Is 2 a random number?

John Kerl

September 12, 2007

Abstract

After briefly sketching basic probability, we discuss how random numbers uniformly distributed on the unit interval may be used to generate random numbers drawn from various probability distributions. The discussion spans pieces of probability, statistics, number theory, and software. Mathematical prerequisites are limited to modular arithmetic and freshman calculus.

These are lecture notes for a talk given to the Graduate Colloquium in the University of Arizona Department of Mathematics on September 12, 2007.

Contents

Contents **2**

1 Introduction **3**

2 Probability measures **4**

3 Independence **5**

4 CDFs and PDFs **5**

4.1 Discrete random variables 5

4.2 Continuous random variables 7

5 Statistics and histograms **9**

5.1 Probability vs. statistics 9

5.2 Histograms for discrete random variables 9

5.3 Histograms for continuous random variables 10

6 Random numbers uniformly distributed on the unit interval **12**

6.1 Random integers mod 2^d 12

6.2 Seeding 13

6.3 Independence and pseudorandomness 13

6.4 Arithmetic congruential generators 13

6.5 Implementations 17

6.6 True randomness 17

7 Getting other distributions from the uniform distribution **18**

7.1 Correctness 18

7.2 Example with invertible CDF 19

7.3 Example with non-invertible CDF 20

7.4 Non-invertible CDFs 21

7.5 Implementations 21

8 Acknowledgements **21**

References **22**

Index **23**

1 Introduction

Suppose you ask me for a random number, and I give you a 2. Is this what you wanted? What if you ask for a hundred random numbers, and I give you all 2's. Is this "random" at all? If I show you a bag full of 2's and tell you that's how I gave you what I did, would you feel comforted or horrified? Or, alternatively, what if I told you I'd rolled a fair 6-sided die 100 times, and it came up 2's each time. Is that somehow less "random" than, say, a mixture of 1's through 6's? What if I gave you forty 2's, and the rest an even mixture of 1's and 3's through 6's. Is that a "random" occurrence? Such questions can become philosophical. My point is that the question of whether 2 is a "random number" is a meaningless question as phrased — furthermore, our intuition gives us debatable answers. The way out of this dilemma is to set meaningful, precise terminology and follow the consequences.

Modern probability theory remains agnostic on subjective issues by making its definitions carefully in terms of *probability measures* and *distribution functions*. When you ask me for a random number, I am entitled to ask: "Okay, drawn from what distribution?" The distribution function describes the likelihood of certain outcomes happening, and one can insert one's philosophy into the discussion separately, outside of the mathematics. We will see, among other things, that a hundred draws from the bag of 2's produces a hundred 2's with probability 1 (certainty); a fair 6-sided die produces a hundred 2's with probability on the order of 10^{-78}.

The short answer to *Is 2 a random number?* — which we'll explore in a bit more detail — is:

> *The term* random number *is a poor choice of wording. What matters, and what contains all the information, is the* probability distribution*. One or more samples ("random numbers") from a given distribution can be described in terms of how surprised we should be if we see them, and how much we should gamble on the possibility of seeing them.*

For this talk, I will focus on generation of random real numbers — rather than, say, random complex numbers, or random matrices, or random permutations. (Probability theory extends to these regions and far beyond. However, this is enough for a one-hour talk. See [**GS**], [**Ker**], or [**FG**] for more information. Also, I encourage you to take a course in probability, no matter what your area of specialization.)

There are two important classes of random numbers[1]:

Discretely distributed random numbers. This means that there are only finitely or countably infintitely many possible values.

- We can think of the flip of a fair coin as generating a 0 (tails, say) with probability 1/2, and a 1 (heads) with probability also 1/2.
- Even if the coin is head-heavy — say, heads have probability 0.6 and tails have probability 0.4 — the flip of that coin still generates 0's and 1's. The probability distribution, though, is different than it was in the fair coin experiment.
- Likewise, the roll of a fair die generates the numbers 1 through 6 with equal probability (1/6) for each. If the die is unfair, each face will have its own probability. However, it's sure that the die will land on *some* face, so the six probabilities need to add up to 1.
- If we flip a coin over and over again until we get heads, then count the total number of coin tosses, then the result can be any positive integer. Such an experiment will, of course, generate 1's, 2's, and

[1] My technical definition of *random number* is as follows: The sample space is the real line; the σ-field is the Borel sets; the probability measure P varies through this document; a random number is a random variable which is the identity function from $\mathbb{R}$ to $\mathbb{R}$.

3's far more often than it generates 20's. This is because flipping 19 tails in a row is an uncommon occurrence.

Continuously distributed random numbers. Here there are uncountably infinitely many possible values. Examples:

- Measurements of people's heights are positive real numbers, with certain ranges of values occurring more often than others. (We assume we have an infinite-precision ruler.)
- Deviations of people's heights from the population average can be positive or negative.
- If I look at the position of the hour hand on an analog clock and divide by twelve, I get (assuming I'm awake all hours of day and night) numbers uniformly distributed between 0 and 1.
- If I square those, I won't get a uniform distribution anymore: in particular, the numbers between 0 and 1/2 have squares between 0 and 1/4. Here the numbers are still all on the unit interval, but differently distributed.

2 Probability measures

Probability is carefully defined using measure theory; see any of the references. We use a **probability measure** P to tabulate the likelihood of various events happening during an experiment. For our purposes, all we need is the following commonsense definition:

- We have an experiment with some possible **outcomes**. The variable name X is conventionally[2] used to denote an outcome. The collection of all outcomes is called the **sample space**. For a single roll of a fair die, the sample space is the numbers 1 through 6.

  For most of this talk, the sample space is the real line and an outcome is simply a real number coming out of a random-number generator.

- Subsets (technically, only measurable subsets) of the sample space are called **events**. For a single die roll, some example events include $\emptyset$ (i.e. $X \neq \mathbb{R}$), $X = 2$, and $X = 1, 3, 5$.

  For most of this talk, an event is either the singleton event $X = k$ or intervals such as $a \leq X \leq b$ or $X \leq b$.

- A probability measure P assigns to each event a number between 0 and 1. We interpret P of an event to be the likelihood of that event occurring. For a single roll of a fair die, we have $P(\emptyset) = 0$, $P(X = 2) = 1/6$, and $P(X = 1, 3, 5) = 1/2$.

