# Tausworthe Generators

Colleen Morse, PT, DPT
Simulation and Modeling, ISYE 6644
May 5, 2021

## Abstract

The history of Tausworthe generators is explored and its mechanics are explained. The generator is reproduced using R-statistical software and its findings are presented with graphical representations and insights. A Tausworthe generator using parameters, $r = 3$, $q = 5$, and $l = 4$ was found to be lacking in uniformity given many gaps in the distribution. The generator with paramters $r = 3$, $q = 13$, and $l = 10$ was found to have a more uniform distribution and passed its chi-square test for uniformity at an alpha level of 0.05. The generator failed its independence tests as shown statistically and graphically. Finally, the variates generated were used to create Normally distributed variates as well as Exponentially distributed variates for use in a wide variaty of models.

# Background

Random numbers are essential when working with a wide variety of models, from finance to manufacturing to scheduling. Observing natural events or rolling a die provides a source of randomness, however these methods are difficult to reproduce and are very time consuming. One of the first algorithmic approaches to finding random variates was developed by John von Neumann. He used a "mid-squares" method involving taking the middle digits of a number, squaring those digits, and then taking the middle digits of that number for the next entry. This method has many shortcomings so it is not used much in practice. Many other generators have since come into practice, including one derived by Tausworthe in the 1960s. His generators used binary numbers to develop random numbers.

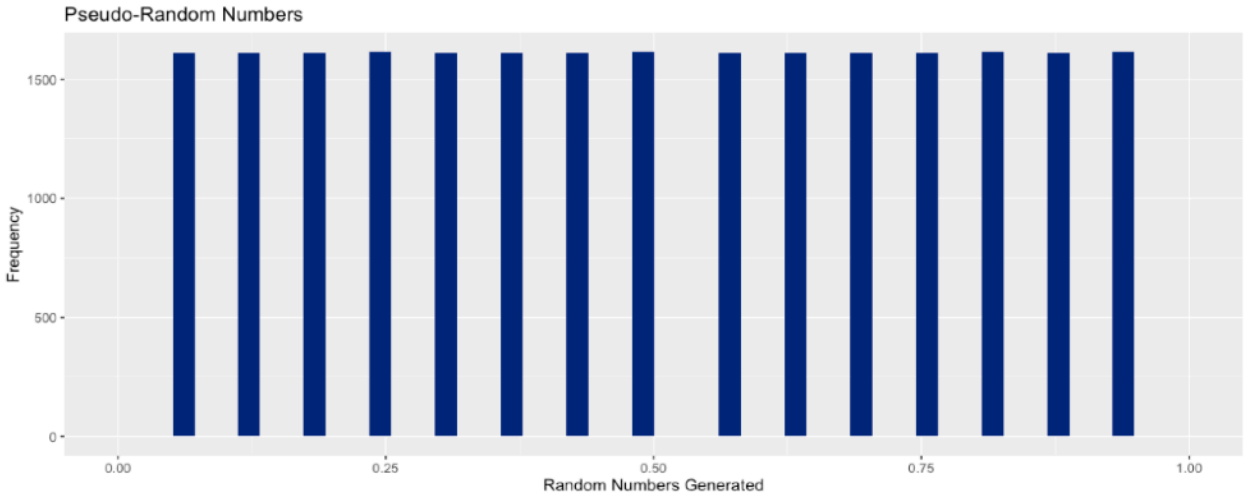
Tausworthe generators require an intial string of bits. First, values need to be chosen for '*r*' and '*q*', such that $0 < r < q$. The first '*q*' number of values in the string of bits are then set. To create the rest of the string of bits, subsequent values of 0 or 1 are chosen using 'exclusive-or' in comparing the $r^{th}$ prior bit to the $q^{th}$ prior bit. If one, and only one, of those two bits is the value '1', then the next bit in line is a 1; if the $r^{th}$ bit is equal to the $q^{th}$ bit then the next bit is a '0'. This pattern continues until the desired length of the bit string is reached.

Using that string of bits, the next step is to divide the string into groups of bits to form binary numbers, each of length '*l*'. Each of those are treated as a number in base 2, which can then be converted to a number in base 10. To obtain the random variate, that number in base 10 is divided by $2^l$ so all outputs exist in the range zero to one. These variates are thought to be uniform in their distribution, ready then to be used as is or converted into the distribution of choice.

# Main Findings

Tausworthe generators are thought to produce uniformly distributed random variates. True randomness can be discarded since these variates are predictable given specified parameters and an initial seed. Pseudo-randomness (or the appearance of randomness) and uniformity, however, remain in question.

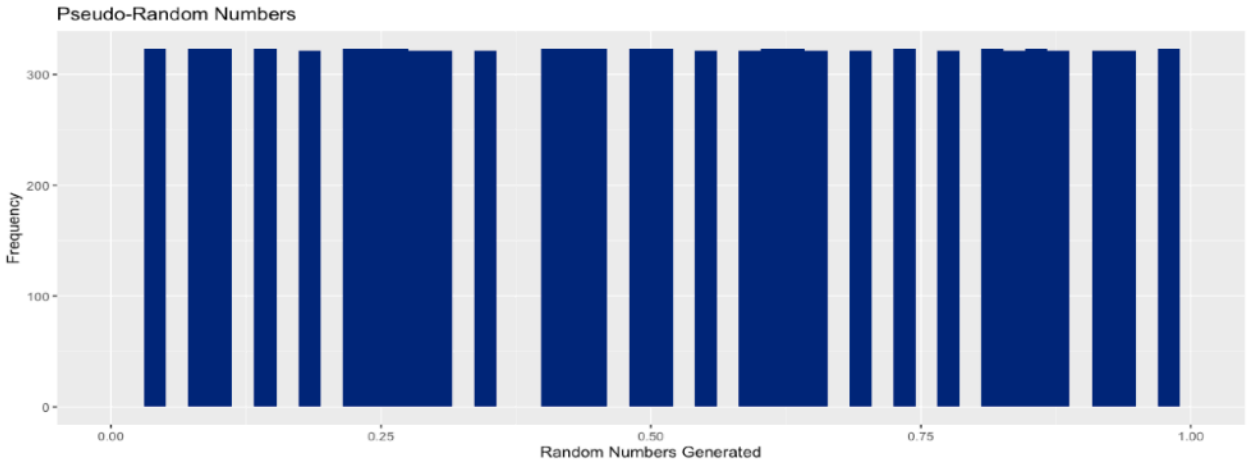
Analysis began with the example provided in the Law text[1]. The '*r*' and '*q*' parameters were set to $r = 3$ and $q = 5$ with a binary number length of $l = 4$. This model successfully generated variates, however a lot of repetition in the variates was observed, as well as frequent gaps between generated numbers, as can be seen in Figure 1: Attempt 1.



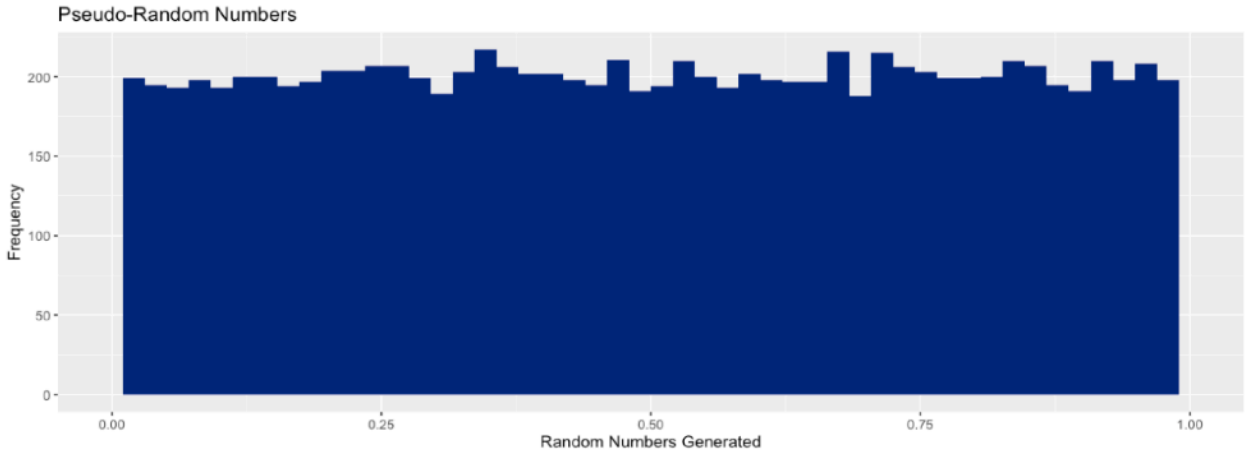

In an attempt to fill in the gaps, the parameters were adjusted to: first, allow for longer binary number lengths and second, to allow for longer periods. The thought process was that not enough binary numbers could be generated with the intial parameters. With longer binary numbers, more numbers could be created. With a longer period, more bits could be generated before their pattern repeats.

