

# STAT 4302 midterm 2

James Laughead

3/26/2022

```
#1a

n <- 2000
y <- rexp(n)
gy <- ifelse(y<=2, 1, 0) * (sin(0.5*pi*y))^2
theta.hat.star <- mean(gy);
theta.hat.star

## [1] 0.3897616

est.var.theta.hat.star <- mean((gy - theta.hat.star)^2) / length(y)
est.var.theta.hat.star

## [1] 6.697388e-05

estandard_error <- (sqrt(est.var.theta.hat.star))
estandard_error

## [1] 0.008183757
```

$$f_X(x) = e^{-x}g(x) = (\sin(0.5\pi y))^2$$

$X_1, \dots, X_{2000}$  is a random sample of 2000 random variables sampled from  $f_X$ , such that  $y \leq 2$

$$\hat{\theta} = \frac{1}{2000} \sum_{x=1}^{2000} g(X^i)$$

$$\text{var}(\hat{\theta}) = \frac{1}{2000} \int_0^2 [g(x) - \theta] f_X(x) dx$$

$$SE(\hat{\theta}) = \sqrt{\text{var}(\hat{\theta})}$$

```
#1b
#95% confidence interval
lower_bound <- theta.hat.star - 1.96*estandard_error
upper_bound <- theta.hat.star + 1.96*estandard_error
c(lower=lower_bound, upper=upper_bound)

##      lower      upper
## 0.3737214 0.4058017

if((lower_bound <= 0.3925579) & (0.3925579 <= upper_bound)){
d <- "CI contains u"
}
if((lower_bound > 0.3925579) & (0.3925579 > upper_bound)){
d <- "CI does not contain u"
```

```
}
print(d)
```

```
## [1] "CI contains u"
```

95% confidence interval is  $(\hat{\theta} - 1.96SE(\hat{\theta}), \hat{\theta} + 1.96SE(\hat{\theta}))$

1c. If this experiment is repeated many times, the confidence intervals will not always contain the true value of theta. This would be the case if the coverage was 100%, but we know the coverage can also be around 95%, or greater or lower than that.

$$f_X(x) = e^{-x} \quad g(x) = (\sin(0.5\pi y))^2 \quad q(x) = \frac{1}{(0.4\sqrt{2\pi})} e^{(\frac{-1}{2})(\frac{x-0.84}{0.4})^2} \quad r(x) = \frac{g(x)f(x)}{q(x)}$$

$\hat{\theta} = \frac{1}{2000} \sum_{x=1}^{2000} \frac{I(0 < X_i < 2)f(X_i)}{q(X_i)}$   $X_i, \dots, X_{2000}$  is a random sample from q

```
#1d
```

```
#entire question
```

```
m2 <- 2000
```

```
theta.hat <- se <- numeric(5)
```

```
x <- rnorm(m2, mean = 0.84, sd = 0.4)
```

```
g <- function(x){
  (((sin(0.5*pi*x))^2)*exp((-x)))*(x>0)*(x<2)
}
```

```
qx <- (1/(0.4*sqrt(2*pi)))*exp((-1/2)*(((x-0.84)/0.4)^2))
```

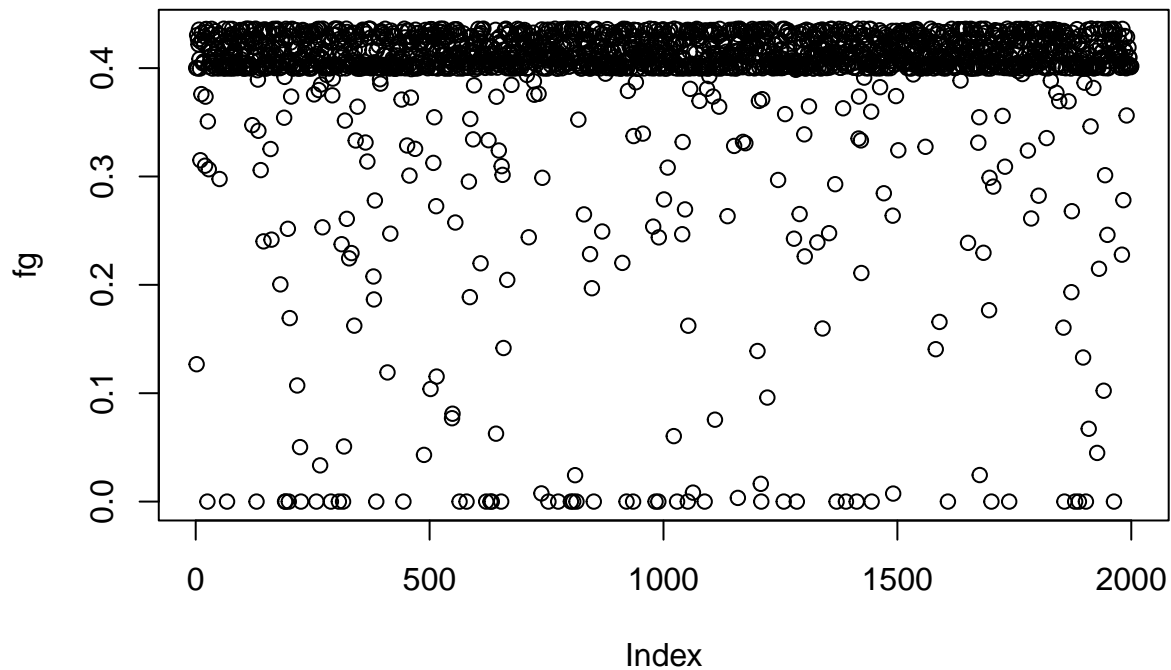
```
fg <- g(x)/qx
```

```
vR <- var(fg)
```

```
print(vR)
```

```
## [1] 0.006643095
```

```
plot(fg)
```



