Chapter 2. Random Variables Generation

All of the random procedures in R, including the function **sample** are based on **runif**. We begin with a few examples.

- If $U \sim$ UNIF$(0, 1)$, then $2U + 5 \sim$ UNIF$(5, 7)$, so we can generate a vector of ten observations from the latter distribution with **2∗runif(10) + 5**. Alternatively, we can use **runif(10, 5, 7)**. The code **runif(10)** is short for **runif(10, 0, 1)**.
- To simulate a roll of two fair dice, we can use **(1:6, 2, rep=T)**. This is equivalent to **ceiling(runif(2, 0, 6))**. Also, to draw one card at random from a standard deck, we can use **ceiling(runif(1, 0, 52))**. Sampling multiple cards without replacement is a little more complicated because at each draw we must keep track of the cards drawn previously, but this record keeping is already programmed into the sample function. So **sample(1:52, 5)** simulates fair dealing of a five-card poker hand.
- Suppose we want to simulate the number X of heads in four tosses of a fair coin. That is, $X \sim$ BINOM$(4, 1/2)$. One method is to generate a vector of four uniform random variables with runif(4) and then compute **sum(runif(4) > 0.5)**, where the expression in parentheses is a logical vec- tor. Functions for simulating random samples from the binomial and several other commonly used probability distributions are available in R, and you can depend on these random functions being very efficiently programmed. So it is preferable to use **rbinom(1, 4, 0.5)**.

1 The Inverse Transform

Theorem 1 *Let U be a uniform (0,1) random variable. For any continuous distribution function F the random variable X defined by*

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

has distribution F.

Example 1 *Suppose we want to generate a random variable X having distribution function*

$$F(x) = x^n, 0 < x < 1.$$

If we let $x = F^{-1}(u)$, then

$$u = F(x) = x^n \text{ or, equivalently, } x = u^{1/n}.$$

Hence we can generate such a random variable X by generating a random number U and then setting $X = U^{1/n}$.

```
> Nsim=10^4
> U=runif(Nsim)
> X=U^(1/n)
> hist(X,freq=F,main="Example 2.1")
> mean(X)
[1] 0.7484784
```

Example 2 *(Page 32, example 2.4 Suess) Generate the Square of a Uniform Random Variable. Let $U \sim UNIF(0,1)$. We seek the distribution of $X = U^2$. Plot the result when 10000 values of U are simulated (left panel) and transformed to values $X = U^2$ (right) in one figure. Each interval in both histograms contains about 1000 simulated values.*

```
# set.seed(1234)
> m = 10000
> u = runif(m); x = u^2
> xx = seq(0, 1, by=.001)
> cut.u = (0:10)/10; cut.x = cut.u^2
> cut.u
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> par(mfrow=c(1,2))
> hist(u, breaks=cut.u, prob=T, ylim=c(0,10))
> lines(xx, dunif(xx), col="blue")
> hist(x, breaks=cut.x, prob=T, ylim=c(0,10))
> lines(xx, .5*xx^(-.5), col="blue")
> par(mfrow=c(1,1))
```

A random variable $X \sim \text{BETA}(\alpha, 1)$ has density function $f(x) = \alpha x^{\alpha-1}$, CDF $F(x) = x^\alpha$ and quantile function $x = F^{-1}(u) = u^{1/\alpha}$, where $0 < x, u < 1$. Thus, X can be simulated as $X = U^{1/\alpha}$. To get a vector of ten independent observations from such a distribution, we could use **qbeta(runif(10), alpha, 1)**, where the constant **alpha** is suitably defined. However, to sample from any beta distribution, it is best to use the R function **rbeta** because it is programmed to use effient methods for all choices of α and β.

Example 3 *Generate exponential random variables with mean 1 or λ.*

If X is an exponential random variable with rate 1, then its distribution function is given by

$$F(x) = 1 - e^{-x}.$$

If we let $x = F^{-1}(u)$, then

$$u = F(x) = 1 - e^{-x} \text{ or, equivalently, } x = -\log(1-u).$$

Note that $1 - u$ is also uniform on (0,1) and thus $-\log(1-U)$ has the same distribution as $-\log U$. In addition, note that if X is exponential with mean 1 then, for any positive

constant c, cX is exponential with mean c. Hence, an exponential random variable X with rate λ (mean $1/\lambda$) can be generated by generating a random number U and setting

$$X = -\frac{1}{\lambda} \log U.$$

Page 45, example 2.1 RC

```
> Nsim=10^4 #number of random variables
> U=runif(Nsim)
> X=-log(U) #transforms of uniforms
> Y=rexp(Nsim) #exponentials from R

> par(mfrow=c(1,2)) #plots
> hist(X,freq=F,main="Exp from Uniform")
> hist(Y,freq=F,main="Exp from R")
```

page 44, problem 2.11 Suess

```
> m = 10000;  lam = 1
> u = runif(m);  x = -log(u)/lam
> cut1 = seq(0, 1, by=.1)
> cut1
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> cut2 = -log(cut1)/lam;
> cut2
 [1]       Inf 2.3025851 1.6094379 1.2039728 0.9162907 0.6931472 0.5108256
 [8] 0.3566749 0.2231436 0.1053605 0.0000000
> cut2[1] = max(x);
  cut2 = sort(cut2)
> cut2
 [1] 0.0000000 0.1053605 0.2231436 0.3566749 0.5108256 0.6931472 0.9162907
 [8] 1.2039728 1.6094379 2.3025851 8.0188233
> hist(x, breaks=cut2, ylim=c(0,lam), prob=T, col="wheat")
>   xx = seq(0, max(x), by = .01)
> lines(xx, lam*exp(-lam*xx), col="blue")
> mean(x);  1/lam
[1] 0.9948941
[1] 1
> median(x); qexp(.5, lam)
[1] 0.6989748
[1] 0.6931472
> hist(u, breaks=cut1, plot=F)$counts
 [1]  975 1018 1018 1031  993 1002  977  962 1007 1017
> hist(x, breaks=cut2, plot=F)$counts
 [1] 1017 1007  962  977 1002  993 1031 1018 1018  975
```

Note that an alternate method to overlay the density curve, use **dexp(xx, lam)** instead of **lam∗exp(-lam∗xx)**.

Some commonly used distributions have CDFs that cannot be expressed in closed form. Examples are normal distributions and some members of the beta and gamma families. Even for these, R provides quantile functions that are accurate to many decimal places, but they require computation by approximate numerical methods. Thus, although one could get two independent standard normal random variables from **qnorm(runif(2))**, it is simpler and faster—and maybe more accurate—to use **rnorm(2)**.