1. Law, A. M. (2015). Simulation Modeling and Analysis (5th ed.), pages 405-407. Columbus, OH: McGraw-Hill Education.

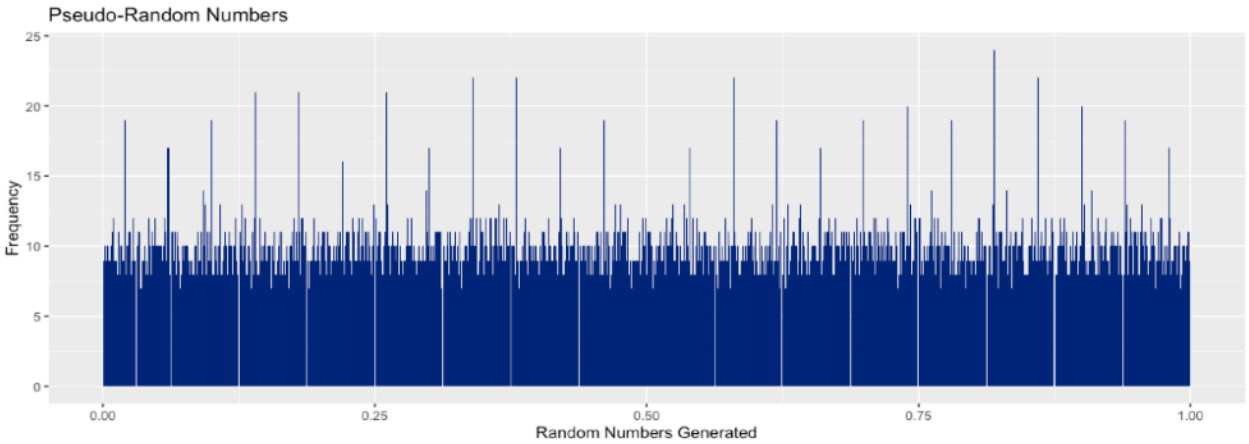
Adjusting the length of the binary numbers (the '*l*' parameter) to 10 (increased from 4), some of the gaps did start to fill in and the distribution started to look more uniform. However, frequent gaps persist in the data, as seen in Figure 2: Attempt 2.




Keeping the longer '*l*' and increasing the '*q*' parameter to 13 (from 5), resulted in a much more uniform-looking distribution of variates, as seen in Figure 3: Attempt 3.



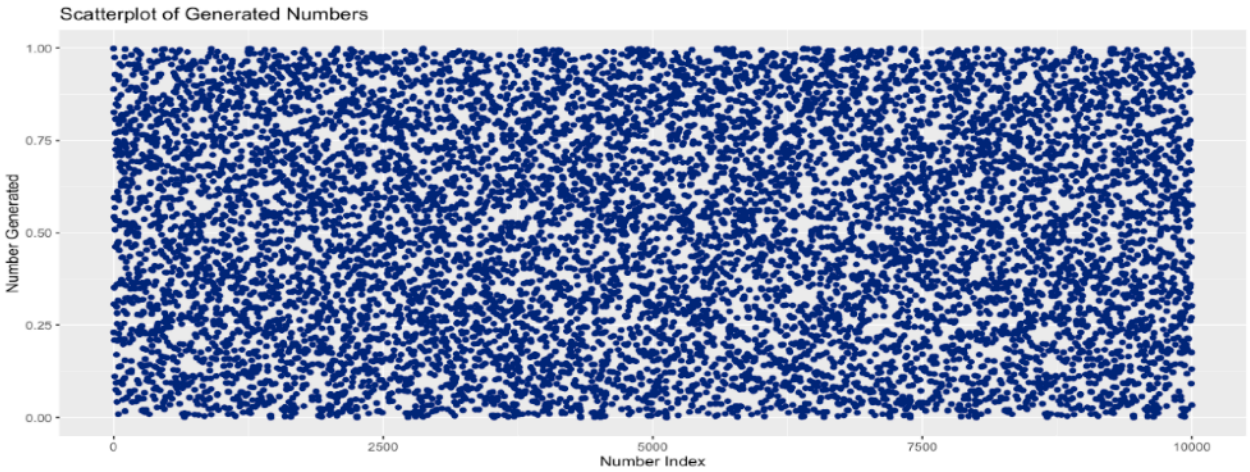

The uniformity falls into question when the values are observed more closely using smaller bin-widths, as seen in Figure 4: Attempt 3 closer look.



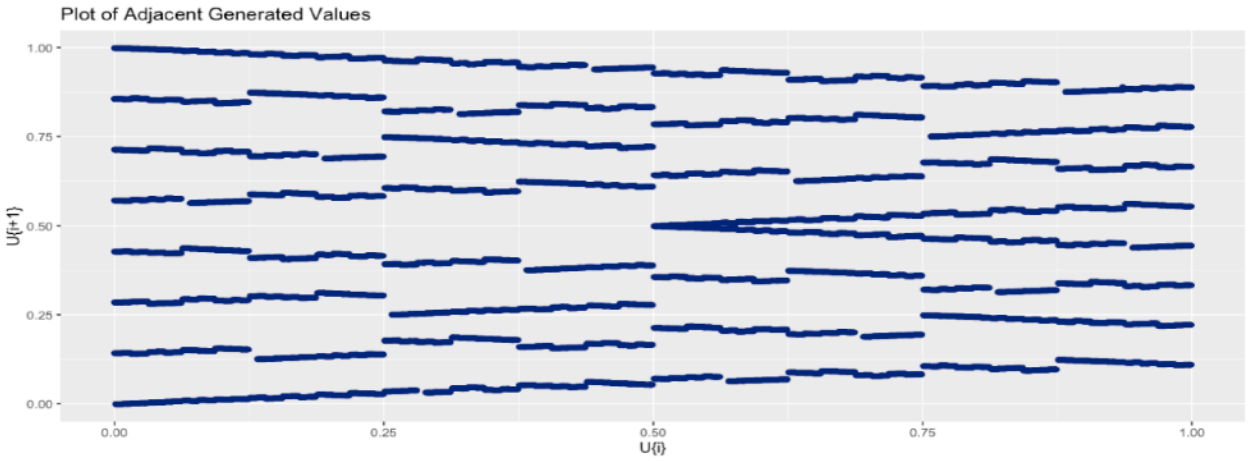

This histogram with shorter bin-widths shows occasional spikes and small gaps in the distribution.

Statistical testing was explored to determine uniformity and independence of the variates generated by this 3rd attempt (using $r = 3$, $q = 13$, and $l = 10$). First, a chi-squared test for distribution was completed using a bin width of 1/50 to match that of the intial histogram. Each bin was expected to contain 200 realizations. The genorator's chi-square value was found to be less than that of its test statistic at an alpha value of 0.05, suggesting this model plausibly follows a uniform distribution. The test was repeated using a bin width of 1/1000 to match that of the second histogram and again the chi-square value was found to be less than that of its test statistic, suggesting uniformity should not be rejected.

Independence tests were then completed and unfortunately the generator did not fare so well. Using the same alpha value, the genorator failed both the Runs Test and the Means Test for Independence. The values initially appear quite random, as seen in Figure 5: Scatterplot of Attempt 3.

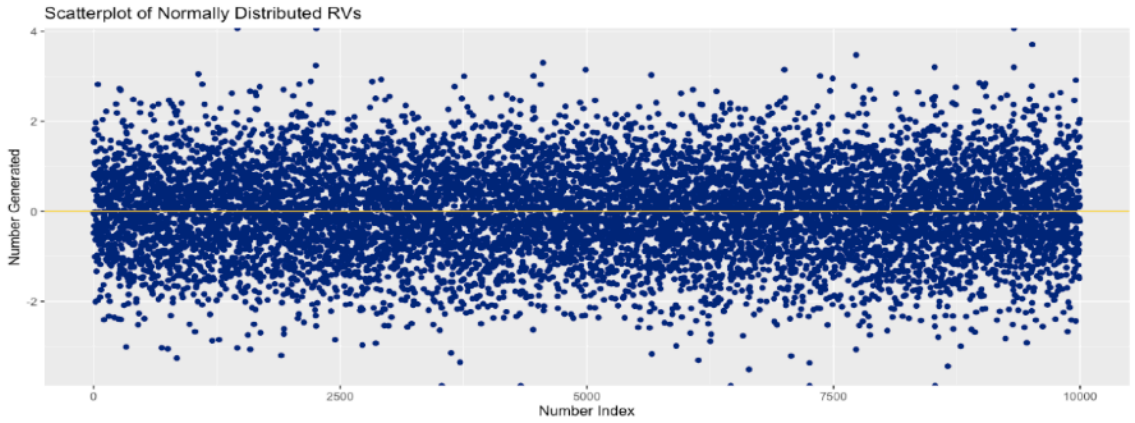



However, Figure 6: Adjacent Values From Attempt 3 shows a definite pattern emerging from the numbers generated (shown below).

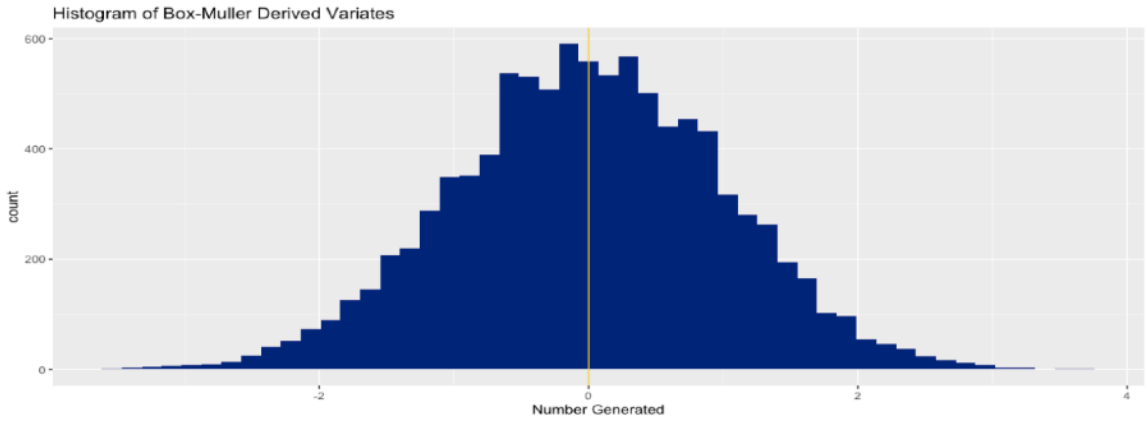



Despite the variates' dependence on each other, this generator may be useful for some models or for creating variates of other distributions. The variates were used to create normally distributed variates and exponentially distributed variates. The Box-Muller method was used for normal variate generation and the results can be seen in the scatterplot of Figure 7 and the histogram of Figure 8.

