EXACT SAMPLING AND PREFIX DISTRIBUTIONS

by
Sebastian Oberhoff

A Thesis Submitted in

Partial Fulfillment of the

Requirements for the Degree of

Master of Science

in Mathematics

at

The University of Wisconsin-Milwaukee

May 2018

ABSTRACT

EXACT SAMPLING AND PREFIX DISTRIBUTIONS

by

Sebastian Oberhoff

The University of Wisconsin-Milwaukee, 2018
Under the Supervision of Professor Chao Zhu

This thesis explores some new means to generate random numbers without incurring any numerical inaccuracies along the way. In the context of continuous distributions this leads to the discussion of prefix distributions – discrete distributions that fully capture a continuous distribution by describing their initial digits. These are first studied graphically, then analytically, which also leads to a general examination of the behavior of the distribution of trailing digits of continuous distributions. Finally, some slightly novel, related results from the theory of computation are presented.

TABLE OF CONTENTS

LIST OF FIGURES

1 Introduction

Imagine we had access to an infinite supply of independent random bits equally likely to be 0 or 1. And suppose we were asked to simulate a weighted random bit with probability $p = 0.25$ for 0 and 0.75 for 1. Using the infinite supply of bits at our command this is a straightforward task. Simply map the first two bits as follows:

$$00 \to 0$$
$$01 \to 1$$
$$10 \to 1$$
$$11 \to 1$$

This much is easy. But what if p was a more complicated number such as $1/\pi$? This seems like a much more challenging task. Indeed, here is a "proof" that it is impossible to perform this task exactly.

Any finite computation can only examine a finite number, say n, of the input bits. This sequence of n input bits takes on every possible value with equal probability of 2^{-n}. No matter what computation we perform on this sequence, ultimately 0 will result with some probability that is a multiple of 2^{-n}. But $1/\pi$ isn't a multiple of 2^{-n} for any n. Even though we can make the error arbitrarily small, some will always remain.

This argument seems very persuasive. And yet there is a loophole. Consider the following procedure.

First, try thinking of the infinite sequence of random bits we have access to as an exact sample from the uniform distribution on the interval $[0, 1]$, which has been given in binary. This sample falls below $1/\pi$ with probability $1/\pi$ exactly. But how do we compare two infinitely long sequences of digits? Here's how: Begin by writing out a few digits of the binary representation of $1/\pi$.

$$\frac{1}{\pi} = 0.0101000101...{}_2$$

Then compare the bits beyond the decimal point with the random bits we've been given.

$$\begin{array}{cccccccccc} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow \\ b_0 & b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & b_8 & b_9 \end{array}$$

If all of these bits miraculously agree with each other, just compute some more bits of $1/\pi$ and compare these with more random bits. Sooner or later we'll find a disagreement between these two sequences. At that point we know which is larger, the rest of the infinite sequence of bits of either number doesn't matter!

We'll refer to this trick, originating as far as we are aware from [1], as *exact Bernoulli sampling*. It circumvents the "proof" given earlier by potentially running arbitrarily long. And yet it is by no means infeasible. The number of required comparisons follows a geometric distribution with $p = 1/2$, so the expected number of required comparisons is a mere 2. Also note that we didn't depend in any way on special properties of $1/\pi$. We could have proceeded in the same way with any computable number imaginable as our p. The only component that can cause worry is the complexity of computing digits of p - though we won't meet any examples where this becomes a critical issue here. The plan for the rest of this discussion is to take exact Bernoulli sampling and milk it as much as possible.

2 Exact Discrete Sampling

2.1 Exact Bisection Sampling

Say our next challenge was to draw exact samples from the Poisson distribution with parameter $\lambda = 5$. Here's how we could do it. First, find a rational number that is close to the median. (Using rationals just makes the procedure simpler.) As figure 1 demonstrates, using 4.5 as our midpoint will do nicely.

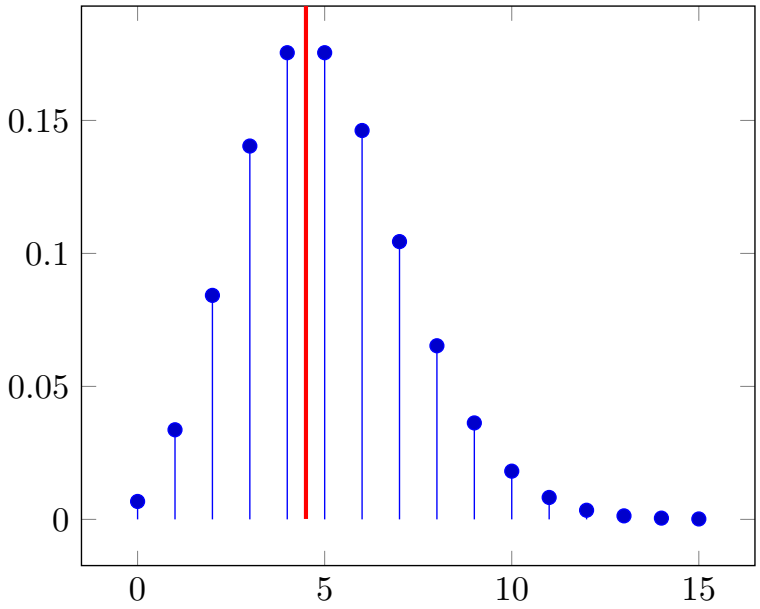

Figure 1: Approximate bisection of the Poisson distribution with $\lambda = 5$ using 4.5 as the mid point

We can then compute the probability that any sample will fall on the left side of this line by computing the leading bits of the cumulative distribution function at 4.5.

$$F(4.5) = 0.0111000011..._2$$

where F is the distribution function of the Poisson distribution with $\lambda = 5$. Using a few of these bits together with exact Bernoulli sampling we can now determine which way to proceed with our search.

For the purpose of illustration, let's assume that our first bisection led us to the right. We then rescale our probabilities by conditioning on the fact that our sample is greater than 4.5 and bisect this new probability distribution as in figure 2.

At this point the idea should be clear. Keep bisecting the remaining range until only a single integer is the sole remaining possibility. That will then be our sample. As long as we don't choose our mid points in a completely silly fashion the number of steps we'll be walking off to the right will follow an approximately geometric distribution. And as soon as we turn back around we'll terminate very soon after.

Of course, none of these considerations were very particular to the Poisson distribution with $\lambda = 5$. So this method is evidently applicable to virtually any discrete distribution you could name.

2.2 Exact Discrete Inverse Sampling

Another procedure that is in many respects quite similar can be constructed from inverse sampling. That is, if $X \sim \mathrm{U}[0, 1]$, then $F^{-1}(X)$ will be distributed according to the Poisson distribution with $\lambda = 5$ (to be exact F^{-1} must be defined to return the smallest $n \in \mathbb{N}$ such that $F(n) \geq X$ (we use the convention that $\mathbb{N}$ includes 0). One way to visually represent this is by labeling the unit interval as in figure 3. Then we drop a sample X onto the line, observe which interval the sample fell in, and choose the next largest corresponding n as our sample.

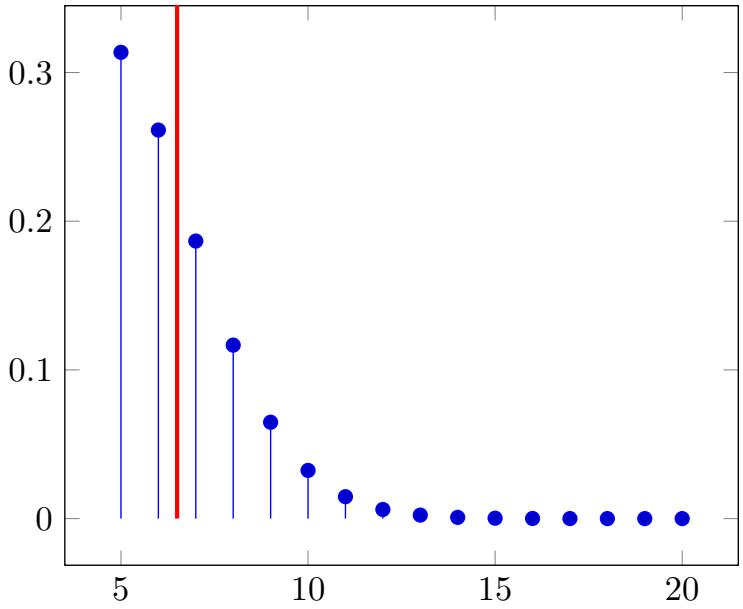

Figure 2: Approximately bisect the Poisson distribution above 4.5 using 6.5 as the new mid point

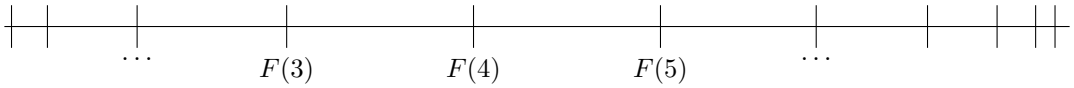

Figure 3: The unit interval divided into sections according to the Poisson distribution function when $\lambda = 5$

We can easily make this procedure exact by sampling the digits of X one by one and computing the leading digits of F for the various relevant values of n. Eventually we'll have X completely fenced in between $F(n)$ and $F(n+1)$ for some n, at which point none of the remaining digits of any of the involved quantities matter, so we can return $n+1$.

3 Exact Rejection Sampling

At this point we'll turn our attention to continuous distributions. Immediately a problem presents itself here. That is a sample from a continuous distribution will typically be an irrational number with infinitely many digits in its decimal representation. How can we hope to draw an exact sample if just printing it out is an indeterminate process?
Clearly, we'll have to manage our expectations as to what we can accomplish here. But if all we ask for is a method that will produce arbitrarily many correct digits of an exact sample, then there are already plenty of methods available.

Take the well known inverse sampling method. In the case of the exponential distribution with $\lambda = 1$ this method amounts to simply computing $-\ln(1 - X)$ where $X \sim \mathrm{U}[0, 1]$. This task can be performed to arbitrary precision. So in the sense just described exact sampling would be quite uninteresting.

But there's another sense in which exact sampling can be accomplished which isn't quite as trivial. We can try to sample just the leading digits of an exact sample so that the remaining digits then no longer depend on the distribution we're sampling from. In particular we'll see a way to sample leading digits, which we'll call a prefix, so that the remaining digits will be uniformly random. A prefix thus completely captures the characteristics of the distribution and so can arguably be referred to as an exact sample.

The tool we'll use to accomplish this is a modification of rejection sampling. Let's focus on the interval [0,1] and let's choose for the purpose of illustration the distribution with density $f(x) = 2x$ as our target. The method of rejection sampling goes back to John von Neumann and is usually stated in terms of some proposal distribution. But we'll focus on the case where the uniform distribution on $[0, 1]$ is our proposal distribution, because that's what our infinite stream of random bits most readily supplies.

Rejection sampling then says to do the following:

1. Draw a sample x from the proposal distribution $\mathrm{U}[0, 1]$
2. Accept x as the sample with probability $\frac{2x}{2} = x$ (we'll do so by comparing $2x$ to another sample from $\mathrm{U}[0, 2]$)
3. If x was rejected, start over

These steps can be visualized as in figure 4. We draw a sample from the uniform distribution on the rectangle enclosing our target density function. If it falls below the density function, the x-coordinate of that point becomes the sample.

