Simulation of Random Variables (spuRs Ch.18)

Most stochastic simulations have the same basic structure:

- Identify a random variable of interest X and write a program to simulate it.
- Generate an iid sample $X_1, \ldots, X_n$ with the same distribution as X.
- Estimate $\mathbb{E}X$ (using $\overline{X}$) and assess the accuracy of the estimate (using a confidence interval).

Step 1 is an example of *model building*. Typically we build up a complex model from simple components, which in this case are independent rv's with known distributions.

It turns out that all random variables can be generated by manipulating $U(0,1)$ rv's.

Quantile Functions and random number generation

Let $F : \mathbb{R} \to [0,1]$ be the cumulative distribution function of a random variable X. Define its *quantile function* $Q : [0,1] \to \mathbb{R}$ by

$$Q(p) = \inf\{y : F(y) \geq p\}, \quad \forall\ 0 < p < 1$$

and

$$Q(1) = \begin{cases} \infty & \text{if } F(y) < 1\ \forall y \in \mathbb{R} \\ \inf\{y : F(y) = 1\} & \text{otherwise} \end{cases}$$

and $Q(0) = -\infty$. It can be easily verified that

(a) The function Q is nondecreasing and $F(Q(t)) \geq t$ for all $0 < t < 1$.

(b) The function Q is left continuous and $Q(F(x)) \leq x$ for all $x \in \mathbb{R}$.

(c) Using (a) and (b), $F(x) \geq t$ iff $x \geq Q(t)$.

(d) Using (c), $Q(U) \overset{d}{\sim} X$ if $U \sim U(0,1)$.

(e) $Q(F(x)) = x \ \forall x$ for which F is 1-1, or equivalently increasing strictly, in a neighborhood of x.

(f) If F is onto the interval $(0,1)$, then $F(Q(t)) = t$ for all $t \in (0,1)$.

(g) If $F : (a,b) \to (0,1)$ is 1-1 and onto for some $a < b$, then the above quantile function Q is the same as the inverse function F^{-1} of F.

For discrete distribution, the quantile function is easily obtained and can be used via (d) to generate a random deviate of the desired distribution. For continuous distribution, if the inverse of its CDF has a closed form then it can be used to generate a random deviate of the desired distribution via (d,g).

Simulating iid uniform samples

We cannot generate truly random numbers on a computer. Instead we generate *pseudo-random numbers*, which have the appearance of random numbers, but are in fact completely deterministic. Pseudo-random numbers can be generated by chaotic dynamical systems, which have the characteristic that the future is very hard to predict given the present.

A very important advantage of using pseudo-random numbers is that, because they are deterministic, any experiment performed using pseudo-random numbers can be repeated exactly.

隨機數的同餘生成法

seed

Congruential generators Congruential generators were the first reasonable class of pseudo-random number generators. R uses a pseudo-random number generator called the *Mersenne-Twister*, which has similar properties to congruential generators.

Given an initial number $X_0 \in \{0, 1, \ldots, m-1\}$ and two big numbers A and B we define a sequence of numbers $X_n \in \{0, 1, \ldots, m-1\}, n = 0, 1, \ldots,$ by

$$X_{n+1} = (AX_n + B) \bmod m.$$

We get a sequence of numbers $U_n \in [0, 1), n = 0, 1, \ldots,$ by putting $U_n = X_n/m$. If m, A, and B are well chosen then the sequence $U_0, U_1, \ldots,$ is almost impossible to distinguish from an iid sequence of $U(0, 1)$ random variables.

In practice it is sensible to discard the value 0 when it occurs, as we often divide by U_n. This is justifiable since for a true uniform, the probability of taking on the value 0 is zero. The value 1 can also be a problem, but note that as defined, $U_n < 1$ for all n.

X_{n+1} = A*X_{n} + B = 103*X_{n} +17, m = 10
=> X_{1} = 103*X_{0} + 17 = 103*2 + 17 = 206 + 17 = 223 mod 10 = 3
=> X_{2} = 103*X_{1} + 17 = 103*3 + 17 = 309 + 17 = 326 mod 10 = 6
=> X_{3} = 103*X_{2} + 17 = 103*6 + 17 = 618 + 17 = 635 mod 10 = 5

If we take $m = 10$, $A = 103$, and $B = 17$, then for $X_0 = 2$, we have

$$\begin{aligned} X_1 &= 223 \bmod 10 = 3 \\ X_2 &= 326 \bmod 10 = 6 \\ X_3 &= 635 \bmod 10 = 5 \\ &\vdots \end{aligned}$$

Clearly the sequence produced by a congruential generator will eventually cycle and thus since there are at most m possible values, the maximum cycle length is m. (The Mersenne-Twister has a cycle length of $2^{19937} - 1$.)

i.e. period

Because computers use binary arithmetic, if we have $m = 2^k$ for some k, then taking $x \bmod m$ is very quick. An example of a good congruential generator is $m = 2^{32}$, $A =$1,664,525, and $B =$ 1,013,904,223. An example of a bad congruential generator is RANDU, which was shipped with IBM computers in the 1970's. RANDU used $m = 2^{31}$, $A =$65,539, and $B = 0$.

Since m = 10, so X_{n+1} = A*X_{n} + B = 3*X_{n} + 7
=> X_{1} = 3*X_{0} + 7 = 3*2 + 7 = 6 + 7 = 13 mod 10 = 3
=> X_{2} = 3*X_{1} + 7 = 3*3 + 7 = 9 + 7 = 16 mod 10 = 6
=> X_{3} = 3*X_{2} + 7 = 3*6 + 7 = 18 + 7 = 25 mod 10 = 5

Seeding The number X_0 is called the seed. If you know the seed (as well as m, A, and B), then you can reproduce the whole sequence exactly. This is a very good idea from a scientific point of view; being able to repeat an experiment means that your results are verifiable.