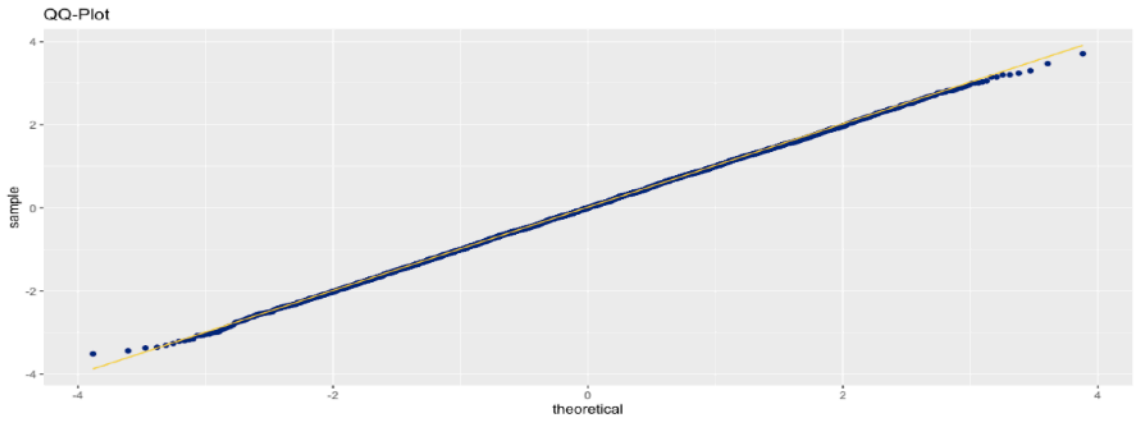
The scatterplot in Figure 7 shows the majority of points focused around 0 and approximately symmetrically distributed above and below zero with fewer numbers further away in each direction



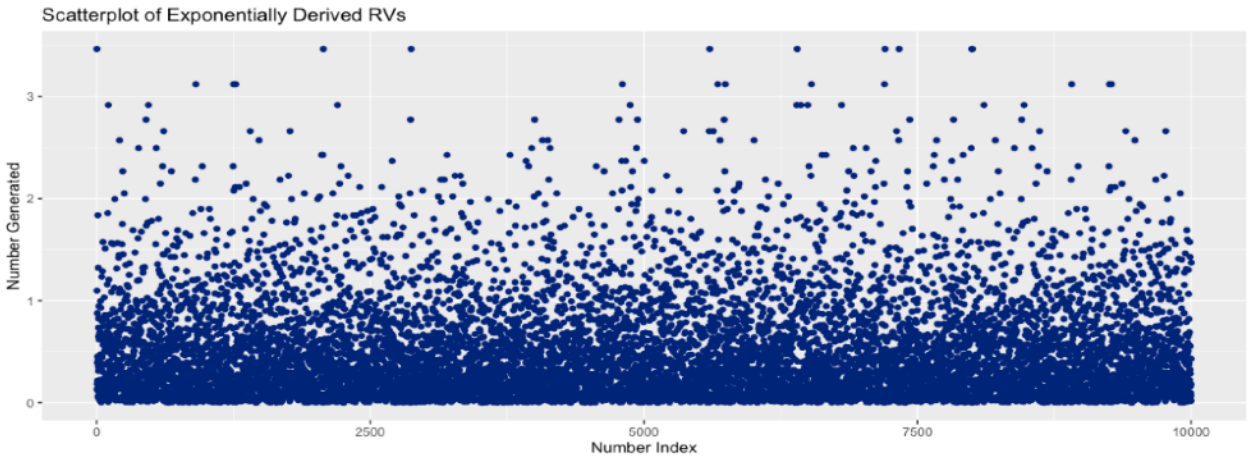

The histogram in Figure 8 shows no sign of skewdness and shows an approximately normal curve.



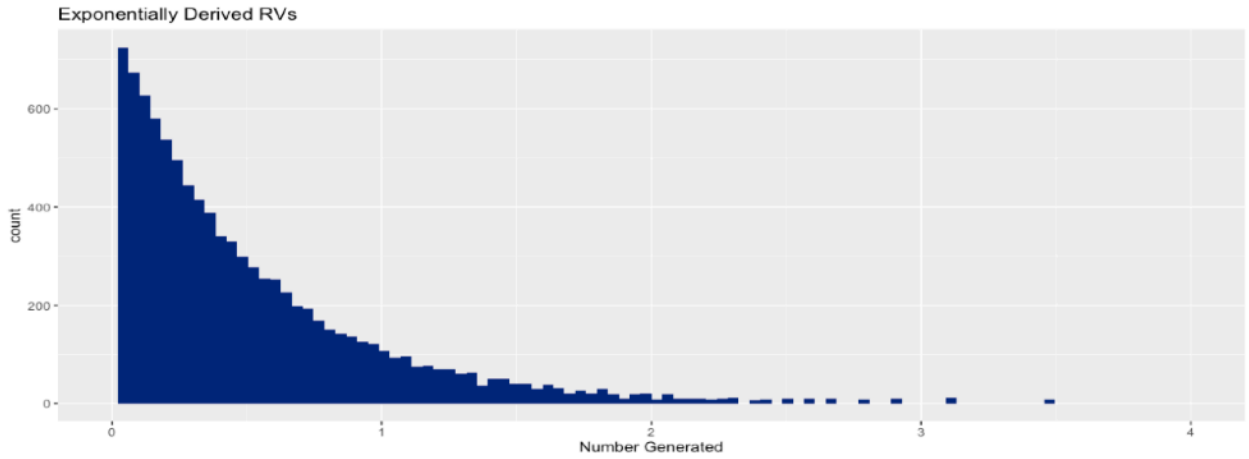

And finally, Figure 9 shows the QQ-Plot of the normal distributed variates with only slight deviation from normality at either tail.



Exponential variates were then generated from the Tausworthe generated variates for use in models requiring interarrival times.




As expected, no values fall below zero and less values are noticed as the values increase.




The distribution indeed appears to be exponential ($\lambda = 2$). These variates are ready to be used in models predicting customer arrivals while the normally distributed variates are ready to be used for the customers' heights, weights, or IQ levels, among other possibilities!

# Conclusion

While they may not be be most "random" of the pseudo-random number generator options, Tausworth generators have many benefits including potentially long periods, relatively fast calculations, and easy reproducibility. For models requiring a large amount of variates, very detailed variates, or variates that are independent from each other, this may not be the best generator to use. Once numbers are generated, they can then be converted into other distributions which may be useful in a wide variety of models.

# Appendix

```
library(tidyverse)
library(DescTools)
```

## Initial Set-up Modeled After Textbook Example

Law, A. M. (2015). Simulation Modeling and Analysis (5th ed.), pages 405-407. Columbus, OH: McGraw-Hill Education.

```
# Set desired length for binary string and initiate empty string
string_length <- 100000
b <- rep(0, string_length)

# Set r and q values
r <- 3
q <- 5

# Set initial seed
b[1:q] <- 1

# Create binary string
for (i in (q+1) : string_length){
  b[i] <- as.numeric((b[i-r] != b[i-q]))
}

# View period
b[1:(2^q)-1]
```

```
##  [1] 1 1 1 1 1 0 0 0 1 1 0 1 1 1 0 1 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0
```

```
# Set length of binary number
l <- 4

# Set up vector to hold binaries
bin_vec <- vector(mode = 'list', length = string_length/l)

# Create binary numbers of length 'l' from the binary string
for (i in 1: (string_length/l)){
  bin_vec[i] <- str_c(b[(i*4-3):(i*4)], collapse = '')
}

# Create dataframe to store binary numbers, their base-10 equivalents,
# and the resulting pseudorandom number they generate
numbers <- data.frame(cbind(base_2 = as.character(bin_vec)))
```

```r
numbers$base_10 <- BinToDec(as.numeric(numbers$base_2))
numbers$U <- numbers$base_10 / (2^1)

head(numbers)
```

```
##   base_2 base_10      U
## 1   1111      15 0.9375
## 2   1000       8 0.5000
## 3   1101      13 0.8125
## 4   1101      13 0.8125
## 5   0100       4 0.2500
## 6   0010       2 0.1250
```

```r
# Create histogram of pseudo-random number values
numbers %>% ggplot(aes(x = U)) +
  geom_histogram(bins = 50, fill = 'navy') +
  scale_x_continuous(limits = c(0, 1)) +
  labs(title = 'Pseudo-Random Numbers',
       x = 'Random Numbers Generated',
       y = 'Frequency')
```

