Midterm Project

Ethan Alteza, Barron Bronson, Holden Ellis, Charlotte Huang

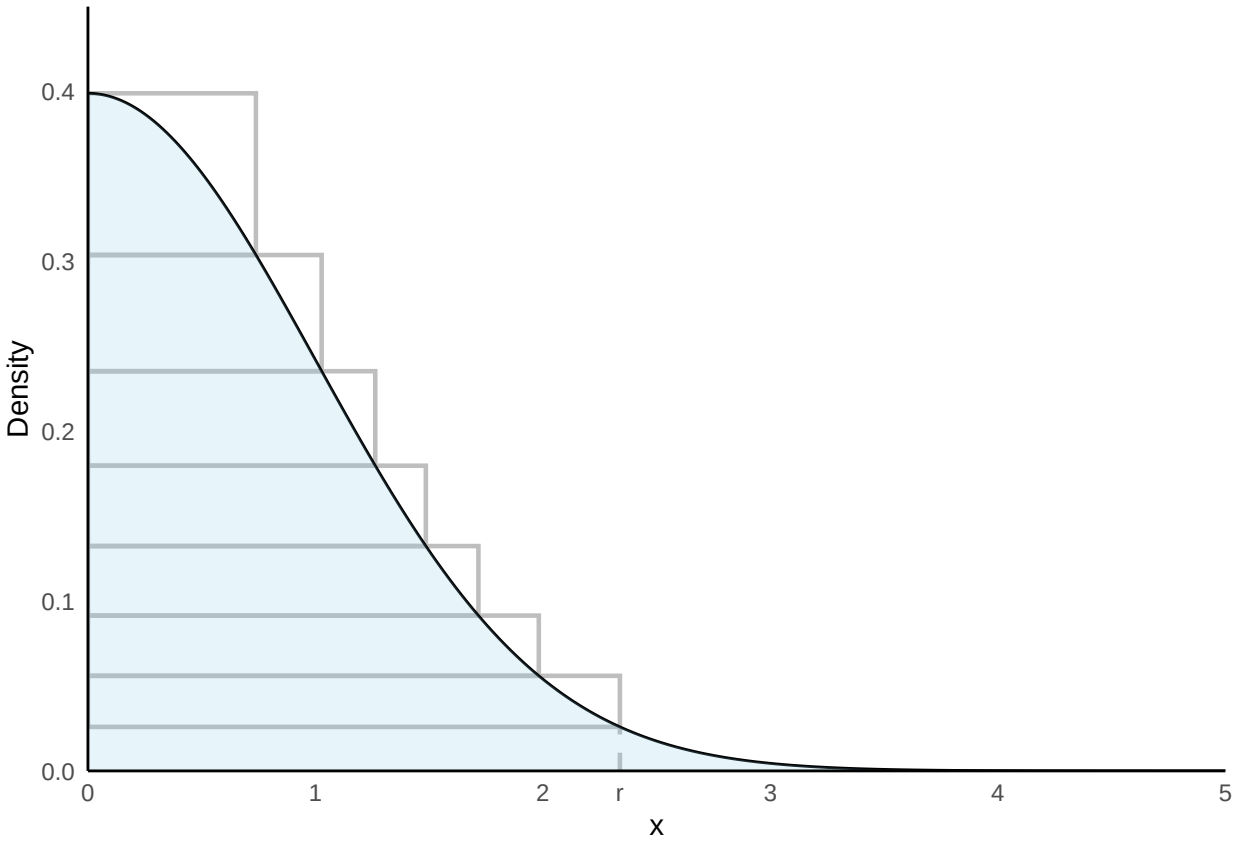
2026-02-09

The Ziggurat Method for Random Number Generation

https://docs.google.com/document/d/1HZTEfuxbhKEsMSGoFqDvulxrU5hy9rIVp31Wf9el38g/edit?usp=sharing

```
source("Ziggurat.R")
source("BoxMuller.R") # if anyone wants to compare, much faster because of how R runs code
```

The Ziggurat Method for Generating Random Variables (2000) by Marsaglia and Tsang outlines a strategy for sampling from probability distributions that are either decreasing or symmetric and unimodal. The method itself was first published by the same authors in 1984, but was made more efficient with this publication. The namesake comes from the Ziggurat structures built in ancient Mesopotamia; these massive, rectangular pyramid structures were temple towers that featured several steps. The Ziggurat method functions by dividing the density into 256 regions of equal area (v). For all regions besides the base layer, the regions are in the shape of a rectangle, but the lowest region includes both a rectangular component and the tail beyond the cutoff, which is sampled using a specialized accept–reject method. These rectangles are mostly encapsulated under the curve, only a small portion of the right edge sticks out of the density.