To generate n pseudo-random numbers in R, use `runif(n)`. R does not use a congruential generator, but is still needs a seed to generate pseudo-random numbers. For a given value of `seed` (assumed integer), the command `set.seed(seed)` always puts you at the same point on the cycle of pseudo-random numbers.
The current state of the random number generator is kept in the vector `.Random.seed`. You can save the value of `.Random.seed` and then use it to return to that point in the sequence of pseudo-random numbers.
If the random number generator is not initialised before you start generating pseudo-random numbers, then R initialises it using a value taken from the system clock.

```
> set.seed(42)
> runif(2)

[1] 0.9148060 0.9370754

> RNG.state <- .Random.seed
> runif(2)

[1] 0.2861395 0.8304476

> set.seed(42)
> runif(4)

[1] 0.9148060 0.9370754 0.2861395 0.8304476

> .Random.seed <- RNG.state
> runif(2)

[1] 0.2861395 0.8304476
```

Simulating discrete random variables

Let X be a discrete random variable taking values in the set $\{0, 1, \ldots\}$ with cdf F and pmf p. The following snippet of code takes a uniform random variable U and returns a discrete random variable X with cdf F.

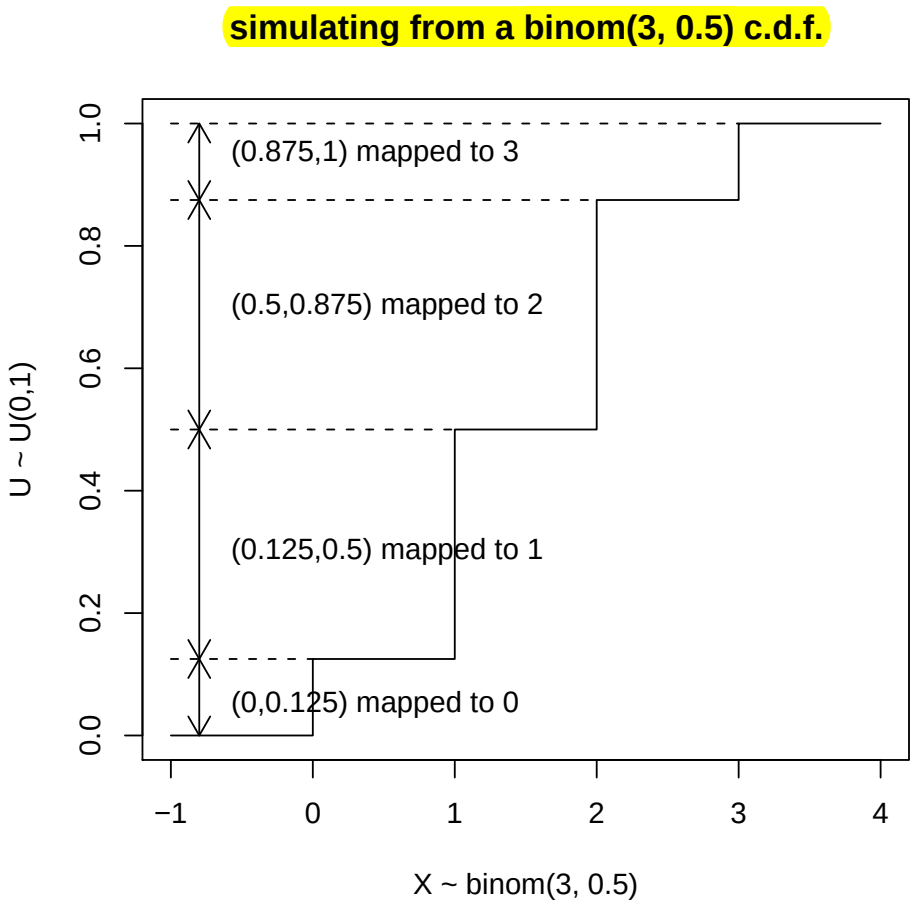
```
# given U ~ U(0,1)
X <- 0
while (F(X) < U) {
    X <- X + 1
}
```

When the algorithm terminates we have $F(X) \geq U$ and $F(X-1) < U$, that is $U \in (F(X-1), F(X)]$. Thus

$$\mathbb{P}(X = x) = \mathbb{P}(U \in (F(x-1), F(x)]) = F(x) - F(x-1) = p(x)$$

as required.

Upper case X should be replaced by lower case x.



See page 335 of the text.

sample(population, k) (to get a random sample of size k without replacement from a finite population
sample(population, k, replace=T) (to get a random sample of size k with replacement from a finite population
sample(x, k, prob) (to get a random sample of size k without replacement from a finite population of elements given in x with probabilities specified by prob)
sample(x, k, replace=T, prob) (to get a random sample of size k with replacement from a finite population of elements given in x with probabilities specified by prob)

To simulate binomial, geometric, negative-binomial or Poisson rv's in R, use `rbinom`, `rgeom`, `rnbinom` or `rpois`. For simulating other discrete rv's R provides

```
sample(x, size, replace = FALSE, prob = NULL)
```

The inputs are

`x` A vector giving the possible values the rv can take;

`size` How many rv's to simulate;

`replace` Set this to `TRUE` to generate an iid sample, otherwise the rv's will be conditioned to be different from each other;

`prob` A vector giving the probabilities of the values in `x`. If omitted then the values in `x` are assumed to be equally likely.