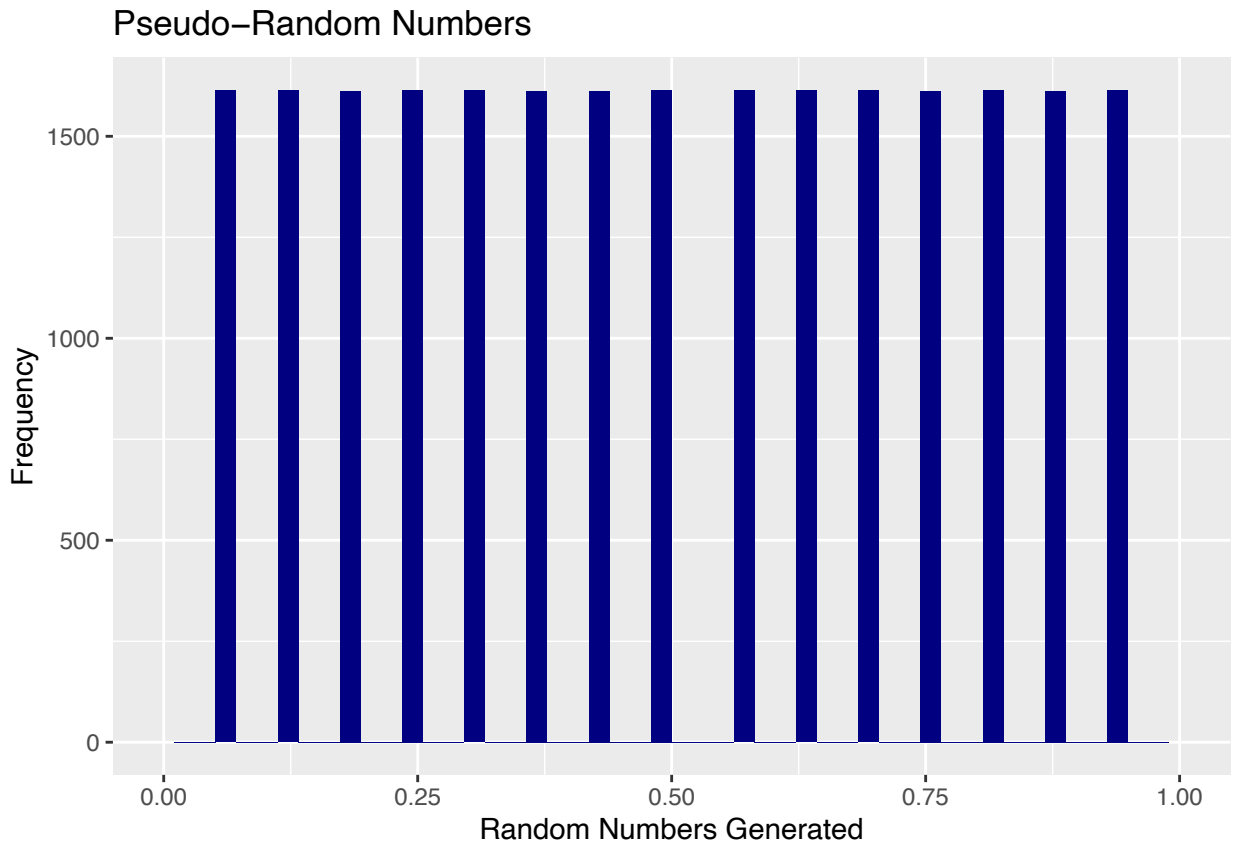



Figure 1: Attempt 1

The many gaps in data suggest this is not a uniform distribution.

## Change input values:

**Start with changing the length of the binary numbers, then change the length of the period if needed.**

```
# Initiate empty string of the same length as before
b <- rep(0, string_length)

# Set r and q values
r <- 3
q <- 5

# Set initial seed
b[1:q] <- 1

# Use XOR to build binary string
for (i in (q+1) : string_length){
  b[i] <- as.numeric((b[i-r] != b[i-q]))
}

# Set length of binary number
l <- 10

# Set up vector to hold binaries
bin_vec <- vector(mode = 'list', length = string_length/l)

# Create binary numbers of length 'l' from the binary string
for (i in 1: (string_length/l)){
  bin_vec[i] <- str_c(b[(i*l-(l-1)):(i*l)], collapse = '')
}

# Create dataframe to store binary numbers, their base-10 equivalents,
# and the resulting pseudorandom number they generate
numbers <- data.frame(cbind(base_2 = as.character(bin_vec)))
numbers$base_10 <- BinToDec(as.numeric(numbers$base_2))
numbers$U <- numbers$base_10 / (2^l)

head(numbers)
```

```
##       base_2 base_10          U
## 1 1111100011     995 0.97167969
## 2 0111010100     468 0.45703125
## 3 0010010110     150 0.14648438
## 4 0111110001     497 0.48535156
## 5 1011101010     746 0.72851562
## 6 0001001011      75 0.07324219
```

```
# Create histogram of pseudo-random number values
numbers %>% ggplot(aes(x = U)) +
  geom_histogram(bins = 50, fill = 'navy') +
  scale_x_continuous(limits = c(0, 1)) +
  labs(title = 'Pseudo-Random Numbers',
       x = 'Random Numbers Generated',
       y = 'Frequency')
```

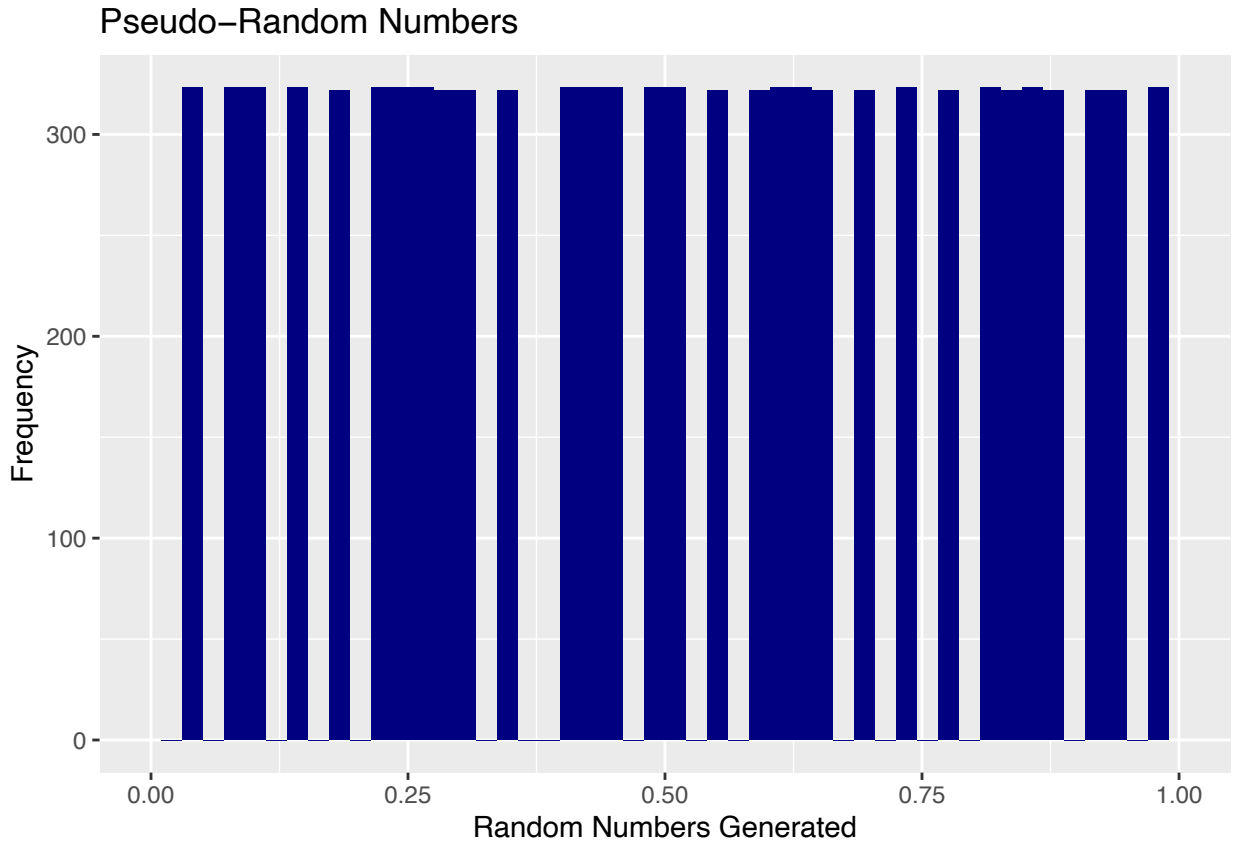



Figure 2: Attempt 2

Changing only the length of the binary number used for input, uniformity appears to have improved. However, continued gaps in the data suggest the output is still not uniformly distributed.

**Increase 'q' value for greater improvement**

```
# Initiate empty string at same length as before
b <- rep(0, string_length)

# Set r and q values
r <- 3
q <- 13

# Set initial seed
b[1:q] <- 1

# Use XOR to build binary string
for (i in (q+1) : string_length){
  b[i] <- as.numeric((b[i-r] != b[i-q]))
}

sprintf('The max period of bits is now %s', (2^q)-1)
```

```
## [1] "The max period of bits is now 8191"
```

```r
# Set length of binary number
l <- 10

# Set up vector to hold binaries
bin_vec <- vector(mode = 'list', length = string_length/l)

# Create binary numbers of length 'l' from the binary string
for (i in 1: (string_length/l)){
  bin_vec[i] <- str_c(b[(i*l-(l-1)):(i*l)], collapse = '')
}

# Create dataframe to store binary numbers, their base-10 equivalents,
# and the resulting pseudorandom number they generate
numbers <- data.frame(cbind(base_2 = as.character(bin_vec)))
numbers$base_10 <- BinToDec(as.numeric(numbers$base_2))
numbers$U <- numbers$base_10 / (2^l)

head(numbers)
```

```
##       base_2 base_10         U
## 1 1111111111    1023 0.9990234
## 2 1110001110     910 0.8886719
## 3 0011101100     236 0.2304688
## 4 0100111010     314 0.3066406
## 5 1101001110     846 0.8261719
## 6 1000101100     556 0.5429688
```

```r
# Create histogram of pseudo-random number values
numbers %>% ggplot(aes(x = U)) +
  geom_histogram(bins = 50, fill = 'navy') +
  scale_x_continuous(limits = c(0, 1)) +
  labs(title = 'Pseudo-Random Numbers',
       x = 'Random Numbers Generated',
       y = 'Frequency')
```

The output appears much more uniformly distributed when both the period and the number length are longer!

```r
numbers %>% ggplot(aes(x = U)) +
  geom_histogram(bins = 1000, fill = 'navy') +
  scale_x_continuous(limits = c(0, 1)) +
  labs(title = 'Pseudo-Random Numbers',
       x = 'Random Numbers Generated',
       y = 'Frequency')
```

However, as the bin-width decreases, varied counts and gaps in the output emerge, questioning uniformity.

