How to Sample from a Probability Distribution?

Agniva Chowdhury

Lecture 9: COMP 480/580
09/25/2025

From Chao Zheng’s slides

A Naive Approach: Inverse Transformation

Now we know we can simulate (pseudo) random numbers from Unif(0,1) in a computer.

Question: How to simulate random numbers from other distributions? (Bernoulli, exponential, Student's t,...)

If the target $\pi(x)$ has a simple cumulative distribution function (CDF) $F(x)$, the following well-known algorithm provides a simple solution.

Algorithm 3: Inverse Transformation Sampling

```
1 draw $U = u$ from Unif(0, 1).;
2 compute $x = F^{-1}(u)$;
3 deliver $X = x$.
```

Inverse Transformation

Theorem 3.1: Inverse Transformation Sampling

The resulting random variable X from Algorithm 3 follows the distribution of $F(x)$.

Proof: To see this, we will need to show $\mathbb{P}(X \leq x) = F(x)$, which means

$$\mathbb{P}(F^{-1}(U) \leq x) = F(x).$$

Since F is a CDF, the monotonicity of F guarantees that

$$\{F^{-1}(U) \leq x\} = \{U \leq F(x)\}.$$

Therefore

$$\mathbb{P}\left(F^{-1}(U) \leq x\right) = \mathbb{P}\left(U \leq F(x)\right) = F(x),$$

where the last equality is due to the property of Unif(0,1), i.e., $\mathbb{P}(U \leq a) = a$.

Example 3.1 Exponential Distribution Exp(λ)

Example 3.1 Derive an inverse transformation sampling algorithm for generating random numbers from Exponential Distribution Exp(λ), and implement it in R. Note that the pdf of Exp(λ) is

$$\pi(x) = \lambda \exp(-\lambda x), \quad x > 0.$$

The CDF for Exp(λ) is easy to calculate:

$$F(x) = \int_0^x \lambda \exp(-\lambda y) dy = 1 - \exp(-\lambda x).$$

Taking the inverse transformation yields

$$x = F^{-1}(y) = -(1/\lambda) \ln(1 - y).$$

Example 3.1 Generating Exponential Distribution Exp(λ)

Algorithm Ex 3.1

```
1 draw $U = u$ from Unif(0,1);
2 compute $x = -(1/\lambda)\ln(1-u)$;
3 deliver $X = x$.
```

Below is an R program implementing the above algorithm :

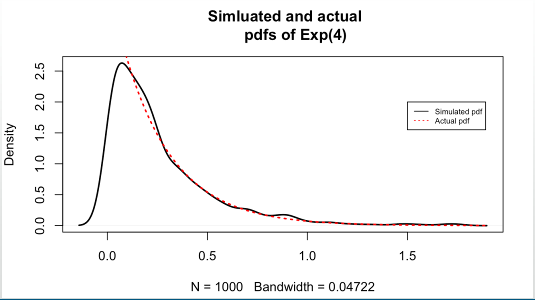
```r
r.myexp <- function(n, lambda){
## This function is for generating a sample of n
## observations from Exp(lambda) distribution.
X <- NULL
for(i in 1:n) {
           u <- runif(1)
           x <- -(log(1-u)/lambda)
           X[i] <- x }
    return(X)
}
```

Example 3.1 Generating Exponential Distribution Exp(λ)

Using the above algorithm, we generate a sample of 1000 random numbers from Exp(4), and plot the simulated pdf and the actual pdf:

```
set.seed(2020) ## fix random seed
mysample <- r.myexp(n=1000, lambda=4)

plot(density(mysample), lwd=2, main=" Simulated and actual
      pdfs of Exp(4)" )
curve(dexp(x,rate=4), from=0, add=T, lwd=2, lty=3, col="red")
legend(x=1.5, y=2, legend=c("Simulated pdf",
"Actual pdf"), lty=c(1,3), col=c("black", "red"), cex=0.6)
```



A-R sampling: Intuition

There are many distributions from which it is difficult, or even impossible, to directly simulate by an inverse transformation.

For example, can you easily write the CDF and its inverse function for normal distribution?

A-R sampling: Intuition

There are many distributions from which it is difficult, or even impossible, to directly simulate by an inverse transformation.

For example, can you easily write the CDF and its inverse function for normal distribution?

Question: How to simulate random numbers from those difficult distributions?

Accept-Reject (A-R) Sampling (sometimes known as rejection sampling or acceptance-rejection sampling) provides an elegant solution to simulate difficult distribution.