See more examples in the subsections
18.2.1 Binomial
18.2.2 Sequence of independent trials

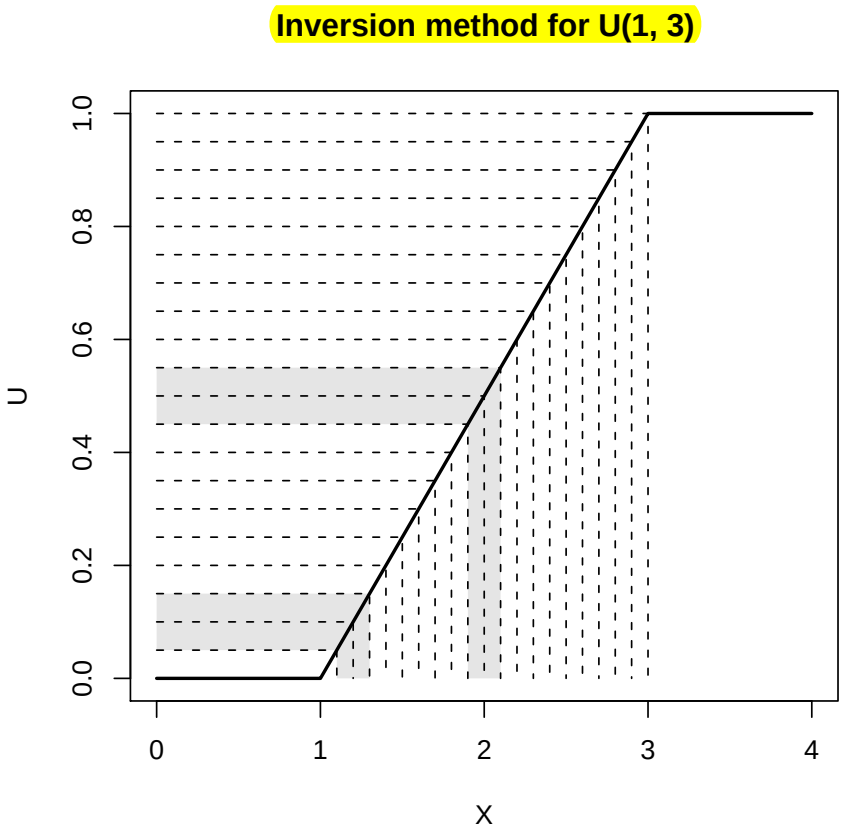
Simulating continuous random variables

Suppose that we are given $U \sim U(0,1)$ and want to simulate a continuous rv X with cdf F_X. Put $Y = F_X^{-1}(U)$ then we have

$$F_Y(y) = \mathbb{P}(Y \leq y) = \mathbb{P}(F_X^{-1}(U) \leq y) = \mathbb{P}(U \leq F_X(y)) = F_X(y).$$

That is, Y has the same distribution as X.

Thus, if we can simulate a $U(0,1)$ rv, then we can simulate any continuous rv X for which we know F_X^{-1}. This is called the *inverse transformation method* or simply the *inversion method*.



If $X \sim U(1,3)$ then $F_X(x) = (x-1)/2$ for $x \in (1,3)$ and thus $F_X^{-1}(y) = 2y+1$ for $y \in (0,1)$.

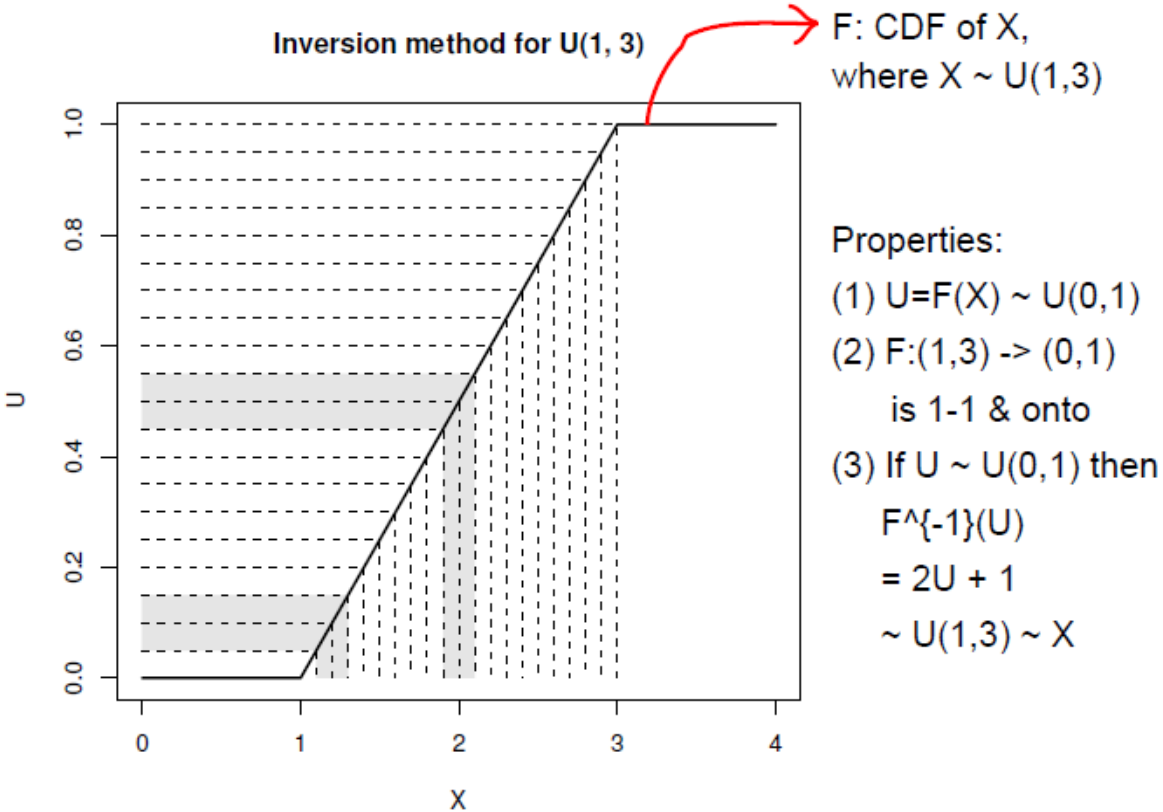


Figure 18.3 *Illustration of the inversion method. A 'uniform rain' of points on the vertical interval* $(0, 1)$ *becomes a uniform rain on the horizontal interval* $(1, 3)$.

