1 Sampling from discrete distributions

A discrete random variable X is a random variable that has a probability mass function $p(x) = P(X = x)$ for any $x \in \mathcal{S}$, where $\mathcal{S} = \{x_1, x_2, ..., x_k\}$ denotes the sample space, and k is the (possibly infinite) number of possible outcomes for the discrete variable X, and suppose $\mathcal{S}$ is ordered from smaller to larger values. Then the CDF, F for X is

$$F(x_j) = \sum_{i \leq j} p(x_i)$$

Discrete random variables can be generated by slicing up the interval $(0, 1)$ into subintervals which define a *partition* of $(0, 1)$:

$$(0, F(x_1)), (F(x_1), F(x_2)), (F(x_2), F(x_3)), ..., (F(x_{k-1}), 1),$$

generating $U = \text{Uniform}(0, 1)$ random variables, and seeing which subinterval U falls into. Let $I_j = \mathcal{I}\left(U \in (F(x_j), F(x_{j+1}))\right)$. Then,

$$P(I_j = 1) = P\left(F(x_{j-1}) \leq U \leq F(x_j)\right) = F(x_j) - F(x_{j-1}) = p(x_j)$$

where $F(x_0)$ is defined to be 0. So, the probability that $I_j = 1$ is same as the probability that $X = x_j$, and this can be used to generate from the distribution of X. As an example, suppose that X takes values in $\mathcal{S} = \{1, 2, 3\}$ with probability mass function defined by the following table:

$p(x)$	x
p_1	1
p_2	2
p_3	3

To generate from this distribution we partition $(0, 1)$ into the three sub-intervals $(0, p_1)$, $(p_1, p_1 + p_2)$, and $(p_1 + p_2, p_1 + p_2 + p_3)$, generate a Uniform$(0, 1)$, and check which interval the variable falls into. The following R code does this, and checks the results for $p_1 = .4$, $p_2 = .25$, and $p_3 = .35$:

```
# n is the sample size
# p is a 3-length vector containing the corresponding probabilities
rX <- function(n, p)
{

   # generate the underlying uniform(0,1) variables
   U <- runif(n)

   # storage for the random variables, X
   X <- rep(0,n)

   # which U are in the first interval
   w1 <- which(U <= p[1])
   X[w1] <- 1
```

```
   # which U are in the second interval
   w2 <- which( (U > p[1]) & (U < sum(p[1:2])) )
   X[w2] <- 2

   # which U are in the third interval
   w3 <- which( U > sum(p[1:2]) )
   X[w3] <- 3


   return(X)

}

X = rX(10000, c(.4, .25, .35))

mean(X == 1)
[1] 0.407

mean(X == 2)
[1] 0.2504

mean(X == 3)
[1] 0.3426
```

The empirical probabilities appear to agree with the true values. Any discrete random variable with a finite sample space can be generated analogously, although the use of a for loop will be necessary when the number of intervals to check is large. The process will be similar when the variable has an infinite sample space– one example of this is the Poisson distribution. The probability mass function for the poisson with parameter λ has the form

$$p(x) = \frac{e^{-\lambda}\lambda^x}{x!}$$

whose sample space is all non-negative integers. The following R program generates from this distribution and compared the empirical mass function with the true mass function for $\lambda = 4$:

```
# function to calculate poission CDF at an integer x
# when lambda = L. This does the same thing as ppois()
pois.cdf <- function(x, L)
{

  # grid of values at which to calculate the mass function
  v <- seq(0, x, by=1)

  # return CDF value
  return( sum( exp(-L)*(L^v)/factorial(v) ) )
```

```
}

# n: sample size, L: lambda
r.pois <- function(n, L)
{

   U <- runif(n)
   X <- rep(0,n)


   # loop through each uniform
   for(i in 1:n)
   {

     # first check if you are in the first interval
     if(U[i] < pois.cdf(0,L))
     {

        X[i] <- 0

     } else
     {

        # while loop to determine which subinterval,I, you are in
        # terminated when B = TRUE
        B = FALSE
        I = 0
        while(B == FALSE)
        {

           # the interval to check
           int <- c( pois.cdf(I, L), pois.cdf(I+1,L) )

           # see if the uniform is in that interval
           if( (U[i] > int[1]) & (U[i] < int[2]) )
           {

              # if so, quit the while loop and store the value
              X[i] <- I+1
              B = TRUE

           } else
           {

             # If not, continue the while loop and increase I by 1
             I=I+1
           }
```

```
      }

    }


  }

  return(X)

}

# generate 1000 Pois(4) random variables
V = r.pois(1000, 4)

# empirical mass function
Mf <- c(0:15)
for(i in 0:15) Mf[i+1] <- mean(V==i)

# plot the observed mass function with the
# the true mass function overlaying it
b <- c(0:15)
plot(b, Mf, xlab="x", ylab="p(x)", main="Empirical mass function", col=2)
lines(b, Mf, col=2)

# overlay with true mass function
points(b, dpois(b,4), col=4)
lines(b, dpois(b,4), col=4)
```

