PSTAT 194CS Midterm Project
RV Generation Methods

Ostapenko, Vasiliy (vostapenko, 774 970 8)
Collaborated with: N/A

Contents

Item 1

Marsaglia Method for N(mu, var)

The Marsaglia method, also called the polar method, is a way to generate pairs of independent normal random variables using pairs of independent uniform RVs. The method falls under the umbrella of rejection sampling algorithms. Indeed, we generate Unif(0, 1) RVs two at a time until they fall within the unit disc. When they do, we return a function of both of them as two independent normal RVs.

The theoretical background behind this algorithm is as follows: if we have a standard uniform RV, then the tuple (cos(2piU), sin(2piU)) is uniform on the unit circle; moreover, multiplying that point by an RV p with cdf $\int_0^x re^{-r^2/2}dr$, we get the point (pcos(2piU), psin(2piU)), whose coordinates are both distributed normally.

Link to Source, i.e. Wikipedia

```
marsaglia = function(mu, sigma) {
  done = FALSE
  while(!done) {
    u_vec = runif(n=2, min=-1, max=1)
    x = u_vec[1]
    y = u_vec[2]
    s = x^2 + y^2
    if(s > 0 & s < 1) {
      done = TRUE
    }
  }
  z_1 = x*sqrt( (-2*log(s))/(s) )
  z_2 = y*sqrt( (-2*log(s))/(s) )
```

```
  return(c(mu+sigma*z_1, mu+sigma*z_2))
}
```

```
get_n_marsaglias = function(n, mu, sigma) {
  if(n%%2 == 1) {
    n_vec = as.vector(
      replicate((n+1)/2, marsaglia(mu=mu, sigma=sigma)))
    n_vec = n_vec[-1]
  } else {
    n_vec = as.vector(
      replicate(n/2, marsaglia(mu=mu, sigma=sigma)))
  }
  return(n_vec)
}
```

Ziggurat Method for Exp(lambda)

The Ziggurat method is a way to generate various different RVs with the help of two uniform ones. In our implementation, we choose to focus on generating exponential random variables. According to the method, we first divide the positive half of the chosen RV's density into horizontal rectangles, called segments. This is done in a way such that the bottom right point of every segment rests directly on the pdf of the RV and the bottom left point of every segment rests on the y axis. Moreover, every segment (rectangle) must have equal area and we try to get the top left point of the top rectangle as close to the pdf evaluated at 0. Now the issue of the bottom rectangle appears - how can we cover an endless tail with a finite rectangle? Actually, the bottom segment is made up of the rectangle plus the rest of the tail area evaluated as an integral. With this setup, we are ready to operate on the segments and the pdf to actualize our rejection sampling Ziggurat algorithm.

The theoretical background behind this algorithm is as follows: we try to cover the density with rectangles (think horizontal Riemann integration) to minimize the proportion of samples which are rejected by the algorithm; we also try to find an optimal cutoff point for the lowest rectangle which separates it from the curve (must be integrated), which, for the exponential density, is approximately 7.697; now we choose a random layer (low-resolution y coordinate), test if the respective x coordinate lies within the density, and return the x if yes; if no, we choose a high resoltuion y coordinate, and perform the rejection test; if we land on the tail, we use a "fallback" algorithm to generate our value.

Link to Source, i.e. Wikipedia

```
f = function(x) {
  return( exp(-x) )
}
f_inv = function(x) {
  return( -1*log(x) )
}

x1 = 7.69711747013104972
y1 = f(x1)
t = exp(-1*x1)
A = x1*y1 + t

n = 256
vals = list()
vals[[1]] = c(x1, y1)
for(i in 2:256) {
  y2 = y1 + A/x1
```

```
  x2 = f_inv(y2)
  vals[[i]] = c(x2, y2)
  y1 = y2
  x1 = x2
}
```

```
ziggurat = function(lambda) {
  done = FALSE
  while(!done) {
    # Step 1
    l = sample.int(255, 1)
    u0 = runif(1, 0, 1)
    u1 = runif(1, 0, 1)

    # Step 2
    x = u0*vals[[l]][1]

    # Step 3
    if(x < vals[[l+1]][1]) {
      done = TRUE
    }

    # Step 4
    if(l == 1) {
      x = vals[[l]][1] - log(u1)
    }

    # Step 5
    y = vals[[l]][2] + u1*(vals[[l+1]][2] - vals[[l]][2])

    # Step 6
    if(y < f(x)) {
      done = TRUE
    }
  }
  return((1/lambda)*x)
}
```

```
get_n_ziggurats = function(n, lambda) {
  n_vec = as.vector(replicate(n, ziggurat(lambda=lambda)))
  return(n_vec)
}
```