  For random numbers considered in this talk, we will want to measure the likelihoods of events such as $P(X = k)$, $P(a \leq X \leq b)$, or $P(X \leq b)$. These probabilities will be given explicitly by functions described in section 4.

- The probability measure satisfies the usual axioms from measure theory, along with the requirement that P of the entire sample space is 1. Those axioms encode commonsense expectations: it is certain that some outcome happens, so $P(X \notin \mathbb{R}) = 0$ and $P(X \in \mathbb{R}) = 1$; $P(X \notin A) = 1 - P(X \in A)$ for all measurable subsets of $\mathbb{R}$; the probabilities of disjoint events add. I used this last property above: $P(X \text{ odd}) = P(X = 1) + P(X = 3) + P(X = 5) = 1/6 + 1/6 + 1/6 = 1/2$.

[2]More generally, one has a sample space Ω with outcomes ω, and a real-valued random variable $X : \Omega \to \mathbb{R}$. As mentioned above, though, for this talk $\Omega = \mathbb{R}$ and X is the identity function, so I refer to X where one would usually refer to ω.

3 Independence

Notation and terminology:

- Let X be a number chosen at random according to some rule which we get to specify.
- If picking such an X has no effect on the result of doing it again, we say that each X generated is **independent** of the others.
- If I use the same rule each time — I don't switch from coin tosses to dice halfway through — then we say that the X's are **identically distributed**.
- If we repeat our experiment so that the X's we generate are independent as well as identically distributed, we say the X's are *independent and identically distributed*, or **IID**. This paper is about generating IID sequences of random numbers . . . almost. We'll find out in section 6 that the difference between random and pseudorandom numbers involves independence.

Here's an example where independence is violated: First, I set the die on the table with the ones-face up. Then, without looking, I pick one of the four sides and tip the die over with my finger. Now, the 1 can't turn up, and neither can the 6 (which was down before). The other four faces occur with equal chance. If I repeat this, the *current* face affects what face can occur *next*.

4 CDFs and PDFs

The central notion in formalizing our intuitive ideas about random numbers is this: Pick a number X according to some rule. Then define

$$F(x) := P(X \leq x),$$

where the P is the probability measure from section 2. This $F(x)$ is the **cumulative distribution function** or **CDF** of X.

Furthermore, if there is an $f(x)$ such that

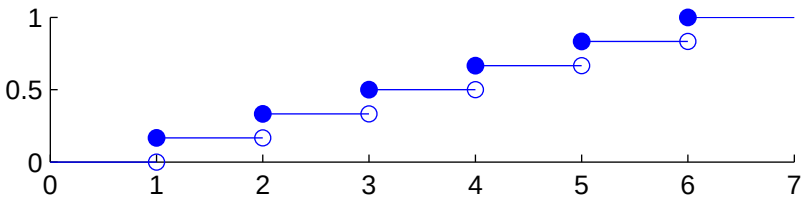
$$F(x) = P(X \leq x) = \int_{-\infty}^{x} f(t)\,dt$$

then we say that $f(x)$ is the **probability density function** or **PDF** of X. In fact, this is the definition of continuous random variable: if such an $f(x)$ exists, then the random variable X is said to be a continuous one. (Note that by construction $F(x)$ is the antiderivative of $f(x)$, and by the second fundamental theorem of calculus $f(x)$ is the derivative of $F(x)$.) I will say more about PDFs in a moment, after first discussing cases where PDFs don't exist.

4.1 Discrete random variables

First suppose there is no such PDF $f(x)$; the CDF isn't the integral of any function. But suppose there are fintely many or countably infinitely many outcomes. Then the CDF is a right-continuous step function.

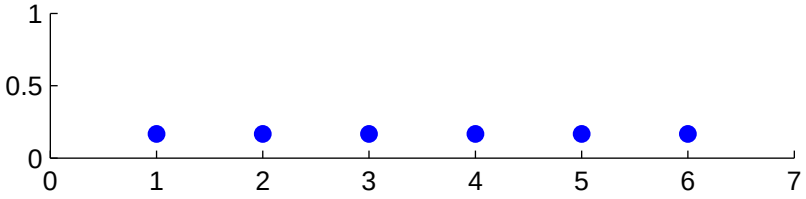
For example, take X to be the result of the roll of a fair die. Outcomes are 1 through 6, each with probability 1/6. Then $P(X \leq -1)$ is 0, and $P(X \leq 7)$ is 1. Also, $P(X \leq 1) = 1/6$, $P(X \leq 2) = 1/3$, and so on. Here is a graph of the CDF:



It also makes sense to plot the probability of each outcome individually. This is

$$f(X) = P(X = x);$$

it is called the **probability mass function** or **PMF** of the discrete random variable X. For the fair die, it looks like this:

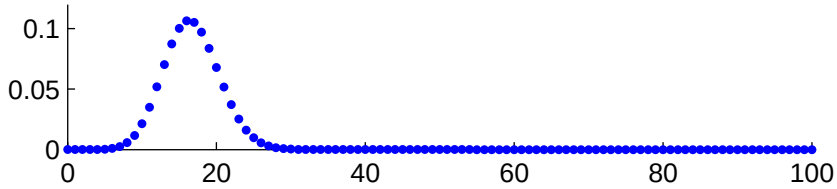


Here are CDFs and PMFs for some other discrete random variables:

- For the bag-of-two's example, the CDF steps up from 0 to 1 at $x = 2$. The PMF has value 1 at $x = 2$ and 0 elsewhere. (Yes, this fits the definition of random variable, even though the outcome is the same each time, and is not "random" at all in the colloquial sense.)
- For a coin with probability p of heads, if we encode tails as 0 and heads as 1, then the CDF steps up from 0 to $(1 - p)$ at $x = 0$, and from $(1 - p)$ up to 1 at $x = 1$. The PMF has value $(1 - p)$ at $x = 0$ and value p at $x = 1$.
- Roll a fair die 100 times and let X count the number of 2's. I won't show the CDF; the algebra is a bit messy. But for the PMF, we can figure that

$$P(X = k) = \binom{100}{k} \left(\frac{1}{6}\right)^k \left(\frac{5}{6}\right)^{100-k}.$$

  That is, we multiply the probability of k 2's and $100 - k$ non-2's, weighting by all the ways to select k items from a set of 100. That PMF looks like this:



  PMFs are highly intuitive — where they are high, those are likely outcomes; where they are low, those are unlikely outcomes. Here, $f(20) = 0.0679$, i.e. you have about 1 in 15 odds of getting 20 2's. On the other hand, $f(100) \approx 10^{-78}$ as I mentioned at the top of the paper. Now, twenty 2's and 100 2's are both legitimate outcomes: it is simply the case that one has a higher probability than the other.