Exercise 1

1. Follow example 2 to use the inverse transform method to generate a random variable having distribution function

$$F(x) = \frac{x^2 + x}{2}, 0 \le x \le 1.$$

2. Two distributions that have explicit forms of the cdf are the logistic and Cauchy distributions. Thus, they are well-suited to the inverse transform method. For each of the following, verify the form of the cdf and then generate 10,000 random variables using the inverse transform. Compare your program with the built-in R functions **rlogis** and **rcauchy**, respectively:

   (a) Logistic

$$\text{pdf: } f(x) = \frac{1}{\beta}\frac{e^{-(x-\mu)/\beta}}{[1+e^{-(x-\mu)/\beta}]^2}, \quad \text{cdf: } F(x) = \frac{1}{1+e^{-(x-\mu)/\beta}}.$$

   (b) Cauchy

$$\text{pdf: } f(x) = \frac{1}{\pi\sigma}\frac{1}{1+(\frac{x-\mu}{\sigma})^2}, \quad \text{cdf: } F(x) = \frac{1}{2} + \frac{1}{\pi}\tan^{-1}((x-\mu)/\sigma).$$

Example 4 *Generate the value of a gamma* (n, λ).

We use the result that a gamma (n, λ) random variable X can be regarded as being the sum of n independent exponentials, each with rate λ, to generate X. Specifically, we can generate a a gamma (n, λ) random variable by generating n random numbers $U_1, \ldots, U_n$ and then setting

$$X = -\frac{1}{\lambda}\log U_1 - \cdots - \frac{1}{\lambda}\log U_n = -\frac{1}{\lambda}\log(U_1 \cdots U_n).$$

ClassWork 1 *Write the codes*

The results of this example can be used to provide an efficient way of generating a set of exponential random variables by first generating their sum and then, conditional on the value of that sum, generating the individual values. For example, we could generate X and

Y, a pair of i.i.d. exponentials having mean 1, by first generating $X+Y$ and then using the result that, given that $X+Y=t$, the conditional distribution of X is uniform on $(0,t)$.

The following algorithm can thus be used to generate a pair of exponentials with mean 1.

Step 1: Generate random numbers U_1 and U_2.

Step 2: Set $t=-\log(U_1U_2)$.

Step 3: Generate a random number U_3.

Step 4: $X=tU_3$, $Y=t-X$.

ClassWork 2 *Write the codes*

Comparing the above with the more direct approach of generating two random numbers U_1 and U_2 and then setting $X=-\log U_1$, $Y=-\log U_2$ shows that the above algorithm saves a logarithmic computation at the cost of two multiplications and the generation of a random number.

ClassWork 3 *Write the codes*

We can also generate k independent exponentials with mean 1 by first generating their sum, say by $-\log(U_1\cdots U_k)$, and then generating $k-1$ additional random numbers $U_1^*,\ldots,U_{k-1}^*$, which should then be ordered. If $U_{(1)}^*<\ldots<U_{(k-1)}^*$ are their ordered values, and if $-\log(U_1\cdots U_k)=t$, then the k exponentials are

$$t[U_{(i)}^*-U_{(i-1)}^*],\ i=1,\ldots,k,\ \text{where } U_{(0)}^*\equiv 0, U_{(k)}^*=1.$$

ClassWork 4 *Write the codes*

2 General Transformation Methods

Example 5 *(Page 46, Example 2.2 RC) Generate some of the random variables starting from an exponential distribution.*

If the X_i's are iid $Exp(1)$ random variables, then

a. $2\sum_{j=1}^{\nu} X_j \sim \chi^2_{2\nu}$,

b. $\beta\sum_{j=1}^{a} X_j \sim Gamma(a,\beta)$,

c. $\sum_{j=1}^{a} X_j / \sum_{j=1}^{a+b} X_j \sim Beta(a,b)$.

Example 6 *Generate χ^2_6 random variables.*

```
U=runif(3*10^6)
U=matrix(data=U,nrow=3) #matrix for sums
X=-log(U) #uniform to exponential
X=2*apply(X,2,sum) #sum up to get chi squares
```

Now, we explore some important relationships among normal, expo- nential, and uniform random variables.

Example 7 *(Example 2.6 Suess Page 36) Target Practice with Normal Errors.*

- Suppose we aim an arrow at a target. The goal is to hit the bull's eye, located at the origin, but shots are random. We assume that errors in the horizontal and vertical directions are independent standard normal random variables, Z_1 and Z_2, respectively, so that any one shot hits at the point (Z_1, Z_2). Thus the squared distance from the origin is $T = R^2 = Z_1^2 + Z_2^2$. Clearly, $T \sim CHISQ(2) = EXP(1/2)$. This distribution has density function $f_T(t) = 0.5e^{-0.5t}$, for $t > 0$, and hence $E(T) = 2$.

```
# set.seed(1212)              # this seed for exact results shown
m = 40000; z1 = rnorm(m); z2 = rnorm(m)
t = z1^2 + z2^2; r = sqrt(t)
hist(t, breaks=30, prob=T, col="wheat")
tt = seq(0, max(t), length=100); dens = 0.5*exp(-0.5*tt)
lines(tt, dens, col="blue")
mean(t); mean(r); mean(r < 2)
> mean(t); mean(r); mean(r < 2)
[1] 2.005963
[1] 1.254416
[1] 0.86445
```

The first numerical result is consistent with the known value $E(T) = 2$. From the second numerical result, we see that on average the arrow misses the bull's eye by about 1.25 units (not $\sqrt{2}$). From the last result, we see that about 86% of the hits are within 2 units of the bull's eye. (Use **pchisq(4, 2)** for the exact value of $P(R < 2) = P(T < 4)$.)

- Also of interest is the angle θ of the hit as measured counterclockwise from the positive horizontal axis. If the hit is in the first quadrant, then $\theta = \arctan(Z_2/Z_1)$. Restricting attention to (Z_1, Z_2) in the first quadrant for simplicity, we can write a program to show that $\theta \sim \text{UNIF}(0, 90)$, measured in degrees.

```
quad1 = (z1 > 0) & (z2 > 0) # logical vector
z1.q1 = z1[quad1]; z2.q1 = z2[quad1]
th = (180/pi)*atan(z2.q1/z1.q1)
hist(th, breaks=9, prob=T, col="wheat")
aa = seq(0, 90, length = 10)
lines(aa, rep(1/90, 10), col="blue")
sum(quad1) # number of hits in quadrant 1
> sum(quad1)
[1] 10032
```

  Keeping track of signs and angles for all 40,000 hits, one gets results consistent with $\theta \sim UNIF(0, 360)$.

Exercise 2

1. Give a method for generating a random variable having density function

$$f(x) = e^x/(e-1), 0 \leq x \leq 1.$$

2. Give a method for generating a random variable having density function

$$f(x) = \begin{cases} \frac{x-2}{2} & \text{if } 2 \leq x \leq 3 \\ \frac{2-x/3}{2} & \text{if } 3 \leq x \leq 6 \end{cases}$$

Example 8 *(Page 47, Example 2.3 RC, Page 38, Example 2.6 Suess, Page 47 Problem 2.14 Suess) The Polar Method (Box-Muller Algorithm) for Generating Normal* $(0, 1)$ *Random Variables.*

Reversing this procedure, we can start with a pair of standard uniform random variates (U_1, U_2) to simulate the location of a random hit in polar coordinates and then transform to rectangular coordinates to obtain two standard normal random variates (X, Y) that express the location of the same hit.