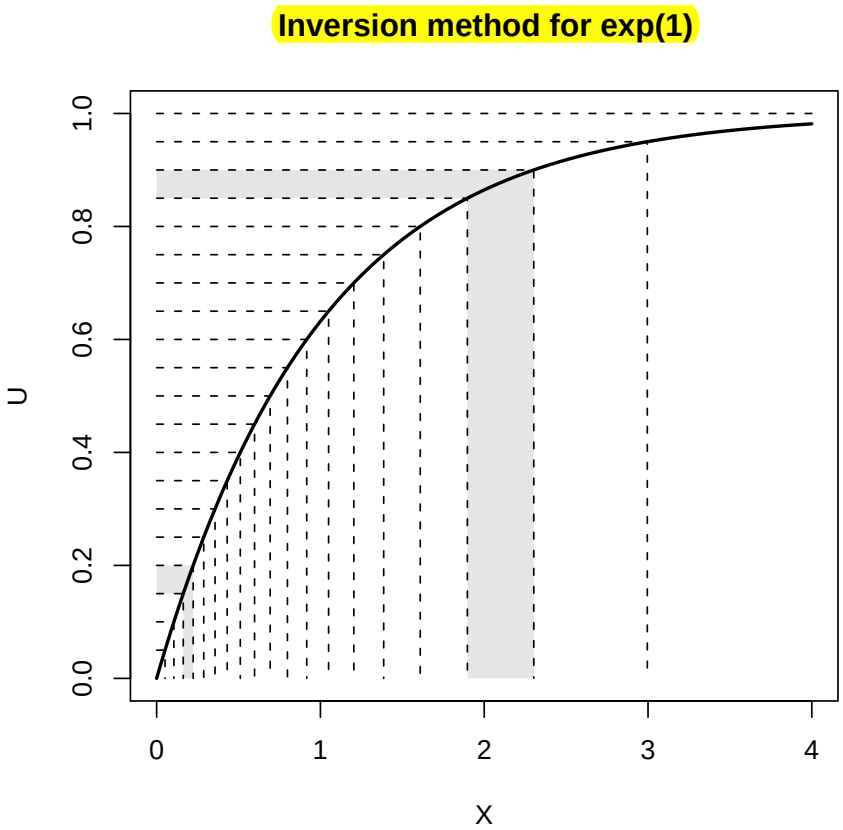
18.3.2 Example: exponential distribution

If $X \sim \exp(\lambda)$ then the pdf is $f_X(x) = \lambda e^{-\lambda x}$, for $x > 0$, and by integrating we find

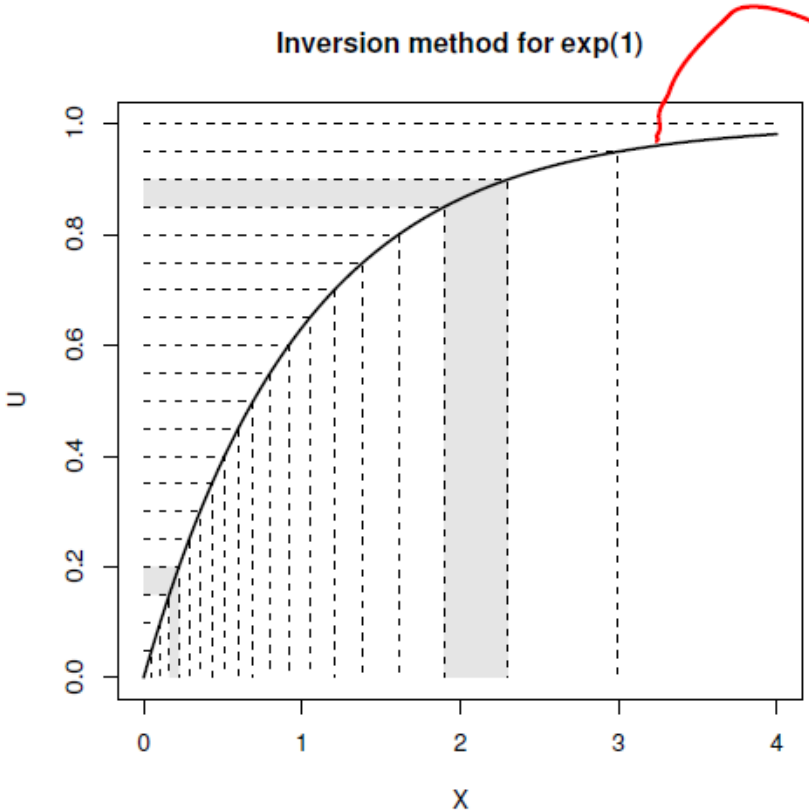
$$F_X(x) = \begin{cases} 0 & \text{for } x < 0; \\ 1 - e^{-\lambda x} & \text{for } x \geq 0. \end{cases}$$

Putting $y = F_X(x)$ we derive the inverse function as follows:

$$\begin{aligned} y &= 1 - e^{-\lambda x} \\ 1 - y &= e^{-\lambda x} \\ \log(1-y) &= -\lambda x \\ x &= -\frac{1}{\lambda}\log(1-y) = F_X^{-1}(y). \end{aligned}$$



If $X \sim \exp(\lambda)$ then $F_X(x) = 1 - e^{-\lambda x}$ for $x \geq 0$ and thus $F_X^{-1}(y) = -\frac{1}{\lambda}\log(1-y)$.