2 Sampling from continuous distributions

Continuous random variables are (informally) those whose sample space is composed of real intervals not exclusively containing integers. In a continuous distribution the probability of taking on any particular value in the sample space in 0; probabilities can only be assigned to intervals in a continuous distribution. For this reason the logic of the previous section does not apply directly and other methods must be used.

2.1 The inversion method

It is a fact that if X has CDF F, then $F(X)$ has a Uniform$(0, 1)$ distribution. The proof of this is a straightforward calculation:

$$\begin{aligned} P(F(X) \leq x) &= P(F^{-1}(F(X)) \leq F^{-1}(x)) \\ &= P(X \leq F^{-1}(x)) \\ &= F(F^{-1}(x)) \\ &= x \end{aligned}$$

So the CDF of $F(X)$ is x, which is the same as the CDF of as Uniform$(0, 1)$. Here F^{-1} denotes the inverse of the CDF (also called the quantile function) and is defined as the function which satisfies:

$$F(x) = y \iff x = F^{-1}(y)$$

In other words, $F(F^{-1}(y)) = y$ and $F^{-1}(F(x)) = x$. The fact above implies that if X has CDF F, then $F^{-1}(U)$ will have CDF F. So, if you are able to calculate F^{-1}, and can generate uniforms, then you can generate a sample from F.

As an example, suppose you want to generate from a distribution with CDF

$$F(x) = \frac{x}{1+x}$$

for $x \in (0, \infty)$. This distribution has mean and variance equal to ∞. To calculate $F^{-1}(y)$, you specify a value for y and solve for x:

$$\begin{aligned} y &= \frac{x}{1+x} \\ (1+x)y &= x \\ y &= x - xy \\ y &= x(1-y) \\ \frac{y}{y-1} &= x \end{aligned}$$

so $F^{-1}(y) = \frac{y}{y-1}$. Therefore $\frac{U}{1-U}$ will have CDF F:

```
# n is the sample size
r.harmonic <- function(n)
{

   # generate uniforms
   U <- runif(n)

   # return F^-1(U)
   return( U/(1-U) )

}
```

```
X <- r.harmonic(1000)

# empirical CDF
v <- seq(0, 20, length=1000)
emp.cdf <- rep(0, 1000)
for(i in 1:1000) emp.cdf[i] <- mean(X <= v[i])

# true CDF
true.cdf <- v/(1+v)

plot(v, emp.cdf, xlab="X", ylab="F(X)", main="Empirical vs True CDF", col=2,type="l")
lines(v,true.cdf,col=4)
```

Due to the long tails, this distribution is a good candidate for a trial distribution in rejection sampling, which we will mention later. As a second example suppose X has CDF

$$F(x) = \left(\frac{1}{1+e^{-x}}\right)^{\theta}$$

where $\theta > 0$ is a parameter. This distribution is known as the skew logistic distribution, which is symmetric when $\theta = 1$, and skewed otherwise. This time the inversion is a bit more involved:

$$\begin{aligned} y &= \left(\frac{1}{1+e^{-x}}\right)^{\theta} \\ 1/y &= (1+e^{-x})^{\theta} \\ (1/y)^{1/\theta} &= 1+e^{-x} \\ -\log\left((1/y)^{1/\theta} - 1\right) &= x \end{aligned}$$