4.2 Continuous random variables

The other kind of random variable we consider here[3] is the kind where the CDF $F(x)$ is a continuous (and of course, still non-decreasing) function. It doesn't make sense to ask about the probability of the event $X = a$ exactly: we have

$$\begin{aligned} P(a \leq X \leq a + \varepsilon) &= P(X \leq a + \varepsilon) - P(X \leq a) \\ &= F(a + \varepsilon) - F(a) \end{aligned}$$

which goes to zero by continuity of F as $\varepsilon \to 0$.

But remember that continuous random variables have the property that the CDF, which is $F(x) = P(X \leq x)$, is the integral of the PDF $f(x)$. If $a \leq b$ then the interval $(-\infty, b]$ may be split disjointly as $(-\infty, a]$ and $(a, b]$. Then from

$$P(X \leq a) = \int_{-\infty}^{a} f(t)\, dt$$

and

$$P(X \leq b) = \int_{-\infty}^{b} f(t)\, dt = \int_{-\infty}^{a} f(t)\, dt + \int_{a}^{b} f(t)\, dt$$

and the disjointness property of probability measures, we have the probability that X is in the interval $[a, b]$, in terms of the PDF:

$$P(a \leq X \leq b) = \int_{a}^{b} f(x)\, dx$$

Also recall that by the Fundamental Theorem of Calculus we can write this in terms of the CDF:

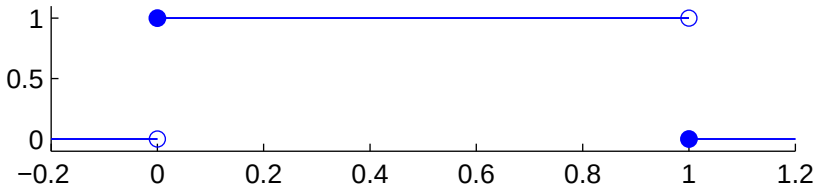
$$P(a \leq X \leq b) = F(b) - F(a).$$

Here are some continuous random variables, with their PDFs and CDFs.

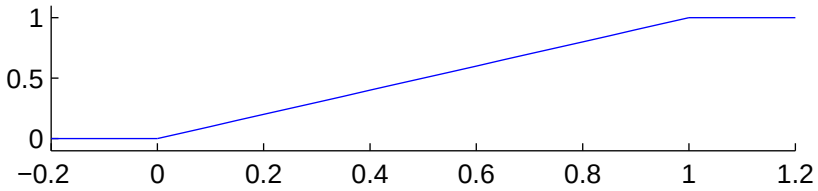
- The **uniformly distributed** random variable on the unit interval is what you might guess:

  $$P(a \leq X \leq b) = b - a$$

  for $0 \leq a, b \leq 1$.



  The CDF is the integral of this: It ramps up from 0 to 1 as x runs from 0 to 1:



[3] There are random variables which are neither continuous nor discrete; they are outside the scope of today's talk. See [**FG**].

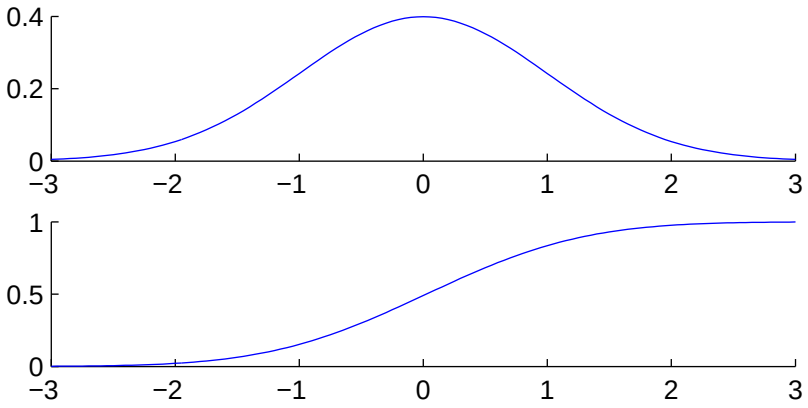
- The **standard normal** random variable is the famous **bell curve**. The PDF is

$$\frac{1}{\sqrt{2\pi}}\, e^{-x^2/2}$$

and there is no simpler way to write the CDF other than as the integral of that:

$$F(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2}\, dt.$$

They look like this:



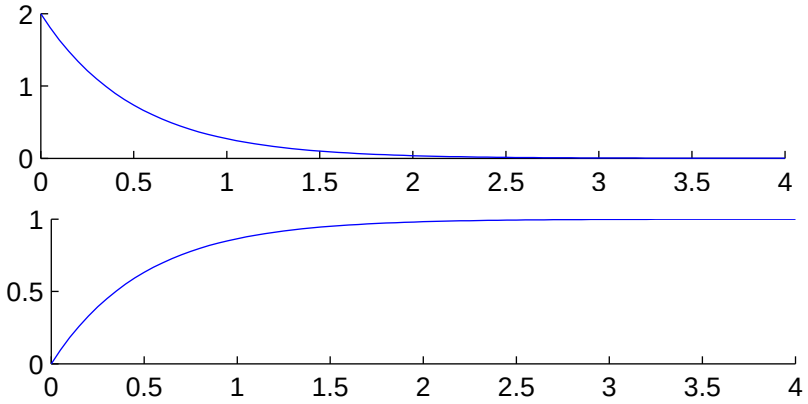
- Here is one you may not have seen before, but I will use it below since it makes a certain example very easy to compute with. Fix a non-negative parameter λ. The random variable X will take non-negative values only. Its PDF is defined to be

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0. \end{cases}$$

Then its CDF is (with $u = -\lambda t$)

$$F(x) = \int_0^x \lambda e^{-\lambda t}\, dt = \begin{cases} 1 - e^{-\lambda x} & x \geq 0 \\ 0 & x < 0. \end{cases}$$

This X is called the **exponential random variable**. The PDF and CDF look like this (taking $\lambda = 2$ here):