F: CDF of X,
where X ~ Exp(1)

Properties:

(1) U=F(X) ~ U(0,1)

(2) F:(0,Inf) -> (0,1)
is 1-1 & onto

(3) If U ~ U(0,1) then
F^{-1}(U)
= - log(1-U)
~ Exp(1) ~ X

~ - log(U)
since U ~ (1-U)

Figure 18.4 *Illustration of the inversion method. A 'uniform rain' of points on the vertical interval* $(0, 1)$ *becomes an 'exponentially distributed rain' on the horizontal axis.*

To simulate exponential, gamma or normal rv's in R, use `rexp`, `rgamma` or `rnorm`.

Most commonly encountered distributions are provided in R, for example the beta, Weibull, t, F and χ^2.

18.4 Rejection method for continuous rv

The inversion method works well if we can find F^{-1} analytically. If not. we can use root-finding techniques to invert F numerically (see Exercise 16), but this can be time-consuming. An alternative method in this situation, which is often faster, is the rejection method.

To motivate the rejection method let us consider a simple example. Say we have a continuous random variable X with pdf f_X concentrated on the interval $(0, 4)$, as illustrated in Figure 18.5. We imagine 'sprinkling' points $P_1, P_2, \ldots,$ uniformly at random under the density function. By sprinkling uniformly, we mean that a small target square under the pdf has the same chance of being hit wherever it is located. Our random points P_i are actually two-dimensional random variables (Xi, Yi), where Xi and Yi are the random coordinates of the i-th point.

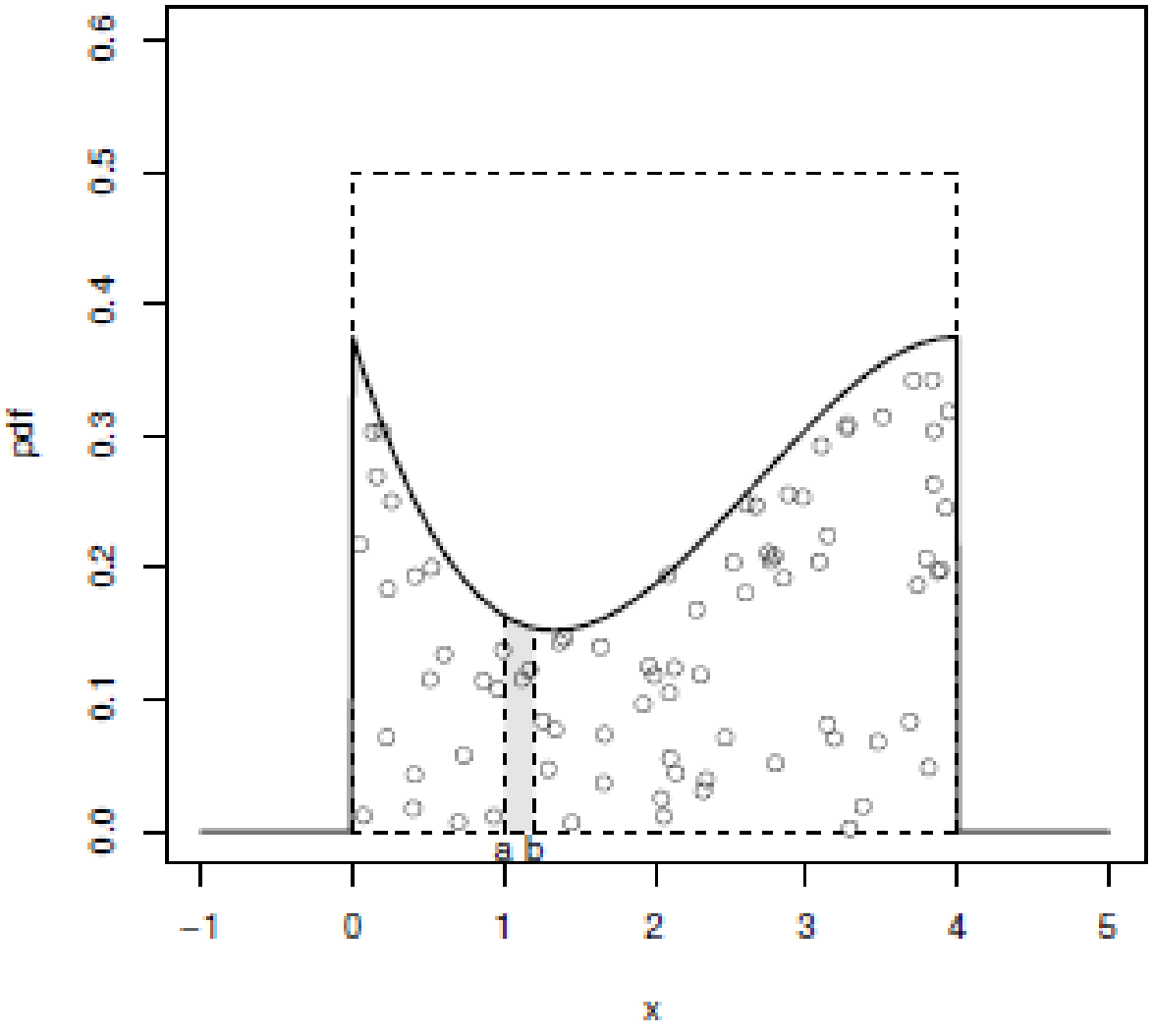


Figure 18.5 *Points uniformly distributed under a pdf.*

Consider the distribution of X_1, the x coordinate of P_1. (Note that all X_i have the same distribution.) Let R be the shaded region under f_X between a and b, as shown in Figure 18.5, then

$$\begin{aligned}
\mathbb{P}(a < X_1 < b) &= \mathbb{P}(P_1 \text{ hits R}) \\
&= \frac{\text{Area of R}}{\text{Area under density}} \\
&= \frac{\int_a^b f_X(x)dx}{1} \\
&= \int_a^b f_X(x)dx.
\end{aligned}$$

Thus by the definition of the pdf, X_1 has the same distribution as X. So we can generate observations on X by taking the x coordinate of random points sprinkled under its pdf f_X. But how do we generate the points P_i uniformly under f_X? The answer is to generate points at random in the rectangle $[0, 4] \times [0, 0.5]$ (dotted in Figure 18.5), and then *reject* those that fall above the pdf, hence the name *rejection method.*

This method extends to any density with finite support that is bounded above. That is, $f_X(x) \leq k$ for all x and some constant k.