There are plenty of expositions in the literature that give a more thorough justification of these steps. The way this is usually implemented is by approximating the uniform distribution using some finite number of bits, inevitably incurring some error. But using exact Bernoulli sampling we can do better.

In essence, we'll consider the rectangle defined by the leading digits of x and y. By drawing additional digits for x and y we can shrink this rectangle arbitrarily small until we're certain that it falls either wholly below or above f. At that point we know whether to accept or reject, without having to examine further digits. Let's walk through an example.

Suppose the leading bits of our uniform sample x for the x-coordinate are 101... . This narrows down our sample to the interval [0.625, 0.75]. We can think of this as defining a δ-environment from basic calculus. This in turn defines a corresponding ϵ-environment [1.25, 1.5] into which $2x$ falls.

Furthermore, let's imagine that the leading bits of our second sample y, drawn from U[0, 2], were 1001... . This allows us to narrow down y to [1.125, 1.375]. Figure 5 shows what's going on.

Now these intervals for $f(x)$ and y overlap. So we can't say for certain whether x should be rejected or not. But that's easy to remedy. Just draw additional bits for x and y. Sooner or later (and chances are sooner) these intervals will separate. And then we'll know whether to accept x or to try again.

But note that something remarkable happens as soon as we accept x. At that point we've determined some of the leading bits of x through careful testing with the rejection method. But the remaining bits can now be drawn from our infinite random bit supply without any further thought! We can forget all about the function $f(x) = 2x$ and just pump out the remaining bits of our sample as fast as our random number generator is able to mint them.

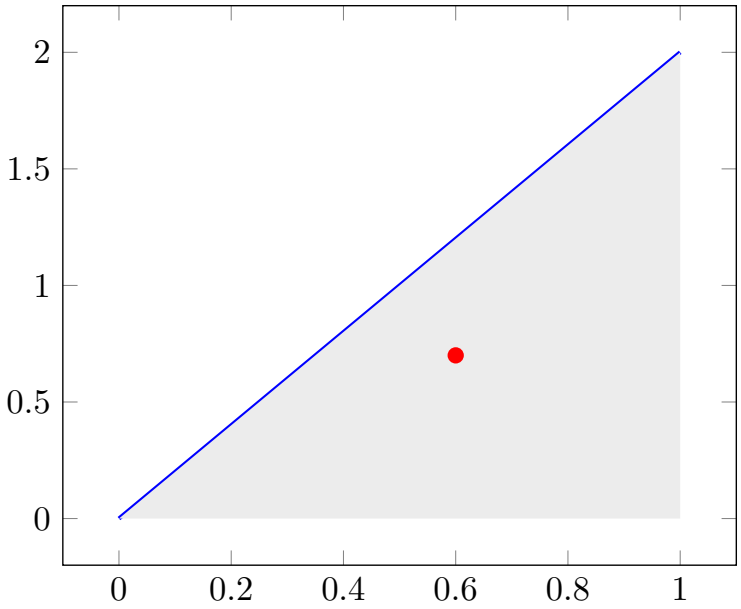

Figure 4: Rejection Sampling — Samples that fall in the shaded region (as shown) are accepted

The previous discussion was loosely inspired by work from C. Karney in [2]. Though Karney only focuses on the normal distribution and its counterpart, the discrete normal distribution, and doesn't employ rejection sampling. Instead it uses a less general tailor made algorithm.

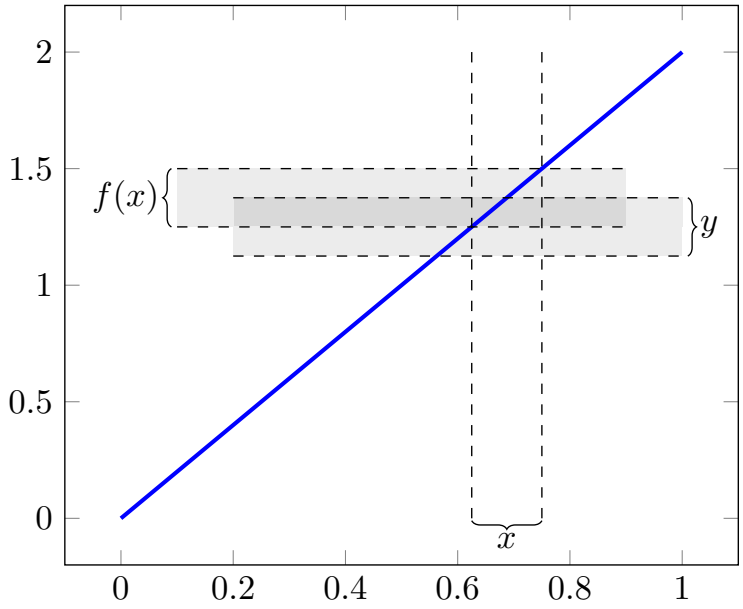

Figure 5: Having drawn $x = 0.101..._2$ and $y = 1.001..._2$ we know that $f(x)$ and y fall somewhere into these intervals

4 Putting It All Together

Now let's take these two previous ideas and put them together. How could we draw an exact sample from the standard normal distribution? At this point it's simple. First, use exact bisection sampling and perform a random walk on the integers, starting at 0. As soon as the sample has been narrowed down to an interval of unit length, the integer part of the sample has been determined. Then use exact rejection sampling to determine the fractional part. That's all there is to it. We can now easily produce the digits of an exactly standard normally distributed sample to our hearts content.

At this point it seems worth reflecting a little more generally. First, because we're eventually just passing through random bits, this algorithm can easily be seen to be asymptotically optimal. We'll examine the exact meaning of "eventually" in section 7. For now, let's direct our attention to the bits required to sample x during the rejection sampling stage. A few of these bits will be discarded while we iterate the rejection method. But the rest is then returned completely unmodified. A different way of looking at this is that we're taking a sample from the uniform distribution and sawing off a little piece at the front. The tail end is then precisely the fractional part of a random variable having practically any desired continuous distribution.

Furthermore, we could have used the exact same sequence to also draw an exact sample from some other unrelated distribution. These two samples would then be correlated in the curious way that, at some offset, eventually all of their digits coincide. Stated differently, if one looks at a sample from the uniform distribution, chances are that, *within eyesight*, one can see the superimposed beginnings of samples from any continuous distribution one can care to mention.

Finally, consider the following. After having determined the leading digits of a sample from some distribution, we could write them down on a slip of paper and pass it to somebody else. We'll call such string of digits a *prefix*. That person wouldn't even have to know which distribution the prefix was drawn from. They would still hold in their hands an *exact* sample. If they're unsatisfied with the precision of the prefix we gave them, they can just add their own random digits! In very much the same way that the symbol π summarizes an infinite sequence of digits, so do the digits on that slip of paper.

As an aside, it is also possible, in a sense, to reconstitute a prefix once many random digits have already been attached. Though this may easily be a different prefix and one does now need the density f of the distribution from which the sample is drawn. The idea is to simply perform rejection sampling using f as a proposal density and the uniform as the target distribution. As soon as a uniform has been successfully drawn, one can be confident that the remaining, unexamined digits are uniform, thus the digits consumed so far constitute a prefix. One caveat here is that one has to condition and rescale f every time a rejection takes place, because, unlike with the uniform distribution, the trailing digits of f are unlikely to again be distributed according to f. While this would be a daunting challenge to program, considering one requires arbitrary precision, there is nothing that prevents this in principle.

Now it is tempting to refer to the distribution of numbers that we could write on that slip of paper as a *prefix distribution*. There's just one slight wrinkle that we have to iron out before we're fully justified to make that designation. The problem is that distinct prefixes may denote the same real number. Take the prefixes 0 and 00. Both of these designate distinct prefixes. One cannot replace one with the other without altering the underlying distribution that one is sampling. But if forced to map these to real numbers one would end up on the same point on the number line.

The way out here is to realize that we're free to add some random bits of our own before passing along the slip of paper. So all we do is sample bits until we hit a 1. At that point our prefix designates a unique real number and we terminate. In effect we've just done a little part of the other person's job. Let's refer to these elongated prefixes as *padded prefixes* as opposed to *unpadded prefixes* for their unmodified counterparts. If it is clear from context we'll also often just use the term *prefix* for either version.

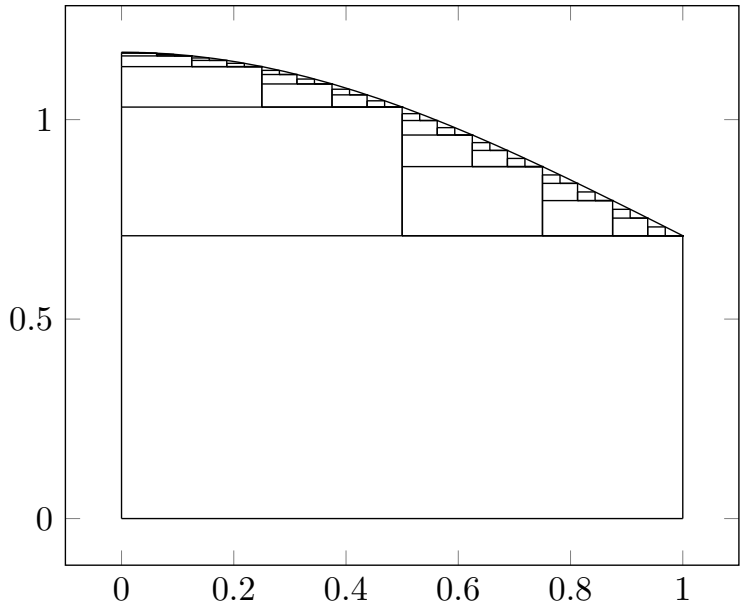

Figure 6: The normal distribution on the interval $[0, 1]$ divided up according to which sections produce distinct prefixes (prefixes up to length 6 have been plotted)

5 Graphing Prefix Distributions

It's usually best to begin building an intuition for something new by producing some visual aids. So what we'll start with is to find ways to graph prefix distributions. For this we'll have to find a way to compute the probabilities of various prefixes occurring. Perhaps surprisingly these probabilities can be represented in a quite natural visual form.

5.1 Unpadded Prefix Probabilities

Let's stick to the example of the normal distribution on the interval $[0, 1]$. And let's momentarily focus only on unpadded prefixes. Then figure 6 shows a graphical representation of the probabilities for various prefixes.

For the following discussion it will be useful to think of our algorithm for exact rejection sampling as being very generous when sampling digits of y, whereas it only samples the absolute minimum required amount of digits for x. And recall that when performing rejection sampling we're eventually placing a uniformly random point on the area below the graph and then using the x-coordinate as our sample.

The large box at the bottom of figure 6 covers all the positions for which we return the empty prefix. It cuts off at the top just at the value 0.709 which is the minimum of the normal distribution on the interval $[0, 1]$ when conditioned on that interval. The reason this box corresponds to the empty prefix is that if we begin drawing samples for the y-coordinate of our uniform sample in $[0, 1] \times [0, 1.169]$ and we find that y falls below this line, then we can already conclude that we're going to accept the x-coordinate as our normal sample, regardless of where it falls. So we don't even need to draw a single digit for x. On the other hand, if y falls above 0.709, then we need further information about x in order to decide whether to accept x or discard it and try again. Since this requires sampling at least a single bit of x, the area above 0.709 corresponds entirely to other prefixes.

