April 28, 2021

A Library of Random Variate Generation Routines

Noelle Richards

ISYE 6644 – Simulation

Spring 2021 Project 2

A Library of Random Variate Generation Routines

ABSTRACT

A library of random variate generation routines is created using various methods, including Inverse Transform and Acceptance-Rejection. When using a solid uniform distribution as a base, these routines can produce random variates that fit each distribution included in the library.

PROBLEM DESCRIPTION

We need a library of random variate generation routines that includes the basic distributions. This library should be able to adjust the random variates based on the parameters given for each distribution (e.g., a Uniform (1, 5) will only return random variates with values between 1 and 5). The distributions included in this library are Bernoulli, Binomial, Geometric, Poisson, Uniform, Exponential, Normal, Erlang, Triangular, Gamma, and Weibull.

MAIN FINDINGS

Uniform Distribution

The one of the most important distributions is the uniform distribution. It provides the base for all the others, thanks to the Inverse Transform Method. I used L'Ecuyer's combined generator to create my uniform RVs.

Bernoulli Distribution

The Bernoulli Distribution is another distribution that provides the foundation to many others. In this library, the user inputs the number of RVs they would like (*q*), then the probability of success. The code then generates *q* uniform variates and checks to see if they are less than or equal to the probability. If they are, the code deems this a success and assigns the value "1" to that RV. If the uniform variate is greater than the probability, then the RV is assigned the value "0".

Binomial Distribution

The binomial distribution is the number of successes in *n* Bernoulli trials with probability *p*. The code uses the same code from the Bernoulli distribution and runs *n* trials for each variate requested.

Geometric Distribution

The geometric distribution is also related to the Bernoulli distribution. It represents the number of failures before a success. In order to calculate the RV from a uniform RV, I used the Inverse Transform Method (Eqn. 1).

$$X = \left\lceil \frac{\ln(U)}{\ln(1-p)} \right\rceil \tag{1}$$

Poisson Distribution

What's a statistician's favorite way to kill people? Poisson Distribution! The Poisson distribution represents the probability of a certain event or events happening within a fixed

interval of time. For example, the probability of 7 arrivals in an hour. In order to generate RVs from the Poisson distribution, the user must supply λ, or the rate at which the event occurs. The Poisson RV is calculated using a variation of the Acceptance-Rejection method, counting the number of uniform RVs until Eqn. 2 becomes true.

$$e^{(-\lambda)} > \prod_{i=1}^{n+1} U_i \tag{2}$$

Exponential Distribution

The exponential distribution is closely related to the Poisson distribution. It uses λ as well as its only parameter. It represents the time between events in a Poisson distribution, e.g. time between arrivals to a barbershop. In this library, the exponential distribution RV is also calculated using the Inverse Transform Method (Eqn. 3).

$$X = -\frac{1}{\lambda}\ln(U) \tag{3}$$

Normal Distribution

The normal distribution is another one of those extremely important distributions. In fact, because of the Central Limit Theorem, many of the other distributions (such as binomial and Poisson) converge to the normal distribution with a high enough sample size. In this code routine, the Box-Muller method is used to generate the standard normal variates (Eqn. 4 and Eqn. 5). The standard normal RVs are then converted to the user's requested mean and variance using Eqn. 6.

$$Z_1 = \sqrt{\text{-2 ln}(U_1)}\, cos(2\pi U_2) \quad (4)$$

$$Z_2 = \sqrt{\text{-2 ln}(U_1)}\, \sin(2\pi U_2) \quad (5)$$

$$X = \mu + \sigma Z \quad (6)$$

Erlang Distribution

The Erlang Distribution is a special case of the gamma distribution which can simplify to the exponential distribution (when the shape parameter is equal to 1). The Erlang RV is generated in this code once again by using the Inverse Transform Method (Eqn. 7).

$$X = -\frac{1}{\lambda} \ln \prod_{i=1}^{k} U_i \quad (7)$$

Triangular Distribution

The triangular distribution RV has different equations based on the value of the uniform RV used to generate it. If the uniform RV is greater than 0.5, the code uses Eqn. 8. Otherwise, it uses Eqn. 9.