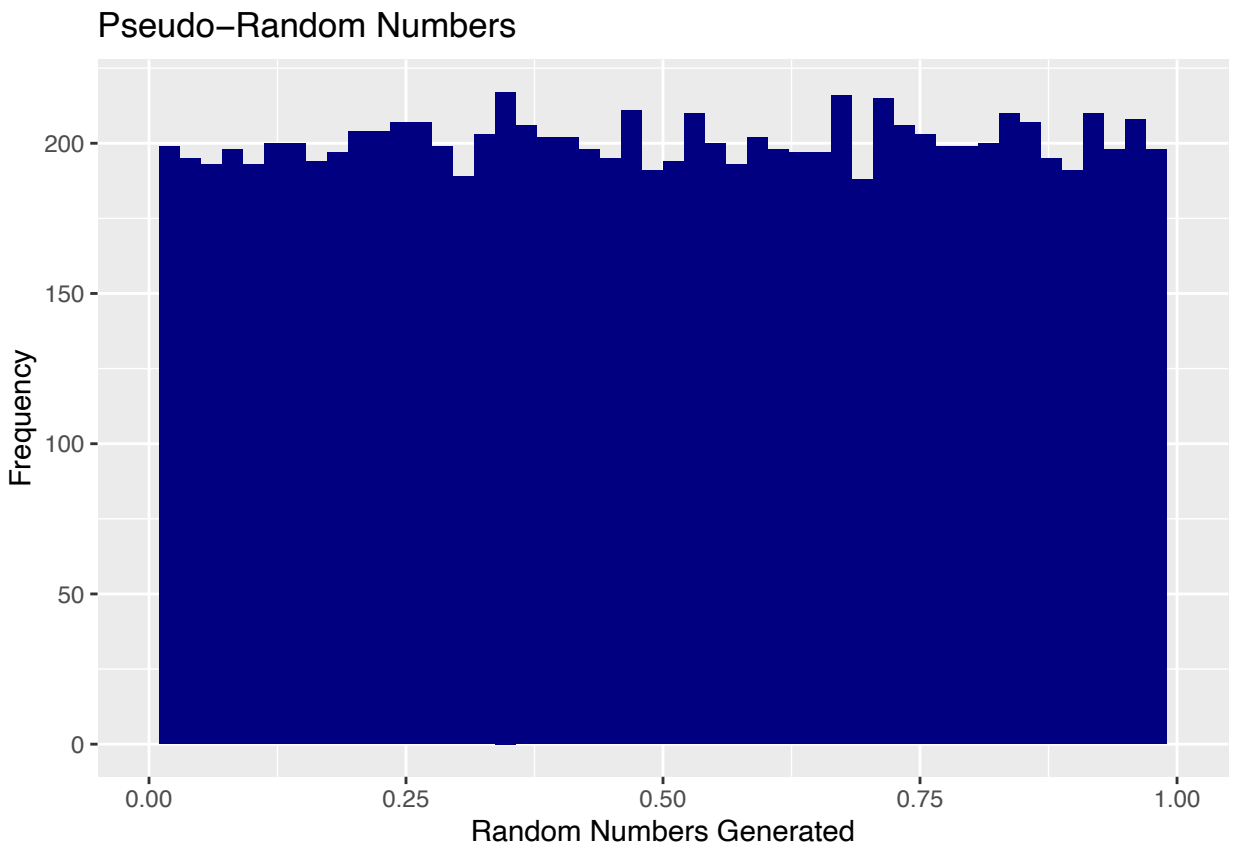



Figure 3: Attempt 3

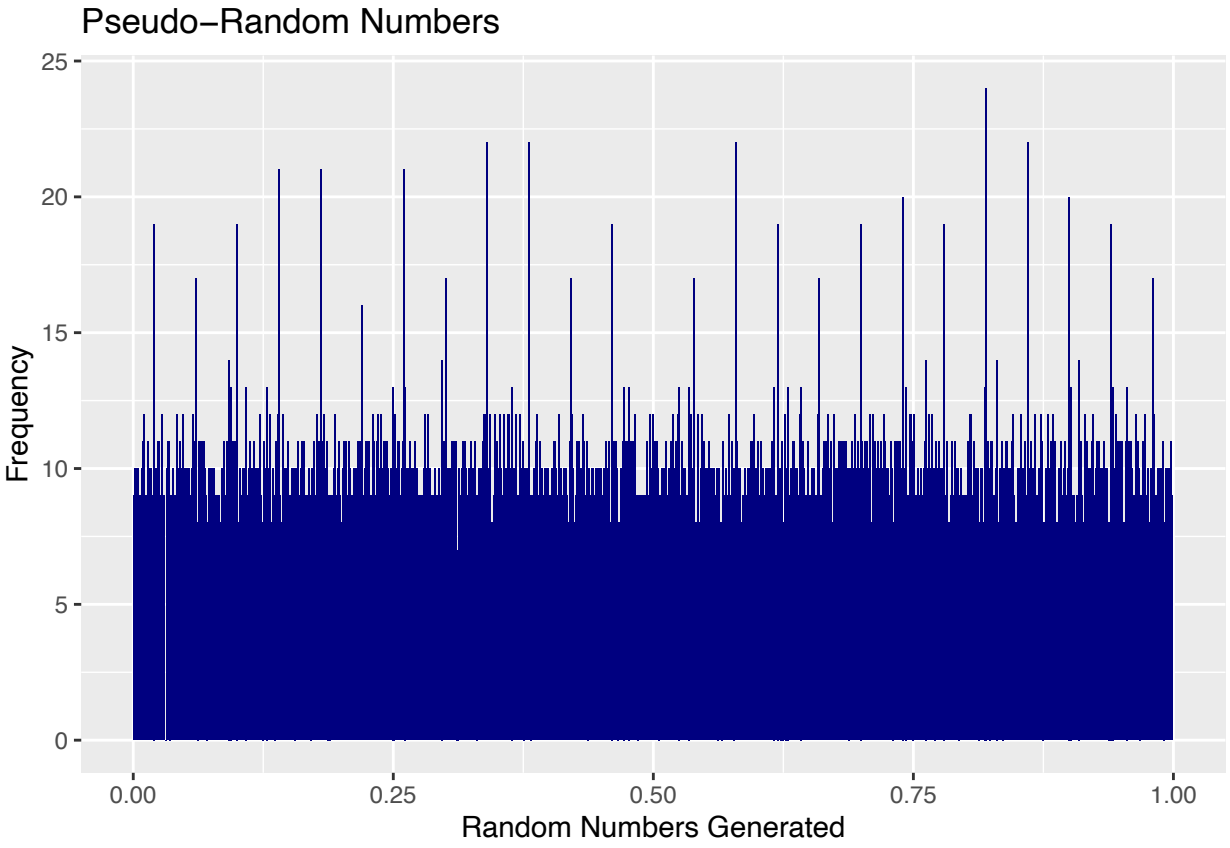



Figure 4: Attempt 3 closer look

## Statistical Testing

### Chi-Square Test for Uniformity

```
bins <- seq(0, 1, by = 1/50)   # create list of boundaries

bin_range <- cut(numbers$U, bins)

chi_table <- data.frame(transform(table(bin_range)))

chi_table$expected <- nrow(numbers) / 50

chi_table$chi_sq <- (chi_table$Freq - chi_table$expected)^2 / chi_table$expected

head(chi_table)
```

```
##     bin_range Freq expected chi_sq
## 1    (0,0.02]  190      200  0.500
## 2 (0.02,0.04]  183      200  1.445
## 3 (0.04,0.06]  209      200  0.405
## 4 (0.06,0.08]  180      200  2.000
## 5  (0.08,0.1]  203      200  0.045
## 6  (0.1,0.12]  190      200  0.500
```

```
alpha <- 0.05

chi_0 <- sum(chi_table$chi_sq)
chi_test <- qchisq(alpha, nrow(chi_table)-1)

chi_0 < chi_test
```

```
## [1] TRUE
```

The Tausworthe generator, using r=3, q=13, and l=10, passes the chi-squared test for uniformity using an alpha level of 0.05 and a bin-width of 0.02.

Re-test with smaller bin size:

```
bins <- seq(0, 1, by = 1/1000)   # create list of boundaries

bin_range <- cut(numbers$U, bins)

chi_table <- data.frame(transform(table(bin_range)))

chi_table$expected <- nrow(numbers) / 1000

chi_table$chi_sq <- (chi_table$Freq - chi_table$expected)^2 / chi_table$expected

alpha <- 0.05

chi_0 <- sum(chi_table$chi_sq)
```

```r
chi_test <- qchisq(alpha, nrow(chi_table)-1)

chi_0 < chi_test
```

```
## [1] TRUE
```

The chi-squared test for uniformity still passes when checking with the smaller bin size.