5 Statistics and histograms

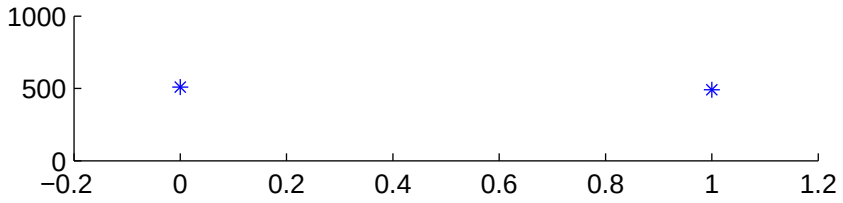
5.1 Probability vs. statistics

The following is a fair question: What's the difference between probability and statistics? Here's the short answer:

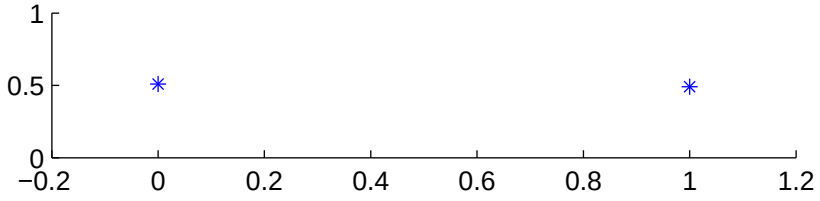
- *Probability* is a branch of mathematics which applies to situations where we know P ahead of time and we want to gauge likelihoods of various events. E.g. *if* a not-quite-fair coin has probability of heads $p = 0.6$, *then* we can describe the probability of a thousand flips turning up anywhere between 520 and 530 heads.
- *Statistics* is an art, some of whose techniques are mathematical. It applies to situations where we are *not* given rules; we *only* have the data and we want to *discover* (or at least hypothesize) what the rules are. Is 520 heads on a thousand coin tosses evidence of an unfair coin? From the number of heads can we estimate what the true p is? Or worse: If we have a only list of numbers between 0 and 1000, it is plausible they were obtained from repetitions of a flip-1000-coins experiment at all?

5.2 Histograms for discrete random variables

A key tool in statistics is the **histogram**. For discrete random variables, they're easy to write down: Run the X-generating experiment a number of times — say, N times. Make a list of all possible outcomes, paired up against the number of times X took on each value. For example, suppose I flipped a coin 1000 times and got 421 heads. Then my histogram[4] looks like this — there is a 421 at $x = 0$ and $1000 - 421 = 579$ at $x = 1$:

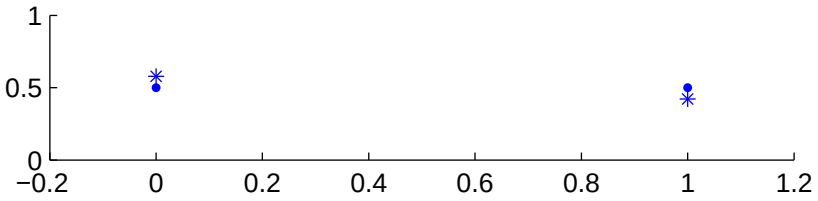


To put this on the same footing as the PMF, I can scale by $1/N$, i.e. plot not 421 and 579 but 0.421 and 0.579. Doing this permits me to see the *fraction* of events which fell into each bin:



I can also superimpose what my expectation was: namely, 500 heads and 500 tails, which scale to 0.500 and 0.500. That is, I can plot the scaled histogram (plotted with asterisks) and the PMF (plotted without asterisks) simultaneously:

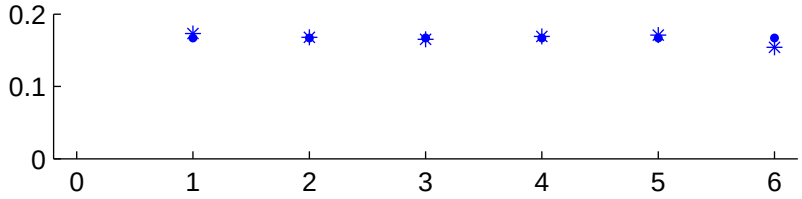
[4]Histograms are traditionally drawn using bar plots.



Likewise for die rolls. Suppose I rolled a die 1000 times and got the following:

Roll	1	2	3	4	5	6
#	173	168	165	169	171	154

Now, my expectation is $1000/6 \approx 167$ of each. I can again plot the histogram, scaled by $1/N$, and the PMF simultaneously:



The histogram looks fairly uniform, or flat — as you would expect from a fair die. It's not perfectly flat — after all, you don't expect to *always* get a pair of 2's on every dozen flips, nor do you expect to *always* have 5 out of 10 coin flips come up heads — but we do expect that as the number of rolls increases, the histogram will look flatter and flatter. This hope is encoded precisely in the **law of large numbers**, which you can read about.

5.3 Histograms for continuous random variables

For continuous random variables, we do something a little different: we divide the range of the random variable into some number of disjoint intervals, called **bins**, which includes all the outcomes that occurred. Then we count the number of events in each bin. Again, I'll scale the bin counts by $1/N$, so I can plot the fraction of events which fell into each bin. And also as before, we can superimpose the expected values over the scaled histogram.

Specifically, I will generate 1000 numbers with the exponential distribution, using $\lambda = 2$ as before. Then I will create histogram bins with width 0.5. So, I'm counting the number of events in the interval $[0, 0.5)$, the number of events in the interval $[0.5, 1.0)$, and so on. Here are the raw data:

Trial	X
1	0.0145829
2	0.1205840
3	1.0430334
4	0.3715260
5	0.2104607
6	0.3936524
7	0.2331679
⋮	⋮
998	1.4285271
999	0.6368516
1000	0.8505719

The smallest number out of all 1000 of those was the 31st, which was 0.0001355; the largest was the 871st, which was 4.1862383. So the highest bin will be $[4.0, 4.5)$. Here are the bins and their counts:

Bin	Count
$[0.0, 0.5)$	656
$[0.5, 1.0)$	215
$[1.0, 1.5)$	97
$[1.5, 2.0)$	20
$[2.0, 2.5)$	8
$[2.5, 3.0)$	2
$[3.0, 3.5)$	1
$[3.5, 4.0)$	0
$[4.0, 4.5)$	1

Since there were 1000 samples, I want to scale the counts by 1000 and plot this:

Bin	Fraction
$[0.0, 0.5)$	0.656
$[0.5, 1.0)$	0.215
$[1.0, 1.5)$	0.097
$[1.5, 2.0)$	0.020
$[2.0, 2.5)$	0.008
$[2.5, 3.0)$	0.002
$[3.0, 3.5)$	0.001
$[3.5, 4.0)$	0.000
$[4.0, 4.5)$	0.001

Now, what did I *expect* to get? For the bin $[0, 0.5)$, or more generally $[a, b)$, I have

$$\begin{aligned} P(a \le X < b) &= F(b) - F(a) \\ &= (1 - e^{-\lambda b}) - (1 - e^{-\lambda a}) \\ &= e^{-\lambda a} - e^{-\lambda b}. \end{aligned}$$

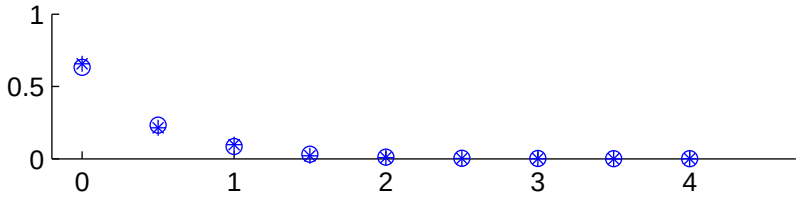
So I want to plot

$$e^{-\lambda x} - e^{-\lambda(x+0.5)}$$

for $x = 0, 0.5, \ldots, 4.5$. Here are those numbers:

Bin $[a, b)$	$P(a \le X \le b) = F(b) - F(a)$
$[0.0, 0.5)$	0.6321206
$[0.5, 1.0)$	0.2325442
$[1.0, 1.5)$	0.0855482
$[1.5, 2.0)$	0.0314714
$[2.0, 2.5)$	0.0115777
$[2.5, 3.0)$	0.0042592
$[3.0, 3.5)$	0.0015669
$[3.5, 4.0)$	0.0005764
$[4.0, 4.5)$	0.0002121

Here are the scaled expected counts (open circles), plotted along with the scaled histogram (asterisks):



That looks like a good fit. If you don't know the distribution and you look at these data, trying to answer some questions about them, you're now doing statistics. You can conjecture various distributions, and see how well each seems to fit visually. Also there are quantitative statistical goodness-of-fit techniques, involving say χ^2 tests, for making this precise.

The task for us is to be able to write a computer program which generates numbers with a given probability distribution. (I just showed you the results of such a program — but I haven't told you how it works yet.) How do we know if it's right? (1) check visually against a histogram, or run goodness-of-fit tests; (2) make it provably correct from the outset. We choose the latter technique, although we'll plot some histograms for good measure.

It turns out that we can take a two-step approach to the problem of generating random numbers with a specified distribution: (1) find out how to generate random numbers which are uniformly distributed on the unit interval; (2) feed those into a function which will produce numbers which have the desired distribution. These two steps have very different flavors: the first is algebraic; the second is analytical.

6 Random numbers uniformly distributed on the unit interval

6.1 Random integers mod 2^d

The first step is in turn a two-step (sub)process: (1a) generate integers which are uniformly distributed on the interval $[0, 2^d)$, for some d, and (1b) scale by $1/2^d$. Notice that the integers mod 2^d are *discrete* random variables, and scaling by $1/2^d$ doesn't change that. Nonetheless we treat the scaled numbers as if they were *continuous* random variables. This feels reasonable as long as d is large — think of it as a software version of approaching a continuum limit. For the sake of discussion, though, let's fix $d = 3$ for a moment, so that we are working mod 8.

Here's one way to generate integers uniformly distributed between 0 and 7:

- Start with 0.
- The next time, give one more than the previous value, mod 8.

This algoritm generates the sequence

$$0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, \ldots.$$

We certainly have uniform distribution: our histogram looks great. What's wrong? The sequence of X's coming out of this algorithm, or **generator**, don't have *independence*.

6.2 Seeding

One problem is that we're starting with 0. The jargon word for start value is **seed**. Practical implementations in software do some variation of the following to seed the random-number generator:

- Take the time of day, including certainly seconds, perhaps milliseconds or microseconds if the system supports it; maybe fold in the day, month, year, etc. This ensures that you get different stuff when you re-run your program.
- Use the Unix process identifier (PID), which is usually a 4- or 5-digit integer. Also maybe fold in some numerical identifier of the machine, such as the IP address. This ensures that two different copies of a program running at the same time on the same or different machine generate different stuff. This is important in parallel processing.
- Take the information from the previous steps and scramble it somehow — compute some function of it modulo 2^d.

6.3 Independence and pseudorandomness

The second problem with the algorithm above is that, given one value of X, we know what the next one is. I am going to share a dirty little secret with you: *most other random-number generators in common use have this same property.* The correct technical term for most generators is **pseudorandom number generator**. Most of us are complicit in the dirty little secret when we omit the *pseudo*. (See section 6.6, however — true randomness does exist in the world.) The current number coming out of the generator is computed from the previous value. What varies among pseudorandom number generators is how well they hide that fact.

If we are generating sequences of numbers mod 2^d, with the current value taken from a function of the previous one, then the generator will repeat itself after at most 2^d repetitions — maybe fewer. The smallest m such that $X_m = X_0$ is called the **period** of the generator.

6.4 Arithmetic congruential generators

Let's modify the above a bit. Instead of adding 1 each time, let's add b. Then

$$X_{n+1} = X_n + b \pmod{2^d}.$$

(The value X_0 is a seed value, chosen as described above.) This is called an **additive congruential generator**. To be concrete, let's take $d = 3$, so we're working mod 8. What if $b = 4$? If $X_0 = 1$, then $X_1 = 5$ and $X_0 = 1$ again. The generator has period 2, and doesn't produce all 8 numbers from 0 through 7 uniformly. Clearly, a minimum requirement is that b have maximal period in the additive group of the ring $\mathbb{Z}/2^d\mathbb{Z}$. This just means that b needs to be odd. The period is 2^d, regardless of the seed value.

Let's look at another approach:

$$X_{n+1} = aX_n \pmod{2^d}.$$

This is a **multiplicative congruential generator**, or **MCG**. Again, it's easy to rule out certain bad choices: $a = 1$ generates a constant sequence; $a = -1$ generates a sequence with period 2. Also any even a is bad: after at most d iterations, the generator produces zeroes. We want a to have maximal order in the multiplicative group of the ring $\mathbb{Z}/2^d\mathbb{Z}$. From algebra (see [**DF**] for example) we know this maximal order is 2^{d-2} for $d \geq 3$. Note however that if $X_0 = 0$, then the generator always gives zeroes. Thus, the period is seed-dependent.