3.2.1 A-R Sampling Algorithm

A-R Sampling: the Algorithm

Given target densities $\pi(x)$ and another density $p(x)$ on $\mathbb{X}$ with $\pi(x) \leq Mp(x)$ for all $x \in \mathbb{X}$ and some constant $M < \infty$, we can generate a sample from π as follows:

Algorithm 2: A-R Sampling

```
1 draw $X = x$ from $p(x)$;
2 draw $Y = y$ from Unif(0,1);
3 if $y \leq \frac{\pi(x)}{Mp(x)}$ then
        accept $X = x$ as a sample from $\pi$;
else
        reject $X = x$.
end
```

We call $p(x)$ as the **instrumental density** or the **proposal density**. Since $\pi(x)$ and $p(x)$ are both pdfs, M **is necessarily larger than 1**.

Theory of A-R Sampling

Theorem 3.3

The distribution of the samples generated by A-R sampling is $\pi(x)$.

Proof.

Theory of A-R Sampling

Theorem 3.3

The distribution of the samples generated by A-R sampling is $\pi(x)$.

Proof. For any (measurable) set $\mathcal{A} \subset \mathbb{X}$,

$$\mathbb{P}(X \in \mathcal{A} | X \text{ is accepted}) = \frac{\mathbb{P}(X \in \mathcal{A}, X \text{ is accepted})}{\mathbb{P}(X \text{ is accepted})}.$$

Note that

$$\begin{aligned}\mathbb{P}(X \in \mathcal{A}, X \text{ is accepted}) &= \int_{\mathbb{X}} \int_0^1 \mathbb{I}_{\mathcal{A}}(x) \mathbb{I}\left(y \le \frac{\pi(x)}{Mp(x)}\right) p(x) dy dx \\ &= \int_{\mathbb{X}} \mathbb{I}_{\mathcal{A}}(x) \frac{\pi(x)}{Mp(x)} p(x) dx \\ &= \pi(\mathcal{A})/M\end{aligned}$$

$$\mathbb{P}(X \text{ is accepted}) = \mathbb{P}(X \in \mathbb{X}, X \text{ is accepted}) = \pi(\mathbb{X})/M = 1/M.$$

Hence, $\mathbb{P}(X \in \mathcal{A} | X \text{ is accepted}) = \pi(\mathcal{A})$, which completes the proof. □

Normalising constants

In practice, often we only know $\pi(x)$ and $p(x)$ **up to some normalising constants C_π and C_p**, respectively; i.e.

$$\pi = \frac{\tilde{\pi}}{C_\pi} \text{ and } p = \frac{\tilde{p}}{C_p}$$

where only $\tilde{\pi}, \tilde{p}$ are known but C_π and C_p are not available. However, we can still use A-R algorithm in this scenario as

$$\frac{\pi(x)}{p(x)} \leq M \Leftrightarrow \frac{\tilde{\pi}(x)}{\tilde{p}(x)} \leq \tilde{M},$$

where $\tilde{M} = M\frac{C_\pi}{C_p}$. This means we can ignore the normalising constants if we can find a $\tilde{M}$ to bound $\frac{\tilde{\pi}(x)}{\tilde{p}(x)}$, then we can apply the A-R algorithm.

Remarks on A-R Sampling

1. We can generate random numbers from some very complicated target pdf $\pi(x)$ even if $p(x)$ is simple.

2. The computational cost for this algorithm depends on the acceptance probability $\mathbb{P}\left(y \leq \frac{\pi(x)}{Mp(x)}\right) = \frac{1}{M}$, which we want it to be as large as possible in simulations. This means we need to choose $p(x)$ properly.

3. The distribution of the random numbers generated from A-R sampling is **exactly** (NOT approximately) $\pi(x)$.

4. The algorithm can be extended to multivariate random number generation.

Example 3.2: Generating Beta(a, b)

Example 3.2 Derive an A-R sampling algorithm for generating random numbers from Beta(a, b) distribution in the situation when $a \geq 1$ and $b \geq 1$, and implement it in R. Note that the pdf of Beta(a, b) is

$$\pi(x) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1}(1-x)^{b-1}\mathbb{I}(0 < x < 1),$$

where $\Gamma(z) = \int_0^\infty x^{z-1}e^{-x}dx$.

It is easy to see that $\pi(x)$ is upper bounded by $\dfrac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\mathbb{I}(0 < x < 1)$,0 and a pdf induced by this upper bound is $p(x) = \mathbb{I}(0 < x < 1)$, which is exactly the pdf of Unif(0,1). Then we can set $M = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}$.