So $F^{-1}(y) = -\log\left((1/y)^{1/\theta} - 1\right)$.

```
# generate n samples from the skew logistic
# distribution with parameter Theta
r.skewlog <- function(n, Theta)
{

    U <- runif(n)
    return( -log( (1/U)^(1/Theta) - 1) )

}

X <- r.skewlog(1000, 4)

# empirical CDF
v <- seq(-10, 10, length=1000)
```

```
emp.cdf <- rep(0, 1000)
for(i in 1:1000) emp.cdf[i] <- mean(X <= v[i])

# true CDF
true.cdf <- (1 + exp(-v))^(-4)

plot(v, emp.cdf, xlab="X", ylab="F(X)", main="Empirical vs True CDF", col=2,type="l")
lines(v,true.cdf,col=4)
```

In many cases you can not symbolically invert the CDF (the normal distribution is an example of this). In principle, you can still use this method in such situations, but you will have to numerically calculate the quantile function. When we get to the section of the course dealing with line searching (optimization), we will do an example like this.

2.2 Transformation method

In some situations you can know mathematically that a particular function of a random variable has a certain distribution. On homework 1, problem 2 you were given an example of this– the transformation introduced there is called the Box-Muller transform, and is actually the preferred way to generate normally distributed variables.

When such relationships are know, it gives a simple way of generating from a distribution. In this example, suppose we wish to generate from the exponential(θ) distribution, and only have access to a computer which generates numbers from the skew logistic distribution, and do not know the inversion method. It turns out that if $X \sim$ SkewLogistic(θ), then $log(1 + e^{-X})$ is exponential(θ):

$$\begin{aligned}
P(\log(1+e^{-X}) \leq y) &= P(1+e^{-X} \leq e^{y}) \\
&= P(e^{-X} \leq e^{y}-1) \\
&= P(-X \leq \log(e^{y}-1)) \\
&= P(X \geq -\log(e^{y}-1)) \\
&= 1 - P(X \leq -\log(e^{y}-1)) \\
&= 1 - \left(\frac{1}{1+e^{\log(e^{y}-1)}}\right)^{\theta} \\
&= 1 - \left(\frac{1}{1+e^{y}-1}\right)^{\theta} \\
&= 1 - e^{-y\theta}
\end{aligned}$$

which is the CDF of an exponential(θ). We can check by simulation that this transformation is correct:

```
X <- r.skewlog(1000, 3)
Y <- log(1 + exp(-X))
```

Several other useful transformation identities are:

- if $X_1, ..., X_k \sim N(0,1)$, then $\sum_{i=1}^{k} X_i^2 \sim \chi_k^2$.
- if $X_1, ..., X_k \sim$ exponential(λ), then $\sum_{i=1}^{k} X_i^2 \sim \text{Gamma}(k, \lambda)$
- if $U \sim \text{Uniform}(0,1)$, then $1 - X^{1/k} \sim \text{Beta}(1, n)$
- if $X_1 \sim \text{Gamma}(\alpha, \lambda)$ and $X_2 \sim \text{Gamma}(\beta, \lambda)$, then $\frac{X_1}{X_2+X_1} \sim \text{Beta}(\alpha, \beta)$
- if $X \sim t(df)$, then $X^2 \sim F(1, df)$

As an exercise for later check by simulation that these identities hold– some useful R functions will be `rnorm()`, `rnorm()`, `rchisq()`, `rbeta()`, `rgamma()`, `rf()`, `rt()`.