Let X and Y be independent standard normal random variables and let R and θ denote the polar coordinates of the vector (X, Y). That is,

$$\begin{aligned} R^2 &= X^2 + Y^2 \\ \tan\theta &= \frac{Y}{X} \end{aligned}$$

Since X and Y are independent, their joint density is the product of their individual densities and is thus given by

$$f(x,y) = \frac{1}{2\pi} e^{-(x^2+y^2)/2}.$$

Making the change of variables

$$d = x^2 + y^2, \theta = \tan^{-1}(\frac{y}{x})$$

to have the joint density function of R^2 and Θ is given by

$$f(d,\theta) = \frac{1}{2}\frac{1}{2\pi} e^{-d/2},\ 0 < d < \infty, 0 < \theta < 2\pi.$$

It follows that R^2 and Θ are independent with R^2 being exponential with mean 2 and Θ being uniformly distributed over $(0, 2\pi)$. We can now generate a pair of independent standard normal random variables X and Y by first generating their polar coordinates and then transforming back to rectangular coordinates. This is accomplished as follows:

Step 1. Generate random numbers U_1 and U_2.

Step 2. Set $R^2 = -2\log U_1$ and $\Theta = 2\pi U_2$.

Step 3. Let

$$X = R\cos\Theta = \sqrt{-2\log U_1}\cos(2\pi U_2)$$

and

$$Y = R\sin\Theta = \sqrt{-2\log U_1}\sin(2\pi U_2).$$

This transformations are known as the Box-Muller transformations.

Problem 2.14 Suess page 48

```
# set.seed(1234)
m = 2*50000; z = numeric(m)
u1 = runif(m/2);  u2 = runif(m/2)
z1 = sqrt(-2*log(u1)) * cos(2*pi*u2)  # half of normal variates
z2 = sqrt(-2*log(u1)) * sin(2*pi*u2)  #    other half
z[seq(1, m, by = 2)] = z1             # interleave
z[seq(2, m, by = 2)] = z2             #    two halves

cut = c(min(z)-.5, seq(-2, 2, by=.5), max(z)+.5)
hist(z, breaks=cut, ylim=c(0,.4), prob=T)
  zz = seq(min(z), max(z), by=.01)
  lines(zz, dnorm(zz), col="blue")

E = m*diff(pnorm(c(-Inf, seq(-2, 2, by=.5), Inf))); E
N = hist(z, breaks=cut, plot=F)$counts; N
Q = sum(((N-E)^2)/E);  Q;  qchisq(c(.025,.975), 9)
```

However, this method is not efficient in computing the sine and cosine trigonometric functions. There is an indirect computation which generates U and then computes the sine and cosine of $2\pi U$. To begin, note that if U is uniform on $(0,1)$ then $2U$ is uniform on $(0,2)$ and so $2U-1$ is uniform on $(-1,1)$. This, if we generate random variables U_1 and U_2 and set

$$V_1 = 2U_1 - 1, \quad V_2 = 2U_2 - 1.$$

Then (V_1, V_2) is uniformly distributed in the square of area 4 centered at $(0,0)$. Suppose now that we continually generate such pairs (V_1, V_2) until we obtain one that is contained in the circle of radius 1 centered at $(0,0)$, i.e., until (V_1, V_2) is such that $V_1^2 + V_2^2 \leq 1$. It now follows that such a pair (V_1, V_2) is uniformly distributed in the circle. If we let R and Θ denote the polar coordinates of this pair, then it can be shown that R^2 and Θ are independent with R^2 being uniformly distributed over $(0,1)$ and Θ being uniformly distributed over $(0, 2\pi)$. It follows that

$$\begin{aligned} \sin\Theta &= \frac{V_2}{R} = \frac{V_2}{(V_1^2+V_2^2)^{1/2}}, \\ \cos\Theta &= \frac{V_1}{R} = \frac{V_1}{(V_1^2+V_2^2)^{1/2}}. \end{aligned}$$

It now follows from the Box-Muller transformation that we can generate independent standard normals by generating a random variable U and setting

$$\begin{aligned} X &= \sqrt{-2\log U}\frac{V_1}{(V_1^2+V_2^2)^{1/2}} \\ Y &= \sqrt{-2\log U}\frac{V_2}{(V_1^2+V_2^2)^{1/2}}. \end{aligned}$$

In fact, since $R^2 = V_1^2 + V_2^2$ is uniformly distributed over $(0,1)$ and is independent of the random angle Θ, we can use it as the random variable U in the Step 3 above. Therefore, letting $S = R^2$, we obtain that

$$\begin{aligned} X &= \sqrt{-2\log S}\frac{V_1}{\sqrt{S}} = V_1\sqrt{-2\log S/S} \\ Y &= \sqrt{-2\log S}\frac{V_2}{\sqrt{S}} = V_2\sqrt{-2\log S/S}. \end{aligned}$$

Thus, the algorithm

Step 1. Generate random numbers U_1 and U_2.

Step 2. Set $V_1 = 2U_1 - 1, V_2 = 2U_2 - 1$, and $S = V_1^2 + V_2^2$.

Step 3. If $S > 1$ return to Step 1.

Step 4. Return the independent standard normals

$$X = V_1\sqrt{-2\log S/S},\ Y = V_2\sqrt{-2\log S/S}.$$

ClassWork 5 *Write the codes*

Example 9 *An antiquated generator for the normal distribution is:*

- *Generate* $U_1, \ldots, U_{12} \sim Uniform[-1/2, 1/2]$
- *Set* $Z = \sum_{i=1}^{12} U_i$

the argument being that the CLT normality is sufficiently accurate with 12 terms.

a. *Show that* $E(Z) = 0$ *and* $Var(Z) = 1$ *(mathematical proof).*

b. *Generate 100 values of* Z *to show your simulated sample mean and sample variance are quite close to theoretical values.*

```
#exercise 2.3 R&C

nsim=10000
u1=runif(nsim)
u2=runif(nsim)
X1=sqrt(-2*log(u1))*cos(2*pi*u2)
X2=sqrt(-2*log(u1))*sin(2*pi*u2)

U=array(0,dim=c(nsim,1))
for(i in 1:nsim)
   U[i]=sum(runif(12,-.5,.5))

par(mfrow=c(1,2))
hist(X1)
hist(U)

a=3
mean(X1>a)
mean(U>a)

## Problem 2.15 Suess page 48
# set.seed(1234)
m = 100000;  n = 12
u = runif(m*n)
UNI = matrix(u, nrow=m)
z = rowSums(UNI) - 6

cut = c(min(z)-.5, seq(-2, 2, by=.5), max(z)+.5)
hist(z, breaks=cut, ylim=c(0,.4), prob=T)
  zz = seq(min(z), max(z), by=.01)
  lines(zz, dnorm(zz), col="blue")
```

```
E = m*diff(pnorm(c(-Inf, seq(-2, 2, by=.5), Inf))); E
N = hist(z, breaks=cut, plot=F)$counts; N
Q = sum(((N-E)^2)/E);  Q;  qchisq(c(.025,.975), 9)
```

3 Discrete Distributions

The quantile transformation method also works for simulating discrete distributions.