Let's discuss one more example. The second largest box in figure 6, situated in the top left, corresponds to the prefix 0. Here y fell above 0.709, so we had to sample the first digit of x. This turned out to be 0, narrowing down our search to the left half. We then sampled y to sufficient accuracy to determine that it fell below 1.031, the value at the midpoint of our distribution function. At that point we're again guaranteed to accept x no matter its further digits. In a similar fashion we can cover the entire area with smaller and smaller rectangles.

Thus, the general method for finding the rectangle corresponding to a given prefix p and density function f is:

- The left boundary is given by attaching infinitely many zeros, or equivalently by mapping p to the directly corresponding real number.
- The right boundary is given by attaching infinitely many ones, or equivalently adding 2^{-n} if p is of length n.
- The bottom boundary is given by removing the last digit of p and finding the upper boundary of that prefix.
- The upper boundary is given by finding the minimum of f over the interval delineated by the left and right boundaries.

Any sample from $\mathbb{R}^2$ that falls to the left or right of these boundaries will have an x-coordinate that disagrees with p. Any sample that falls within the interval, but below the lower boundary, will have been accepted before sampling all of p. And any sample that falls within the interval, but above the upper boundary, will require more information about the digits of the x-coordinate than p can provide.

Conversely, any sample from $\mathbb{R}^2$ that falls within these boundaries corresponds to p, because, by the same arguments as in the previous paragraph, it doesn't correspond to any other prefix.

Furthermore, because we're picking the sample uniformly in the area below f and because the total area below f is exactly 1, the area of each of these rectangles is exactly equal to the probability that we'll pick the corresponding prefix.

This gives us not only a simple visualization of the probabilities involved, but also the means to compute these probabilities. In the following we use $\#p$ to denote the number of digits of p. p_{-k} denotes p with k digits removed at the end. And $[p]$ is shorthand for the interval prefixed by p - that is $[p] \coloneqq [p, p + 2^{-\#p}]$ where the right-hand side is abusing notation slightly to use p as a real number, not just as a string of bits. Then for a given unpadded prefix p its probability to be picked is

$$P_{\text{up}}(p) \coloneqq \underbrace{2^{-\#p}}_{\text{width of rectangle}} \cdot \underbrace{(\min_{[p_{-1}]} f - \min_{[p]} f)}_{\text{height of rectangle}} .$$

(The up in P_{up} stands for unpadded.) The only nontrivial hurdle that needs to be taken is to find means to minimize the given distribution function f on arbitrary intervals. Though all the distribution functions we'll examine here will be so well behaved that this will be of no real practical concern.

5.2 Padded Prefix Probabilities

Now that we've seen how to determine the probabilities of unpadded prefixes we're ready to tackle the probabilities of padded prefixes. The only difference between these was that if a unpadded prefix didn't terminate with a 1, we would attach uniformly random bits until we hit a 1, giving us a padded prefix.

Let's interpret what this means in terms of figure 6. Take again the large box at the bottom corresponding to the empty prefix. This prefix doesn't end with a 1 (a special case which doesn't end with a 0 either), so it'll have to be padded. Half of the padded prefixes will have a 0 as their first bit, the other half a 1. We can visualize this by splitting the area corresponding to the empty prefix down the middle and erasing the top boundary. Doing this we're in effect "adding" half the probability of the empty prefix to the prefix 0 and half to the prefix 1.

The prefix 1 ends with a 1, so we're done there. This is not so with the prefix 0. It will have to be split again and have its upper boundary erased. The same will have to be done with the prefix 00, 000, etc. but also the prefix 10, 1010 and any other prefix ending with a 0. Performing this splitting and merging recursively for figure 6 ultimately produces what can be seen in figure 7.

The fact that padded prefixes correspond to unique real numbers can now be appreciated visually. Whereas in figure 6 there where many rectangles whose right-end boundaries were above the same point on the x-axis, in figure 7 there's an infinite recursion occurring above each rectangle on the right. So all right-hand boundaries, if projected down onto the x-axis, would now hit a unique real number.

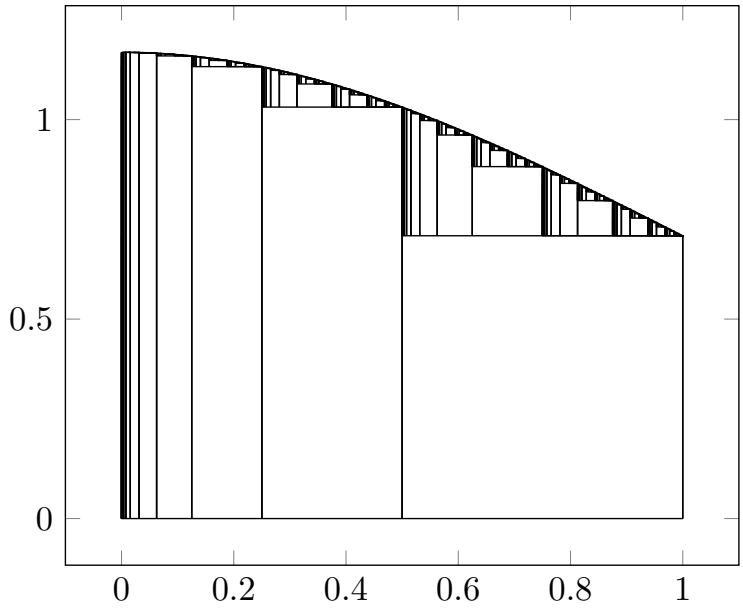

Figure 7: The normal distribution on the interval $[0, 1]$ divided up according to which sections correspond to which padded prefixes (prefixes up to length 9 have been plotted)

Indeed, one could use figure 7 to devise a slightly alternate algorithm to sample from prefix distributions. One could draw a uniformly random sample from the region below f using rejection, determine the sample until it was fully contained within one of the rectangles, and then return the x-coordinate of the right-hand boundary as the sample.

With this image in mind we'll now turn to actually computing the probabilities involved. Given that we already know how to compute probabilities for unpadded prefixes using P_{up}, it is relatively straightforward to compute the probabilities of padded prefixes. For a prefix p return

- 0 if p doesn't end with a 1
- $$\sum 2^{-k} \cdot P_{\text{up}}(p_{-k})$$

  if p does end with a 1. The sum here ranges over k starting at 0 until p_{-k} either reaches the empty prefix (included) or another prefix ending with 1 (excluded).

The summation here can be thought of as scanning backward from p. Every prefix p_{-k} has a $P_{\text{up}}(p_{-k})$ chance to get picked and a 2^{-k} chance to lead to the prefix p.

For instance, if $p = 1001$, then it can either be picked straight away with probability $P_{\text{up}}(1001)$. Or the prefix 100 might be picked, leading to 1001 with probability $1/2$. Or 10 could be picked, which leads to 1001 with probability $1/4$.

We're now finally in the position to graph a prefix distribution. Figure 8 illustrates the probabilities of the various prefixes of the normal distribution on the interval $[0, 1]$. One interesting feature of this graph is that it is quite self-similar at small scales. This was more obvious for the preceding graphs, it is a little less clear to the naked eye here. The little clusters that can be seen, for instance, above 0.6 and 0.8 are, if magnified, almost exact copies of the entire graph.

Similarly, we can produce a graph for the cumulative distribution function as seen in figure 9. This picture looks slightly crooked which is just the result of many small jumps giving the impression of a continuously upward sloping function.

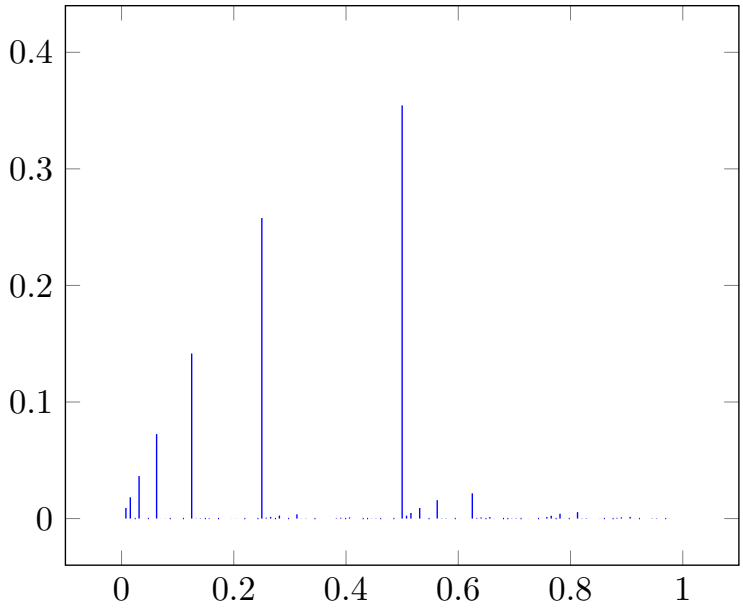

Figure 8: The probability mass function of the prefix distribution of the normal distribution on the interval $[0, 1]$ (prefixes up to length 7 have been plotted)

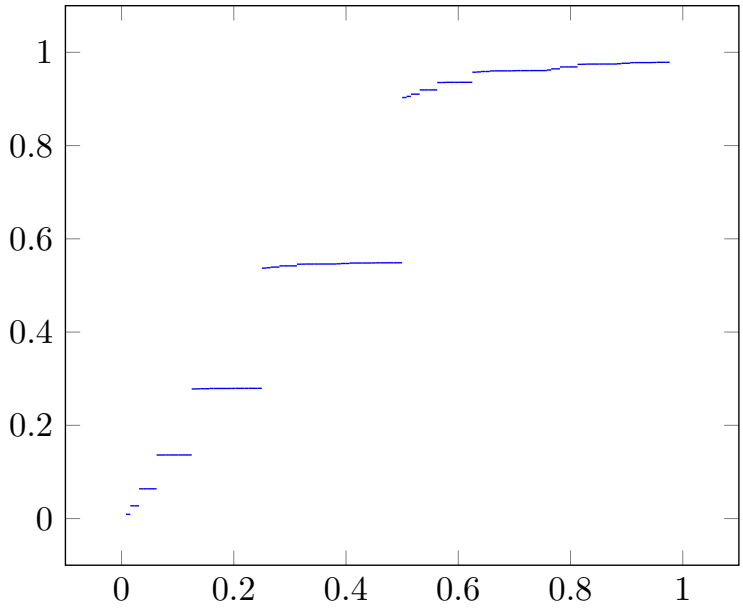

Figure 9: The cumulative distribution function of the prefix distribution of the normal distribution on the interval $[0, 1]$ (probabilities for prefixes up to length 7 have been summed)

5.3 Plots, Plots, Plots

Finally, we can can use the tools we've seen so far to produce plots not only for the interval $[0, 1]$ but for all of $\mathbb{R}$. This is done simply by dividing $\mathbb{R}$ up into unit length intervals from one integer to the next. Within each of these intervals we can produce a plot for the conditional distribution on that interval. We then simply piece all of these plots together, producing the final result.

The area in each rectangle will shrink as we rescale the conditional distributions so that the entire area below the graph will again be 1. But in every case it will be shrunk by exactly the proportionate amount,

accurately reflecting the probability that a given prefix, now including the integer component, will be chosen.