Rejection method (uniform envelope) Suppose that f_X is non-zero only on $[a, b]$, and $f_X \leq k$.

1. Generate $X \sim U(a, b)$ and $Y \sim U(0, k)$ independent of X (so $P = (X, Y)$ is uniformly distributed over the rectangle $[a, b] \times [0, k]$).
2. If $Y < f_X(X)$ then return X, otherwise go back to step 1.

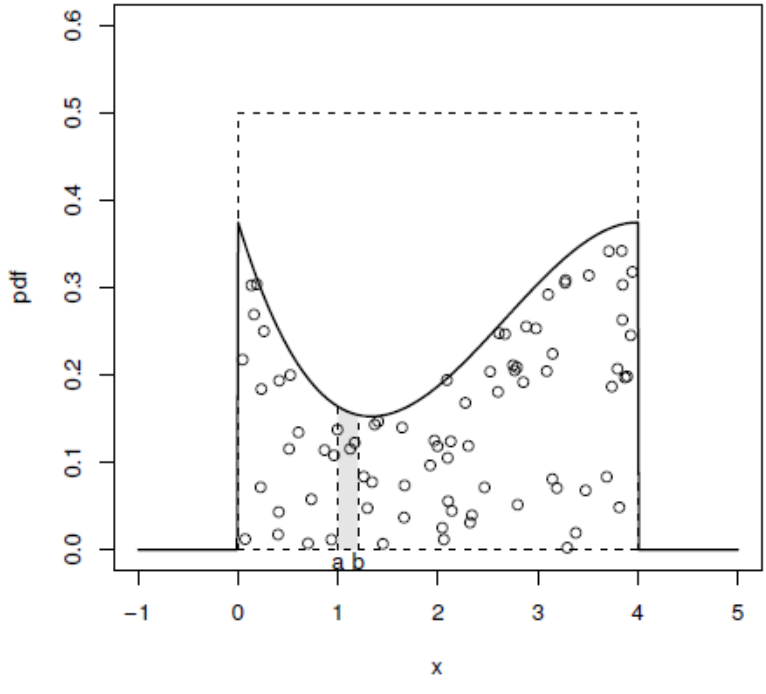


Figure 18.5 *Points uniformly distributed under a pdf.*

18.4.1 Example: triangular density

Consider the triangular pdf f_X defined as

$$f_X(x) = \begin{cases} x & \text{if } 0 < x < 1; \\ (2-x) & \text{if } 1 \leq x < 2; \\ 0 & \text{otherwise.} \end{cases}$$

We apply the rejection method as follows:

```
# program spuRs/resources/scripts/rejecttriangle.r

rejectionK <- function(fx, a, b, K) {
  # simulates from the pdf fx using the rejection algorithm
  # assumes fx is 0 outside [a, b] and bounded by K
```

```
  # note that we exit the infinite loop using the return statement
  while (TRUE) {
    x <- runif(1, a, b)
    y <- runif(1, 0, K)
    if (y < fx(x)) return(x)
  }
}

fx<-function(x){
  # triangular density
  if ((0<x) && (x<1)) {
    return(x)
  } else if ((1<x) && (x<2)) {
    return(2-x)
  } else {
    return(0)
  }
}

# generate a sample
set.seed(21)
nreps <- 3000
Observations <- rep(0, nreps)
for(i in 1:nreps)   {
  Observations[i] <- rejectionK(fx, 0, 2, 1)
}

# plot a scaled histogram of the sample and the density on top
hist(Observations, breaks = seq(0, 2, by=0.1), freq = FALSE,
     ylim=c(0, 1.05), main="")
lines(c(0, 1, 2), c(0, 1, 0))
```

The output is given in Figure 18.6.