Sampling is done by picking a random level of the ziggurat and then drawing from a uniform within it. The optimization here is that most of these points (over 99%) fall under the rectangle above and we are able to immediately accept them by a simple check, without ever computing the density function. This paper is quite significant to RNG, since the Ziggurat method became the standard technique in sampling for many distributions; it also is one of the fastest general methods. In NumPy, one of the most popular Python packages, generators for the Normal, Exponential, and Gamma distributions now utilize the Ziggurat method as of 2019. These generators are 2-10 times faster than the original implementation, which goes to show how efficient this method is.

Our Application

We implemented two Ziggurat R functions, one for the Normal distribution and one for the Exponential distribution: rnormzig and rexpzig. Separate functions need to be written for each distribution for two reasons. First, the most effective method for drawing from the tail distribution is unique. For the normal, Marsaglia recommends

$$X = -ln(U_1)/r, Y = -ln(U_2)U_1, U_2 \sim \text{Uniform}(0,1) \text{if } 2Y > X^2, Z_{tail} = r + X$$

and for the exponential he recommends

$$X = r - log(U), U \sim \text{Uniform}(0,1)$$

Second, the Normal distribution is symmetric about zero, whereas the Exponential distribution is not. After generating a positive Normal x value using the Ziggurat construction, symmetry is enforced by randomly assigning a sign to that value. This step is not necessary when sampling from the Exponential distribution because the distribution is defined only on $[0, infty)$. To draw from the rectangles, the code just requires the pdf and the inverse pdf for the distribution. This is done by first computing the breakpoints of each

rectangle, which is handled by the function zigtable(). This is done by choosing a random value,r, in the domain such that

$$v = rf(r) + \int_r^{\text{inf}} f(x)dx$$

r becomes x_n and x_{n-1} is generated iteratively by

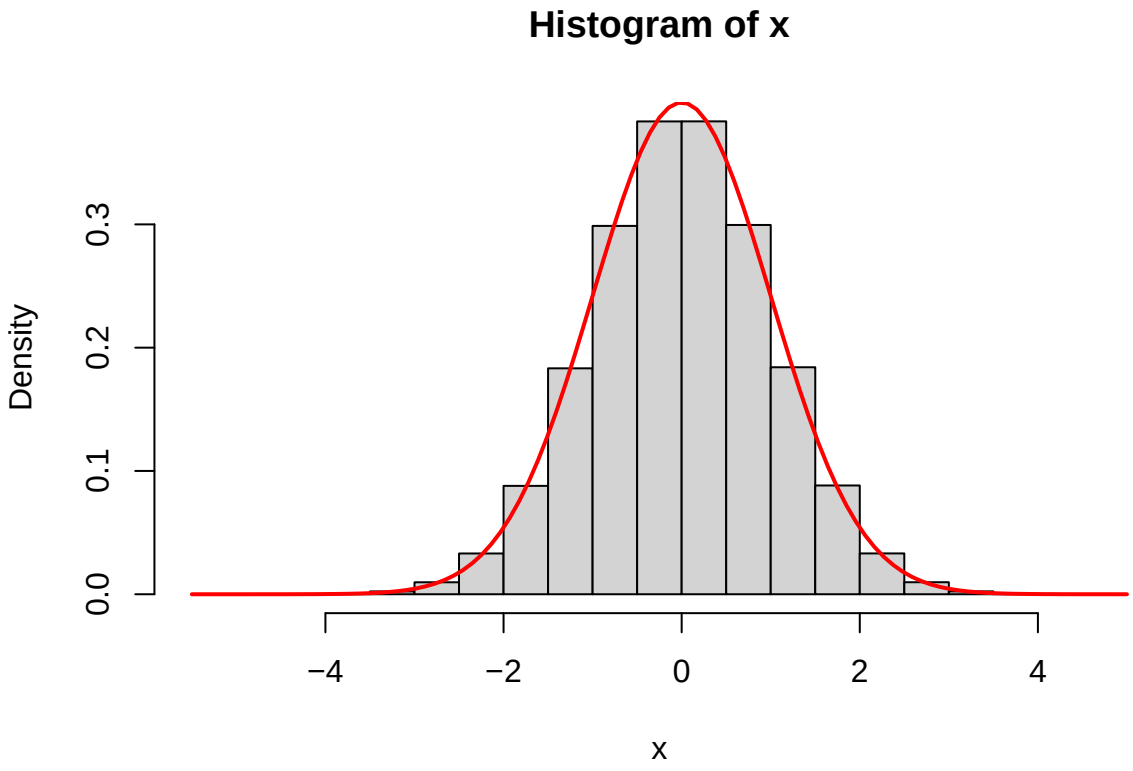
$$x_{i-1} = f^{-1}(\frac{v}{x_i} + f(x_i))$$