The following figures show this idea applied to all the plots we've seen so far in this section. Also, plots for both the exponential distribution with $\lambda = 1$ and a sinusoidal distribution with density $f(x) = \sin(4\pi x) + 1$ on the interval $[0, 1]$ have been added for comparison.

Finally, it is worth considering the fact that, unpadded prefixing all the way from the very beginning, we could have had this entire discussion not just in base 2, but in any base whatever. This would've produced completely analogous algorithms for sampling exactly from discrete distributions. And it would've produced an analogous algorithm to perform exact rejection sampling, in turn producing analogous prefix distributions. We'll end this section by showcasing a few more graphs (figure 22 and following) of the normal distribution on the interval $[0, 1]$ in base 10, which illuminate the logic behind prefix distributions somewhat more vividly than the binary paradigm.

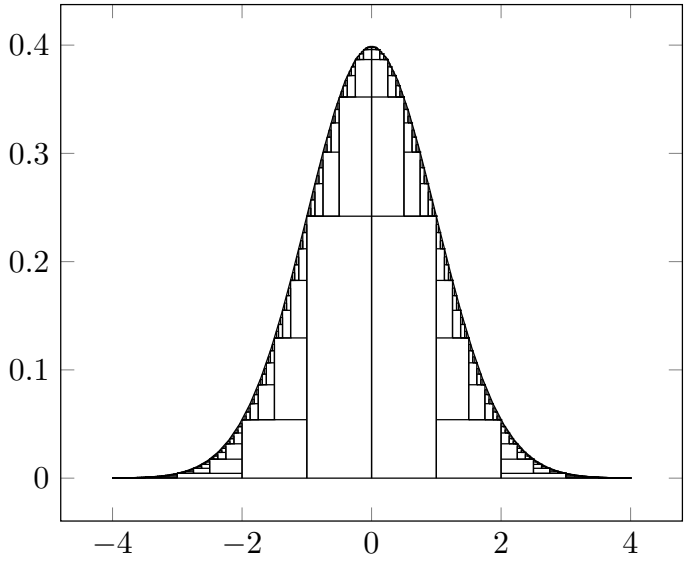

Figure 10: The normal distribution on the interval $[-4, 4]$ divided up by the unpadded prefix probabilities (prefixes up to length 5 have been plotted)

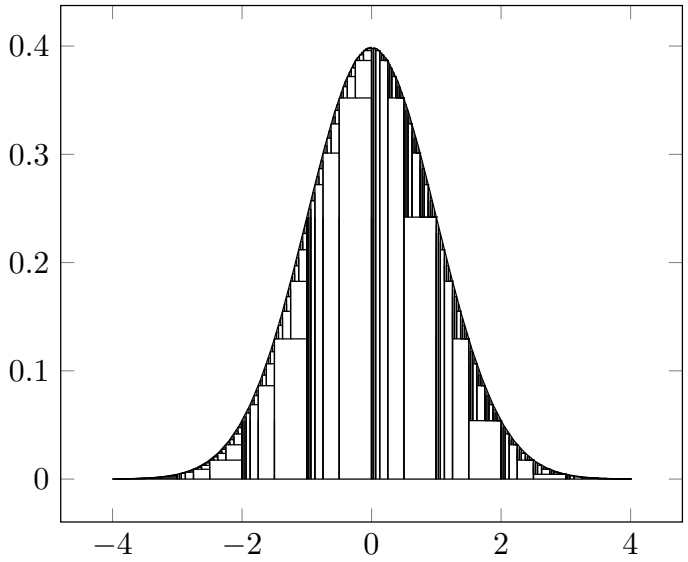

Figure 11: The normal distribution on the interval $[-4, 4]$ divided up by the padded prefix probabilities (prefixes up to length 5 have been plotted)

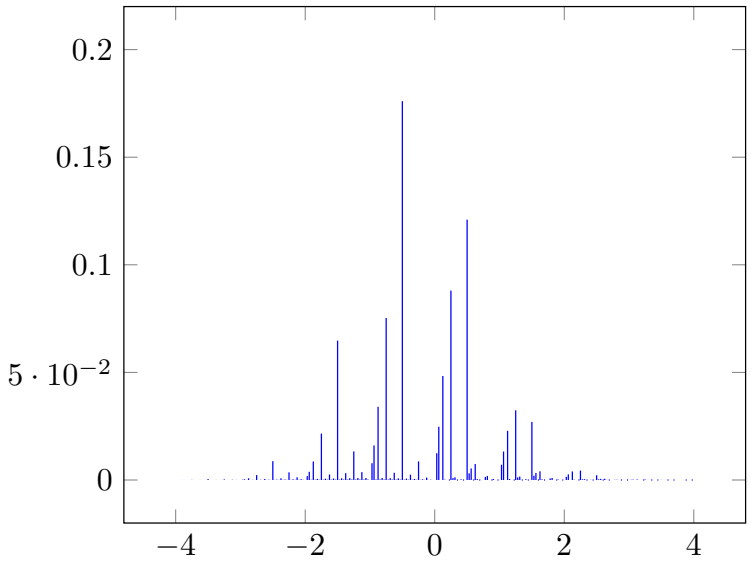

Figure 12: The probability mass function of the prefix distribution of the normal distribution on the interval $[-4, 4]$ (prefixes up to length 5 have been plotted)

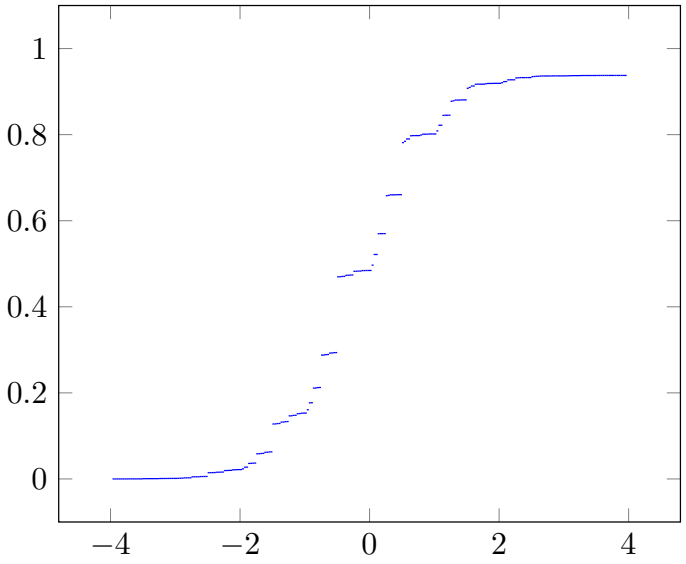

Figure 13: The cumulative distribution function of the prefix distribution of the normal distribution on the interval $[-4, 4]$ (probabilities for prefixes up to length 5 have been summed)

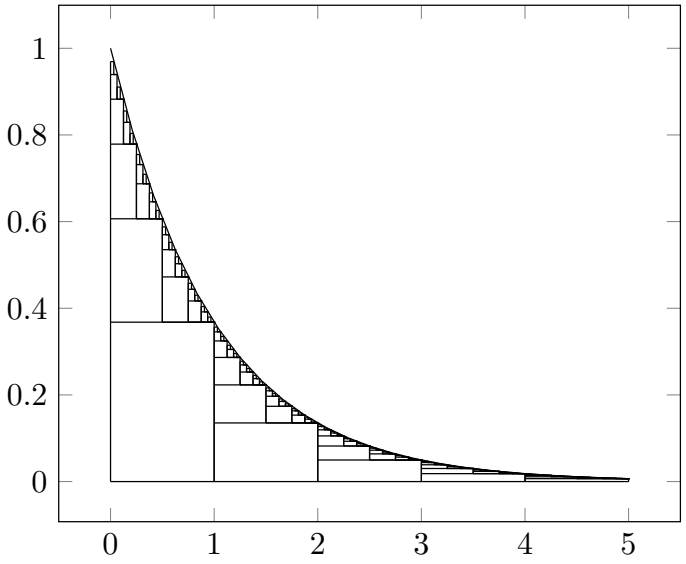

Figure 14: The exponential distribution on the interval $[0, 5]$ divided up by the unpadded prefix probabilities (prefixes up to length 5 have been plotted)

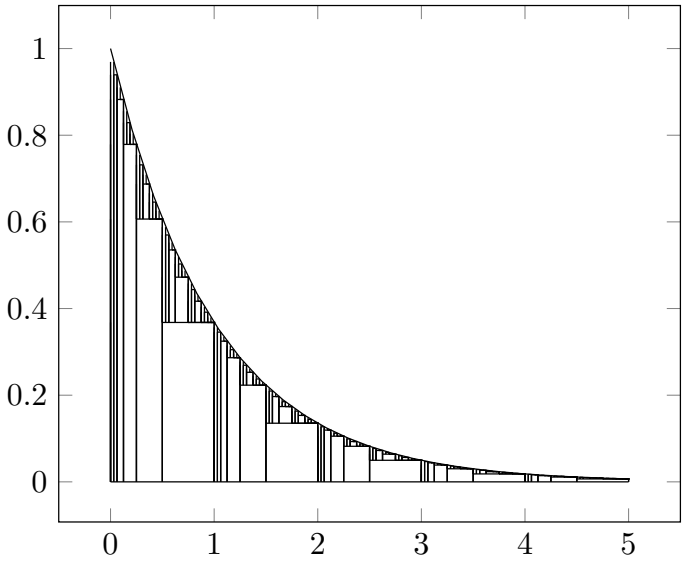

Figure 15: The exponential distribution on the interval $[0, 5]$ divided up by the padded prefix probabilities (prefixes up to length 5 have been plotted)

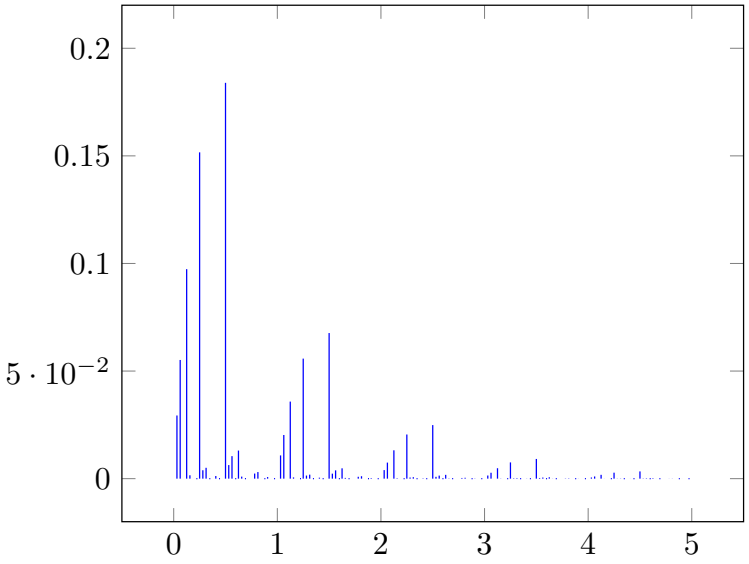

Figure 16: The probability mass function of the prefix distribution of the exponential distribution on the interval $[0, 5]$ (prefixes up to length 5 have been plotted)