2.3 Rejection Sampling

Rejection sampling is a general algorithm to generate samples from a distribution with density p_1 (called the target density) based only on the ability to generate from another distribution with density p_2 (usually referred to as the trial density) such that

$$\sup_x \frac{p_1(x)}{p_2(x)} \leq M < \infty$$

This is essentially saying that the tails of the trial density, p_2, are not infinitely fatter than those of the target density, p_1. The basic algorithm is:

1. Generate $U \sim \text{Uniform}(0,1)$
2. Generate $X \sim p_2$
3. If $U \leq \frac{p_1(X)}{M \cdot p_2(X)}$ then accept X as a realization from p_1, otherwise throw out X and try again

Showing the distribution of accepted draws is the same as the target distribution is a simple conditional probability calculation. First we will show that the rejection rate is $1/M$:

$$\begin{aligned}
P(\text{accept}) &= \sum_x P(\text{Accept}|X = x) P(\text{Drawing } X = x) \\
&= \sum_x P\left(U \leq \frac{p_1(x)}{M \cdot p_2(x)} | X = x\right) p_2(x) \\
&= \sum_x \frac{p_1(x)}{M \cdot p_2(x)} \cdot p_2(x) \\
&= \sum_x \frac{p_1(x)}{M} = 1/M
\end{aligned}$$

where $P(\text{accept}|X) = \frac{p_1(X)}{M p_2(X)}$, follows from the basics about the uniform CDF. Now to show that the distribution of accepted draws is the same as the target distribution:

$$\begin{aligned} P(X|\text{accept}) &= \frac{P(X, \text{accept})}{P(\text{accept})} \\ &= \frac{P(\text{accept}|X)P(\text{Drawing } X)}{P(\text{accept})} \\ &= \frac{P(\text{accept}|X)p_2(X)}{1/M} \\ &= \frac{p_1(X)}{Mp_2(X)} \cdot Mp_2(X) \\ &= p_1(X) \end{aligned}$$

The second lines follows from Bayes' Rule. The third line follows from the fact that $P(\text{Drawing } X) = p_2(X)$, and $P(\text{accept}) = 1/M$.

One drawback of this method is that you will end up generating more uniforms than necessary, since some of them will get rejected. Theoretically, as long as you know that p_1 indeed has longer tails then p_2, you can choose M to be ridiculously large and this will still yield a valid algorithm. Since the rejection rate is equal to $1/M$, you will have to, on average, generate $M \cdot n$ draws from the trial distribution **and** from the uniform distribution just to get n draws from the target distribution. So choosing M to be too large will yield an inefficient algorithm, and it pays to find a sharp bound for M.

The Cauchy distribution is one that will have larger tails than almost any other distribution you will ever deal with, so usually makes a good trial density. Also, for most non-negative random variables, the first distribution we demonstrated the inversion method can be used as a trial density, which we will use here.

As an example we will generate from the exponential(λ) distibution using rejection sampling. Recall the CDF $F(x) = x/(1+x)$ from a previous example; the density corresponding to this CDF is

$$\begin{aligned} \frac{\partial F(x)}{\partial x} &= \frac{\partial \dfrac{x}{1+x}}{\partial x} \\ &= \frac{(1+x) - x}{(1+x)^2} \\ &= \left(\frac{1}{1+x}\right)^2 \end{aligned}$$

We will take this to be p_2. In this case p_1 is the exponential density, $p_1(x) = \lambda e^{-\lambda x}$. To get a sharp bound for M, we now have to determine the maximum of

$$G(x) = \frac{p_1(x)}{p_2(x)} = \lambda e^{-\lambda x}(1+x)^2$$

We calculate the derivative of this function:

$$\begin{aligned} G'(x) &= \lambda \left(-\lambda e^{-\lambda x}(1+x)^2 + 2(1+x)e^{-\lambda x} \right) \\ &= \lambda e^{-\lambda x} \left(-\lambda(1+x)^2 + 2(1+x) \right) \\ &= \lambda e^{-\lambda x} \left(-\lambda x^2 + 2(1-\lambda)x + (2-\lambda) \right) \end{aligned}$$