A third approach is a **linear congruential geneator**, or **LCG**. It does the following:

$$X_{n+1} = aX_n + b \pmod{2^d}.$$

The numbers a and b are carefully chosen. Before we get theoretical, here are a couple of experiments: the first with $a = 3, b = 4$, and the second with $a = 5, b = 3$. Both start with the seed $X_0 = 1$.

$a = 3, b = 4$:	$a = 5, b = 3$
1	1
7	0
1	3
7	2
1	5
7	4
1	7
7	6
1	1
7	0
1	3
7	2
1	5
7	4
1	7
7	6
1	1
7	0
1	3
⋮	⋮

Here is the histogram data over a hundred runs (I won't make a plot — I'll just show the pair of tables):

Bin	Count
0	0
1	50
2	0
3	0
4	0
5	0
6	0
7	50

Bin	Count
0	13
1	13
2	13
3	13
4	12
5	12
6	12
7	12

Clearly the one on the left isn't doing too well: just 1's and 7's over and over. The one on the right covers all values from 0 to 7 evenly, and it does so in a less predictable way than simply counting up mod 8.

Surprisingly, this basic idea extended to larger d (typically $d = 32$ or $d = 64$) is widely used — as naive as it seems.

OK so let's look at this a bit — how can we choose a and b? The venerable resource *Numerical Recipes* [**NR**] gives the following. They consider linear recurrences mod M, where for us $M = 2^d$.

- b is relatively prime to M. For $M = 2^d$, this means b is odd.
- $p \mid a - 1$ for all prime factors of M. For $M = 2^d$, this condition becomes quite weak; it only means a is odd.
- $4 \mid a - 1$ iff $4 \mid M$. For $M = 2^d$, of course this means $a \equiv 1 \pmod 4$.
- $a, b, X_0 < M$, which is obviously not a problem since we can just reduce mod M: reduction mod M is a homomorphism so we can do it before or after computing $aX + b$.

In particular, [**NR**] suggests

$$a = 1664525,\ b = 1013904223.$$

Here is some Python code to run this algorithm for 1000 iterations, starting with seed value $X_1 = 123456789$:

```
#!/usr/bin/python -Wall

a = 1664525
b = 1013904223
mask = (1<<32) - 1
M = 1.0 * (1<<32)

X = 123456789
N = 1000
for k in range(1, N+1):
    print "%4d %11d 0x%08x %11.7f" % (k, X, X, X/M)
    X = (a*X + b) & mask
```

I have it print out the X's in decimal and hexadecimal, along with their values scaled to the unit interval. Here's the output:

k	X_k decimal	X_k hex	$X_k/2^{32}$
1	123456789	`0x075bcd15`	0.0287445
2	920370032	`0x36dbbb70`	0.2142903
3	3761641487	`0xe036180f`	0.8758254
4	2252023330	`0x863b2622`	0.5243400
5	1475571481	`0x57f36f19`	0.3435583
6	2340457892	`0x8b808da4`	0.5449303
7	1600748723	`0x5f697cb3`	0.3727034
⋮	⋮	⋮	⋮
998	4048274180	`0xf14bc304`	0.9425623
999	3093262995	`0xb85f7293`	0.7202064
1000	3511244502	`0xd14956d6`	0.8175253

When I make a histogram with bins at each tenth, I get this:

Bin	Count
$[0.0, 0.1)$	112
$[0.1, 0.2)$	106
$[0.2, 0.3)$	92
$[0.3, 0.4)$	90
$[0.4, 0.5)$	111
$[0.5, 0.6)$	116
$[0.6, 0.7)$	102
$[0.7, 0.8)$	78
$[0.8, 0.9)$	98
$[0.9, 1.0)$	95

— which looks like a pretty flat histogram. As before, we'd expect it to flatten with larger N. And again, there are formal tests to show whether it flattens enough as N increases to indicate that the true underlying distribution is uniform. Several things could go wrong: Maybe I have a bug in my program (I hope that's unlikely since it's so short!); maybe the coefficients are badly chosen (but I trust *Numerical Recipes*!); maybe the idea of using a linear-congruential generator is a bad choice from the outset. It can be shown that LCGs aren't optimal for all applications.

However, here is one saving grace of LCGs — doing $aX + b \bmod M$ tends to spread iterates around the range from 0 to $2^{32} - 1$. In fact, let's start with seed values $X_1 = 123456787$, values $Y_1 = 123456788$, and values $Z_1 = 123456789$. Upon iteration, the similarities disappear:

k	X_k	Y_k	Z_k	$X_k/2^{32}$	$Y_k/2^{32}$	$Z_k/2^{32}$
1	123456787	123456788	123456789	0.0287445	0.0287445	0.0287445
2	917040982	918705507	920370032	0.2135152	0.2139028	0.2142903
3	2982502077	3372071782	3761641487	0.6944179	0.7851216	0.8758254
4	665391352	3606190989	2252023330	0.1549235	0.8396318	0.5243400
5	1157603319	1316587400	1475571481	0.2695255	0.3065419	0.3435583
6	910523626	3772974407	2340457892	0.2119978	0.8784641	0.5449303
7	3767895873	2684322298	1600748723	0.8772816	0.6249925	0.3727034
8	1833154476	3684444433	1240767094	0.4268145	0.8578516	0.2888886
9	3017395995	2657275708	2297155421	0.7025423	0.6186952	0.5348482
10	1621506494	3806421355	1696368920	0.3775364	0.8862515	0.3949667

It's clear that a linear-congruential generator *does* produce patterns — the X's that come out aren't independent. The important caveat for practical computational work is for those patterns not to jive with patterns in your computer program or in the data being manipulated. There are extensive references to this subject: see in particular Knuth vol. 2 and *Numerical Recipes* ([**Knu2**] and [**NR**]). Also, I chose to talk about LCGs in this paper because they're easy to understand, but they aren't the cutting edge of pseudorandom number generation. Which takes me to my next section ....