### Runs Test for Independence

```r
for (i in 2:(nrow(numbers))){
  numbers$run[i] <- ifelse(numbers$U[i-1] < numbers$U[i],
                           'INCREASE', 'DECREASE')
  numbers$change[i] <- ifelse(numbers$run[i-1] != numbers$run[i],
                              TRUE, FALSE)
}

head(numbers)
```

```
##       base_2 base_10         U      run change
## 1 1111111111    1023 0.9990234     <NA>     NA
## 2 1110001110     910 0.8886719 DECREASE     NA
## 3 0011101100     236 0.2304688 DECREASE  FALSE
## 4 0100111010     314 0.3066406 INCREASE   TRUE
## 5 1101001110     846 0.8261719 INCREASE  FALSE
## 6 1000101100     556 0.5429688 DECREASE   TRUE
```

```r
n <- nrow(numbers)

A0 <- sum(as.numeric(numbers$change),
          na.rm = TRUE) + 1              # +1: need to count the first "run"

A_exp <- (2*n - 1) / 3
A_var <- (16*n - 29) / 90

Z0 <- (A0 - A_exp) / sqrt(A_var)

Z_test <- qnorm(alpha/2)

abs(Z0) < abs(Z_test)
```

```
## [1] FALSE
```

The Tausworthe generator, using r=3, q=13, and l=10, fails the Runs test for Independence using an alpha level of 0.05.

### Means Test for Independence

```
numbers$above_mean <- ifelse(numbers$U > mean(numbers$U),
                             'above','below')

nabove <- sum(numbers$above_mean == 'above', na.rm = TRUE)
nbelow <- sum(numbers$above_mean == 'below', na.rm = TRUE)

B_exp <- ((2*nabove*nbelow) / n) + (1/2)
B_var <- ((2*nabove*nbelow)*(2*nabove*nbelow - n)) / (n^2*(n-1))

Z0 <- (nabove - B_exp) / sqrt(B_var)

abs(Z0) < abs(Z_test)
```

```
## [1] FALSE
```

This generator also fails the Means test for Independence using an alpha level of 0.05.

**Plotting the numbers generated:**

```
numbers %>% ggplot(aes(x = seq(1, n), y = U)) +
  geom_point(color = 'navy') +
  labs(title = 'Scatterplot of Generated Numbers',
       x = 'Number Index',
       y = 'Number Generated')
```

The overall scatterplot appears random. No clear patterns are apparent.

**Plotting adjacent generated numbers:**

```
for (i in 1:(n-1)){
 numbers$y[i] <- numbers$U[i+1]
}

numbers %>% ggplot(aes(x = U, y = y)) +
  geom_point(color = 'navy') +
  labs(title = 'Plot of Adjacent Generated Values',
       x = 'U{i}',
       y = 'U{i+1}')
```

Definite patterns emerge when plotting adjacent values! This is expected given the failed independence tests.

**Generate Normals**

```
z1 <- rep(0, n/2)
z2 <- rep(0, n/2)
for (i in 1:(n/2)){
```

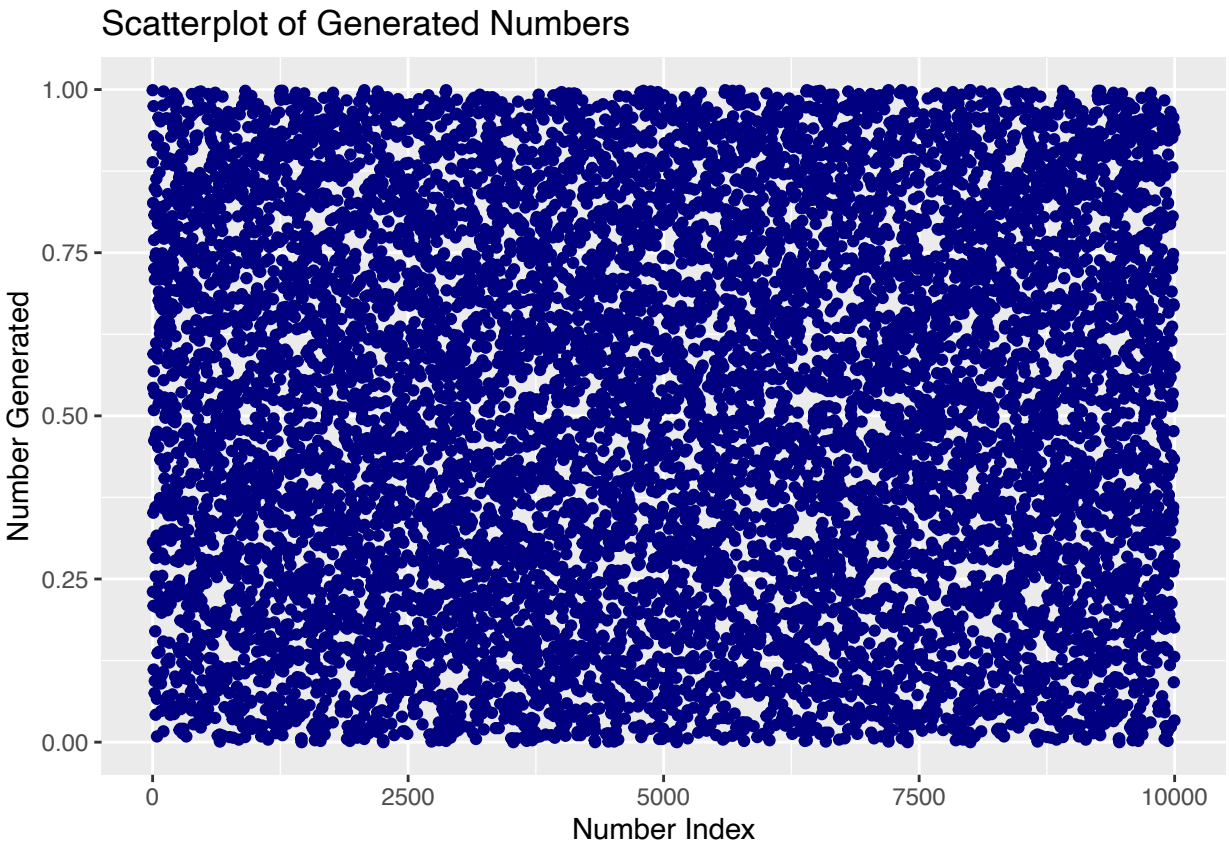



Figure 5: Scatterplot of Attempt 3

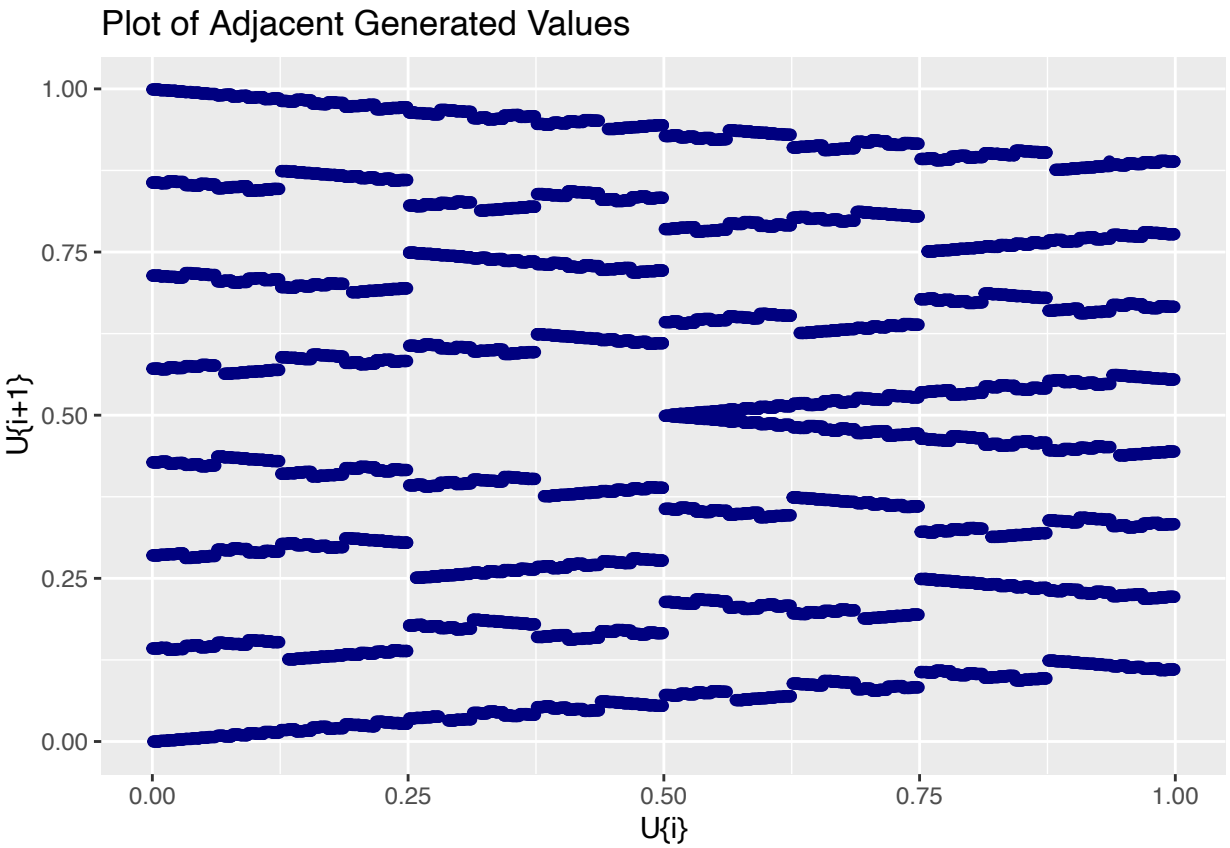



Figure 6: Adjacent Values From Attempt 3

```
  u1 <- numbers$U[i]
  u2 <- numbers$U[(n/2)+i]
  z1[i] <- sqrt(-2*log(u1))*cos(2*pi*u2)
  z2[i] <- sqrt(-2*log(u1))*sin(2*pi*u2)
}

numbers$z <- c(z1, z2)

numbers %>% ggplot(aes(x = seq(1, n), y = z)) +
  geom_point(color = 'navy') +
  geom_hline(yintercept = 0, color = 'gold') +
  labs(title = 'Scatterplot of Normally Distributed RVs',
       x = 'Number Index',
       y = 'Number Generated')
```