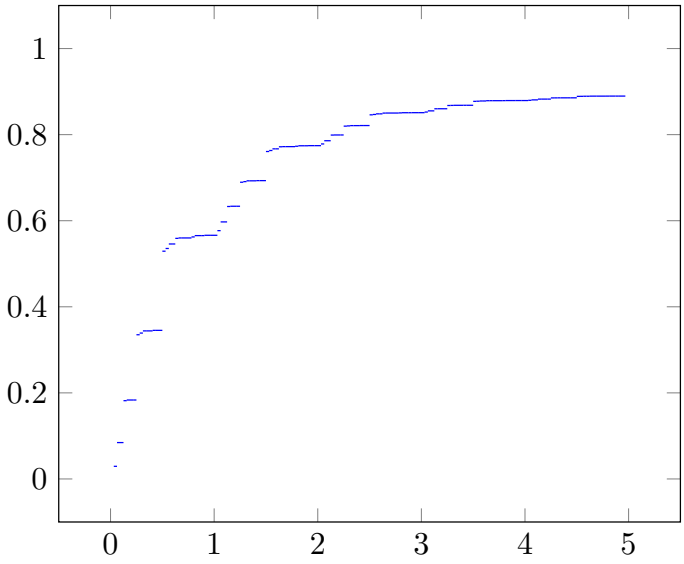

Figure 17: The cumulative distribution function of the prefix distribution of the exponential distribution on the interval $[0, 5]$ (probabilities for prefixes up to length 5 have been summed)

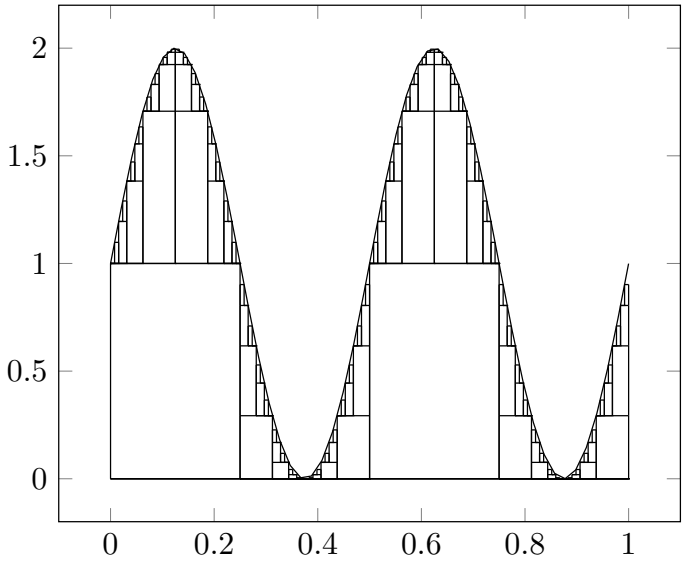

Figure 18: The sinusoidal distribution on the interval $[0, 1]$ divided up by the unpadded prefix probabilities (prefixes up to length 7 have been plotted)

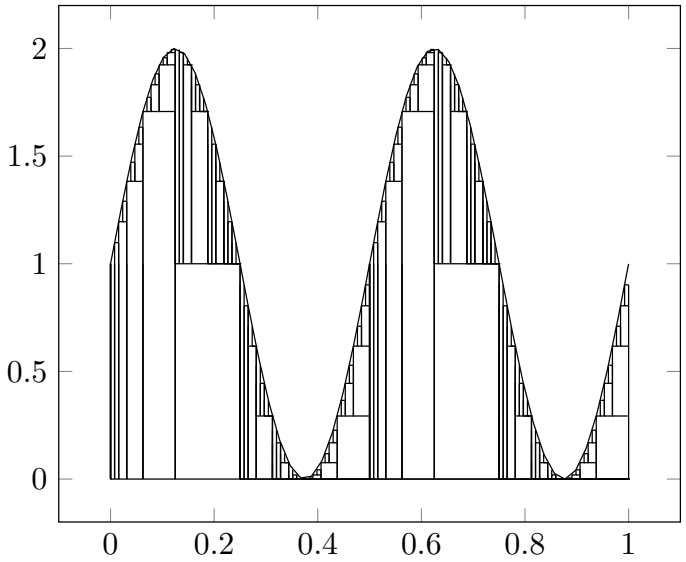

Figure 19: The sinusoidal distribution on the interval $[0, 1]$ divided up by the padded prefix probabilities (prefixes up to length 7 have been plotted)

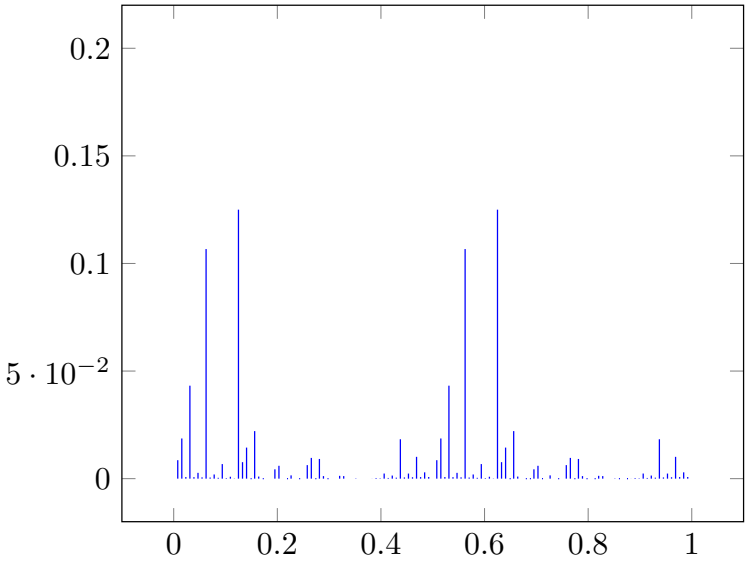

Figure 20: The probability mass function of the prefix distribution of the sinusoidal distribution on the interval $[0, 1]$ (prefixes up to length 7 have been plotted)

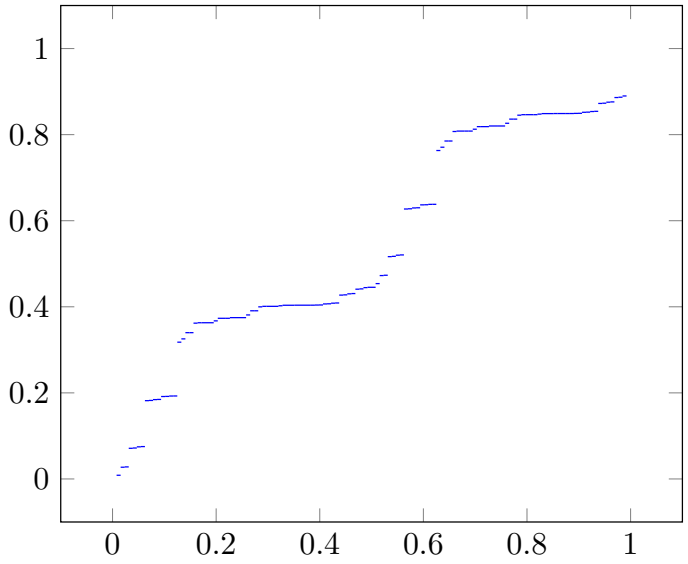

Figure 21: The cumulative distribution function of the prefix distribution of the sinusoidal distribution on the interval $[0, 1]$ (probabilities for prefixes up to length 7 have been summed)

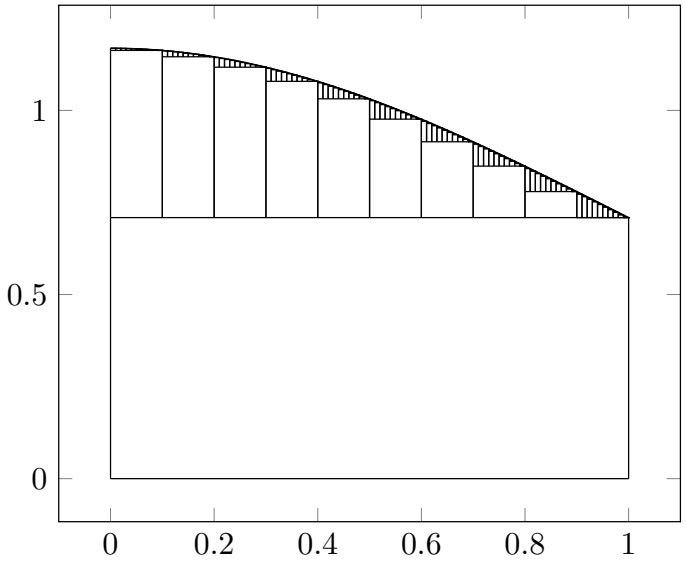

Figure 22: The normal distribution on the interval $[0, 1]$ divided up by the unpadded prefix probabilities in base 10 (prefixes up to length 3 have been plotted)

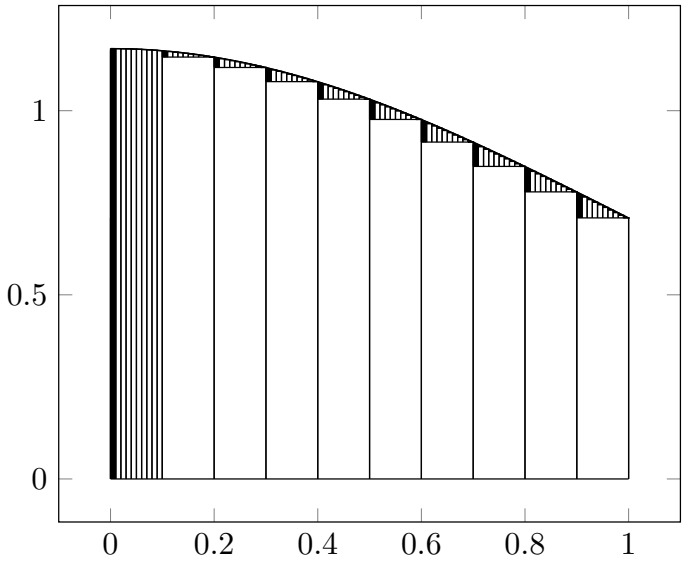

Figure 23: The normal distribution on the interval $[0, 1]$ divided up by the padded prefix probabilities in base 10 (prefixes up to length 3 have been plotted)

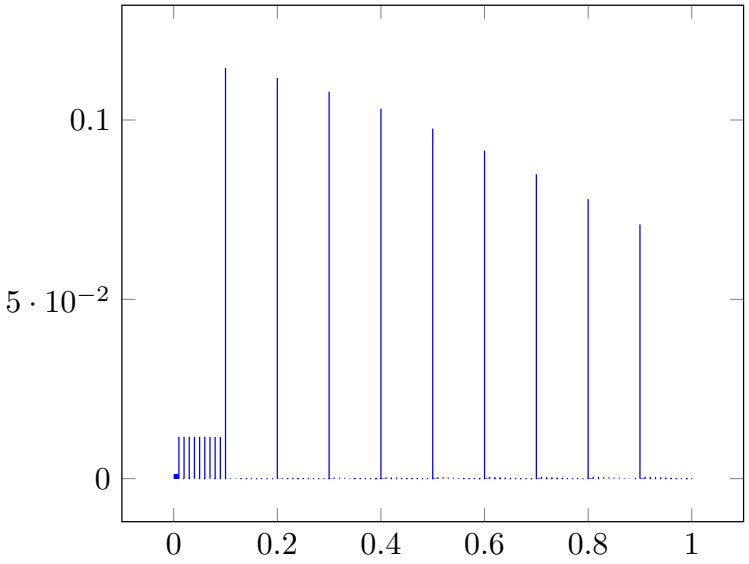

Figure 24: The probability mass function of the prefix distribution of the normal distribution on the interval $[0, 1]$ in base 10 (prefixes up to length 3 have been plotted)