This process repeats until $x_0 = 0$. The challenge here is choosing the optimal v so that the iterative process doesn't overshoot or undershoot. This is done R's `uniroot` function Once the breakpoints are computed, it is easy to choose a random area and draw from the uniform within. First, to choose what rectangle we are sampling from, we randomly sample an integer between 2 and 257 (first index is the tail region) from the discrete uniform distribution using `sample()`. `sample()` is relatively slow compared with C implementations, where the index can be extracted cheaply from a single 32-bit random integer using the last 9 bits. After obtaining the rectangle we are sampling from, we check if the rectangle is the base layer. If the rectangle selected is the base layer then we randomly choose a x-value in the using `runif()` and check if $x_i < x_{i-1}$ or that the x-value is in the fast-accept region. If the x-value is not in the fast-accept region the code switches to the distribution-specific tail generator (defined for exponential and normal above) rather than using the rectangular accept/reject logic. The process is the same if the chosen rectangle is not the bottom layer, but if the x-value is not in the fast-accept region we generate an additional uniform height and accept the point (x, y) only if it lies under the pdf curve.

```
zigtable(function(x) { exp(-x) }, function(y) { -log(y) },8)
```

```
## $x
## [1] 0.0000000 0.4338930 0.7888599 1.1387148 1.5163812 1.9560291 2.5166676
## [8] 3.3489677
##
## $v
## [1] 0.1527383
```

```
set.seed(292026)
x <- rnormzig(1000000)
hist(x, freq=FALSE)
curve(dnorm(x), add=TRUE, col="red", lwd=2) # looks good
```




```
set.seed(777)

#testing: alpha = 0.01
samples1 <- rnormzig(4999)
print("4999 Samples")
```

```
## [1] "4999 Samples"
```

```
shapiro.test(samples1) # no evidence against normal; p-value > alpha = 0.01
```

```
##
##  Shapiro-Wilk normality test
##
## data:  samples1
## W = 0.99954, p-value = 0.2852
```

```
ks.test(samples1, "pnorm", mean=mean(samples1), sd=sd(samples1)) # same
```

```
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  samples1
## D = 0.0083523, p-value = 0.8766
## alternative hypothesis: two-sided
```

```
print("1,000,000 ziggurat samples")
```

```
## [1] "1,000,000 ziggurat samples"
```

```r
# testing 1 mil samples...
ks.test(x, "pnorm", mean=mean(x), sd=sd(x)) # passed
```

```
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  x
## D = 0.00067577, p-value = 0.7511
## alternative hypothesis: two-sided
```

```r
mean(x) # close to 0
```

```
## [1] 0.001020289
```

```r
sd(x)   # close to 1
```

```
## [1] 1.000076
```

```r
print("1,000,000 rnorm samples")
```

```
## [1] "1,000,000 rnorm samples"
```

```r
# additional test:
x2 <- rnorm(1000000)
ks.test(x2, "pnorm", mean=mean(x2), sd=sd(x2)) # passed
```

```
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  x2
## D = 0.00069352, p-value = 0.722
## alternative hypothesis: two-sided
```