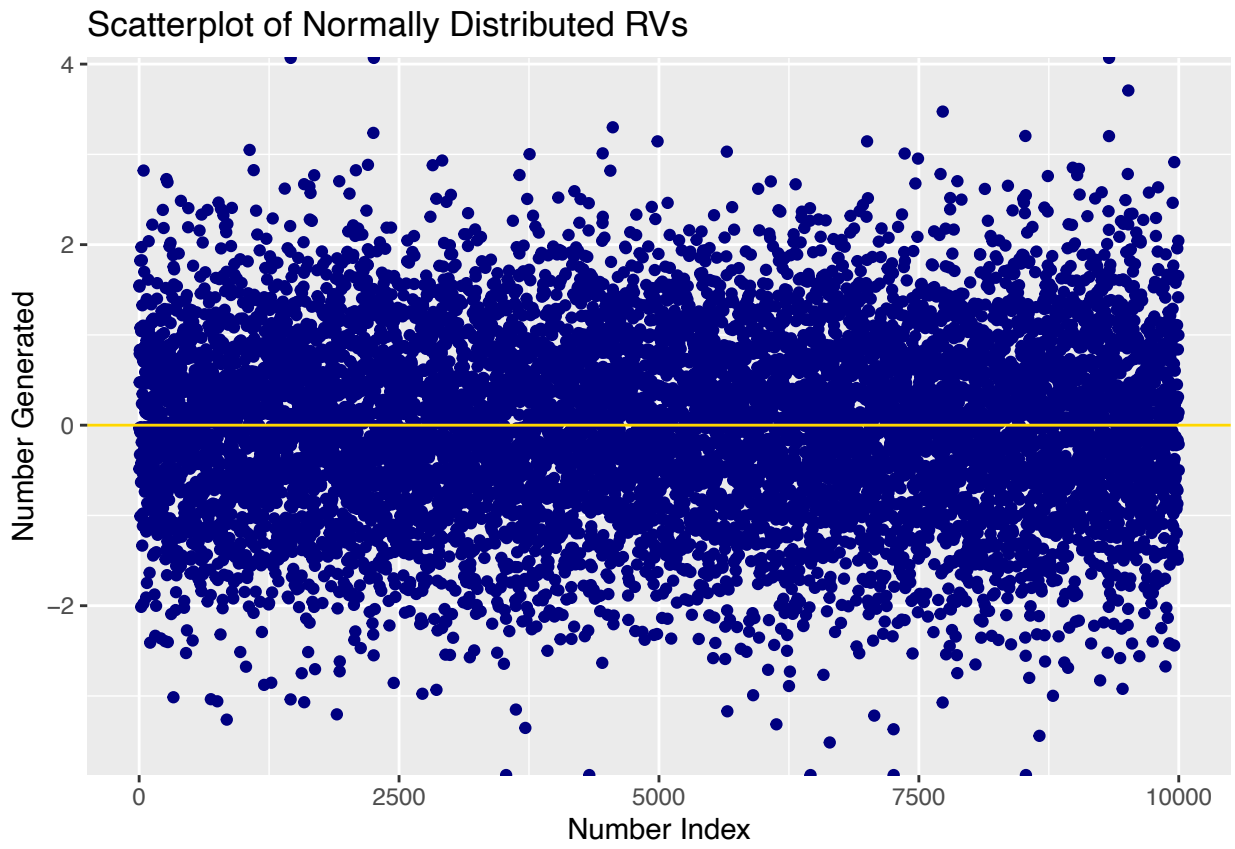



Figure 7: Scatterplot of Normals Generated From Attempt 3

Variates appear randomly scattered and closer to the expected value of 0 than the extreme values (> abs(2)).

```
# Generate a histogram for the normal variates
numbers %>% ggplot(aes(z)) +
  geom_histogram(fill = 'navy', bins = 50) +
  geom_vline(xintercept = 0, color = 'gold') +
  labs(title = 'Histogram of Normal-Derived Variates',
       x = 'Number Generated')
```

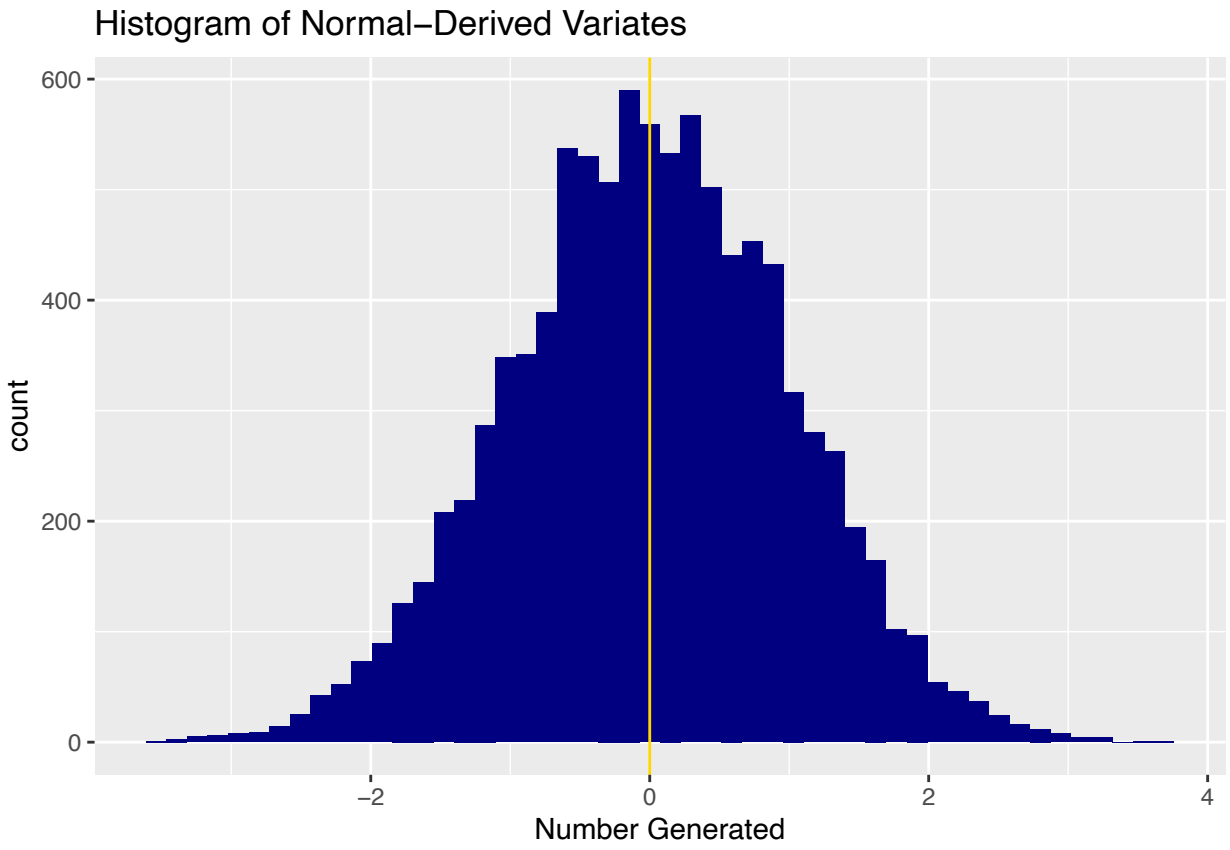



Figure 8: Histogram of Normals Generated From Attempt 3

No clear skewdness is present. The curve appears approximately normal.

```
# View QQ-plot for normal variates
numbers %>% ggplot(aes(sample = z)) +
  stat_qq(color = 'navy') +
  stat_qq_line(color = 'gold') +
  labs(title = 'QQ-Plot')
```