$$X = \sqrt{2\text{U}} \quad (8)$$

$$X = 2 - \sqrt{2(1-U)} \quad (9)$$

Gamma Distribution

The version of a gamma RV generator included in this code only works if the shape parameter (*b*) inputted by the user is an integer. If the shape parameter is an integer, the sum of *b* exponential RVs has a gamma distribution (Eqn. 10).

$$X = -\alpha \ln\left(\prod_{i=1}^{b} U_i\right) \tag{10}$$

Weibull Distribution

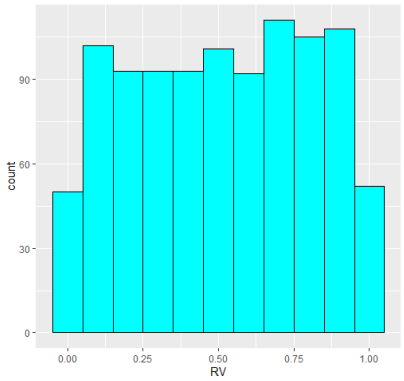
The last distribution this library includes is the Weibull distribution. It is often used in engineering to model reliability. The Weibull RV is produced using the Inverse Transform Method (Eqn. 11).

$$X = -\frac{1}{\lambda}[-ln(U)]^{1/\beta} \tag{11}$$

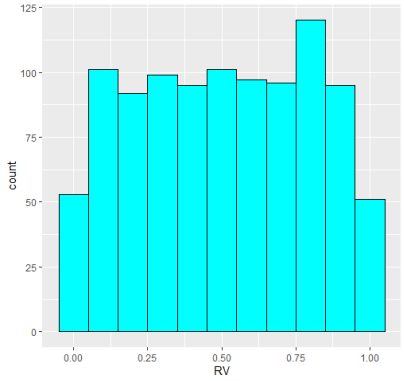
CONCLUSION

Now that the RVs can be generated, the final question to be answer is how "good" are these generators? Unfortunately, I was unable to pass the chi-squared or other goodness-of-fit tests with either my code-generated RVs or using RVs generated using R's already existing functions. Instead, we will do a visual check of my code against the built-in functions, assuming the built-in functions do pass any g-o-f tests. We will look at the uniform, normal, and gamma distributions with 1000 RVs each.

The uniform distribution, *U*(0,1), shows a slight difference but overall, the graphs look very similar and seem to all be near 100 for each histogram bin (as expected).

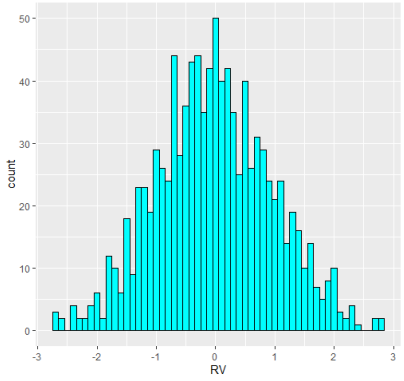


Student Generated RVs - Uniform

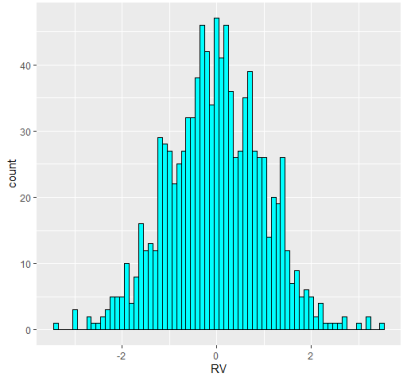


R Generated RVs - Uniform

The normal distribution, Nor(0,1), has a bigger difference in look, suggesting that using the R generated variates would be a better bet.