Example 3.2: Generating Beta(a, b)

In this case, we have $\frac{\pi(x)}{Mp(x)} = x^{a-1}(1-x)^{b-1}$.
Therefore, the A-R sampling algorithm for Beta(a, b) is as follow:

Algorithm Ex 3.2

```
1 draw $X = x$ from Unif(0,1);
2 draw $Y = y$ from Unif(0, 1) independently from $X$.
3 if $y \leq x^{a-1}(1-x)^{b-1}$, accept $X = x$;
  otherwise reject.
```

Suppose we want to generate n samples from the distribution of Beta(a, b), we just simply deliver x to Z when $X = x$ is accepted, and repeat the Steps 1.-3. until collecting n entries in Z.

Example 3.2: Generating Beta(a, b)

An R program implementing above algorithm:

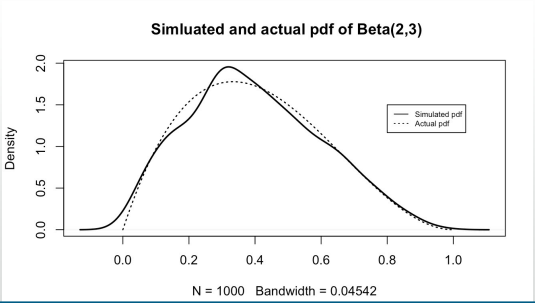
```r
r.mybeta <- function(n, a, b){
## This function is for generating a sample of n
## observations from Beta(a,b) distribution with the
## restriction that a >=1 and b>=1.
X <- NULL
for(i in 1:n) {
           repeat { x <- runif(1)
                    y <- runif(1)
                    if(y <= x^(a-1)*(1-x)^(b-1)) {
                       X[i] <- x
                       break}
                  }
              }
    return(X)
}
```

Example 3.2: Generating Beta(a, b)

Using the above algorithm, we generate a sample of 1000 random numbers from Beta(2,3), and plot the simulated pdf and the actual pdf:

```
set.seed(2020)
mysample <- r.mybeta(n=1000, a=2, b=3)

plot(density(mysample),lwd=2, main=" Simulated and actual
      pdf of Beta(2,3)" );
curve(dbeta(x,2,3),from=0,to=1, add=T, lwd=1.5, lty=3);
legend(x=0.8, y=1.5, legend=c("Simulated pdf", "Actual pdf"),
      lty=c(1,3), cex=0.6)
```



Example 3.3: Generating $N(0,1)$

Example 3.3 Derive an A-R sampling algorithm for generating random numbers from $N(0,1)$ from a double exponential distribution $\mathcal{L}(\lambda)$ with $\lambda > 0$, which has the density form

$$f(x) = \frac{\lambda}{2}\exp(-\lambda|x|),$$

and implement it in R.

Example 3.3: Generating $N(0,1)$

Example 3.3 Derive an A-R sampling algorithm for generating random numbers from $N(0,1)$ from a double exponential distribution $\mathcal{L}(\lambda)$ with $\lambda > 0$, which has the density form

$$f(x) = \frac{\lambda}{2}\exp(-\lambda|x|),$$

and implement it in R.

A standard normal distribution has the pdf

$$\pi(x) = \sqrt{\frac{1}{2\pi}}\exp(-x^2/2).$$

It is easy to see that, when $x > 0$

$$\frac{\pi(x)}{f(x)} = \frac{1}{\lambda}\sqrt{\frac{2}{\pi}}\exp\left(-\frac{x^2 - 2\lambda|x|}{2}\right) \leq \frac{1}{\lambda}\sqrt{\frac{2}{\pi}}\exp\left(\frac{\lambda^2}{2}\right),$$

hence we can let $M = \frac{1}{\lambda}\sqrt{\frac{2}{\pi}}\exp\left(\frac{\lambda^2}{2}\right)$.

Example 3.3: Generating $N(0,1)$

Also, $\frac{\pi(x)}{Mf(x)} = \exp\left(-\frac{(|x|-\lambda)^2}{2}\right)$.

Therefore, the request A-R sampling algorithm should be:

Algorithm Ex 3.3

```
1 draw $X = x$ from $\mathcal{L}(\lambda)$;
2 draw $Y = y$ from Unif(0, 1);
3 if $y \leq \exp\left(-\frac{(|x|-\lambda)^2}{2}\right)$, accept $X = x$;
  otherwise reject.
```

Note that in Step 1, we need to draw $X = x$ from $\mathcal{L}(\lambda)$, to implement this we can apply the inverse transformation sampling.