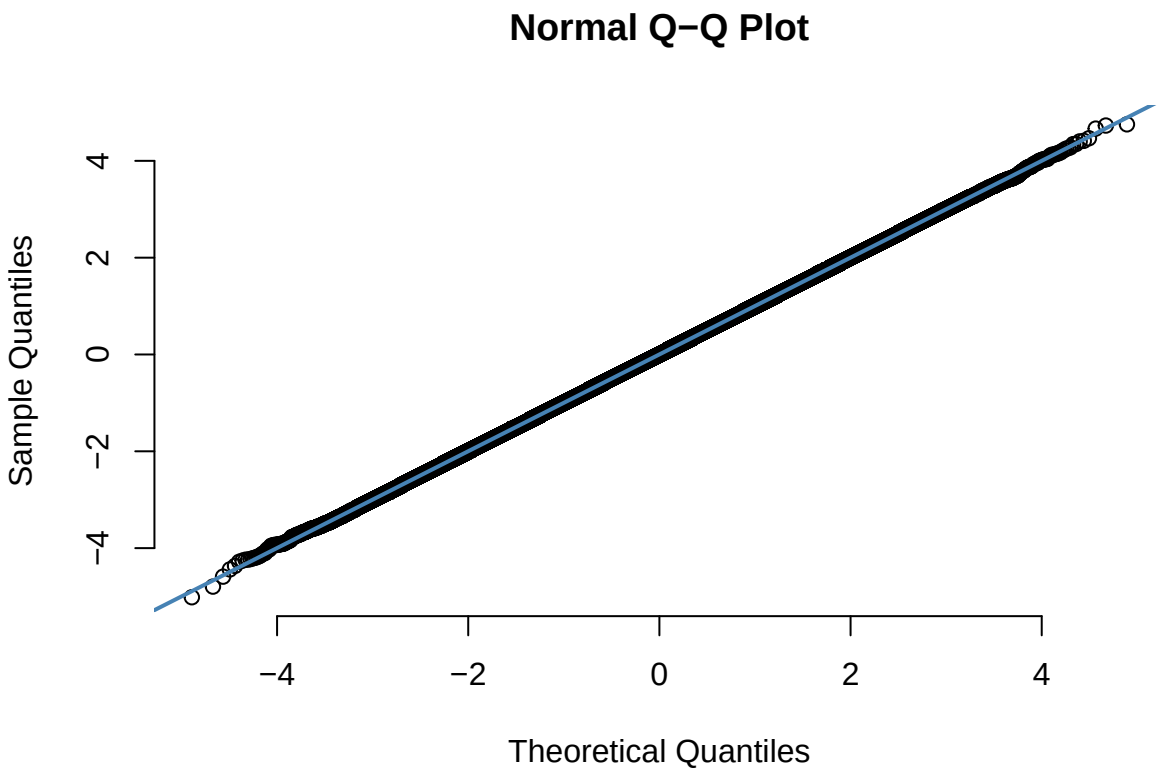
```r
mean(x2)  # close to 0
```

```
## [1] 0.000669734
```

```r
sd(x2)    # close to 1
```

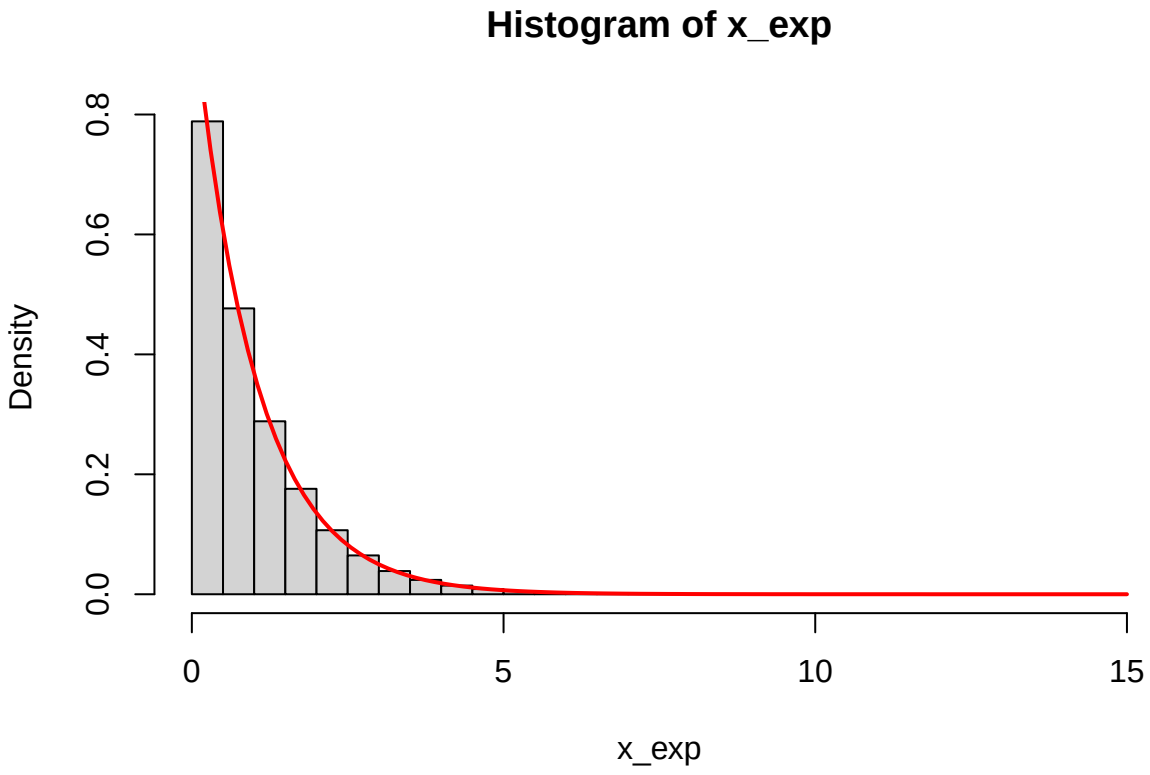
```
## [1] 1.001078
```

```r
# QQ plot
qqnorm(x, pch = 1, frame = FALSE)
qqline(x, col = "steelblue", lwd = 2) # basically normal
```