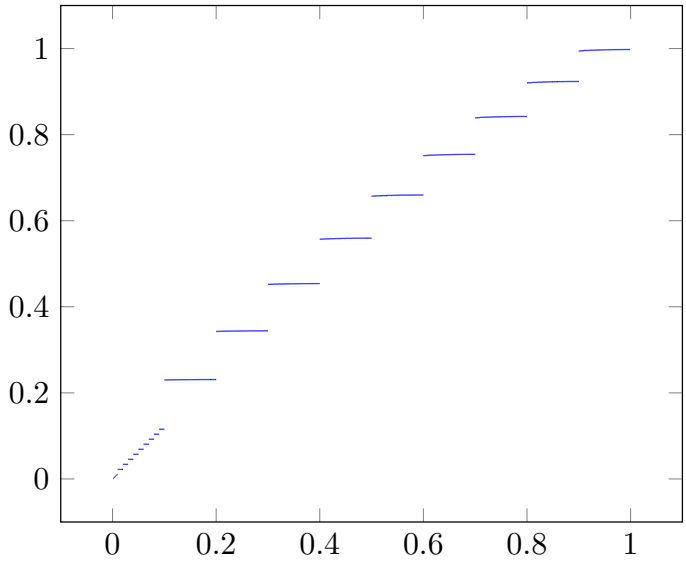

Figure 25: The cumulative distribution function of the prefix distribution of the normal distribution on the interval $[0, 1]$ in base 10 (probabilities for prefixes up to length 3 have been summed)

6 Letting The Base Approach Infinity

Looking at figures 23, 24, and 25 it seems evident that, if one increases the base further and further, one gets closer and closer to the distribution one initially unpadded prefixed with. The question is in what sense.

It won't be the case that the probability mass function of the padded prefixes converges back to f, since a) the probabilities in the pmf converge to 0 and b) the pmf never becomes a continuous function. Instead, reminiscent of the central limit theorem, we'll have to argue that the cumulative distribution functions converge.

Proposition 1. *If F is the original distribution function and F_b is the distribution function for padded prefixes in base b, then*

$$\forall x : \lim_{b\to\infty} F_b(x) = F(x)$$

Proof. First, as the base b increases the probability that a padded prefix has length 1 approaches 1. This is a simple consequence of the fact that the probabilities corresponding to length 1 prefixes are effectively a lower Riemann sum. This is particularly clear from figure 23. Since the area under the entire function is 1, so is the limit of the Riemann sums and thereby the limit of the probabilities. The only minor issue is the leftmost interval, which corresponds to prefixes of longer length. But since this interval decreases in length to 0, so does the probability of selecting a prefix in that interval. So for all x, if we let R_b be a random variable distributed according to the base b prefix distribution, we can write

$$\begin{aligned}
\lim_{b\to\infty} F_b(x) &= \lim_{b\to\infty} \mathop{\mathbb{P}}_{b}(R_b \leq x) \\
&= \lim_{b\to\infty} \left[\mathop{\mathbb{P}}_{b}(\#R_b = 1, R_b \leq x) + \mathop{\mathbb{P}}_{b}(\#R_b > 1, R_b \leq x)\right] \\
&= \lim_{b\to\infty} \mathop{\mathbb{P}}_{b}(\#R_b = 1, R_b \leq x)
\end{aligned}$$

where we're again using # to denote prefix length. We then conclude

$$\lim_{b\to\infty} \mathop{\mathbb{P}}_{b}(\#R_b = 1, R_b \leq x) = F(x)$$

because, again, $\mathop{\mathbb{P}}_{b}(\#R_b = 1, R_b \leq x)$ is just a lower Riemann sum on the interval $[-\infty, x]$, which converges to $F(x)$. □

This result may seem quite benign. But it does suggest that if one is on the lookout for any surprising features one should stick to small bases, since larger bases more and more just resemble the original distribution.

7 The Distribution Of Prefix Lengths

So far we haven't worried much about the length of the prefixes we're producing. None of the computations we've performed gave any reason for particular concern since anytime there was a potentially unbounded iteration going on, the number of iterations was roughly geometrically distributed. We'll now double check this intuition.

Let's start with some numerical evidence. We have already seen how to compute the probabilities of specific prefixes. This allows us to estimate the expected length of prefixes of the examples we've seen so far. The following results take into account all unpadded prefixes of length 15 or less:

- Standard normal distribution ≈ 0.1846
- Exponential distribution ($\lambda = 1$) ≈ 0.5626
- Sinusoidal distribution ≈ 3.4979

This isn't a bad start. But computing these values is fairly inconvenient if one doesn't already have the required program at hand. A more general approach seems desirable. To see what kind of general result is possible it's best to first consider what sort of distribution might cause the corresponding prefix lengths to run away and become arbitrarily long.

Consider the continuous distribution whose support is only on the interval $[0, 2^{-n}]$. Sampling a prefix from that interval will necessitate producing at least n digits, since all shorter prefixes run the risk of producing a number outside the target interval when concatenating random bits to it. Hence the expected prefix length can in principle become arbitrarily long.

Note however, that in order for such a density to still integrate to 1 it will have to be spiked extremely sharply. This suggests that we might be able to make progress by adding an assumption that prevents such sharp spikes. One natural way of doing this is by assuming that the derivative of the density be bounded in absolute value or, slightly more generally, that the density be globally Lipschitz-continuous.

Defintion 1. *(Lipschitz-Continuity) We say that a function $f : \mathbb{R} \to \mathbb{R}$ is (globally) Lipschitz-continuous if there exists a K such that for all x, y*

$$|f(x) - f(y)| \leq K|x - y|\,. \qquad K, x, y \in \mathbb{R}$$

We'll also say that such a function is K-Lipschitz. And we'll often say that a function is K-Lipschitz only on some finite interval $[a, b]$.

But even after having added the assumption that the density be Lipschitz-continuous, there's still another issue lurking. The problem is that, instead of being sharply spiked on the interval $[0, 2^{-n}]$, the density might also manage to integrate to 1 by repeating many little spikes on the intervals $[k, k + 2^{-n}]$ for $k \in \mathbb{Z}$. How do we also rule out these?

Again, the most natural way of preventing such pathologies is by insisting that the density have support only on some restricted interval $[a, b]$. However, in contrast with the assumption that f be Lipschitz-continuous, which seemed fairly benign, this new assumption raises the question whether we're throwing out the baby with the bathwater. After all, many of our most cherished distributions, such as the normal or the exponential distribution, *don't* have their densities restricted to an interval.

The way we'll address this issue is by splitting up the real line into intervals of unit length. We then condition on the prefix falling into one of these intervals, use the bounds we obtain for each interval, and put the whole thing back together. We'll see an example of this at the end of this section. But for now, we'll continue by establishing the following.

Proposition 2. *Let f be a probability density and let R be a unpadded prefix for f. Assuming that f is K-Lipschitz and has support $[i, i + 1]$ where $i \in \mathbb{Z}$ it follows that*

$$\mathbb{P}\left(\#R \leq n\right) > 1 - 2^{-n}K$$

Proof. First, without loss of generality we'll assume that $i = 0$. Now consider figures 26 and 27. As these two figures demonstrate, the area covered by the probabilities of length n or less prefixes is identical to the area covered by the lower Riemann sum corresponding to the partition

$$\{k \cdot 2^{-n} | k \in \mathbb{Z}\}.$$

So we can estimate the probability that a prefix has length n or less by simply estimating this lower Riemann sum. In particular, we'll do so by finding an upper bound for $\mathbb{P}(\#P > n)$, which is equivalent to the error of the lower Riemann sum.

Let L_n be the lower Riemann sum with the aforementioned partition. And let m_k and M_k be the min and max respectively of f on the interval $[k2^{-n}, (k+1)2^{-n}]$. Then L_n consists of terms that are each of the form

$$\underbrace{2^{-n}}_{\text{width}} \cdot \underbrace{m_k}_{\text{height}}.$$

On the other hand, the actual area under the curve on such an interval is at most

$$2^{-n} \cdot M_k.$$

Thus the error on each interval is at most $2^{-n}(M_k - m_k)$. Using the fact that f is K-Lipschitz we then get

$$2^{-n}(M_k - m_k) \leq 2^{-2n}K$$

as an upper bound on each interval. Thus, the total error is bounded as follows:

$$\int_0^1 f(x)\,dx - L_n \leq 2^n \cdot 2^{-2n}K = 2^{-n}K.$$

In other words $\mathbb{P}(\#R > n) \leq 2^{-n}K$. Hence $\mathbb{P}(\#R \leq n) > 1 - 2^{-n}K$. □

Also note that all our inequalities become equalities in the case where f is a straight line. The bound on expected values is now essentially just a corollary.

Proposition 3. *Using the previous definitions and assumptions we have for the expected unpadded prefix length*

$$\mathbb{E}[\#R] \leq K$$

Proof. The proof is straightforward:

$$\begin{aligned}
\mathbb{E}[\#R] &= \sum_{n=0}^{\infty} n\,\mathbb{P}(\#R = n) \\
&= \sum_{n=1}^{\infty} n\left(\mathbb{P}(\#R \geq n) - \mathbb{P}(\#R \geq n+1)\right) \\
&= \sum_{n=1}^{\infty} \mathbb{P}(\#R \geq n) \\
&\leq \sum_{n=1}^{\infty} 2^{-n}K \\
&= K.
\end{aligned}$$

□

Finally, we can apply this to a distribution with unbounded support.

Proposition 4. *Using the previous definitions and assumptions, but where f has arbitrary support we have*

$$\mathbb{E}\left[\#R\right] \leq \sum_{i\in\mathbb{Z}} K_i$$

where K_i is the Lipschitz constant of f corresponding to the interval $[i, i+1]$.

Proof. We'll use the fact that the Lipschitz constant for f conditioned on the interval $[i, i+1]$ is the same as the Lipschitz constant of f itself on that interval, but normalized by dividing it by $\mathbb{P}\left(R \in [i, i+1]\right))$.

$$\begin{aligned}
\mathbb{E}\left[\#R\right] &= \sum_{i\in\mathbb{Z}} \mathbb{E}\left[\#R | R \in [i, i+1]\right] \cdot \mathbb{P}\left(R \in [i, i+1]\right)) \\
&\leq \sum_{i\in\mathbb{Z}} \frac{K_i}{\mathbb{P}\left(R \in [i, i+1]\right))} \cdot \mathbb{P}\left(R \in [i, i+1]\right)) \\
&= \sum_{i\in\mathbb{Z}} K_i
\end{aligned}$$

□

The infinite sum $\sum_{i\in\mathbb{Z}} K_i$ may look intimidating. But in all but the most pathological cases this will be a quite manageable finite value. For example, we can bound the sum using the absolute values of the first and second derivatives of f.

Lemma 1. *Given a differentiable function $f : \mathbb{R} \to \mathbb{R}$ we have*

$$\sup_{x\in[a,b]} |f(x)| \leq \int_a^b \frac{|f(x)|}{b-a} + |f'(x)|\, dx$$

for any $a \leq b$ where $a, b \in \mathbb{R}$.