6.5 Implementations

Here are some ways you can get various machines to generate unit-uniform pseudorandom numbers for you:

- The C library's `random` routine on GNU/Linux systems uses a 31-stage non-linear additive recurrence relation to generate 32-bit pseudorandom integers.
- Another popular generator for C programming, called `drand48`, uses an LCG (as described in this document) with $M = 2^{48}$ and scales down to $[0, 1)$.
- Python 2.5's `random` module uses the *Mersenne twister* algorithm which uses linear algebra over the finite field $\mathbb{F}_2$. There's a nice Wikipedia article on this generator.
- Matlab has a `random('unif', 0, 1)` routine. Also (see `help rand`) it can generate pseudorandom numbers from several different distribiutions.
- The TI-83 calculator has a `random` function, although from the manual I can't tell what algorithm it uses.

From this brief list it's clear that on whatever you compute, there is probably a pseudorandom number generator you can use. Or, you can code up your own, by porting the code snippet above into your favorite language. Just remember that pseudorandom number generators generate sequences that *look* IID, but really aren't. A good piece of advice I've heard several times is to try your simulation runs (in whatever your area of expertise) using at least two (completely) different pseudorandom number generators.

6.6 True randomness

Despite my caveats in section 6.3 about pseudorandom numbers, there are sources of true randomness (by which I mean sequences which are not just ID but IID) in the world.

One, on Linux systems, is the `/dev/random` pseudofile. You can read uniformly distributed unsigned bytes from it (i.e. 0-255), or get 32-bit integers by reading 4-tuples of bytes, etc. These numbers are truly random; they are generated by doing computations based on environmental noise: interpacket arrival time, taken from the computer's network card, and/or time between keystrokes on the keyboard. However, my experiments have shown that I get only a few dozen bytes per second of throughput from `/dev/random`. This may not feed your Monte Carlo simulation nearly fast enough.

Another source of true randomness is described in a paper I read a few years ago ... I've lost the reference but it was created by someone at Xilinx Corporation. There are certain kinds of programmable electronic circuits, called *FPGAs* (for *field programmable gate arrays*) which can be set up so that certain outputs are connected back to certain inputs in such a way that chaotic voltage oscillation occurs. However, these are special-purpose hardware devices which most of us don't have available to plug into our computers.

Hence, pseudorandom generators still very much have their place in the world.

7 Getting other distributions from the uniform distribution

Finally — one way or another — we know how to generate uniformly distributed (pseudo)random sequences of integers mod 2^d, typically for $d = 32$ or $d = 64$. Then we can scale by $1/2^d$ and we'll get numbers uniformly distributed in the interval $[0, 1)$.

Let U be such a random number. Now we want to generate random numbers V with a desired CDF $F(x)$. The claim is that

$$V = F^{-1}(U)$$

works. First I want to assume that $F(x)$ is a strictly increasing function — it has no flat spots. Then we'll be able to invert it, as required by the algebra below. Second, I'll show what one can do when the CDF has flat spots — we can still define a sort of inverse. As is uncharacteristic for me, I'll do the abstract work first, and then an example.

7.1 Correctness

Suppose that $F(x)$ is invertible. Let $V = F^{-1}(U)$, for unit-uniform U. Let $G(x)$ be the CDF of V. We want to show that $G(x) = F(x)$. We have

$$\begin{aligned} G(x) &= P(V \leq x) \\ &= P(F^{-1}(U) \leq x) \\ &= P(U \leq F(x)). \end{aligned}$$

Now, there are three cases:

- If $1 \leq F(x)$, then $P(U \leq F(x))$ is 1: the unit-uniform random variable U is always less than or equal to 1.
- If $F(x) < 0$, then $P(U \leq F(x))$ is 0: U is never less than 0.
- If $0 \leq F(x) < 1$, then $P(U \leq F(x))$ is $F(x)$: in general, for $a \in [0, 1)$, $P(U \leq a) = a - 0 = a$.

But there really aren't three cases: the CDF $F(x)$ maps the real line to $(0, 1)$, so F^{-1} maps $(0, 1)$ back to $\mathbb{R}$. Keeping only the third case, we have $G(x) = F(x)$ which is what we wanted to show.

7.2 Example with invertible CDF

Here's a sample computation using the exponential distribution. Recall that the PDF of this random variable is

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

and its CDF is

$$F(x) = \int_0^x \lambda e^{-\lambda t}\, dt = \begin{cases} 1 - e^{-\lambda x} & x \geq 0 \\ 0 & x < 0. \end{cases}$$

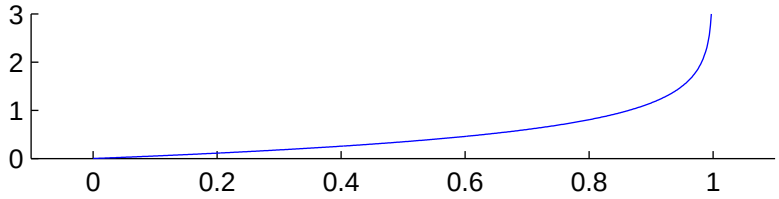
Now we want to invert the CDF. Doing the algebra, we obtain

$$F^{-1}(u) = \frac{-\ln(1-u)}{\lambda}, \quad u \in [0, 1).$$

Using the prescription above, all we need to do to generate pseudorandom numbers V with the exponential distribution with parameter λ is to start with unit-uniform random numbers U and compute

$$V = \frac{-\ln(1-U)}{\lambda}.$$

Here's a graph of V as a function of U:



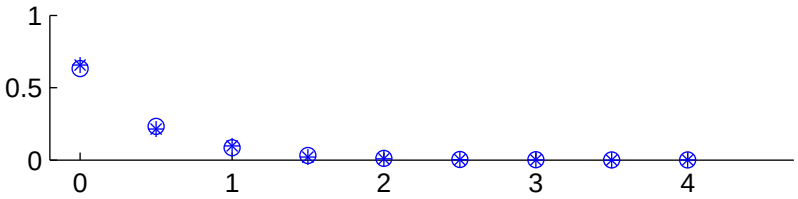
I'm going to take the very same sequence of U's as in section 6.4 (although there I called them $X_k/2^{32}$). Then, with $\lambda = 2$ again, I'll apply the above formula. Here is the output:

k	U_k	V_k
1	0.0287445	0.0145829
2	0.2142903	0.1205840
3	0.8758254	1.0430334
4	0.5243400	0.3715260
5	0.3435583	0.2104607
6	0.5449303	0.3936524
7	0.3727034	0.2331679
⋮	⋮	⋮
998	0.9425623	1.4285271
999	0.7202064	0.6368516
1000	0.8175253	0.8505719