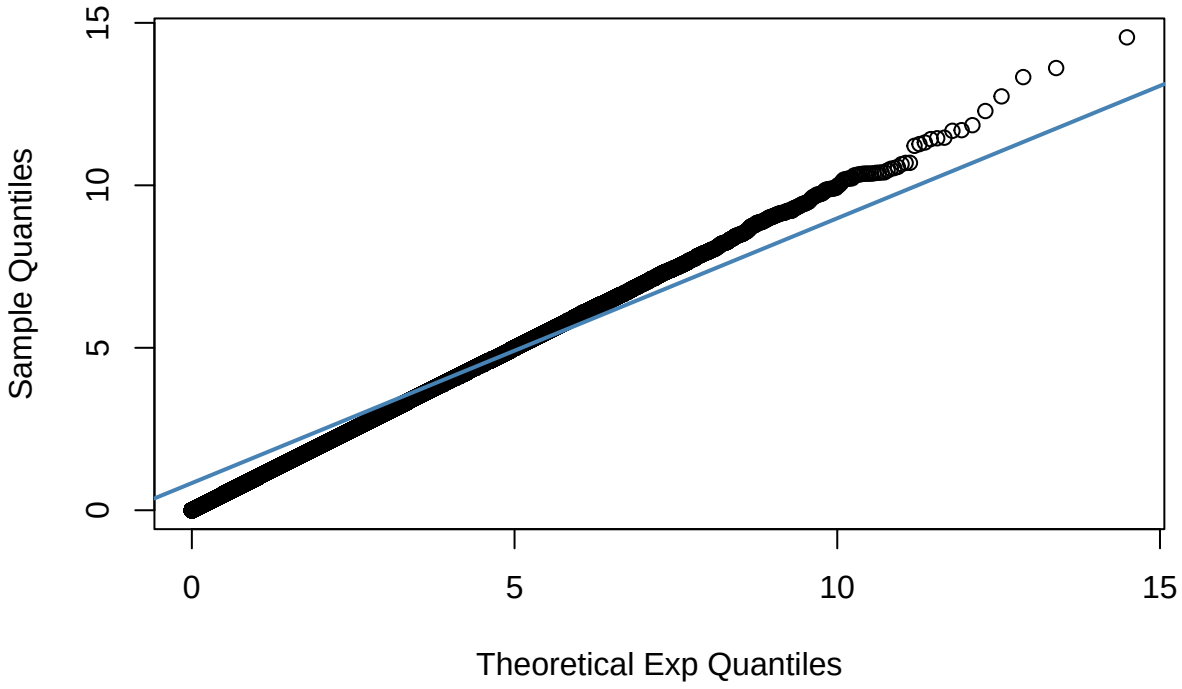


Assume we generate 100000 with Ziggurat method and they're actually normal, using rnorm() for now

```
set.seed(777)
x_exp <- rexpzig(1000000)
hist(x_exp, freq = FALSE)
curve(dexp(x), add=TRUE, col="red", lwd=2) # looks good
```




```
set.seed(777)
qqplot(qexp(ppoints(length(x_exp)), rate = 1/mean(x_exp)),
       sort(x_exp),
       xlab = "Theoretical Exp Quantiles",
       ylab = "Sample Quantiles")
qqline(x_exp, col = "steelblue", lwd = 2) # basically normal
```




```r
print("1,000,000 Ziggurat samples")
```

```
## [1] "1,000,000 Ziggurat samples"
```

```r
mean(x_exp) #Should be close to 1
```

```
## [1] 0.9985969
```

```r
sd(x_exp) #Should be close to 1
```

```
## [1] 0.9979412
```

```r
ks.test(x_exp, "pexp", rate = mean(x_exp)) #large p-value -> fail to reject expoential distribution
```

```
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  x_exp
## D = 0.0014393, p-value = 0.03175
## alternative hypothesis: two-sided
```

```r
print("1,000,000 rnorm samples")
```

```
## [1] "1,000,000 rnorm samples"
```

```r
# additional test:
x2_exp <- rexp(1000000)
ks.test(x2_exp, "pexp", rate = mean(x2_exp)) # passed
```

```
## Warning in ks.test.default(x2_exp, "pexp", rate = mean(x2_exp)): ties should
```

```
## not be present for the one-sample Kolmogorov-Smirnov test
```

```
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  x2_exp
## D = 0.00064725, p-value = 0.7963
## alternative hypothesis: two-sided
```

```
mean(x2_exp)  # basically 1
```

```
## [1] 0.9999385
```

```
sd(x2_exp)  # 1
```

```
## [1] 1.001024
```

Testing For Accuracy

In order to validate that the Ziggurat method is actually generating the random variables of interest, we performed the chi-square goodness of fit test (Knuth 1997). This test is used to check the distribution of the random variables generated by quantizing the horizontal axis of the probability density function into k bins and deriving a single value as a quality metric from the determined actual and expected number of samples in each bin. We constructed an R function based on the chi-square statistic formula: 2k-1 =i=1k(Yi-tpi)2/ tpi.

$$\chi^2_{k-1} = \sum_{i=1}^{k} \frac{(Y_i - tp_i)^2}{tp_i}$$