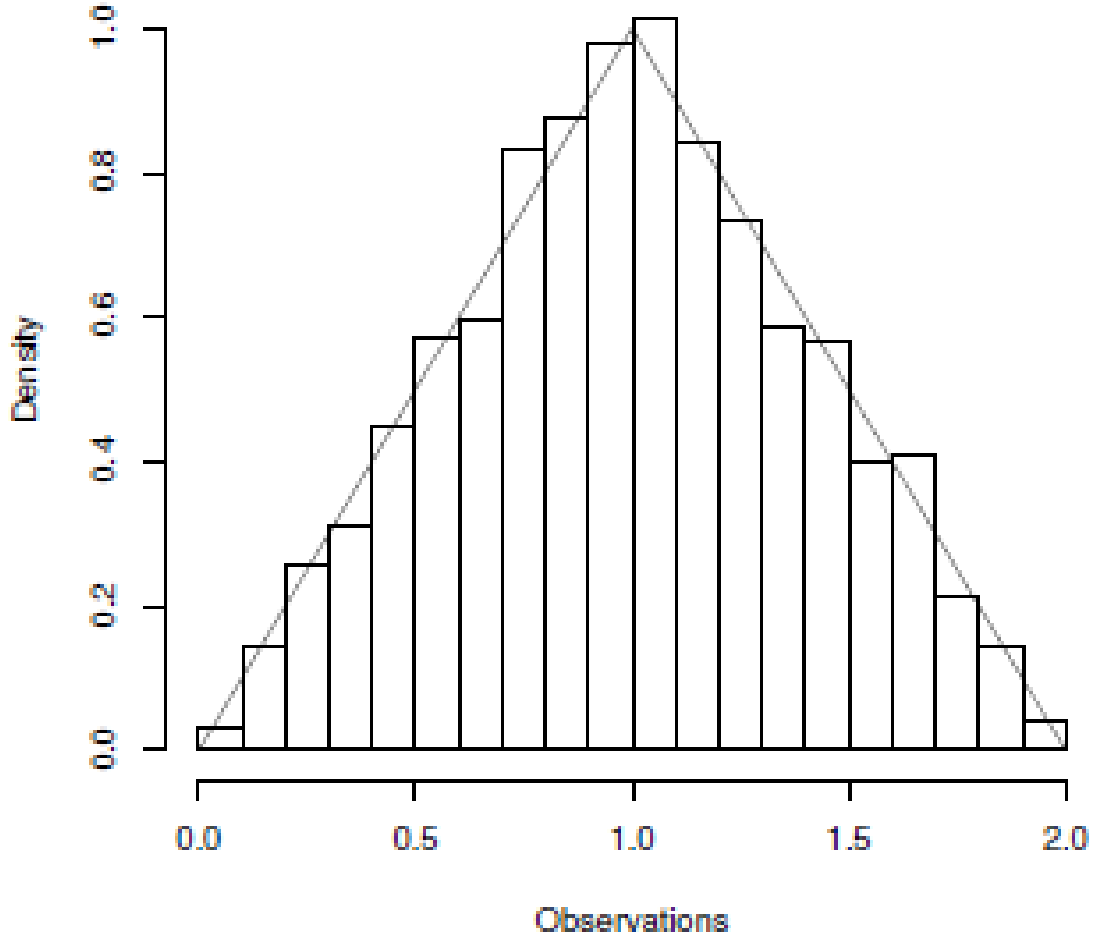


Figure 18.6 *Empirical pdf of the triangular distribution, simulated using the rejection method.*

18.4.2 General rejection method

Our rejection method above uses a rectangular envelope to cover the target density f_X, then generates candidate points uniformly within the rectangle. However if the rectangle is infinite, then we cannot generate points uniformly within it, because it has infinite area. Instead we need a shape with finite area, within which we can simulate points uniformly.

Let X have pdf h and, given X, let $Y \sim U(0, kh(X))$ (so the range of Y depends on X), then (X, Y) is uniformly distributed over the region A defined by the curve kh above and 0 below. To see this we use conditional probability:

$$\begin{aligned}
&\mathbb{P}((X,Y) \in (x, x+dx) \times (y, y+dy)) \\
&= \mathbb{P}(Y \in (y, y+dy) \,|\, X \in (x, x+dx))\mathbb{P}(X \in (x, x+dx))
\end{aligned}$$

= (dy / kh(x)) h(x)dx = 1/k dxdy, where k is the area of A.

Suppose we wish to simulate from the density f_X. Let h be a density we can simulate from, and choose k such that

$$k \geq k^* = \sup_x \frac{f_X(x)}{h(x)}.$$

Note that $k^* \geq 1$, with equality if and only if f_X and h are identical. Then kh forms an envelope for f_X, and we can generate points uniformly within this envelope. By accepting points below the curve f_X, we get the general rejection method:

General rejection method

To simulate from the density f_X, we assume that we have envelope density h from which you can simulate, and that we have some $k < \infty$ such that $\sup_x f_X(x)/h(x) \leq k$.

1. Simulate X from h.
2. Generate $Y \sim U(0, kh(X))$.
3. If $Y < f_X(X)$ then return X, otherwise go back to step 1.

18.4.3 Efficiency

The efficiency of the rejection method is measured by the expected number of times you have to generate a candidate point (X, Y). The area under the curve kh is k and the area under the curve f_X is 1, so the probability of accepting a candidate is $1/k$. Thus the number of times N we have to generate a candidate point has distribution $1+\text{geom}(1/k)$, with mean $\mathbb{E}N = 1+(1-1/k)/(1/k) = k$. So, the closer h is to f_X, the smaller we can choose k, and the more efficient the algorithm is.

General Rejection Method

To simulate from the pdf f_X, assume that

- an envelop pdf h exists so that $f_X \leq kh$ for some constant $k \geq 1$,
- it is easy to generate a random observation V from h, and
- h and f_X have same support s.t. $\sup_x \frac{f_X(x)}{h(x)} < \infty$

To generate a random observation $X \sim f_X$

(1) Generate a candidate $V \sim h$

(2) Given the above V, generate $U \sim U(0, kh(V))$

(3) If $U < f_X(V)$ then set $X = V$ (accept V)
otherwise return to step (1) (reject V)

Motivation as well as proof.

$$
\begin{aligned}
P(X \le x) &= \int_{-\infty}^{x} f_X(v)dv = \frac{c\int_{-\infty}^{x} f_X(v)dv}{c} \qquad \text{for all nonzero c} \\
&= \frac{\frac{1}{k}\int_{-\infty}^{x} f_X(v)dv}{\frac{1}{k}} \qquad \text{by taking } c = 1/k \\
&= \frac{\int_{-\infty}^{x}\int_{0}^{f_X(v)} \frac{1}{k}dudv}{\int_{-\infty}^{\infty}\int_{0}^{f_X(v)} \frac{1}{k}dudv} = \frac{\int_{-\infty}^{x}\int_{0}^{f_X(v)} \frac{1}{kh(v)}h(v)dudv}{\int_{-\infty}^{\infty}\int_{0}^{f_X(v)} \frac{1}{kh(v)}h(v)dudv} \\
&= \frac{P(V \le x \text{ and } U < f_X(V))}{P(U < f_X(V))} \\
&= P(V \le x | U < f_X(V)) = P(V \le x | \text{ accept } V)
\end{aligned}
$$

Efficiency. Note that

$$k = \int_{-\infty}^{\infty} \int_{0}^{kh(v)} du dv$$

is the area of the envelop region R hat is bounded by the curves $y = kh(x)$ and $y = 0$. Also (U, V) is uniformly distributed on R and thus

$$P(U \leq f_X(V)) = \frac{\text{area under } f_X}{\text{area of } R} = \frac{1}{k}.$$

In practice, it is desirable to use a small k so as to increase the above acceptance probability of the method. Let N be the number of (U, V) pairs needed for accepting V to be an X. It is clear that N can be viewed as the number of Bernoulli trials for getting a success. Thus, $N \sim geometric(1/k)$ and $E(N) = 1/(1/k) = k$. To optimize the rejection algorithm, we minimize k.