Proof. Because f is differentiable, it is also continuous. So there exists an x_1 such that

$$f(x_1) = \sup_{x\in[a,b]} |f(x)|\,.$$

Furthermore, by the mean value theorem, there exists an $x_2 \in [a, b]$ such that

$$f(x_2) = \frac{1}{b-a}\int_a^b f(x)\, dx\,.$$

Therefore

$$\begin{aligned}
\sup_{x\in[a,b]} |f(x)| - \left|\frac{1}{b-a}\int_a^b f(x)\, dx\right| &= |f(x_1)| - |f(x_2)| \\
&\leq |f(x_1) - f(x_2)| \\
&= \left|\int_{x_2}^{x_1} f'(x)\, dx\right| \\
&\leq \int_{x_2}^{x_1} |f'(x)|\, dx \\
&\leq \int_a^b |f'(x)|\, dx
\end{aligned}$$

which yields

$$\sup_{x\in[a,b]} |f(x)| \leq \left|\frac{1}{b-a}\int_a^b f(x)\, dx\right| + \int_a^b |f'(x)|\, dx \leq \int_a^b \frac{|f(x)|}{b-a} + |f'(x)|\, dx\,.$$

□

This lemma is immediately applicable to expected prefix lengths.

Proposition 5. *Using the previous definitions and assumptions, if f is twice differentiable, we have*

$$\mathbb{E}\left[\#R\right] \leq \sum_{i\in\mathbb{Z}} K_i \leq \int_{\mathbb{R}} |f'(x)| + |f''(x)|\, dx$$

Proof. For a differentiable function f the Lipschitz constant K_i is nothing but $\sup_{x\in[i,i+1]} |f'(x)|$. So we can simply apply the previously established bound to each K_i and sum the results. □

Because f integrates to 1, the only way this upper bound could fail to be finite is if f oscillates with exceedingly vanishing amplitude – aberrant to say the least.

Let's conclude this section by taking the standard normal distribution as an example. First, the sum of the K_i: Because the normal distribution is symmetric we can just sum over the positive intervals and double the result. For the first interval $[0, 1]$ the maximum slope in absolute value is attained at the right-hand side. For all further intervals it is attained at the left-hand side. Therefore the expected prefix length is bounded by

$$2\left(|\varphi'(1)| + \sum_{i=1}^{\infty} |\varphi'(i)|\right) \approx 1.2115\,.$$

Using the integrals as an upper bound instead produces

$$\sqrt{\frac{2}{\pi}} + 2\sqrt{\frac{2}{e\cdot\pi}} \approx 1.7658\,.$$

Similar considerations for both the exponential and the sinusoidal distribution allow us to give the following summary.

	numerical estimate	$\sum_i K_i$	$\int \lvert f'\rvert + \lvert f''\rvert\, dx$
standard normal	0.1846	1.2115	1.7658
exponential	0.5626	1.5820	2
sinusoidal	3.4979	12.5664	108.5310

These examples showcase both scenarios where the upper bounds are reasonably tight and where they are not.

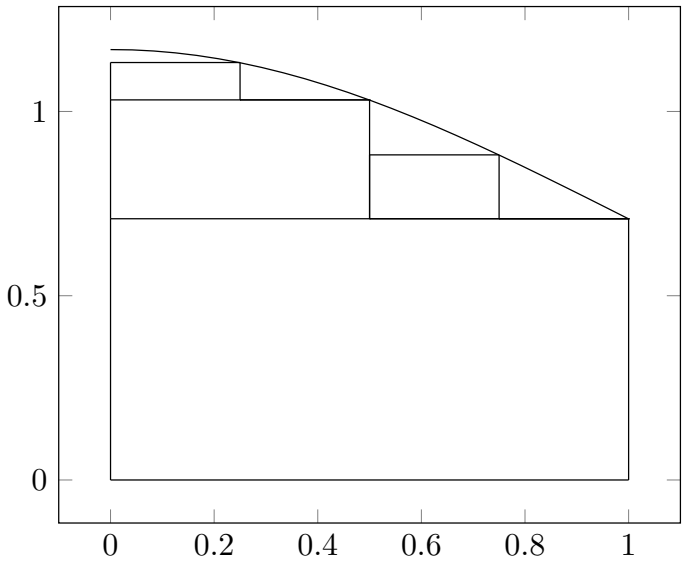

Figure 26: The normal distribution on the interval $[0, 1]$ divided up according to which sections produce distinct prefixes (prefixes up to length 2 have been plotted)

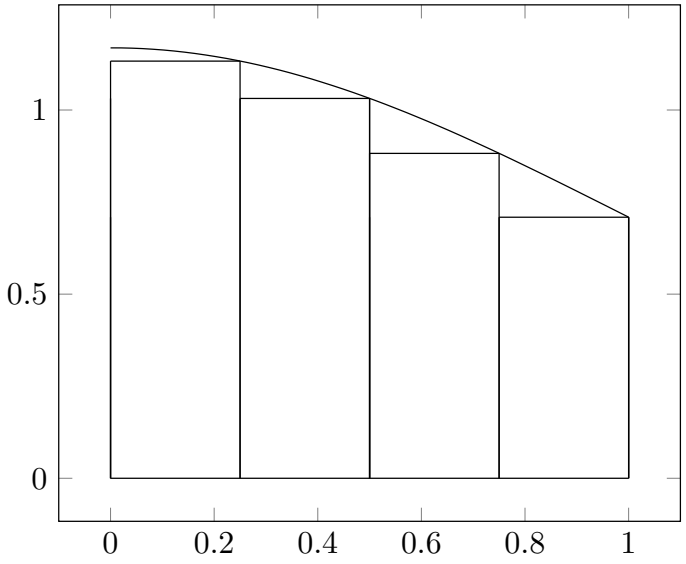

Figure 27: This figure is identical to figure 26 except that the thinnest rectangles have been vertically extended

8 The Uniformity Of Trailing Digits

We've now seen that under relatively minor assumptions the prefixes of continuous distributions are quite short. This also means that the trailing digits of continuous distributions rapidly converge to the uniform distribution. We now make this statement more exact. The following discussion won't rely on insights based on prefixes and is instead essentially a generalization of an argument that is suggested by Arif Zaman in [3].

Theorem 1. *Let X be a random variable that is distributed continuously according to a Riemann integrable density function f. Then the trailing digits n places beyond the decimal point, when interpreted as a real number in the interval $[0, 1]$, converge in distribution to the standard uniform distribution as $n \to \infty$.*

Proof. We begin by denoting the density of the fractional part of X as $\{X\}$. Then the density of $\{X\}$ is given by

$$\tilde{f}(x) = \begin{cases} \sum_{i \in \mathbb{Z}} f(i + x) & 0 < x \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

The case $0 < x \leq 1$ follows from the fact that, if we denote the corresponding cumulative distribution functions F and $\tilde{F}$, then clearly for $0 < x \leq 1$ we have

$$\tilde{F}(x) = \sum_{i \in \mathbb{Z}} F(i + x) - F(i)\,.$$

We're just summing up all the probabilities corresponding to intervals $[i, i + x]$ where the fractional part is less than x. Then differentiate both sides with respect to x.

$$\begin{aligned} \tilde{f}(x) &= \frac{d}{dx} \sum_{i \in \mathbb{Z}} F(i + x) - F(i) \\ &= \frac{d}{dx} \sum_{i \in \mathbb{Z}} \int_0^x f(i + x)\, dx \\ &= \frac{d}{dx} \int_0^x \sum_{i \in \mathbb{Z}} f(i + x)\, dx \qquad \text{by Fubini's theorem} \\ &= \sum_{i \in \mathbb{Z}} f(i + x)\, dx \end{aligned}$$

Now that we know the density of $\{X\}$, the density for digits that are trailing n places beyond the decimal point is close at hand. If we continue expressing our numbers in binary it'll be the density of the random variable $\{2^n X\}$. After all, multiplying by 2^n just shifts all the bits n places to the left.

The density f_{2^n} of $2^n X$ is given by transforming the density for X as

$$f_{2^n}(x) = \frac{1}{2^n} f\left(\frac{x}{2^n}\right)\,.$$

Thus, just as before, we get the following density for $\{2^n X\}$:

$$\tilde{f}_{2^n}(x) = \begin{cases} \sum_{i \in \mathbb{Z}} \frac{1}{2^n} f\left(\frac{i+x}{2^n}\right) & 0 < x \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

Notice that this is really just a Riemann sum again. For any fixed x we're evaluating f at distances 2^{-n} apart and multiplying the result by the width of the corresponding interval. Since f integrates to 1, we get

$$\lim_{n \to \infty} \tilde{f}_{2^n}(x) = \begin{cases} 1 & 0 < x \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

which is exactly the density of the standard uniform distribution. □

Further, we can see that this convergence tends to be extremely rapid. We can measure this, for instance, in terms of the L_1-distance between f_{2^n} and the standard uniform density. The argument follows the same lines as the previous section.

Proposition 6. *Given a density function f, if f is K-Lipschitz and has support $[i, i+1]$, $i \in \mathbb{Z}$, then*

$$\int_{\mathbb{R}} \left| \mathbf{1}_{[0,1]}(x) - \tilde{f}_{2^n}(x) \right| \, dx \leq 2^{-n} K$$

where $\mathbf{1}_{[0,1]}$ is the indicator function for the interval $[0,1]$ and $\tilde{f}_{2^n}$ is the density of the trailing digits beyond the nth bit.

Proof. $\mathbf{1}_{[0,1]}$ and $\tilde{f}_{2^n}$ are both equal everywhere outside the interval $[0,1]$. On the interval $[0,1]$ on the other hand, $\mathbf{1}_{[0,1]} = 1$ and $\tilde{f}_{2^n}(x)$ is a Riemann sum for any fixed x. Since it is the Riemann sum for f which integrates to 1, $\left|1 - \tilde{f}_{2^n}(x)\right|$ is really just the error of the Riemann sum. And we've already seen in the proof of proposition 2 how we can bound this error above by $2^{-n}K$ □

This leads straight to:

Proposition 7. *Using the previous definitions and assumptions, if f has arbitrary support, we have*

$$\int_{\mathbb{R}} \left| \mathbf{1}_{[0,1]}(x) - \tilde{f}_{2^n}(x) \right| \, dx \leq 2^{-n} \sum_{i \in \mathbb{Z}} K_i$$

where K_i is the Lipschitz constant of f corresponding to the interval $[i, i+1]$.

Proof. For densities with arbitrary support we get an error of at most $2^{-n}K_i$ on every interval of unit length by the previous result. We simply sum these errors over all $i \in \mathbb{Z}$. □

Of course the sum over the K_i is a familiar face from the previous section. So we can immediately also give the upper bound $2^{-n} \int_{\mathbb{R}} |f'(x)| + |f''(x)| \, dx$, provided that f is twice differentiable.

Arif Zaman was brought to the distribution of fractions by examining a wheel of fortune and asking what the distribution of the resting position for the wheel might be. He assumed the circumference of the wheel to be normed to 1 and that the total distance traveled was a normally distributed random variable X. Since the wheel effectively removes the integer part of X after each rotation, the resting position will then be $\{X\}$.