Page 35 Suess, Figure 2.6. Binomial CDF and quantile function.

```
> plot(x,cumsum(dbinom(x,5,0.6)),type="s",ylab="CDF")
> points(x,cumsum(dbinom(x,5,0.6)),pch=20)
> abline(0,0)
> abline(1,0)
> u=cumsum(dbinom(x,5,0.6))
> u
[1] 0.01024 0.08704 0.31744 0.66304 0.92224 1.00000
> plot(u,x,type="S",ylab="Quantile")
> points(u,x,pch=20)
> abline(v=0)
> abline(v=1)
> lines(y=c(3,3),x=c(u[3],u[4]),col="blue",lwd = 3)

> set.seed(1234); rbinom(10, 5, 0.4)
 [1] 1 2 2 2 3 2 0 1 2 2
> set.seed(1234); qbinom(runif(10), 5, 0.4)
 [1] 1 2 2 2 3 2 0 1 2 2
> set.seed(1234); rbinom(10, 5, 0.6)
 [1] 4 3 3 3 2 3 5 4 3 3
> set.seed(1234); 5 - qbinom(runif(10), 5, 0.4)
 [1] 4 3 3 3 2 3 5 4 3 3
> set.seed(1234); qbinom(runif(10), 5, 0.6)
 [1] 2 3 3 3 4 3 0 2 4 3

> qbinom(0.5, 5, 0.6)
[1] 3
> pbinom(3, 5, 0.6)
[1] 0.66304
> pbinom(3.5, 5, 0.6)
[1] 0.66304
> qbinom(0.66304, 5, 0.6)
[1] 3
```

To generate a discrete random variable $X \sim P_\theta$ with the probability mass function

$$P\{X = x_j\} = p_j, j = 0, 1, \ldots, \sum_j p_j = 1.$$

Approach:

- Generate a random number U;

- Set

$$X = \begin{cases} x_0 & \text{if } U < p_0 \\ x_1 & \text{if } p_0 \le U < p_0 + p_1 \\ \vdots & \\ x_j & \text{if } \sum_{j=0}^{j-1} p_i \le U < \sum_{j=0}^{j} p_i \\ \vdots & \end{cases}$$

Or, we can calculate— once for all, assuming we can store them— the probabilities

$$p_0^* = P_\theta(X \le x_0), \quad p_1^* = P_\theta(X \le x_1), \quad p_2^* = P_\theta(X \le x_2), \ldots,$$

and then generate $U \sim Uniform(0,1)$ and take

$$X = x_k, \text{ if } p_{k-1}^* \le U < p_k^*.$$

A more efficient procedure is the following:

Step 1. Generate a random number U;

Step 2. If $U < p_0$ set $X = x_0$ and stop;
If $U < p_0 + p_1$ set $X = x_1$ and stop;
If $U < p_0 + p_1 + p_2$ set $X = x_2$ and stop;
$\vdots$

Example 10 *Simulate a random variable X such that $p_1 = 0.20, p_2 = 0.15, p_3 = 0.25, p_4 = 0.40$ where $p_j = P(X = j)$.*

ClassWork 6 *Write the codes*

Example 11 *Generating the discrete uniform random variables.*

Suppose we want to generate the value of X which is equally likely to take on any of the values $1, \ldots, n$. That is, $P(X = j) = 1/n, j = 1, \ldots, n$. We can use the preceding results as the followings:

- generate a random number U;
- Set $X = j$ if $(j-1)/n \le U < j/n$. That is, $X = Int(nU) + 1$.

ClassWork 7 *Write the codes*

Example 12 *To generate Binomial (n, p).*

Generate the value of a binomial(n, p) random variable X–

$$P(X = i) = \frac{n!}{i!(n-i)!} p^i (1-p)^{n-i}, i = 0, 1, \ldots, n.$$

To do so, we employ the inverse transform method by making use of the recursive identity

$$P(X = i+1) = \frac{n-i}{i+1} \frac{p}{1-p} P(X = i).$$

Algorithm:

1. Generate U.
2. $c = p/(1-p), i = 0, pr = (1-p)^n, F = pr$.
3. If $U < F$, set $X = i$ and stop.
4. $pr = [c(n-i)/(i+1)]pr, F = F + pr, i = i + 1$.
5. Go to step 3.

To generate $X \sim Binomail(10, 0.3)$ in R: **pbinom** $(k, 10, 0.3)$ gives probabilities

$$p_0 = 0.028, p_1 = 0.149, p_2 = 0.382, \ldots, p_{10} = 1.$$

ClassWork 8 *Write the codes*

Example 13 *To generate Poisson* (λ)*.*

The random variable X is Poisson with mean λ if

$$p_i = P(X = i) = e^{-\lambda}\frac{\lambda^i}{i!}, i = 0, 1, \ldots.$$

The key to using the inverse transform method to generate such a random variable is

$$p_{i+1} = \frac{\lambda}{i+1}p_i, i \geq 0.$$

Algorithm:

1. Generate U.
2. $i = 0, p = e^{-\lambda}, F = p$.
3. If $U < F$, set $X = i$ and stop.
4. $p = \lambda p/(i+1), F = F + p, i = i + 1$.
5. Go to step 3.

ClassWork 9 *Write the codes*

Example 14 *To generate Poisson random variables for large values of* λ *in R. [Example 2.5, page 49 RC]*

```
> Nsim=10^4; lambda=100
> spread=3*sqrt(lambda)
> t=round(seq(max(0,lambda-spread),lambda+spread,1))
> prob=ppois(t, lambda)
> X=rep(0,Nsim)
> for (i in 1:Nsim){
+ u=runif(1)
+ X[i]=t[1]+sum(prob<u) }
```

Example 15 *To generate Geometric* (p).

Recall that X is said to be a geometric random variable with parameter p if

$$P(X = i) = pq^{i-1}, i \geq 1, \text{ where } q = 1 - p.$$

Since

$$\begin{aligned} \sum_{i=1}^{j-1} P(X = i) &= 1 - P(X > j - 1) \\ &= 1 - P(\text{ first } j - 1 \text{ trials are all failures}) \\ &= 1 - q^{j-1}, j \geq 1. \end{aligned}$$

We can generate the value of X by generating a random number U and setting X equal to that value j for which

$$1 - q^{j-1} \leq U < 1 - q^j \quad \text{or} \quad q^j < 1 - U \leq q^{j-1}.$$

That is, we can define X by

$$X = \min\{j : q^j < 1 - U\}.$$

Hence,

$$\begin{aligned} X &= \min\{j : j > \log(1 - U)/\log(q)\} \\ &= Int(\log(1 - U)/\log(q)) + 1 \\ &= Int(\log(U)/\log(q)) + 1. \end{aligned}$$

ClassWork 10 *Write the codes*

Example 16 *To generate Negative Binomial* (r, p).