18.4.4 Example: gamma

For $m, \lambda > 0$ the $\Gamma(\lambda, m)$ density is $f(x) = \lambda^m x^{m-1} e^{-\lambda x}/\Gamma(m)$, for $x > 0$. There is no explicit formula for the cdf F or its inverse, so we will use the rejection method to simulate from f.

We will use an exponential envelope $h(x) = \mu e^{-\mu x}$, for $x > 0$. Using the inversion method we can easily simulate from h using $-\log(U)/\mu$, where $U \sim U(0,1)$. To envelop f we need to find

$$k^* = \sup_{x>0} \frac{f(x)}{h(x)} = \sup_{x>0} \frac{\lambda^m x^{m-1} e^{(\mu-\lambda)x}}{\mu\Gamma(m)}.$$

Clearly k^* will be infinite if $m < 1$ or $\lambda \leq \mu$. For $m = 1$ the gamma is just an exponential. Thus we will assume $m > 1$ and choose $\mu < \lambda$. For $m \in (0,1)$ the rejection method can still be used, but a different envelope is required.

To find k^* we take the derivative of the right-hand side above and set it to zero, to find the point where the maximum occurs. You can check that this is at the point $x = (m-1)/(\lambda - \mu)$, which gives

by setting (f/h)'(x) = 0

$$k^* = \frac{\lambda^m (m-1)^{m-1} e^{-(m-1)}}{\mu(\lambda-\mu)^{m-1}\Gamma(m)}.$$ (is a function of mu)

To improve efficiency we would like to choose our envelope to make k^* as small as possible. Looking at the formula for k^* this means choosing μ to make $\mu(\lambda-\mu)^{m-1}$ as large as possible. Setting the derivative with respect to

μ to zero, we see that the maximum occurs when $\mu = \lambda/m$. Plugging this back in we get $k^* = m^m e^{-(m-1)}/\Gamma(m)$.

We can now code up our rejection algorithm.

```
# program spuRs/resources/scripts/gamma.sim.r

gamma.sim <- function(lambda, m) {
  # sim a gamma(lambda, m) rv using rejection with an exp envelope
  # assumes m > 1 and lambda > 0
  f <- function(x) lambda^m*x^(m-1)*exp(-lambda*x)/gamma(m)
  h <- function(x) lambda/m*exp(-lambda/m*x)
  k <- m^m*exp(1-m)/gamma(m)
  while (TRUE) {
    X <- -log(runif(1))*m/lambda
    Y <- runif(1, 0, k*h(X))
    if (Y < f(X)) return(X)
  }
}

set.seed(1999)
n <- 10000
g <- rep(0, n)
for (i in 1:n) g[i] <- gamma.sim(1, 2)
hist(g, breaks=20, freq=F, xlab="x", ylab="pdf f(x)",
  main="theoretical and simulated gamma(1, 2) density")
x <- seq(0, max(g), .1)
lines(x, dgamma(x, 2, 1))
```

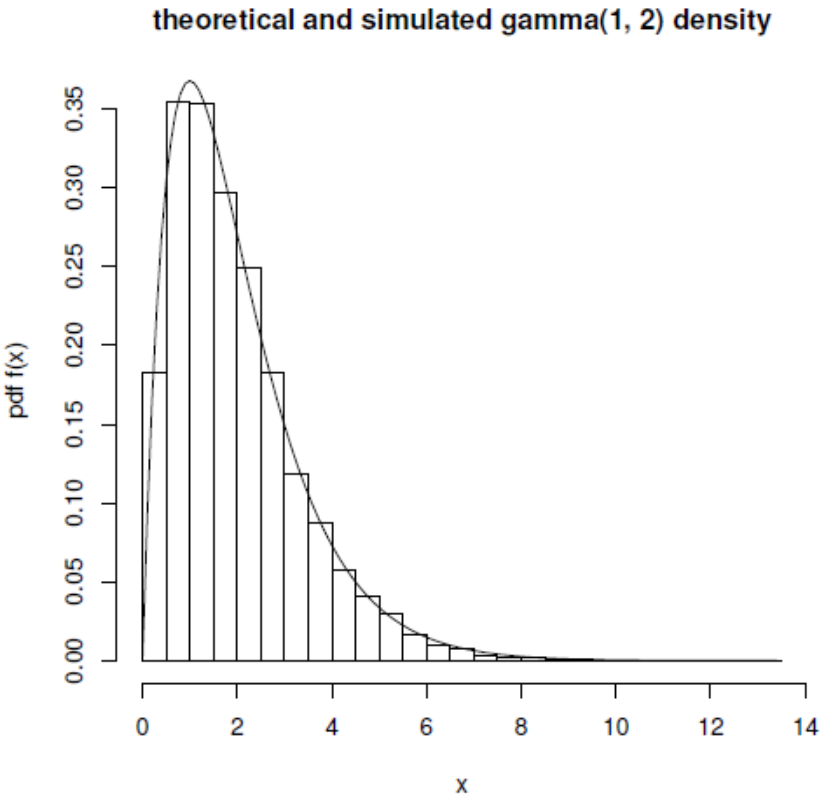


Figure 18.7 $\Gamma(1, 2)$ *density estimated from a sample generated using the rejection method.*