t = number of observations

$p_i = $probability that each observation falls into the the category i

Y_i = the number of observation that actually do fall into the category i

Using the rnormzig() function to generate 100,000 random variates, the distribution generated was tested based on 200 bins spaced uniformly over $[-7, 7]$ which we chose based on the paper that also analyzes the Ziggurat method (Leong et al. 2005). The calculated 2 value was 174.2026 and 184.4201 for the Ziggurat method and the rnorm() function in R, respectively. The critical value for a chi-square test with 199 degrees of freedom at 95% confidence level ($\alpha = 0.05$) is 232.912. Our implementation was also ran over increasing number of samples– one hundred thousand, a million, etc– which all had a 2 value less than the critical value of 232.912. Thus, we have successfully generated random variables that follow a normal distribution.

is used to check the distribution of the random variables generated. This test quantizes the horizontal axis of the probability density function into k bins and derives a single value as a quality metric from the determined actual and expected number of samples in each bin.

```
set.seed(999)

n = 100000
samples <- rnorm(n)

chi_squared_test <- function(x, k){
  t <- length(x) # number of observations
  bins <- seq(-7, 7, length.out = k+1) #200 bins spaced uniformly over data edge of min and max

  Yi <- hist(x, breaks = bins, plot = FALSE)$counts # observed counts
  # expected probabilites (p_i) of standard normal
  p_i <- numeric(k)
  for (i in 1:k) {
```

```
    p_i[i] <- pnorm(bins[i+1]) - pnorm(bins[i])
}
  # using computing chi square using equation from the paper
  chi_sq <- sum((Yi - t * p_i)^2 / (t * p_i))
  df <- k - 1
  return(data.frame(chi_square = chi_sq,
                    df = df))
}
```

```
set.seed(777)

k <- 200
results <- chi_squared_test(x, k)
results2 <- chi_squared_test(x2, k)

results$Method <- "Ziggurat"
results2$Method <- "Rnorm"

combined <- rbind(results, results2)
combined <- combined[, c("Method", setdiff(names(combined), "Method"))]
kable(combined, caption = "Chi-squared value: Ziggurat vs Rnorm")
```