Here are the bins and counts for V:

Bin	Count
$[0.0, 0.5)$	656
$[0.5, 1.0)$	215
$[1.0, 1.5)$	97
$[1.5, 2.0)$	20
$[2.0, 2.5)$	8
$[2.5, 3.0)$	2
$[3.0, 3.5)$	1
$[3.5, 4.0)$	0
$[4.0, 4.5)$	1

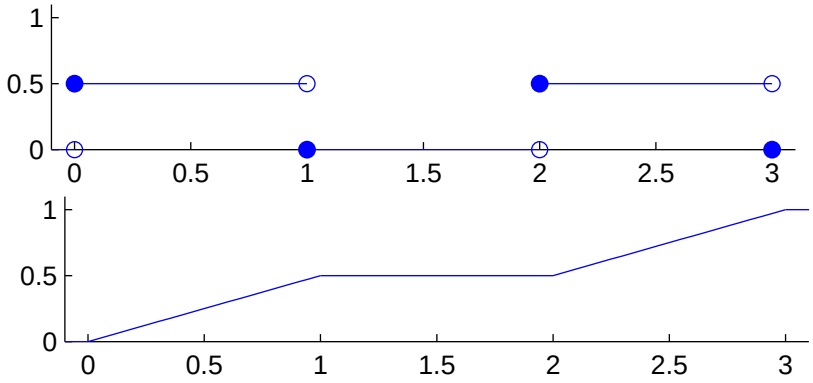
This should look familiar — these are the data I used in section 5.3. The difference is that now you know where it came from. Here again is the scaled histogram, along with expectations:



Having the correctness proof in section 7.1 tells me I'm attempting to do the right thing; having the histogram shows me I succeeded. At this point I feel confident my computer program does what it should.

7.3 Example with non-invertible CDF

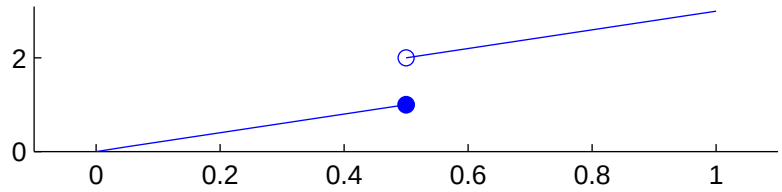
The second case above was when the CDF is non-invertible. Since CDFs are non-decreasing, this means there is one or more flat spots. Here I'll proceed by example. Imagine random numbers distributed uniformly, half on $[0, 1)$ and half on $[2, 3)$. The PDF $f(x)$ and CDF $F(x)$ look like this, respectively:



Now, we can't invert the CDF since it's flat from 1 to 2. But, that's not a problem. Remember that the CDF is useful as follows:

$$P(1 \leq V \leq 2) = F(2) - F(1).$$

Since $F(2) = F(1)$, there's zero probability of generating an V between 1 and 2. So, we can try to graph F^{-1}, using the usual freshman technique of swapping the horizontal and vertical axes. For the part that fails the vertical-line test, take its minimum value: define $F^{-1}(0.5)$ to be 1. Then we have this:



The formula for this is

$$V = \begin{cases} 2\,U, & 0 \le U < 1/2 \\ 2\,U + 1, & 1/2 \le U < 1. \end{cases}$$

This is perhaps what you would have guessed.

7.4 Non-invertible CDFs

I should point out that sometimes it's not possible to find a closed-form expression for the inverse of the CDF. This is the case, in particular, with the standard normal random variable. There are lots of techniques; you can do a web search. You will find references to Newton-Raphson iteration, the Box-Muller method (described in [**NR**]), the Ziggurat algorithm, and of course [**Knu2**].

7.5 Implementations

Almost any computing environment you are likely to find will have a unit-uniform random-number generator — and if not, you now know how to create your own. What about other distributions, though — do you need to do the CDF-inverting business above, or can you re-use someone else's efforts? Here it varies:

- A bare-bones library will give you a function which returns integers uniformly distributed between 0 and `RAND_MAX` (which is some pre-defined value contained in a header file). For example, the C library's `random` function does this.
- Often a library will also do scaling by 1/`RAND_MAX` for you, giving you unit-uniform random numbers: e.g. the C library's `drand48` routine.
- Python 2.5's `random` module and Matlab both implement several different distributions. In Python, do: `import random` followed by `help('random')`; in Matlab, `help random`.

8 Acknowledgements

This paper is intentionally interdisciplinary; I've drawn on personal experiences obtained over the course of many years and several jobs. However, I must particularly acknowledge Tom Kennedy's Math 564 (probability) course at the University of Arizona, from which much of the content of this paper (excluding section 6) has been eagerly and shamelessly appropriated.

References

[**DF**] D.S. Dummit and R.M. Foote. *Abstract Algebra* (2nd ed.). John Wiley and Sons, 1999.

[**FG**] Fristedt, B. and Gray, L. *A Modern Approach to Probability Theory.* Birkhauser, 1997.

[**GS**] Grimmett, G. and Stirzaker, D. *Probability and Random Processes*, 3rd ed. Oxford, 2001.

[**Ker**] Kerl, J. *Probability notes.* `http://math.arizona.edu/~kerl/doc/prb.pdf`

[**KK**] Kaplan, M. and Kaplan, E. *Chances Are . . . : Adventures in Probability.* Penguin, 2007.

[**Knu2**] Knuth, D.E. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, 3rd ed. Addison Wesley, 1997.

[**NR**] Press, W. et al. *Numerical Recipes* (2nd ed.). Cambridge, 1992.

Index

A
additive congruential generator 14

B
bell curve 8
bins 10
Box-Muller method 21

C
CDF 5
continuous random variable 5
Continuously distributed random numbers 4
cumulative distribution function 5

D
Discretely distributed random numbers 3
drand48 17

E
events 4
exponential random variable 8

G
generator 13

H
histogram 9

I
identically distributed 5
IID 5
independent 5

L
law of large numbers 10
LCG 14
linear congruential geneator 14

M
MCG 14
Mersenne twister 17
multiplicative congruential generator 14

O
outcomes 4

P
PDF 5
period 13
PMF 6
probability density function 5
probability mass function 6
probability measure 4
pseudorandom number generator 13

R
right-continuous 5

S
sample space 4
seed 13
standard normal 8, 21
step function 5

U
uniformly distributed random variable 7

Z
Ziggurat algorithm 21