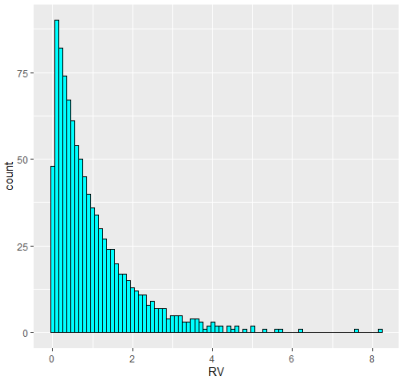


Student Generated RVs - Normal

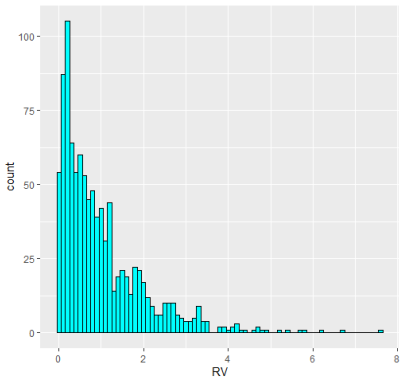


R Generated RVs - Normal

The gamma distribution, Γ(1,1), actually shows more variation in shape with the R generated RVs, which could mean that the student generated RVs are a better fit for the distribution.



Student Generated RVs - Gamma



R Generated RVs - Gamma

In conclusion, using the Inverse Transform and Acceptance-Rejection methods allows for decent RV generation from a variety of distributions.