Generate the value of a negative binomial(r, p) random variable X–

$$P(X = j) = \frac{(j-1)!}{(j-r)!(r-1)!} p^r (1-p)^{j-r}, j = r, r+1, \ldots.$$

To do so, we employ the inverse transform method by making use of the recursive identity

$$P(X = j + 1) = \frac{j(1-p)}{j+1-r} P(X = j).$$

We may also use the relationship between negative binomial and geometric random variables ($X = \sum_{i=1}^{r} X_i$, where X_i's are i.i.d Geometric (p)) and the results of Example 5 to obtain an algorithm for simulating from this distribution.

ClassWork 11 *Write the codes*

Exercise 4

1. Give an efficient algorithm to simulate the value of a random variable X such that

$$P(X = 1) = 0.3,\ P(X = 2) = 0.2,\ P(X = 3) = 0.35,\ P(X = 4) = 0.15.$$

   Also, Find these probabilities by simulation.

2. A deck of 100 cards–numbered $1, 2, \ldots, 100$–is shuffled and then turned over one card at at time. Say that a "hit" occurs whenever card i is the ith card to be turned over, $i = 1, \ldots, 100$. Write a simulation program to estimate the expectation and variance of the total number of hits.

3. Using 100 random numbers explain how you could find an approximation for $\sum_{i=1}^{N} e^i/N$, where $N = 10,000$.

4. A pair of fair dice are to be continuously rolled until all the possible outcomes $2, 3, \ldots, 12$ have occurred at least once. Develop a simulation study to estimate the expected number of dice rolls that are needed.

5. **Exercise 2.11, page 57 (Casella).** In both questions, the comparison between generators is understood in terms of efficiency via the **system.time** function.

   (a) Generate a binomial $Bin(n, p)$ random variable with $n = 25$ and $p = 0.2$. Plot a histogram for a simulated sample and compare it with the binomial mass function. Compare your generator with the R binomial generator.

   (b) For $\alpha \in 2[0, 1]$, show that the R code

```
> u=runif(1)
> while(u > alpha) u=runif(1)
> U=u
```

   produces a random variable U from $Unif([0, \alpha])$. Compare it with the transform αU, $U \sim Unif(0, 1)$, for values of α close to 0 and close to 1, and with **runif(1,max=alpha)**. Use **hist(Wait(1000,.5))** and **hist(Trans(1000,.5))** to see the corresponding histograms. Vary n and α.

6. **Exercise 2.15, page 58 (Casella).** The Poisson distribution POIS(λ) is connected to the exponential distribution through the Poisson process in that it can be simulated by generating exponential random variables until their sum exceeds 1. That is, if $X_i \sim$ Exp(λ) and if K is the first value for which $\sum_{i=1}^{K+1} X_i > 1$, then $K \sim$ POIS(λ). Compare this algorithm with **rpois** and the algorithm of Example 2.5 (Casella) [Example 12] for both small and large values of λ.

4 The Composition Approach or Mixture Representations

Suppose that we had an efficient method to simulate the value of a random variable having either of the two probability mass functions $\{p_j^{(1)}, j \geq 0\}$ or $\{p_j^{(2)}, j \geq 0\}$, and that we wanted to simulate the value of the random variable X having mass function

$$P(X = j) = \alpha p_j^{(1)} + (1 - \alpha) p_j^{(2)}, j \geq 0, \tag{1}$$

where $0 < \alpha < 1$. One way to simulate such a random variable X is to note that if X_1 and X_2 are random variables having respective mass functions $\{p_j^{(1)}, j \geq 0\}$ and $\{p_j^{(2)}, j \geq 0\}$, then random variable X defined by

$$X = \begin{cases} X_1 & \text{with probability } \alpha \\ X_2 & \text{with probability } 1 - \alpha \end{cases}$$

will have its mass function given by (1). From this it follows that we can generate the value of such a random variable by first generating a U and then generating a value of X_1 if $U < \alpha$ and of X_2 if $U > \alpha$.

Example 17 *Suppose we want to generate the value of a random variable X such that*

$$p_j = P(X = j) = \begin{cases} 0.05 & \textit{for } j = 1, 2, 3, 4, 5, \\ 0.15 & \textit{for } j = 6, 7, 8, 9, 10. \end{cases}$$

By noting that $p_j = 0.5p_j^{(1)} + 0.5p_j^{(2)}$, where

$$p_j^{(1)} = 0.1, j = 1, \ldots, 10, \text{ and } p_j^{(2)} = \begin{cases} 0 & \text{for } j = 1, 2, 3, 4, 5 \\ 0.2 & \text{for } j = 6, 7, 8, 9, 10 \end{cases}$$

We can simulate X as follows:

Step 1: Generate U_1.

Step 2: Generate U_2.

Step 3: If $U_1 < 0.5$, set $X = Int(10U_2) + 1$. Otherwise, set $X = Int(5U_2) + 6$.

ClassWork 12 *Write the codes*

If $F_i, i = 1, \ldots, n$ are distribution functions and $\alpha_i, i = 1, \ldots, n$, are nonnegative numbers summing to 1, then the distribution function F given by

$$F(x) = \sum_{i=1}^{m} \alpha_i F_i(x)$$

is said to be a **mixture** or a **composition**, of the distribution function $F_i, i = 1, \ldots, n$. One way to simulate from F is first to simulate a random variable I, equal to i with probability

$\alpha_i, i = 1, \ldots, n$, and then to simulate from the distribution F_I. This approach to simulating from F is often referred to as the **composition method**.

Note that we can also write the pdf of the above **mixture distribution** in the form

$$f(x) = \int_{\mathcal{Y}} g(x|y)p(y)dy \text{ or } f(x) = \sum_{i \in \mathcal{Y}} p_i f_i(x),$$

where g and p are standard distributions that can be easily simulated. To generate a random variable X using such a representation, we can first generate a variable Y from the **mixing** distribution and then generate X from the selected conditional distribution. That is,

- if $y \sim p(y)$ and $X \sim f(x|y)$, then $X \sim f(x)$ (if continuous);
- if $\gamma \sim P(\gamma = i) = p_i$ and $X \sim f_\gamma(x)$, then $X \sim f(x)$ (if discrete).

Homework 1 *Show that the Student's t density with ν degrees of freedom $\mathcal{T}_\nu$ can be expressed in terms of a mixture, where*

$$X|y \sim Normal(0, \nu/y) \text{ and } Y \sim \chi^2_\nu.$$

Homework 2 *If $X \sim Neg(n, p)$, then X has the mixture representation*

$$X|y \sim Poisson(y) \text{ and } Y \sim Gamma(n, \beta),$$

where $\beta = (1-p)/p$. [Example 2.6 and Figure 2.3, pages 50–51 (Casella)]

The following R code generates from this mixture

```
> Nsim=10^4
> n=6;p=.3
> y=rgamma(Nsim,n,rate=p/(1-p))
> x=rpois(Nsim,y)
> hist(x,main="",freq=F,col="grey",breaks=40)
> lines(1:50,dnbinom(1:50,n,p),lwd=2,col="sienna")
```

See the proofs for the above two examples in the notes.