Example 3.2: Generating $N(0,1)$

By simple calculation, the CDF for $\mathcal{L}(\lambda)$ is

$$F(x) = \int_{-\infty}^{x} \frac{\lambda}{2} \exp\left(-\lambda|z|\right) dz = \begin{cases} \frac{1}{2}\exp(\lambda x), & \text{if } x \le 0 \\ 1 - \frac{1}{2}\exp(-\lambda x), & \text{if } x > 0. \end{cases}$$

Thus, taking the inverse transformation yields

$$F^{-1}(u) = \begin{cases} \lambda^{-1}\ln(2u), & \text{if } u \le 1/2 \\ -\lambda^{-1}\ln(2-2u), & \text{if } u > 1/2. \end{cases}$$

Example 3.2: Generating $N(0,1)$

The new sampling algorithm for $N(0,1)$ is as follow:

Algorithm Ex 3.3

```
1 draw $U = u$ from Unif(0,1);
2 compute $X = x = F^{-1}(u)$;
3 draw $Y = y$ from Unif(0, 1);
4 if $y \leq \exp\left(-\frac{(|x|-\lambda)^2}{2}\right)$, accept $X = x$;
  otherwise reject.
```

Example 3.3: Generating $N(0,1)$

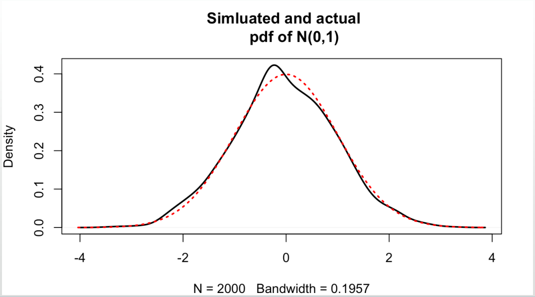
An R program implementing above algorithm:

```
r.mynormal <- function(n, lambda){
## This function is for generating a sample of n
## observations from N(0,1)
X <- NULL
for(i in 1:n) {
           repeat { u <- runif(1)
                    if (u<=0.5){
                       x <- lambda^(-1)*log(2*u)
                    } else{
                       x <- -lambda^(-1)*log(2-2*u)
                    }
                       y <- runif(1)
                    if (y <= exp(-(abs(x)-lambda)^2/2)) {
                       X[i] <- x
                       break}
                  }
              }
    return(X)
}
```

Example 3.3: Generating $N(0,1)$

Using the above algorithm, let $\lambda = 1$ in $\mathcal{L}(\lambda)$, we generate a sample of 2000 random numbers from $N(0,1)$, and plot the simulated pdf and the actual pdf:

```r
set.seed(2020)
mysample <- r.mynormal(n=2000, lambda=1)
plot(density(mysample),lwd=2, main=" Simulated and actual
      pdf of N(0,1)" )
curve(dnorm(x), add=T, lwd=2, lty=3, col="red")
legend(x=2, y=0.4, legend=c("Simulated pdf",
"Actual pdf"), lty=c(1,3), cex=0.6)
```



Acceptance Probability/Efficiency

The probability of a generated X being accepted is $\tau = 1/M$. We define τ as the **efficiency** of A-R sampling.

Further, denote by T the number of trials to achieve an acceptance of $X = x$. Then

$$\mathbb{P}\left(T = t\right) = \left(1 - \tau\right)^{t-1}\tau,$$

which means T follows a Geometric(τ) distribution with $\mathbb{E}(T) = \tau^{-1} = M$.

This implies that the larger the τ is, the smaller is T expected to be, thus the higher is the efficiency.

Acceptance Probability and Efficiency

Recall **Example 3.2**, if $a = 2$ and $b = 3$, then we have $M = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} = 12$. The efficiency for A-R sampling is $1/M = 1/12$. We can also verify this through simulation.

```
r.mybeta <- function(n, a, b){
X <- NULL
count <- 0 ## count the number of total samples
for(i in 1:n) {
           repeat { x <- runif(1)
                    y <- runif(1)
                 count <- count + 1
                   if(y <= x^(a-1)*(1-x)^(b-1)) {
                       X[i] <- x; break}  }  }
    return(list(X=X, count=count))
}

set.seed(2020)
totalsamples <- r.mybeta(n=1000, a=2, b=3)$count
```