# Custom distribution of random numbers

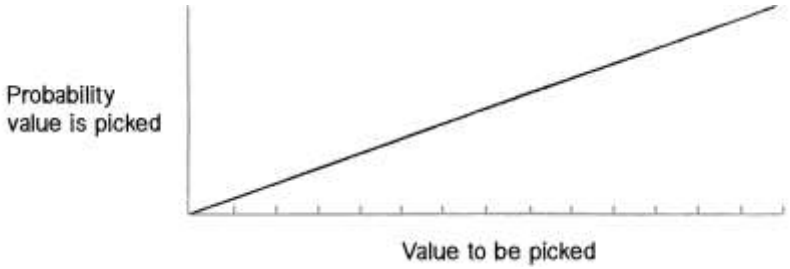
There will come a time in your life when you do not want a uniform distribution of random values, or a Gaussian one. Let's imagine for a moment that you are a random walker in search of food. Moving randomly around a space seems like a reasonable strategy for finding something to eat. After all, you don't know where the food is, so you might as well search randomly until you find it. The problem, as you may have noticed, is that random walkers return to previously visited locations many times (this is known as "oversampling"). One strategy to avoid such a problem is to, every so often, take a very large step. This allows the walker to forage randomly around a specific location while periodically jumping very far away to reduce the amount of oversampling. This variation on the random walk (known as a Lévy flight) requires a custom set of probabilities. Though not an exact implementation of a Lévy flight, we could state the probability distribution as follows: the longer the step, the less likely it is to be picked; the shorter the step, the more likely.

Earlier in this section, we saw that we could generate custom probability distributions by filling an array with values (some duplicated so that they would be picked more frequently) or by testing the result of `random()`. We could implement a Lévy flight by saying that there is a 1% chance of the walker taking a large step.

```
var r = random(1);
// A 1% chance of taking a large step
if (r < 0.01) {
  xstep = random(-100, 100);
  ystep = random(-100, 100);
} else {
  xstep = random(-1, 1);
  ystep = random(-1, 1);
```

```
}
```

However, this reduces the probabilities to a fixed number of options. What if we wanted to make a more general rule—the higher a number, the more likely it is to be picked? 3.145 would be more likely to be picked than 3.144, even if that likelihood is just a tiny bit greater. In other words, if `x` is the random number, we could map the likelihood on the y-axis with `y = x`.




Nature of Code Image

Figure I.4

If we can figure out how to generate a distribution of random numbers according to the above graph, then we will be able to apply the same methodology to any curve for which we have a formula.

One solution is to pick two random numbers instead of one. The first random number is just that, a random number. The second one, however, is what we'll call a "qualifying random value." It will tell us whether to use the first one or throw it away and pick another one. Numbers that have an easier time qualifying will be picked more often, and numbers that rarely qualify will be picked infrequently. Here are the steps (for now, let's consider only random values between 0 and 1):

1. Pick a random number: R1
2. Compute a probability P that R1 should qualify. Let's try: P = R1.
3. Pick another random number: R2
4. If R2 is less than P, then we have found our number—R1!

5. If R2 is not less than P, go back to step 1 and start over.

   Here we are saying that the likelihood that a random value will qualify is equal to the random number itself. Let's say we pick 0.1 for R1. This means that R1 will have a 10% chance of qualifying. If we pick 0.83 for R1 then it will have a 83% chance of qualifying. The higher the number, the greater the likelihood that we will actually use it.

   Here is a function (named after the Monte Carlo method, which itself was named after the Monte Carlo casino) that implements the above algorithm, returning a random value between 0 and 1. This program uses the values to size ellipses, but we could use those values for many things.