Table 1: Chi-squared value: Ziggurat vs Rnorm

Method	chi_square	df
Ziggurat	158.4568	199
Rnorm	184.4201	199

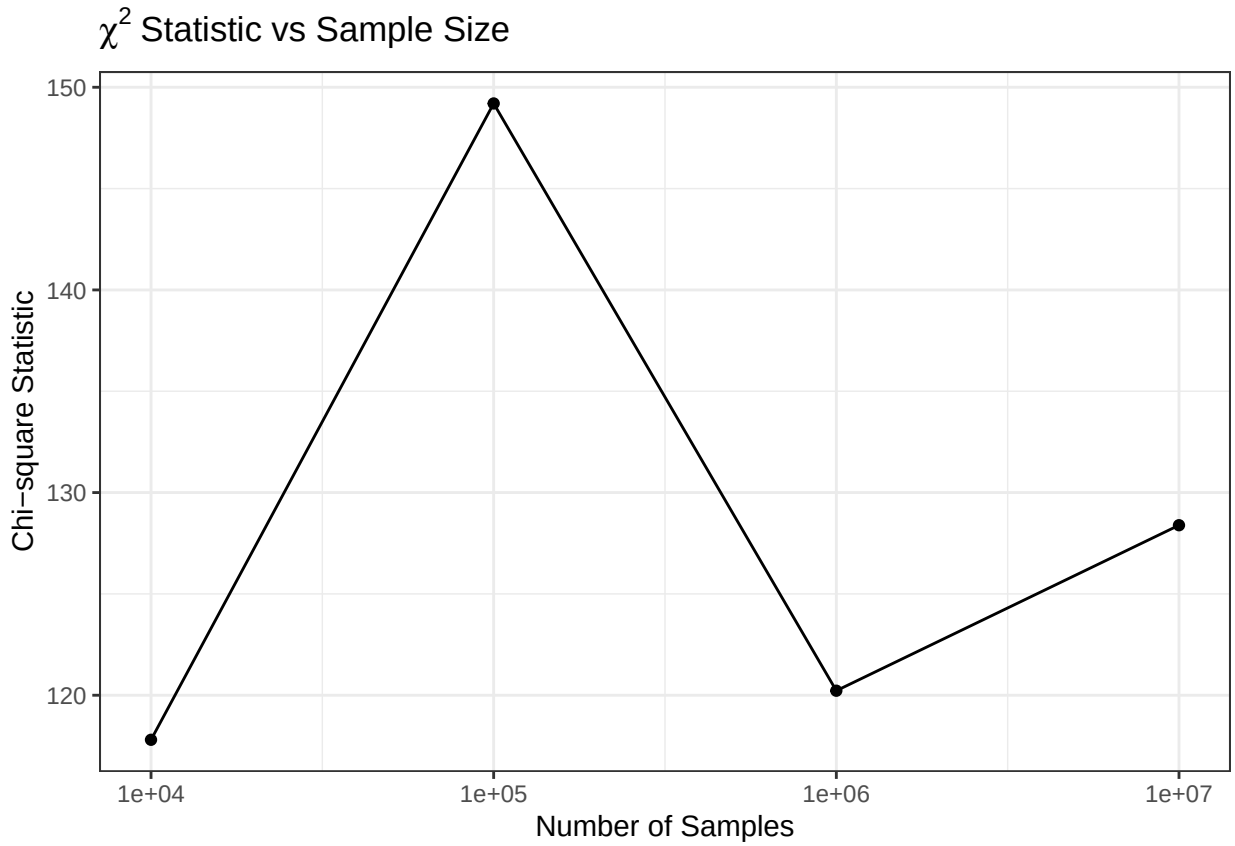
```
n <- c(10000, 100000, 1000000, 10000000)
set.seed(123)
k <- 200
value <- numeric(length(n))
for(i in 1:length(n)){
  vectors <- rnormzig(n[i])
  chi_sq <- chi_squared_test(vectors, k)
  value[i] <- chi_sq$chi_square

}


df <- data.frame(
  n = n,
  chi_square = value
)

ggplot(df, aes(x = n, y = chi_square)) +
  geom_line() +
  geom_point() +
  scale_x_log10() +
  labs(
    x = "Number of Samples",
    y = "Chi-square Statistic",
    title = expression(chi^2 ~ "Statistic vs Sample Size")
```

```
) +
theme_bw()
```



Works Cited