You can check that when $G'(x) = 0$ has a solution, it is indeed a maximum. Since $\lambda e^{-\lambda x}$ is strictly positive, the solution to $G'(x) = 0$ must be the same as the solution to

$$Q(x) = -\lambda x^2 + 2(1-\lambda)x + (2-\lambda) = 0,$$

a quadratic equation with discriminant

$$4(1-\lambda)^2 - 4\lambda(2-\lambda)$$

which is always 4 independently of λ. So any solution of this quadratic will have the form

$$\frac{-2(1-\lambda) \pm 2}{-2\lambda} = \frac{(1-\lambda) \pm 1}{\lambda} = (-1 + 2/\lambda, -1)$$

since $x \geq 0$, the only feasible solution is $x^\star = -1 + 2/\lambda$. Thus when $x^\star = -1 + 2/\lambda \geq 0$, $M = G(x^\star)$ constitutes a sharp bound. When $-1 + 2/\lambda \leq 0$ (i.e. $\lambda \leq 2$), there is no feasible solution. However, when $\lambda \leq 2$,

$$Q(0) = (2-\lambda) \leq 0,$$

and furthermore,

$$\begin{aligned} Q'(x) &= -2\lambda x + 2(1-\lambda) \\ &= 2 - 2\lambda(1+x) \\ &\leq 2 - 2(1+x) && \text{(since } \lambda \leq 2\text{)} \\ &\leq 0 && \text{(since } -2(1+x) \downarrow\text{)} \end{aligned}$$

and so Q is monotonically decreasing which, combined with $Q(0) \leq 0$, implies Q is negative for any x. Therefore, $G'(x) = \lambda e^{-\lambda x} Q(x) \leq 0$, so G is also monotonically decreasing, and $x^\star = 0$. Finally this implies $M = G(0)$ when $\lambda \geq 2$. To summarize:

- If $\lambda \leq 2$, $M = G(-1 + 2/\lambda)$
- If $\lambda \geq 2$, $M = G(0)$.

We can plot M as a function of λ:

```
Lambda <- seq(.1, 5, length=1000)
```

```
M <- rep(0, 1000)
for(j in 1:1000)
{

   G <- function(x) Lambda[j]*( (1+x)^2 )*exp(-Lambda[j]*x)
   if( Lambda[j] < 2 )
   {

      M[j] <- G(-1 + 2/Lambda[j])
   } else
   {

      M[j] <- G(0)

   }

}
plot(Lambda, M)
```

Now that we have an explicit bound we can carry out rejection sampling. We will use $\lambda = 3/2$ in this example. We will also keep track of the total number of iterations required to collect $N = 1000$ samples, stored in the variable `niter`.

```
Lambda = 1.5

# sample size
N = 1000

# target density
p1 <- function(x) Lambda*exp(-x*Lambda)

# trial density
p2 <- function(x) (1 + x)^(-2)

# function to generate from trial density
r.trial <- function(n)
{
   U <- runif(n)
   return( U/(1-U) )
}

# bound on p1(x)/p2(x)
if(Lambda < 2)
{
   M = p1(-1 + 2/Lambda)/p2(-1 + 2/Lambda)
} else
{
   M = p1(0)/p2(0)
}
```

```
M = 100

X <- rep(0, N)

# number of iterations used
niter = 0

for(j in 1:N)
{

   # keep going until you get an accept
   B <- FALSE

   while(B == FALSE)
   {

      # add to the number of iterations
      niter = niter + 1

      # generate a candidate
      V <- r.trial(1)

      # calculate the ratio, evaluated at V
      R <- p1(V)/(M*p2(V))

      # generate a single uniform
      U <- runif(1)

      # if the ratio is lower the uniform,
      # accept this as a draw
      if(U < R)
      {

         X[j] <- V
         B <- TRUE

      }

   }

}

M
[1] 1.617415
mean(X)
[1] 0.6927637
var(X)
[1] 0.449128
```

```
niter
[1] 1625
```

Notice the sample mean and variance appear to agree with the theoretical mean and variance, which are 2/3, and 4/9, respectively. Also note the number of iterations required was 1625, which is approximately equal to $n \cdot M = 1617.415$. Now consider if we were lazy and just chose $M = 100$, and carried out the sampling (cut and paste the same code but uncomment the part `M=100`).

```
> mean(X)
[1] 0.6609276
> var(X)
[1] 0.4499482
> niter
[1] 98168
```

Notice we still appear to have sampled from the right distribution, but we used nearly 90 times the number of iterations, emphasizing the need for a sharp bound on M.