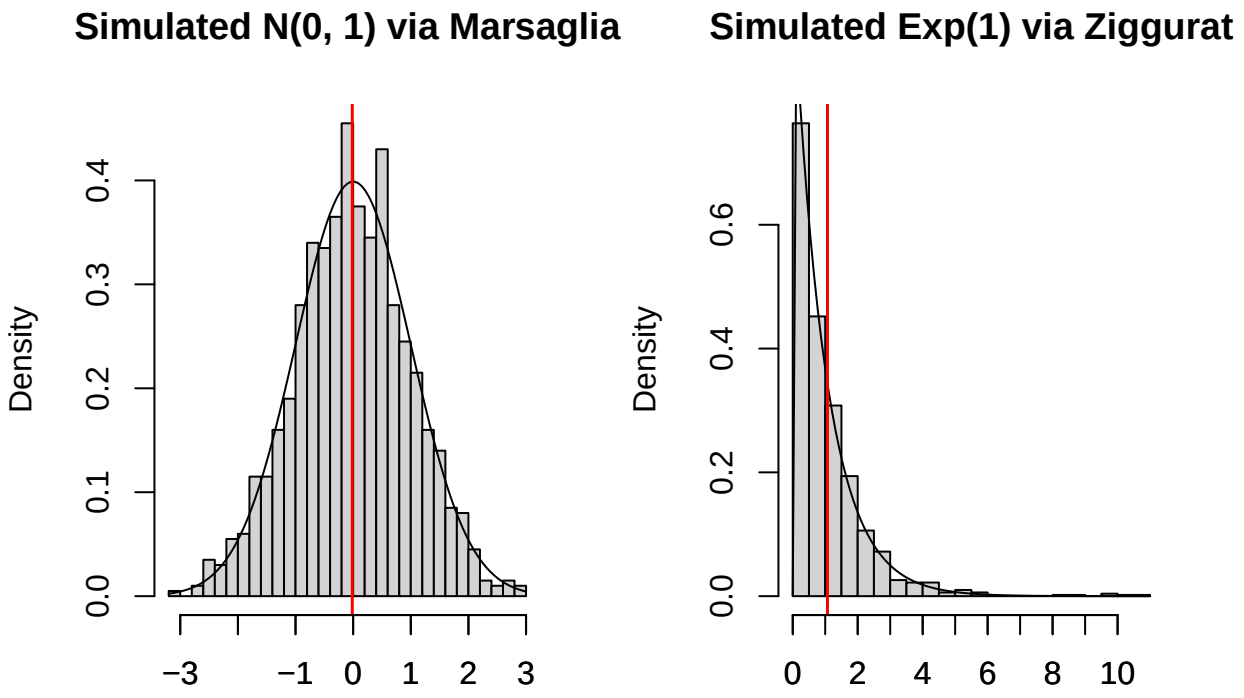
Testing RV Generation Algorithms and Plotting

```
mu = 0
sigma = 1
m = get_n_marsaglias(n=1000, mu=mu, sigma=sigma)

lambda = 1
z = get_n_ziggurats(n=1000, lambda=lambda)
```

```
par(mfrow=c(1, 2))
{
```

```
  {
    x = seq(mu - 3*sigma, mu + 3*sigma, by=0.01)
    hist(m, breaks=30, freq=FALSE, xlab="",
         main=paste0("Simulated N(", mu, ", ", sigma, ") via Marsaglia"))
    curve(dnorm(x, mean=mu, sd=sigma), add=TRUE)
    abline(v=mean(m), col="red", lwd=1.5)
    axis(side=1, at=c(-3:3))
  }
  {
    x = seq(0, 10, by=0.01)
    hist(z, breaks=30, freq=FALSE, xlab="",
         main=paste0("Simulated Exp(", lambda, ") via Ziggurat"))
    curve(dexp(x, rate=lambda), add=TRUE)
    abline(v=mean(z), col="red", lwd=1.5)
    axis(side=1, at=c(0:10))
  }
}
```



KS Tests

```
test1 = ks.test(x=m, y="pnorm", mu, sigma)
test1
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
```

```
## data:  m
## D = 0.017, p-value = 0.9
## alternative hypothesis: two-sided
```

For the n=1000 independent N(0, 1) RVs, we conduct a one-sample, two-sided KS test to determine whether they indeed come from the stated distribution. Under the null, our sample (m) comes from a standard normal distribution. Under the alternative, our sample does not come from a standard normal distribution. With a p-value of 0.9253, which is much greater than our chosen cutoff of 0.05, we fail to reject the null. Thus we can assume our sample comes from the N(0, 1) distribution.

```
test2 = ks.test(x=z, y="pexp", lambda)
test2
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  z
## D = 0.028, p-value = 0.4
## alternative hypothesis: two-sided
```

Likewise, for the n=1000 independent Exp(1) RVs, we conduct a one-sample, two-sided KS test to determine whether they indeed come from the stated distribution. Under the null, our sample (z) comes from an exponential distribution with rate parameter 1. Under the alternative, our sample does not come from that distribution. With a p-value of 0.4233, which is much greater than our chosen cutoff of 0.05, we fail to reject the null. Thus we can assume our sample comes from the Exp(1) distribution.

Final project proposal: we will use a dataset of honeybee wing size and vein density and apply bootstrapping resampling method to it. We will study whether there is indeed a linear relationship to the features and we will estimate the true slope of the line.

Item 2

Jake suggested that we use the Metropolis-Hastings method to generate normal random variables and to use those RVs in modelling log daily returns of a stock with a geometric random walk. Arthur suggested that we use a dataset of laptop computer features and prices and to build a neural network model to predict the price given the features. The algorithm used in training a neural network is gradient descent. We chose to go with the bootstrapping project because it seemed the most interesting and one of our friends had a dataset readily avaiable for us to use. Moreover, we are excited to learn bootstrapping in this class, so getting exposure to it now would be good.

Item 3

Arthur had a very nice, easy-to-read write up. He also participated in group discussions and made a suggestion for the final project. He also pointed out how to change one of my plots to make it look nicer. Jake had a very interesting method for his project and the write up made it clear how he used it. He also contributed an idea for the final project and pointed out that I could use my method for another RV generation.

Arthur Starodynov: 15/15 Jake Bentley: 15/15