APPENDIX – SOURCE CODE

```
# Library of Random Variate Generation Routines.
#
# Discrete Distributions included: Bernoulli, Binomial, Negative Binomial,
#   Geometric, Poisson
#
# Continuous Distributions included: Uniform, Exponential, Normal, Erlang,
#   Triangular, Gamma, Weibull

# Clear Environment and load relevant libraries
rm(list=ls())
seed.number <- 42
set.seed(seed.number)
library(ggplot2)
library(dplyr)

#Function definition

# User Input
RVG <- function(){

 #Uniform Distribution function
 unif <- function(seed.number, q) {
  x1 <- numeric(q+3)
  x1[1]<-seed.number+1
  x1[2]<-seed.number+2
  x1[3]<-seed.number+3

  x2 <- numeric(q+3)
  x2[1]<-seed.number+4
  x2[2]<-seed.number+5
  x2[3]<-seed.number

  Y <- numeric(q)
  counter <- seq(from=4, to=q+3, by=1)

  for (i in counter) {
   x1[i]<- (1403580*x1[i-2]-810728*x1[i-3])%%((2^32)-209)
   x2[i]<- (527612*x2[i-1]-1370589*x2[i-3])%%((2^32)-22853)
   Y[i-3]<- (x1[i]-x2[i])%%((2^32)-209)
  }
  RV <- Y/((2^32)-209)
  return(RV)
 }
 dst <- menu(c("Bernoulli", "Binomial", "Geometric", "Poisson", "Uniform",
           "Exponential", "Normal", "Erlang", "Triangular", "Gamma", "Weibull"),
          title="Please select which distribution from which to generate random variates. ")

 q <- as.integer(readline(prompt = "Please enter the number of random variates you would like generated: "))

#####################################################################
 #Bernoulli
 if (dst == 1){
  p <- as.numeric(readline(prompt = "Please enter the probability of success: "))
  std.unif <- unif(seed.number, q)
  #RV Generation
  RV <- as.integer(std.unif <= p)
#####################################################################
 #Binomial
 } else if ( dst == 2){
  n <- as.numeric(readline(prompt = "Please enter the number of Bernoulli trials: "))
  p <- as.numeric(readline(prompt = "Please enter the probability of success: "))
  #RV Generation
  RV <- numeric(q)
```

```
  for (j in 1:q){
    std.unif <- unif(sample.int(1, n=(2^(16))), n)
    #Bernoulli
    RV.b <- as.integer(std.unif <= p)
    RV[j] <- sum(RV.b)
  }
#####################################################################
 #Geometric
 } else if (dst == 3){
  p <- as.numeric(readline(prompt = "Please enter the probability of success: "))
  #RV Generation
  std.unif <- unif(seed.number, q)
  RV <- ceiling(log(std.unif)/(log(1-p)))
#####################################################################
  ###################################################################
 #Poisson
 } else if (dst == 4){
  lambda <- as.numeric(readline(prompt = "Please enter lambda: "))
  RV <- numeric(q)
  #RV Generation
  a <- exp(-lambda)
  RV <- numeric(q)
  for (i in 1:q){
    p <- 1
    X <- -1
    while (p >= a){
      std.unif <- unif(sample.int(1, n=(2^(16))),1)
      p<-p*std.unif
      X<- X+1
    }
    RV[i] <- X
  }
#############################################################
 #Uniform
 } else if (dst == 5){
  #Uniform
  a <- as.numeric(readline(prompt = "Please enter the lower bound: "))
  b <- as.numeric(readline(prompt = "Please enter the upper bound: "))
  #RV Generation
  std.unif <- unif(seed.number, q)
  RV <- ((b-a)*std.unif) + a
#####################################################################
 #Exponential
 } else if (dst == 6){
  lambda <- as.numeric(readline(prompt = "Please enter lambda: "))
  #RV Generation
  std.unif <- unif(seed.number,q)
  RV <- (-1/lambda)*log(1-std.unif)
#####################################################################
 #Normal
 } else if (dst == 7){
  #Normal
  mu <- as.numeric(readline(prompt = "Please enter the mean: "))
  sigma <- as.numeric(readline(prompt = "Please enter the standard deviation: "))
  #RV Generation
  w<- q/2
  u1 = unif(seed.number, w)
  u2 = unif(seed.number*25, w)

  z1=rep(0,w)
  z2=rep(0,w)

  for (i in 1:w){
    z1[i] = sqrt(-2*log(u1[i]))*cos(2*pi*u2[i])
    z2[i] = sqrt(-2*log(u1[i]))*sin(2*pi*u2[i])
  }
```

```
   std.RV <- c(z1,z2)
   RV <- mu + sigma*std.RV
#####################################################################
  } else if (dst == 8){
   #Erlang
   k <- as.numeric(readline(prompt = "Please enter k: "))
   lambda <- as.numeric(readline(prompt = "Please enter lambda:  "))
   RV <- numeric(q)
   #RV Generation
   for (i in 1:q){
    std.unif <- unif(sample.int(1, n=(2^(16))), k)
    RV[i] <-(-1/lambda)*log(prod(std.unif))
   }
#####################################################################
  } else if (dst == 9){
   #Triangular
   a <- as.numeric(readline(prompt = "Please enter the lower bound: "))
   b <- as.numeric(readline(prompt = "Please enter the upper bound: "))
   c <- as.numeric(readline(prompt = "Please enter the mean: "))
   #RV Generation
   std.unif <- unif(seed.number, q)
   RV <- numeric(q)
   Fc <- (c-a)/(b-a)
   for (i in 1:q) {
    if (std.unif[i] < Fc){
     RV[i] <- a+(std.unif[i]*(b-a)*(c-a))^.5
    } else {
     RV[i] <- b-((1-std.unif[i])*(b-a)*(b-c))^.5
    }
   }
#####################################################################
  } else if (dst == 10){
   #Gamma
   alpha <- as.numeric(readline(prompt = "Please enter alpha:  "))
   beta <- as.numeric(readline(prompt = "Please enter the shape parameter (integers only): "))
   RV <- numeric(q)
   #RV Generation
   for (i in 1:q){
    std.unif <- unif(seed.number*i, beta)
    RV[i] <- (-alpha)*log(prod(std.unif))
   }
#####################################################################
  } else if (dst == 11){
   #Weibull
   lambda <- as.numeric(readline(prompt = "Please enter lambda:  "))
   beta <- as.numeric(readline(prompt = "Please enter beta: "))
   #RV Generation
   std.unif <- unif(seed.number, q)
   RV <- (1/lambda)*(-log(1-std.unif))^(1/beta)
  }
  return(RV)
}

RV <- RVG()

#Graph
RV.df <- data.frame(RV)
ggplot(RV.df, aes(x=RV)) +
  geom_histogram(binwidth = .1, colour = 'black', fill = 'cyan')
```