Inverse transform sampling

Matt Bonakdarpour *University of Chicago*

See here for a PDF version of this vignette.

Prerequisites

This document assumes basic familiarity with probability theory.

Overview

Inverse transform sampling is a method for generating random numbers from any probability distribution by using its inverse cumulative distribution, $F^{-1}(x)$. Recall that the cumulative distribution for a random variable X is $F_X(x) = P(X \leq x)$. In what follows, we assume that our computer can, on demand, generate independent realizations of a random variable U uniformly distributed on $[0, 1]$.

Algorithm

Continuous distributions

Assume we want to generate a random variable X with cumulative distribution function (CDF) F_X. The inverse transform sampling algorithm is simple:

1. Generate $U \sim \text{Unif}(0, 1)$.
2. Let $X = F_X^{-1}(U)$.

Then X will follow the distribution governed by the CDF F_X, which was our desired result.

Note that this algorithm works in general but is not always practical. For example, inverting F_X is easy if X is an exponential random variable, but it is harder if X is normal random variable.

Discrete distributions

Now we will consider the discrete version of the inverse transform method. Assume that X is a discrete random variable such that $P(X = x_i) = p_i$. The algorithm proceeds as follows:

1. Generate $U \sim \text{Unif}(0, 1)$.
2. Determine the index k such that $\sum_{j=1}^{k-1} p_j \leq U < \sum_{j=1}^{k} p_j$, and return $X = x_k$.

(Notice that the second step requires a *search*.)

Assume our random variable X takes on one of K values with probabilities $\{p_1, \ldots, p_K\}$. We implement the algorithm below, assuming these probabilities are stored in a vector called "pvec".

```
discrete_inv_transform_sample <- function (pvec) {
  K <- length(pvec)
  u <- runif(1)
  if (u <= pvec[1])
    return(1)
  for (k in seq(2,K)) {
    if(sum(pvec[seq(1,k-1)]) < u && u <= sum(pvec[seq(1,k)]))
      return(k)
  }
}
```

Note that this is this an inefficient implementation given here for pedagogical purposes.

Continuous example: exponential distribution

Assume Y is an exponential random variable with rate parameter $\lambda = 2$. Recall that the probability density function is $p(y) = 2e^{-2y}$, for $y > 0$. First, we derive the CDF:

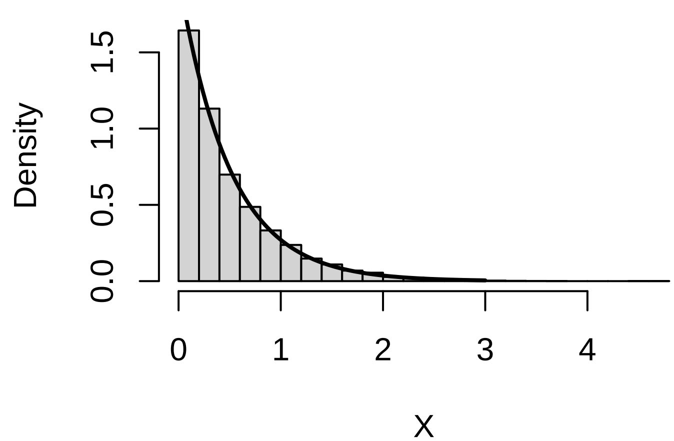
$$F_Y(x) = P(Y \leq x) = \int_0^x 2e^{-2y} dy = 1 - e^{-2x}.$$

Solving for the inverse CDF, we get that

$$F_Y^{-1}(y) = -\tfrac{1}{2}\log(1 - y).$$

Using our algorithm above, we first generate $U \sim \text{Unif}(0,1)$, then set $X = -\log(1-U)/2$. We do this in the R code below and compare the histogram of our samples with the true density of Y.

```
set.seed(1)
num_samples <- 10000
u <-  runif(num_samples)
x <- -log(1-u)/2
hist(x,n = 32,freq = FALSE,xlab = 'X',main = "")
curve(dexp(x,rate = 2),0,3,lwd = 2,add = TRUE)
```



Indeed, the random draws appear are following the intended distribution.

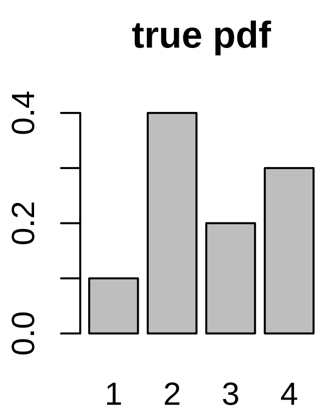
Discrete example

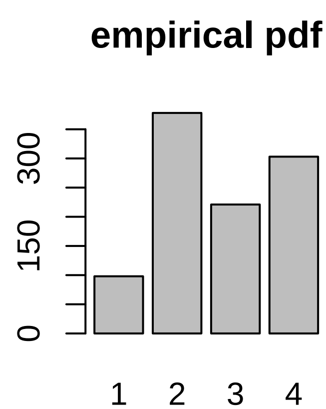
Let's assume we want to simulate a discrete random variable X that follows the following distribution:

k	$\Pr(X = k)$
1	0.1
2	0.4
3	0.2
4	0.3

Below we simulate from this distribution using the "discrete_inv_transform_sample" function above, and plot both the true probability vector, and the empirical proportions from our simulation.

```r
par(mfcol = c(1,2))
num_samples <- 1000
pvec        <- c(0.1,0.4,0.2,0.3)
names(pvec) <- 1:4
samples     <- rep(0,num_samples)
for(i in seq_len(num_samples)) {
  samples[i] <- discrete_inv_transform_sample(pvec)
}
barplot(pvec,main = "true pdf")
barplot(table(samples),main = "empirical pdf")
```





Again, the plot supports our claim that we are drawing from the correct probability distribution.