Now a wheel of fortune may seem like a rather droll thing to study, but in fact there are wheels of fortune that are of significant practical importance. Take the generation of random numbers in computers. At the most fundamental level this is often accomplished by measuring, for instance, the current CPU temperature to very high precision and using the bits beyond some position n as a source of randomness. But this is nothing other than a wheel of fortune that has been normed to length 2^{-n}! We can now see much more clearly why these methods are so appropriate.

9 Some Impossibility Results

Most of this thesis has focused on continuous distributions and their prefixes. We'll now close with two results regarding discrete distributions from a completely different arena – the theory of computation.

The fact that we could sample exactly from virtually any discrete distribution might have come as a slight surprise to some. And it raises the question where the limits of the tools we have considered are. In fact, are there *any* discrete distributions which one can *not* sample exactly using exact bisection sampling or similar means? Indeed, there are – though they are admittedly somewhat esoteric.

To begin with, we'll have to recall the undecidability of the halting problem established by Alan Turing in [4]. Because the argument is so disarmingly simple, we'll briefly recount it here.

Theorem 2. *There is no algorithm H (alternatively Turing machine or computer program), accepting as inputs the description of another algorithm A and a string x, that can within a finite amount of time determine whether A, when run on x, halts.*

Proof. As with all the proofs in this section, we argue by contradiction. Suppose such an algorithm H exists. Then one can easily modify it into a new algorithm $\overline{H}$ that only takes a single argument A defined as

$$\overline{H}(A) = \begin{cases} \text{enter an infinite loop} & \text{if } H(A,A) = \text{ true} \\ \text{terminate} & \text{if } H(A,A) = \text{ false} \end{cases}$$

Because $H(A,x)$ supposedly correctly determines whether A halts on x, this is equivalent to

$$\overline{H}(A) = \begin{cases} \text{enter an infinite loop} & \text{if } A(A) \text{ halts} \\ \text{terminate} & \text{if } A(A) \text{ doesn't halt} \end{cases}$$

We use "terminate" because it really doesn't matter what $\overline{H}$ outputs in that case. It only matters that $\overline{H}$ halts. Now the question is: Does $\overline{H}(\overline{H})$ halt? There are two options.

- $\overline{H}(\overline{H})$ **halts**: But then by the definition of $\overline{H}$ it doesn't halt. ↯
- $\overline{H}(\overline{H})$ **doesn't halt**: But then by the definition of $\overline{H}$ it halts. ↯

Either way, we get a contradiction. So we're forced to accept that H can't exist. □

How do we turn this into a discrete distribution from which it is impossible to sample exactly? Like so:

Proposition 8. *Let (A_n) be an enumeration of all possible algorithms. And let (A_{n_k}), k starting at 1, be a subsequence consisting of all the non-halting algorithms (algorithms that don't halt when given themselves as input).*

Then there doesn't exist any randomized algorithm (an algorithm that has access to random bits during its computation) that can sample exactly from the distribution with probability mass function

$$P(i) = \begin{cases} 2^{-k} & \text{if } i = n_k \text{ for some } k \\ 0 & \text{otherwise} \end{cases}$$

where $i \in \mathbb{N}$.

In other words, we've just placed some discrete distribution with infinite support, the geometric distribution with $p = 1/2$ to be exact, on the set of non-halting algorithms.

Proof. It is an elementary result in the theory of computation that the set of non-halting algorithms can't be enumerated (in any order, possibly with repetitions). Suppose we could. Then we could solve the halting problem for any algorithm A by doing the following in alternating fashion.

- Run A for a couple steps and see if it has halted yet.
- List out a few non-halting algorithms and see if A shows up.

This process would sooner or later determine whether A halts. But we've already seen that this cannot be done. Now it is definitely possible to run A for a few steps at a time. So it must be that it is impossible to enumerate the non-halting algorithms.

Since we can't even list the non-halting algorithms it will come as no surprise that we can't sample from them either. Usually these impossibility results proceed via a reduction. Just as in the previous paragraph, one demonstrates that, if it was possible to perform some task T, then one could indirectly solve the halting problem. Hence T is impossible. But how do we use a random process to solve a problem such as the halting problem which demands a definite yes-or-no answer?

The solution is to use a strategy known as *derandomization*. This strategy usually entails taking a fast randomized algorithm and significantly shrinking the sample space of that algorithm. It may then become computationally feasible to exhaustively check through the entire sample space, yielding a deterministic algorithm that is now guaranteed to produce the correct result.

We can reuse this idea of exhaustive checking to derandomize the supposed randomized algorithm that can sample from the indices of non-halting Turing machines. If we call this algorithm N then one can turn this algorithm into a determinstic algorithm by making the random bits it has access to an input r of N. We then repeatedly call N on all possible finite sequences, one after the other.

Because the indices of halting algorithms are sampled with probability 0, these will never appear as the output of N. On the other hand, for any given non-halting algorithm A_{n_k} there is a non-zero probability of n_k being produced by N. In other words, there must exist an input r_k such that $N(r_k) = n_k$. Sooner or later we'll reach r_k, thus listing out n_k. And it is easy to use that to list out A_{n_k} itself. So we have found a procedure that enumerates all non-halting algorithms, which is impossible. ↯ □

The second impossibility result concerns exact Bernoulli sampling. All the way back in the introduction there was the innocuous remark that Bernoulli sampling can be performed for any computable number, which is a number whose digits can be listed out by some algorithm. It is easy to show that there are numbers that aren't computable. After all, there are only countably many algorithms, yet uncountably many real numbers. So there must be quite a few real numbers that won't be listed out by any algorithm. Now one might ask if there's perhaps an even more remarkable trick which can't just perform exact Bernoulli sampling for computable numbers, but also for a few uncomputable ones.

Proposition 9. *There exists no randomized algorithm that can sample exactly from the distribution with probability mass function*

$$P(i) = \begin{cases} p & i{=}0 \\ 1-p & i{=}1 \end{cases}$$

where $i \in \{0, 1\}$ and p is not a computable number.

Proof. Suppose this was possible using some algorithm N using random bits r as its only input. We will again derandomize N – looping through all possible values of r – to produce an algorithm that can exactly list out the digits of p.

When iterating through the possible values of r, there will be many for which N hasn't terminated yet. We ignore those. But suppose we reach a value r_k so that $N(r_k) = 0$. We will then also know that N outputs 0 for any string of bits that are a continuation of r_k. So if $\#r_k$ is the number of bits in r_k, we can conclude that there is at least a $2^{-\#r_k}$ chance that N will output 0. We can add this probability to a tally we're keeping for the probability of sampling 0.

If we do the same thing for the output 1, then we'll be able to track two partial sums that are converging to p both from above and below. If they didn't converge, then either N wouldn't be sampling with the correct probabilities or there would exist a non-zero probability that N never terminates. The gap between these partial sums will eventually close far enough to determine every digit of p, one after the other. But that's impossible. ↯ □

References

[1] Donald Knuth & Andrew Yao, 1976, The complexity of nonuniform random number generation, Algorithms and Complexity: New Directions and Recent Results, pages 357–428

[2] Charles F. F. Karney, 2016, Sampling exactly from the normal distribution, ACM Transactions on Mathematical Software, Volume 42, Issue 1

[3] Arif Zaman, 2004, The Density of the Fractional Part of a Normal Distribution

[4] Alan Turing, 1937, On Computable Numbers, with an Application to the Entscheidungsproblem, Proceedings of the London Mathematical Society

10 Appendix

The following is a listing of the code that was used to compute the probabilities of various prefixes. It is written in Clojure, a descendant of Lisp.

```
(ns exact-sampling.plot
  (:require [clojure.math.numeric-tower :refer [expt]]
            [clojure.string :as str])
  (:import (org.apache.commons.math.optimization MultiStartUnivariateRealOptimizer
      GoalType)
           (org.apache.commons.math.optimization.univariate BrentOptimizer)
           (org.apache.commons.math.random JDKRandomGenerator)
           (org.apache.commons.math.analysis UnivariateRealFunction)))

(def minimize
  "Finds the minimum of a function f on the interval [a,b]."
  (memoize
    (fn [f a b]
      (-> (MultiStartUnivariateRealOptimizer. (BrentOptimizer.) 100 (
          JDKRandomGenerator.))
          (.optimize (reify UnivariateRealFunction (value [_ x] (f x))) GoalType/
              MINIMIZE a b)
          (f)))))

(defn prefix-to-number [[base digits]]
  "Given a base and a sequence of digits in that base, returns the corresponding
      number."
  (->> digits
       (map-indexed #(* (expt base (- (inc %1))) %2))
       (reduce +)))

(defn prefix-to-interval-length [[base digits]]
  "Given a base and a sequence of digits in that base, returns the width of the
      interval
  corresponding to all numbers in that base beginning with those digits."
  (expt base (- (count digits))))

(defn prefix-to-interval [[base digits]]
  "Given a base and a sequence of digits in that base, returns a tuple [a,b]
      denoting the interval
   corresponding to all numbers in that base beginning with those digits."
  (let [a (prefix-to-number [base digits])]
    [a (+ a (prefix-to-interval-length [base digits]))]))

(defn prefix-to-probability [f [base digits]]
  "Given a probability density function f, a base, and a sequence of digits, returns
       the probability
  that an unpadded prefix in the given base with the given digits will be selected."
  (if (empty? digits)
    (minimize f 0 1)
    (let [current-min (apply minimize f (prefix-to-interval [base digits]))
          previous-min (apply minimize f (prefix-to-interval [base (butlast digits)
              ]))]
      (* (- current-min previous-min) (prefix-to-interval-length [base digits])))))

(defn leading-digits [digits]
  "Given a sequence of digits, returns a sequence of prefix sequences ending with
      the empty sequence.
  E.g. 101 yields (10, 1, [])."
```

```
  (concat (take-while some? (iterate butlast (butlast digits))) [[]]))

(defn padded-prefix-to-probability [f [base digits]]
  "Given a probability density function f, a base, and a sequence of digits, returns
      the probability
  that a padded prefix in the given base with the given digits will be selected."
  (if (or (empty? digits) (zero? (last digits)))
    0
    (->> (leading-digits digits)
         (take-while #(or (empty? %) (zero? (last %))))
         (map-indexed #(* (expt base (- (inc %1))) (prefix-to-probability f [base
             %2])))
         (reduce +)
         (+ (prefix-to-probability f [base digits])))))

(defn digits-up-to-length-n [base n]
  "Returns a sequence of all sequences of digits up to length n in the given base."
  (->> (iterate #(mapcat (fn [digits] (map (partial conj digits) (range base))) %)
      [[]])
       (take (inc n))
       (apply concat)))

(defn probability-map-0-1 [f base n]
  "Takes a probability density function f with support [0,1], a base, and a maximum
      prefix length n.
  Returns a map from the padded prefixes (represented as doubles) to their
      corresponding probabilities."
  (->> (digits-up-to-length-n base n)
       (map #(vector (prefix-to-number [base %]) (padded-prefix-to-probability f [
           base %])))
       (filter #(pos? (second %)))
       (into (sorted-map))))

(defn probability-map [f base n a b]
  "Takes a probability density function f with arbitrary support, a base, and a
      maximum prefix length n.
  Returns a map from the padded prefixes (represented as doubles) to their
      corresponding probabilities."
  (->> (range a b)
       (mapcat #(->> (probability-map-0-1 (comp f (partial + %)) base n)
                     (map (fn [[x p]] [(+ x %) p]))
                     (into (sorted-map))))))
```