Exercise 5

1. Give a method for simulating from the probability mass function p_j, $j = 5, 6, \ldots, 14$, where
$$p_j = \begin{cases} 0.11 & \text{when } j \text{ is odd and } 5 \leq j \leq 13, \\ 0.09 & \text{when } j \text{ is even and } 6 \leq j \leq 14. \end{cases}$$
[Exercise 14, page 60]

2. Suppose that the random variable X can take on any of the values $1, \ldots, 10$ with respective probabilities $0.06, 0.06, 0.06, 0.06, 0.06, 0.15, 0.13, 0.14, 0.15, 0.13$. Use the composition approach to give an algorithm that generates the value of X. [Exercise 15, page 60]

3. Present a method to generate the value of X, where

$$P(X = j) = \left(\frac{1}{2}\right)^{j+1} + \frac{(\frac{1}{2})2^{j-1}}{3^j}, j = 1, 2, \ldots.$$

[Exercise 16, page 60]

4. A random selection of m balls is to be made from an urn that contains n balls, n_i of which have color type i, $\sum_{i=1}^{r} n_i = n$. Discuss efficient procedures for simulating $X_1, \ldots, X_r$, where X_i denotes the number of withdrawn balls that have color type i. [Exercise 19, page 61]

5 The Acceptance–Rejection Methods

There are many distributions for which the inverse transform method and even general transformations fail to be able to generate the required random variables. For these cases, we must turn to **indirect** methods; that is, methods in which we generate a candidate random variable and only accept it subject to passing a test.

These methods often are called the **rejection method** or the **acceptance–rejection method**, which only requires us to know the functional form of the density f of interest (called the **target density**) up to a multiplicative constant. We use a simpler (to simulate) density g, called the **instrumental** or **candidate density**, to generate the random variable for which the simulation is actually done. The only constraints we impose on this candidate density g are that

(i). f and g have compatible supports (i.e., $g(x) > 0$ when $f(x) > 0$).

(ii). There is a constant M with $f(x)/g(x) \leq M$ for all x.

In this case, X can be simulated as follows. First, we generate $Y \sim g$ and, independently, we genereate $U \sim Uniform(0,1)$. If $U \leq \dfrac{1}{M}\dfrac{f(Y)}{g(Y)}$, then we set $X = Y$. If the inequality is not satisfied, we then discard Y and U and start again.

Algorithm 1 (Accept-Reject Method)

1. Generate $Y \sim g$, $U \sim Uniform(0,1)$;
2. Accept $X = Y$ if $U \leq f(Y)/(Mg(Y))$;
3. Return to Step 1 otherwise.

The R implementation of this algorithm is straightforward: If **randg** is a function that delivers generations from the density g, in the same spirit as **rnorm** or **rt**, a simple R version of Algorithm 1 is

```
> u=runif(1)*M
> y=randg(1)
> while (u>f(y)/g(y)){
+     u=runif(1)*M
+     y=randg(1)}
```

which produces a single generation y from f.

Why does this method work? We find

$$\begin{aligned} P(Y \leq x | U \leq f(Y)/[Mg(Y)]) &= \frac{P(Y \leq x, U \leq f(Y)/[Mg(Y)])}{P(U \leq f(Y)/[Mg(Y)])} \\ &= \frac{\int_{-\infty}^{x} \int_{0}^{f(y)/[Mg(y)]} dug(y)dy}{\int_{-\infty}^{\infty} \int_{0}^{f(y)/[Mg(y)]} dug(y)dy} \\ &= \frac{\int_{-\infty}^{x} f(y)/[Mg(y)]g(y)dy}{\int_{-\infty}^{\infty} f(y)/[Mg(y)]g(y)dy} \\ &= \frac{\int_{-\infty}^{x} f(y)dy}{\int_{-\infty}^{\infty} f(y)dy} = P(X \leq x). \end{aligned}$$

Theorem 2 (Continous Case)

(i) The random variable generated by the rejection method has density f.

(ii) The number of iterations of the algorithm that are needed is a geometric random variable mean M.

Equivalently, we can use this as the basis for simulating a random variable X from the distribution having mass function $\{p_j, j \geq 0\}$ by first simulating a random variable Y having mass function $\{q_j\}$ and then accepting this simulating value with a probability proportional to p_Y/q_Y.

Theorem 3 (Discrete Case) *The acceptance-rejection algorithm generating a random variable X such that $P(X = j) = p_j$, $j = 0, \ldots,$. In addition, the number of iterations of the algorithm needed to obtain X is a geometric random variable with mean M.*

Proof: we determine the probability that a single iteration produces the accepted value j. First note that

$$P(Y = j, \text{it is accepted}) = P(Y = j)P(\text{accept}|Y = j) = q_j \frac{p_j}{Mq_j} = p_j/M.$$

Summing over j yields the probability that a generated random variable is accepted:

$$P(\text{accepted}) = \sum_j p_j/M = 1/M.$$

As each iteration independently results in an accepted value with probability $1/M$, we see that the number of iterations needed is Geometric with mean M. Also,

$$P(X = j) = \sum_n P(j \text{ accepted on iteration } n) = \sum_n \left(1 - \frac{1}{M}\right)^{n-1} \frac{p_j}{M} = p_j.$$

Remark It should note that the way in which we "accept the value Y with probability $\frac{p_Y}{Mq_Y}$" is by generating a random number U and then accepting Y if $U \leq \frac{p_Y}{Mq_Y}$.

Example 18 *Suppose we wanted to simulate the value of a random variable X that takes one of the values $1, 2, \ldots, 10$ with respective probabilities $0.11, 0.12, 0.09, 0.08, 0.12, 0.10, 0.09, 0.09, 0.10, 0.10$.*

One possibility is to use the inverse transform algorithm, another approach is to use the rejection method with q being the discrete uniform density on $1, \ldots, 10$. That is, $q_j = 1/10, j = 1, \ldots, 10$. For this choice of $\{q_j\}$ we can choose M by

$$M = \max \frac{p_j}{q_j} = 1.2$$

and so the algorithm would be as follows:

Step 1: Generate U_1 and set $Y = Int(10U_1) + 1$.

Step 2: Generate U_2.

Step 3: If $U_2 < p_Y/.12$, set $X = Y$ and stop. Otherwise, return to Step 1.

ClassWork 13 *Write the codes*

Exercise 6

1. Give two methods for generating a random variable X such that

$$P(X = i) = \frac{e^{-\lambda}\lambda^i/i!}{\sum_{j=0}^{k} e^{-\lambda}\lambda^j/j!}, \; i = 0, \ldots, k.$$

   Also, generate a sample of size 20 from this probability mass function and give a frequency of this sample and compute the corresponding sample mean and sample variance. [Exercise 12, page 59]

2. Let X be a binomial random variable with parameters n and p. Suppose that we want to generate a random variable Y whose probability mass function is the same as the conditional mass function of X given that $X \geq k$, for some $k \leq n$. Let $\alpha = P(X \geq k)$. [Exercise 13, page 59]

   (a) Compute the values of α.

