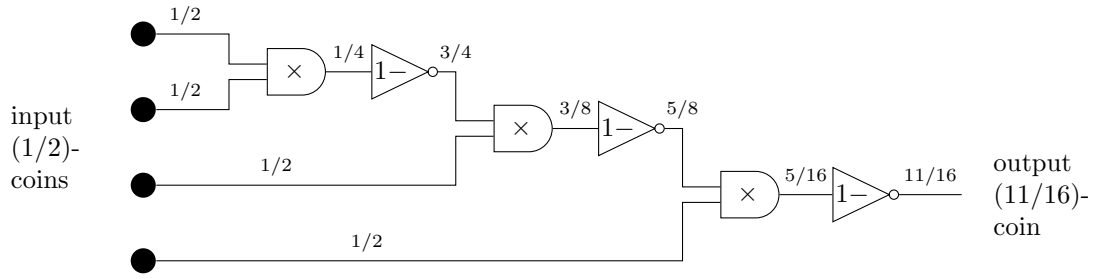

Figure 6: Logic circuit representation of an algorithm to produce one (11/16)-coin flip from four (1/2)-coin flips using AND and NOT gates

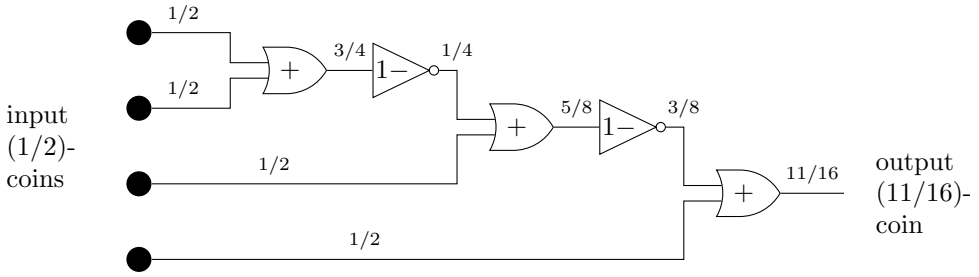

Figure 7: Logic circuit representation of an algorithm to produce one (11/16)-coin flip from four (1/2)-coin flips using OR and NOT gates

equivalent to $\times$ and NOT is equivalent to $1-$, we find the following decomposition.

$$\begin{aligned}\frac{11}{16} &= 1-\frac{5}{16} = 1-\frac{1}{2}\times\frac{5}{8}\\ &= 1-\frac{1}{2}\times\left(1-\frac{3}{8}\right) = 1-\frac{1}{2}\times\left(1-\frac{1}{2}\times\frac{3}{4}\right)\\ &= 1-\frac{1}{2}\times\left(1-\frac{1}{2}\times\left(1-\frac{1}{4}\right)\right) = 1-\frac{1}{2}\times\left(1-\frac{1}{2}\times\left(1-\frac{1}{2}\times\frac{1}{2}\right)\right)\end{aligned}$$

This can be represented as a logic circuit with four (1/2)-coins as inputs, as shown in Figure 6.

Because we are only using (1/2)-coins as inputs, and NOT operators leave the denominator invariant while AND operators multiply the denominators, this procedure can only produce constants that can be written as $k/2^n$. By the same arguments it can also be deduced that the number of (1/2)-coins required as inputs is always equal to n. In fact, it is the case that any constant that can be written as $k/2^n$ (equivalently, having a terminating binary expansion) can be simulated by such a circuit, although this is not obvious.

Nonetheless, accepting this proposition, it is easy to show that any constant can therefore be approximately simulated to arbitrary precision, and that the approximation error decreases exponentially in the number of (1/2)-coins used. Given $c \in (0,1)$ and tolerance ε, we truncate the binary expansion of c after $-\lfloor\log_2(\varepsilon)\rfloor - 1$ places, and write the resulting constant as a fraction. This fraction will necessarily be of the form $k/2^n$, with $n \leq -\lfloor\log_2(\varepsilon)\rfloor - 1$, and the approximation error is bounded by $|c-(k/2^n)| \leq 2^{-n-1} \leq 2^{\lfloor\log_2(\varepsilon)\rfloor} \leq \varepsilon$. By our proposition, this fraction can be simulated using at most $-\lfloor\log_2(\varepsilon)\rfloor - 1$ queries of a (1/2)-coin. Therefore truncating the binary expansion of c to one more place decreases the approximation error by a factor of 2, and we conclude that the approximation error decreases exponentially in the number of (1/2)-coins used.

It is also possible to construct the same class of constants using a combination of OR and NOT operators. Figure 7 shows the logic circuit representing a decomposition of the same constant 11/16 using OR operators instead of AND operators. Although the OR gates are labelled $+$, this is a logical addition and doesn't correspond to a mathematical addition in the way AND corresponds to $\times$. The number of inputs is the same as for the AND circuit, for the same reasons, and so the same result holds regarding the scaling of the approximation error for constants that can't be simulated exactly.

The principle in computer science states that any logic circuit can be decomposed into only NAND or only NOR gates; however these representations typically use many more logic gates than their unconstrained counterparts. With application to manufacturing computer processors, it turns out to be better to use more gates of the same

type; but for this application we are quite happy to mix operators if it saves computing time — although it is worth keeping in mind that our primary concern is the number of p-coin queries, not the amount of post-processing, which is typically much cheaper. Nonetheless a natural question is, (when) can we reduce the number of operations by mixing operators, and by how much?

References

Keane, M. and O'Brien, G. L. (1994), 'A bernoulli factory', *ACM Transactions on Modeling and Computer Simulation (TOMACS)* **4**(2), 213–219.

Von Neumann, J. (1951), 'Various techniques used in connection with random digits', *Appl. Math Ser* **12**(36-38), 3.