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 its 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 can take on any 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 `p.vec`.

```r
# note: this inefficient implementation is for pedagogical purposes
# in general, consider using the rmultinom() function
discrete.inv.transform.sample <- function( p.vec ) {
  U  <- runif(1)
  if(U <= p.vec[1]){
    return(1)
  }
  for(state in 2:length(p.vec)) {
    if(sum(p.vec[1:(state-1)]) < U && U <= sum(p.vec[1:state]) ) {
      return(state)
    }
  }
}
```

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 compute the CDF:

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

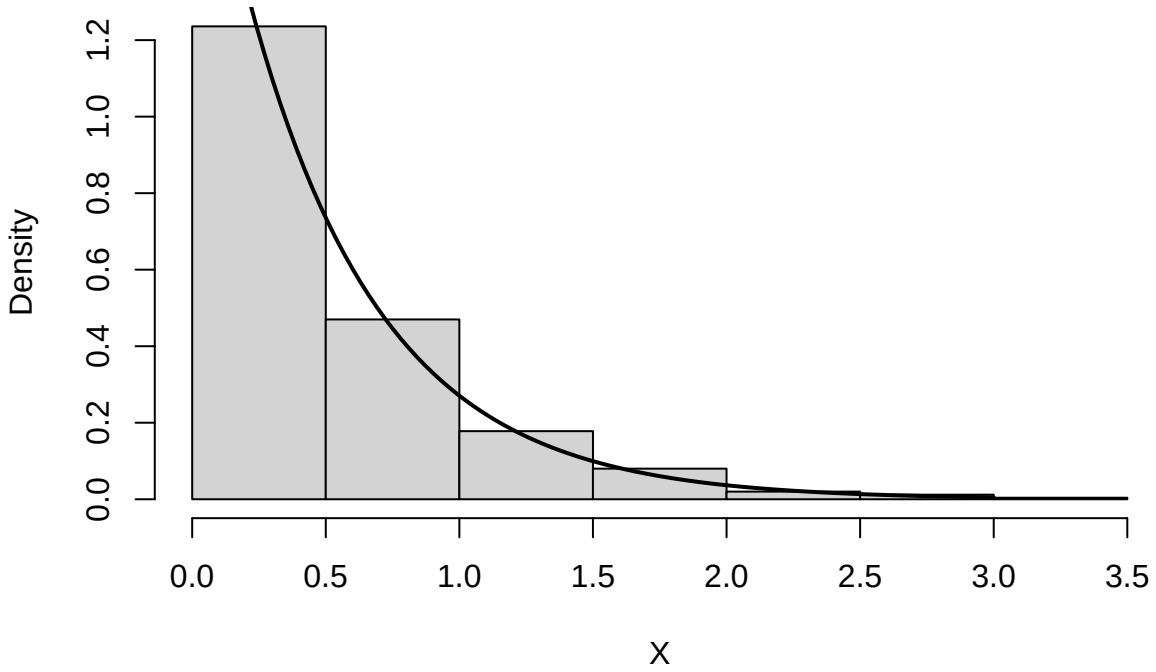
Solving for the inverse CDF, we get that

$$F_Y^{-1}(y) = -\frac{\ln(1-y)}{2}$$

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

```r
# inverse transform sampling
num.samples <-  1000
U           <-  runif(num.samples)
X           <- -log(1-U)/2

# plot
hist(X, freq=F, xlab='X', main='Generating Exponential R.V.')
curve(dexp(x, rate=2) , 0, 3, lwd=2, xlab = "", ylab = "", add = T)
```



Indeed, the plot indicates that our random variables are following the intended distribution.