   (b) Compute the conditional mass function of X given that $X \geq k$, for some $k \leq n$.

   (c) Give the inverse transform method for generating Y.

   (d) Give a second method for generating Y.

Example 19 *Use the rejection method to generate a random variable having density function*

$$f(x) = 20x(1-x)^3, 0 < x < 1.$$

Since this random variable (which is beta with parameters 2 and 4) is concentrated in the interval (0, 1), we can consider the rejection method with $g(x) = 1, 0 < x < 1$. To determine the smallest constant M such that $f(x)/g(x) \leq M$, we use calculus to determine the maximum value of

$$\frac{f(x)}{g(x)} = 20x(1-x)^3,$$

which is attained when $x = 1/4$ and thus $M = 135/64$, and the rejection procedure is as follows:

Step 1: Generate U_1 and U_2.

Step 2: If $U_2 \leq \frac{256}{27}U_1(1-U_1)^3$, stop and set $X = U_1$. Otherwise, return to Step 1.

The average number of times that Step 1 will be performed is $M = 135/64 \approx 2.11$.

ClassWork 14 *Write the codes*

Example 20 (General Cases) *Construct an algorithm to simulate Beta(α, β) random variables based on the Accept-Reject method, using as the instrumental distribution the uniform $(0, 1)$ when both α and β are larger than 1. (The generic* **rbeta** *function does not impose this restriction.) [Examples 2.7 and 2.8, pages 53–55 (Casella)]*

The upper bound M is then the maximum of the beta density, obtained for instance by **optimize** (or its alias **optimise**):

```
> optimize(f=function(x){dbeta(x,2.7,6.3)},
+ interval=c(0,1),max=T)$objective
[1] 2.669744
```

Since the candidate density g is equal to one, the proposed value Y is accepted if $M \times U < f(Y)$, that is, if $M \times U$ is under the beta density f at that realization. Note that generating $U \sim U[0,1]$ and multiplying by M is equivalent to generating $U \sim U[0,M]$. For $\alpha = 2.7$ and $\beta = 6.3$, an alternative R implementation of the Accept-Reject algorithm is

```
> Nsim=2500
> a=2.7;b=6.3
> M=2.67
> u=runif(Nsim,max=M) #uniform over (0,M)
> y=runif(Nsim) #generation from g
> x=y[u<dbeta(y,a,b)] #accepted subsample
```

and the left panel in Figure 2.4 of Casella shows the results of generating 2500 pairs (Y,U) from $U[0,1] \times U[0,M]$. The black dots $(Y, Ug(Y))$ that fall under the density f are those for which we accept $X = Y$, and we reject the grey dots $(Y, Ug(Y))$ that fall outside. It is again clear from this graphical representation that the black dots are uniformly distributed over the area under the density f. Since the probability of acceptance of a given simulation is $1/M$ (Exercise 2.5), with $M = 2.67$ we accept approximately $1/2.67 = 37\%$ of the values.

Consider instead simulating $Y \sim Beta(2,6)$ as a proposal distribution. This choice of g is acceptable since

```
> optimize(f=function(x){dbeta(x,2.7,6.3)/dbeta(x,2,6)},
+ max=T,interval=c(0,1))$objective
1.671808

> x=NULL
> while (length(x)<Nsim){
          y=runif(Nsim*M)
          x=c(x,y[runif(Nsim*M)*M<dbeta(y,a,b)])
  }
> x=x[1:Nsim]
```

This modification of the proposal thus leads to a smaller value of M and a correspondingly higher acceptance rate of 58% than with the uniform proposal. The right panel of Figure 2.4 of Casella shows the outcome of the corresponding Accept-Reject algorithm and illustrates the gain in effciency brought by simulating points in a smaller set.

Example 21 *Use the rejection method to generate a random variable having the gamma* $(3/2, 1)$ *density*

$$f(x) = Kx^{1/2}e^{-x}, x > 0,$$

where $K = \frac{1}{\Gamma(3/2)} = \frac{2}{\sqrt{\pi}}$.

Because this random variable is concentrated on the positive axis and has a mean 3/2, it is natural to try the rejection method with an exponential random variable with the same mean,

$$g(x) = \frac{2}{3}e^{-2x/3}, x > 0.$$

The same way as Example 12 to determine the maximum value of

$$\frac{f(x)}{g(x)} = \frac{3K}{2}x^{1/2}e^{-x/3},$$

which is attained when $x = 3/2$ and thus $M = \dfrac{3^{3/2}}{(2\pi e)^{1/2}}$. Thus, a gamma $(3/2, 1)$ can be generated by the rejection procedure as follows:

Step 1: Generate U_1, U_2, and set $Y = -\frac{3}{2}\log U_1$.

Step 2: If $U_2 \leq (\frac{2eY}{3})^{1/2}e^{-Y/3}$, stop and set $X = Y$. Otherwise, return to Step 1.

The average number of iterations will be needed is $M = \dfrac{3^{3/2}}{(2\pi e)^{1/2}} \approx 1.257$.

ClassWork 15 *Write the codes*

Exercise 7

1. Let X be an exponential random variable with mean 1. Give an efficient algorithm for simulating a random variable whose distribution is the conditional distribution of X given that $X < 0.05$. That is, its density function is

   $$f(x) = \frac{e^{-x}}{1 - e^{-0.05}},\ 0 < x < 0.05.$$

   Generate 1000 such variable and use them to estimate of $E(X|X < 0.05)$. Then determine the exact value of $E(X|X < 0.05)$. [Exercise 6, page 82]

2. Give algorithms for generating random variables from the following distributions. [Exercise 8, page 82]

   (a)

   $$F(x) = \frac{x + x^3 + x^5}{3},\ 0 \leq x \leq 1.$$

   (b)

   $$F(x) = \begin{cases} \frac{1-e^{-2x}+2x}{3}, & 0 < x < 1, \\ \frac{3-e^{-2x}}{3}, & 1 < x < \infty. \end{cases}$$

3. Exercises 2.5–2.8, pages 55–56 (Casella).

Example 22 *Generating a Normal Random Variable.*

To generate a standard normal random variable Z, we first note that the absolute value of Z has probability density function

$$f(x) = \frac{2}{\sqrt{2\pi}} e^{-x^2/2}, x > 0. \tag{2}$$

We then start by generating from the preceding density function by using the rejection method with g being the exponential density function with mean 1,

$$g(x) = e^{-x}, x > 0.$$

Now

$$\frac{f(x)}{g(x)} = \sqrt{\frac{2}{\pi}} e^{x - x^2/2},$$

and so the maximum value is attained when $x = 1$ and thus $M = \sqrt{\frac{2e}{\pi}}$. Because

$$\frac{f(x)}{Mg(x)} = \exp(-\frac{(x-1)^2}{2}),$$

it follows that we can generate the absolute value of a standard normal random variable as follows:

Step 1: Generate U_1, U_2, and set $Y = -\log U_1$.

Step 2: If $U_2 \leq \exp(-\frac{(Y-1)^2}{2})$, stop and set $X = Y$. Otherwise, return to Step 1.