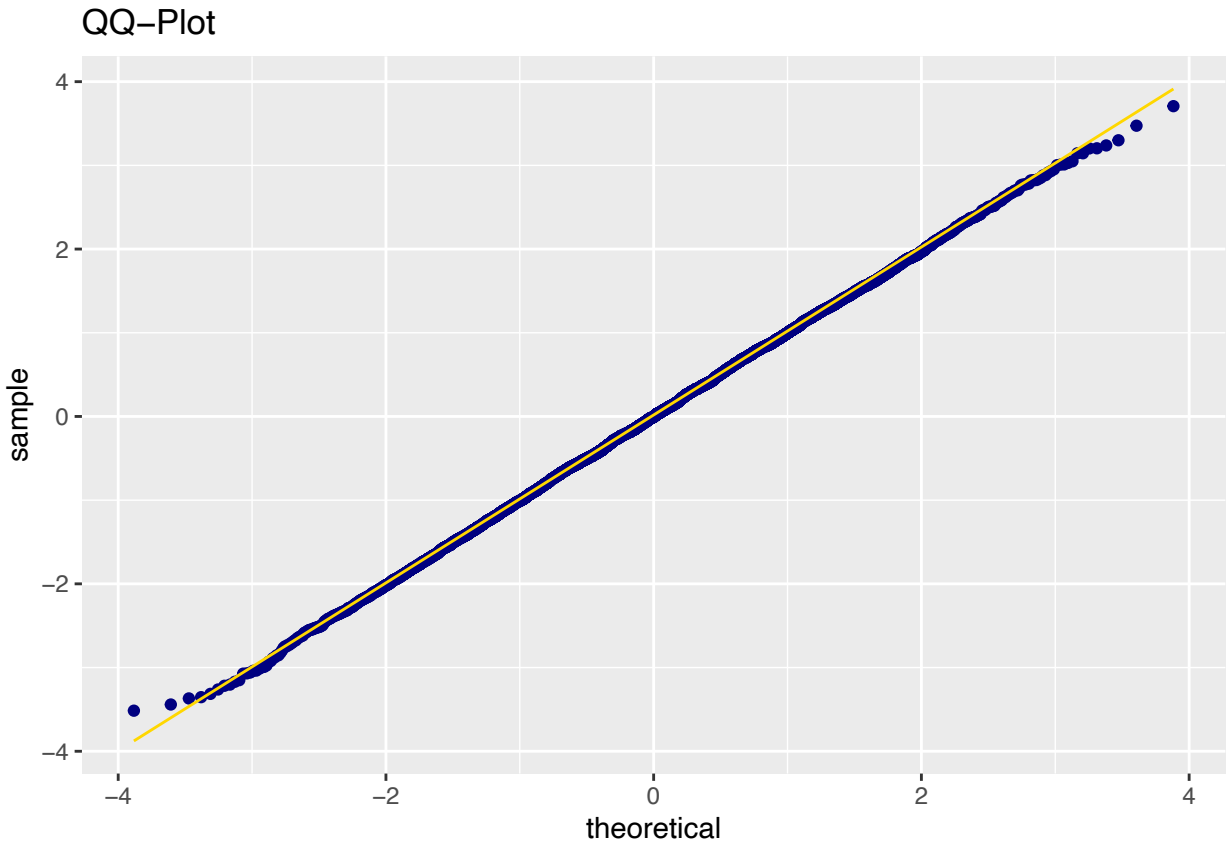



Figure 9: QQ-Plot of Normals Generated From Attempt 3

Some deviation from normality is seen at both tails, however overall, these variates appear to closely follow the normal distribution.

### Generating Exponentials

```
lambda <- 2

for (i in 1:n){
  numbers$exp[i] <- -1/2*log(1 - numbers$U[i])
}

numbers %>% ggplot(aes(x = seq(1, n), y = exp)) +
  geom_point(color = 'navy') +
  labs(title = 'Scatterplot of Exponentially Derived RVs',
```

```
       x = 'Number Index',
       y = 'Number Generated')
```

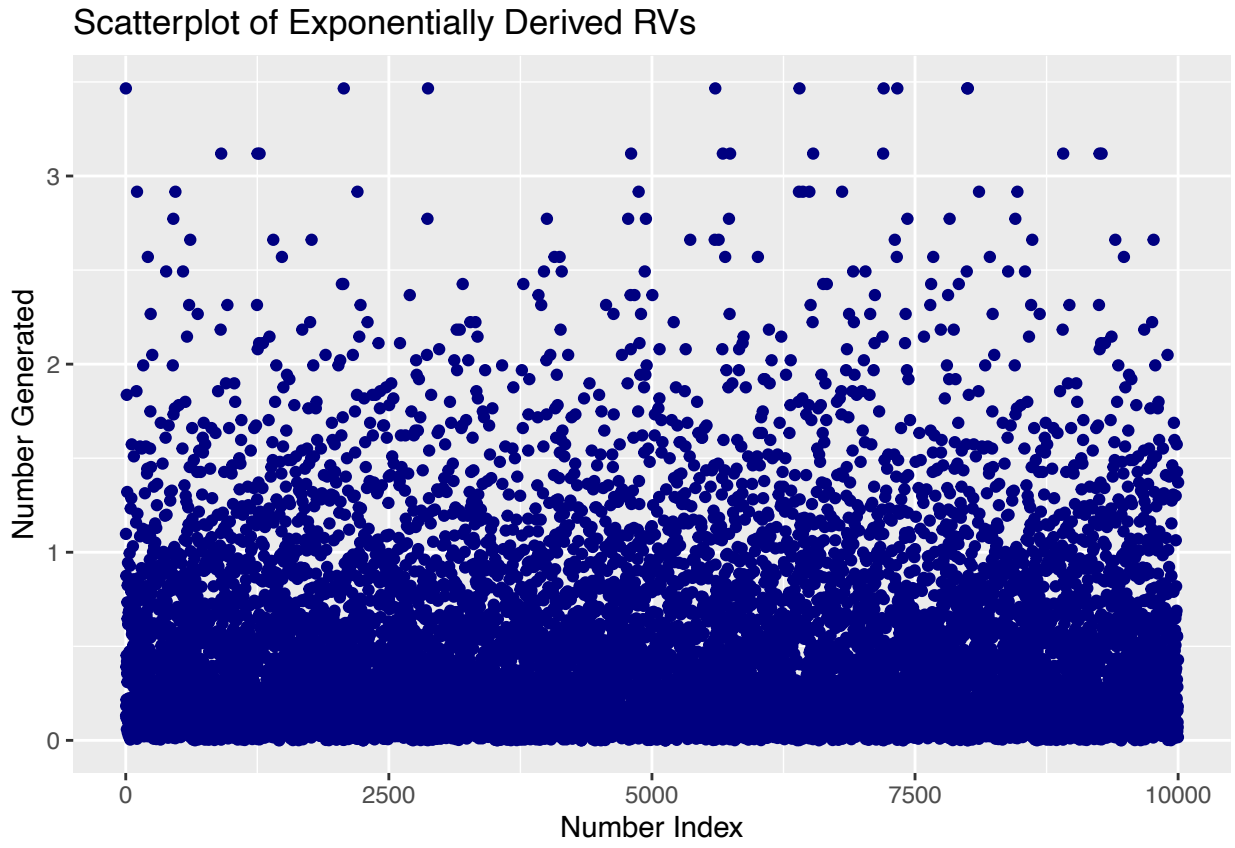



Figure 10: Scatterplot of Exponentials Generated From Attempt 3

```
numbers %>% ggplot(aes(exp)) +
  geom_histogram(fill = 'navy', bins = 100) +
  scale_x_continuous(limits = c(0, 4)) +
  labs(title = 'Exponentially Derived RVs',
       x = 'Number Generated')
```

The exponential variates appear exponential, indeed. These can now be used as interarrival times when generating customers in a model.

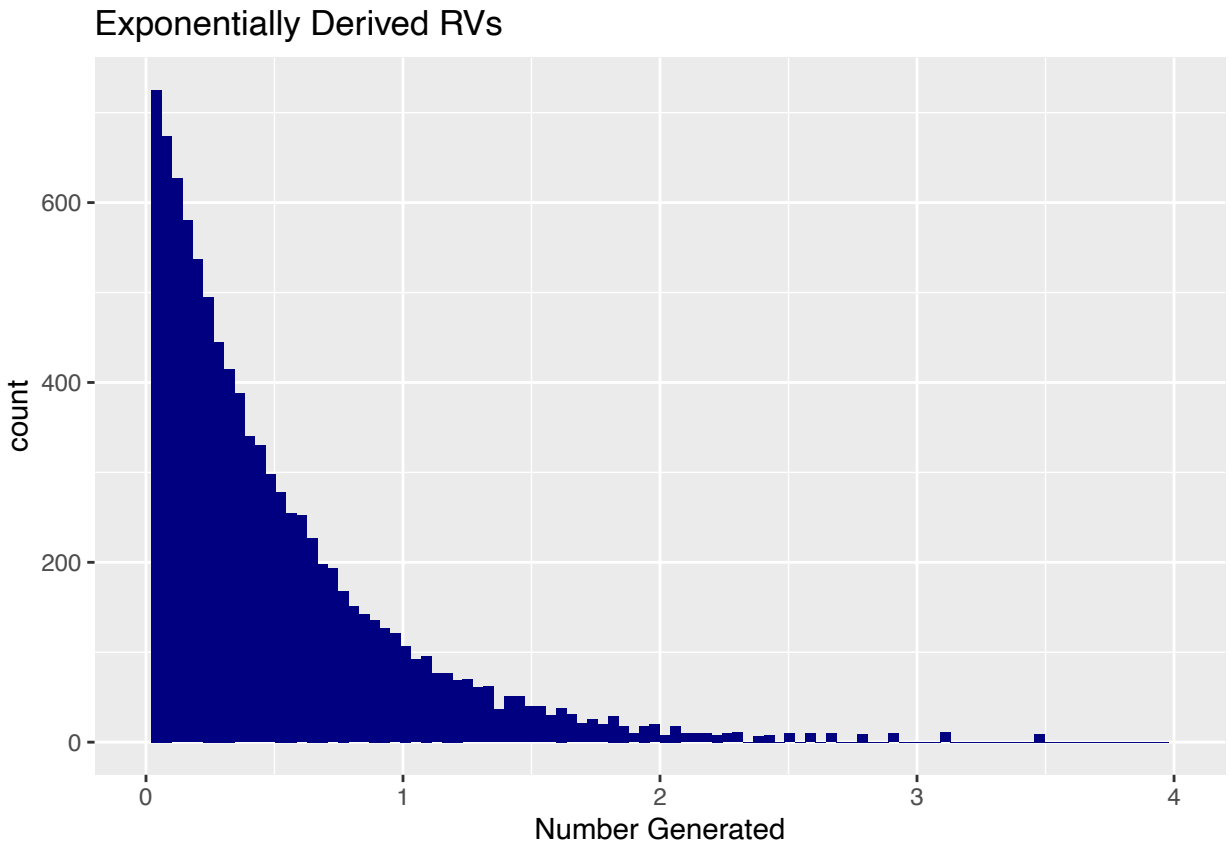



Figure 11: Histogram of Exponentials from Attempt 3