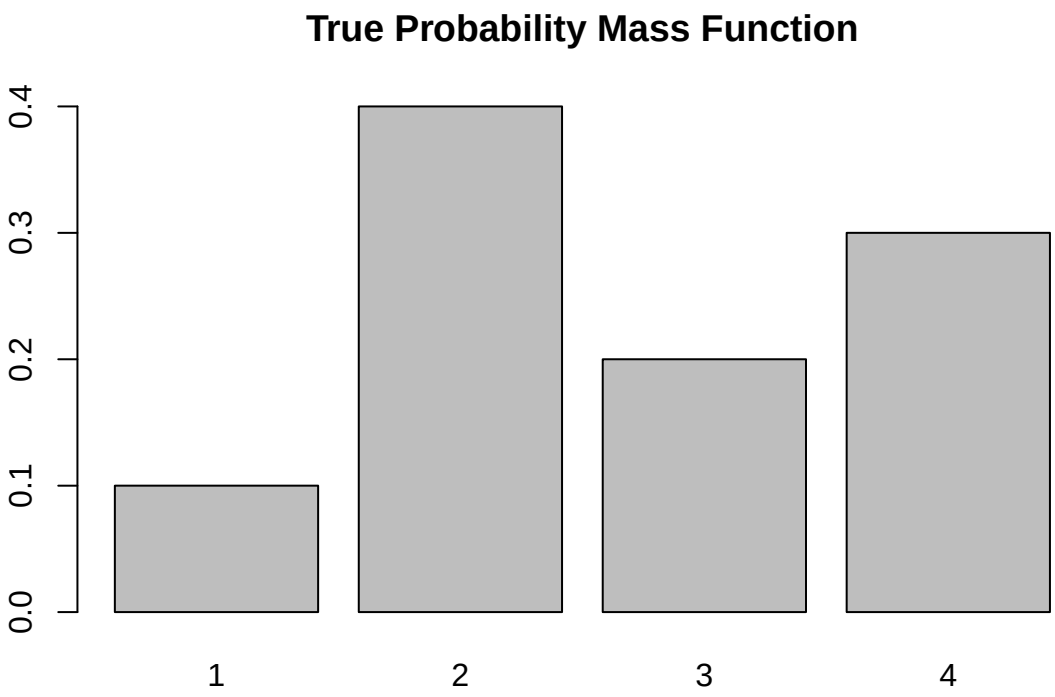
Discrete Example

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

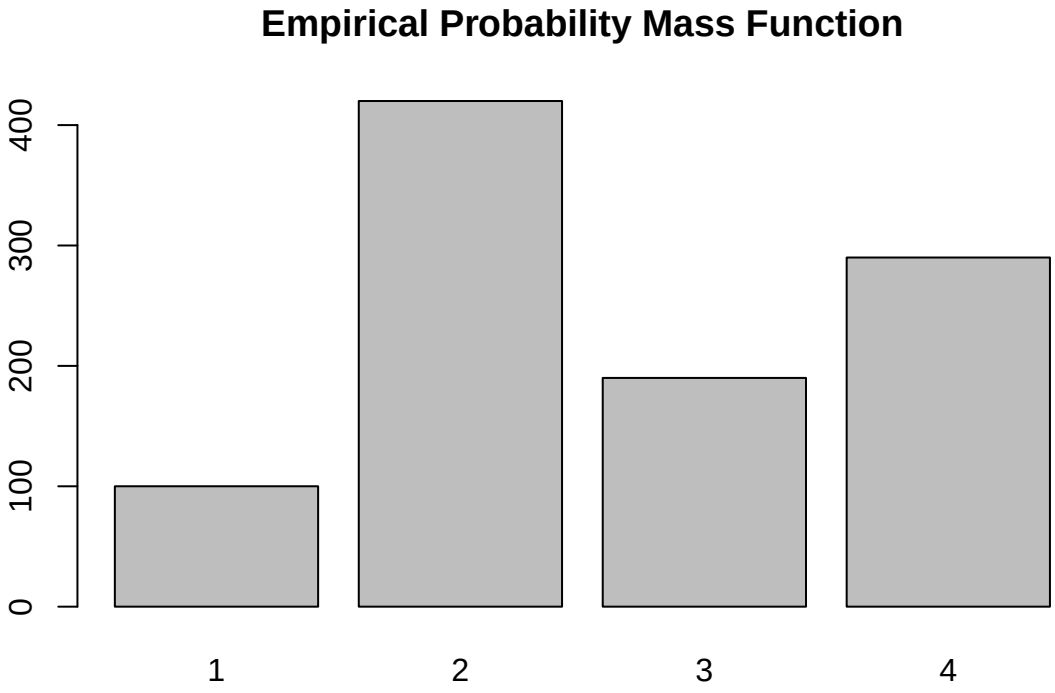
x_i	P(X=x_i)
1	0.1
2	0.4
3	0.2
4	0.3

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.

```
num.samples <- 1000
p.vec        <- c(0.1, 0.4, 0.2, 0.3)
names(p.vec) <- 1:4
samples     <- numeric(num.samples)
for(i in seq_len(num.samples) ) {
  samples[i] <- discrete.inv.transform.sample(p.vec)
}
barplot(p.vec, main='True Probability Mass Function')
```




```
barplot(table(samples), main='Empirical Probability Mass Function')
```



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