Once we have simulated a random variable X having density function as in Equation (2), we can then obtain a standard normal Z by letting Z be equally likely to be either X or $-X$.

In Step 2, the value Y is accepted if $U \leq \exp\{-(Y-1)^2/2\}$, which is equivalent to $-\log U \geq (Y-1)^2/2$. However, $-\log U$ is exponential with rate 1, and so the procedure can be changed as:

Step 1: Generate U_1, U_2, and set $Y_1 = -\log U_1$ and $Y_2 = -\log U_2$.

Step 2: If $Y_2 \geq (Y_1 - 1)^2/2$, stop and set $X = Y_1$. Otherwise, return to Step 1.

By memoryless property, $Y_2 - (Y_1 - 1)^2/2$ is exponential with rate 1 (independent of X). Therefore, the following algorithm will generate an exponential with rate 1 and an independent standard normal random variable:

Step 1: Generate U_1, U_2, and set $Y_1 = -\log U_1$ and $Y_2 = -\log U_2$.

Step 2: If $Y_2 \geq (Y_1-1)^2/2$, stop and set $Y = Y_2-(Y_1-1)^2/2$ and go to step 3. Otherwise, return to Step 1.

Step 3: Generate U and set

$$Z = \begin{cases} Y_1 & \text{if } U \leq 1/2 \\ -Y_1 & \text{if } U > 1/2 \end{cases}$$

Then $Z \sim N(0,1)$ and $Y \sim Exp(1)$ are independent.

ClassWork 16 *Write the codes*

Some **key properties** of the Accept-Reject algorithm, which should always be considered when using it, are the following:

1. Only the ratio f/g is needed, so the algorithm does not depend on the normalizing constant.
2. The bound $f \leq Mg$ need not be tight; the algorithm remains valid (if less efficient) when M is replaced with any larger constant.
3. The probability of acceptance is $1/M$, so M should be as small as possible for a given computational effort.

One criticism of the Accept-Reject algorithm is that it generates "useless" simulations from the proposed g when rejecting, even those necessary to validate the output as being generated from the target f; the method of importance sampling (Section 3.3) can be used to bypass this problem.

Homework 3 *Given a sampling density $f(x|\theta)$ and a prior density $\pi(\theta)$, if we observe $\boldsymbol{x} = (x_1, \ldots, x_n)$, the posterior distribution of θ is*

$$\pi(\theta|\boldsymbol{x}) = \pi(\theta|x_1, \ldots, x_n) \propto \prod_i f(x_i|\theta)\pi(\theta),$$

where $\prod_i f(x_i|\theta) = L(\theta|x_1, \ldots, x_n)$ is the likelihood function. For estimating a normal mean, a robust prior is the Cauchy. For $X_i \sim N(\theta, 1)$, $\theta \sim Cauchy(0, 1)$, the posterior distribution is

$$\pi(\theta|\boldsymbol{x}) \propto \frac{1}{\pi}\frac{1}{1+\theta^2}\frac{1}{2\pi}\prod_{i=1}^{n} e^{-(x_i-\theta)^2/2}.$$

Set $\theta_0 = 3$, $n = 10$, and generate $X_1, \ldots, X_n \sim N(\theta_0, 1)$. Use the Accept-Reject algorithm with a Cauchy(0, 1) candidate to generate a sample from the posterior distribution. Evaluate how well the value θ_0 is recovered. How much better do things get if n is increased?

Solution: Since the ratio $\pi(\theta|\boldsymbol{x}) = \pi(\theta)$ is the likelihood, it is obvious that the optimal bound M is the likelihood function evaluated at the MLE (assuming π is a true density and not an improper prior). Simulating from the posterior can then be done via

```
theta_0=3;n=100;N=10^4
x=rnorm(n)+theta0 lik=function(the){prod(dnorm(x,mean=the))}
M=optimise(f=function(the){prod(dnorm(x,mean=the))},
int=range(x),max=T)$obj theta=rcauchy(N)
res=(M*runif(N)>apply(as.matrix(theta),1,lik));print(sum(res)/N)
while (sum(res)>0){le=sum(res);theta[res]=rcauchy(le)
res[res]=(M*runif(le)>apply(as.matrix(theta[res]),1,lik))}
```

The rejection rate is given by 0.9785, which means that the Cauchy proposal is quite inefficient. An empirical confidence (or credible) interval at the level 95% on θ is (2.73, 3.799). Repeating the experiment with $n = 100$ leads (after a while) to the interval (2.994, 3.321), there is therefore an improvement.

Generating Random Vectors

A random vector $X_1, \ldots, X_n$ can be simulated nu sequentially generating the X_i. That is, first generate X_1; then generate X_2 from its conditional distribution given the generated value of X_1; then generate X_3 from its conditional distribution given the generated value of X_1 and X_2; and so on.

Example 23 *Simulate a random vector having a multinomial distribution.*

Consider n independent trials, each of which results in one of the outcomes $1, 2, \ldots, r$ with respective probabilities $p_1, p_2, \ldots, p_r$, $\sum_{i=1}^{r} p_i = 1$. If X_i denote the number of trials that result in outcome i, then the random vector $(X_1, \ldots, X_r)$ is said to be a multinomial random vector. Its joint probability mass function is given by

$$P(X_i = x_i, i = 1, \ldots, r) = \frac{n!}{x_1! \cdots x_r!} p_1^{x_1} \cdots p_r^{x_r}, \sum_{i=1}^{r} x_i = n.$$

- If r is large relative to n, so that many of the outcomes do not occur on any of the trials, then it is probably best to simulate the random variables by generating the outcomes of the n trials. That is, first generate independent random variables $Y_1, \ldots, Y_n$ such that
$$P(Y_j = i) = p_i, i = 1, \ldots, r, j = 1, \ldots, n,$$
and then set $X_i =$ number of j, $j = 1, \ldots, n : Y_j = i$. (That is, the generated value of Y_j represents the result of trial j , and X_i is the number of trials that result in outcome i.)

- If n is large relative to r, then $X_1, \ldots, X_r$ can be simulated in sequence. That is, first generate X_1, then X_2, then X_3, and so on. So X_1 is binomial (n, p_1). We can generate X_1 with the method in Section 3.3. Suppose its generated value is x_1. Then, given that x_1 of the n trials resulted in outcome 1, it follows that each of the other $n - x_1$ trials independently results in outcome with probability
$$P(2| \text{ not } 1) = \frac{p_2}{1 - p_1}.$$

Therefore, the conditional distribution of X_2, given that $X_1 = x_1$ is binomial $(n - x_1, p_2/(1 - p_1))$. Then, we can again make use of Section 3.3 to generate X_2. Consequently, the conditional distribution of X_3 given that $X_i = x_i, i = 1, 2$ is binomial $(n - x_1 - x_2, p_3/(1 - p_1 - p_2))$. We then use this fact to generate X_3, and continue on until all the values $X_1, \ldots, X_r$ have been generated.

ClassWork 17 *Write the codes*