As we can see from the plot, between 0 and 2, the ratio function is nearly constant because it usually produces a value around 0.4

```
#1e

m2 <- 2000
theta.hat <- se <- numeric(5)

x <- rnorm(m2, mean = 0.84, sd = 0.4)

#x <- rexp(m2, rate = 1)
?rnorm
?rexp
g <- function(x){
  (((sin(0.5*pi*x))^2)*exp((-x)))*(x>0)*(x<2)
}
qx <- (1/(0.4*sqrt(2*pi)))*exp(((1/2)*((x-0.84)/0.4)^2))

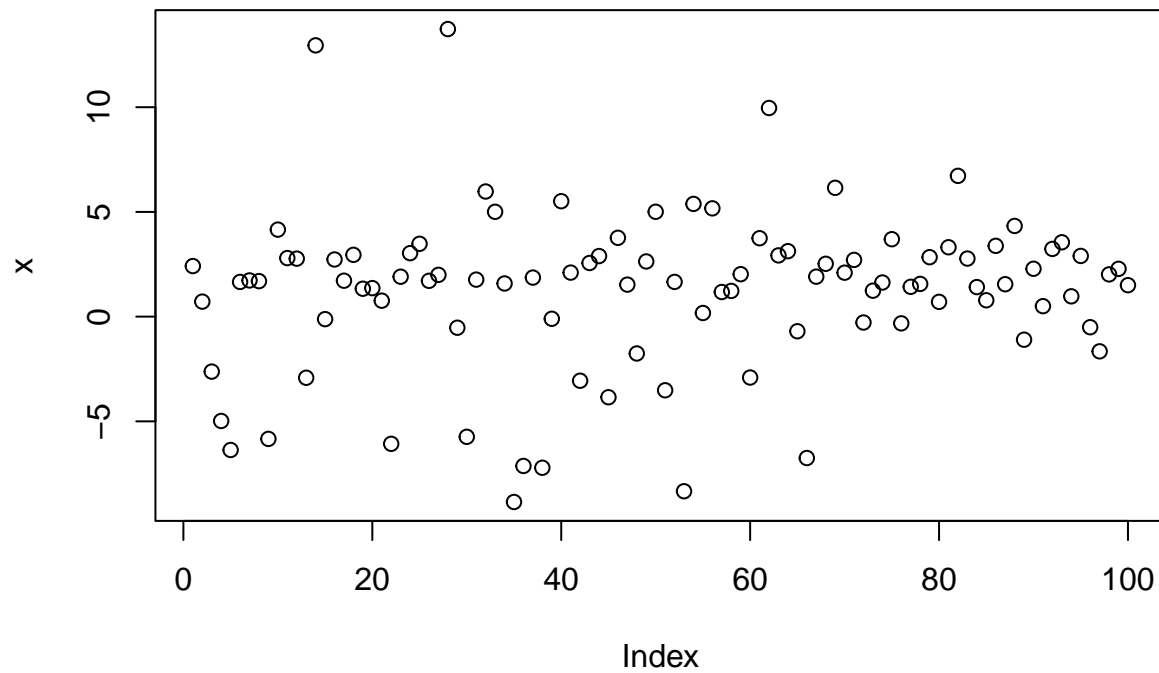
fg <- g(x)/qx
#Our Monte Carlo estimate is:
print(mean(fg))

## [1] 0.3939388
```

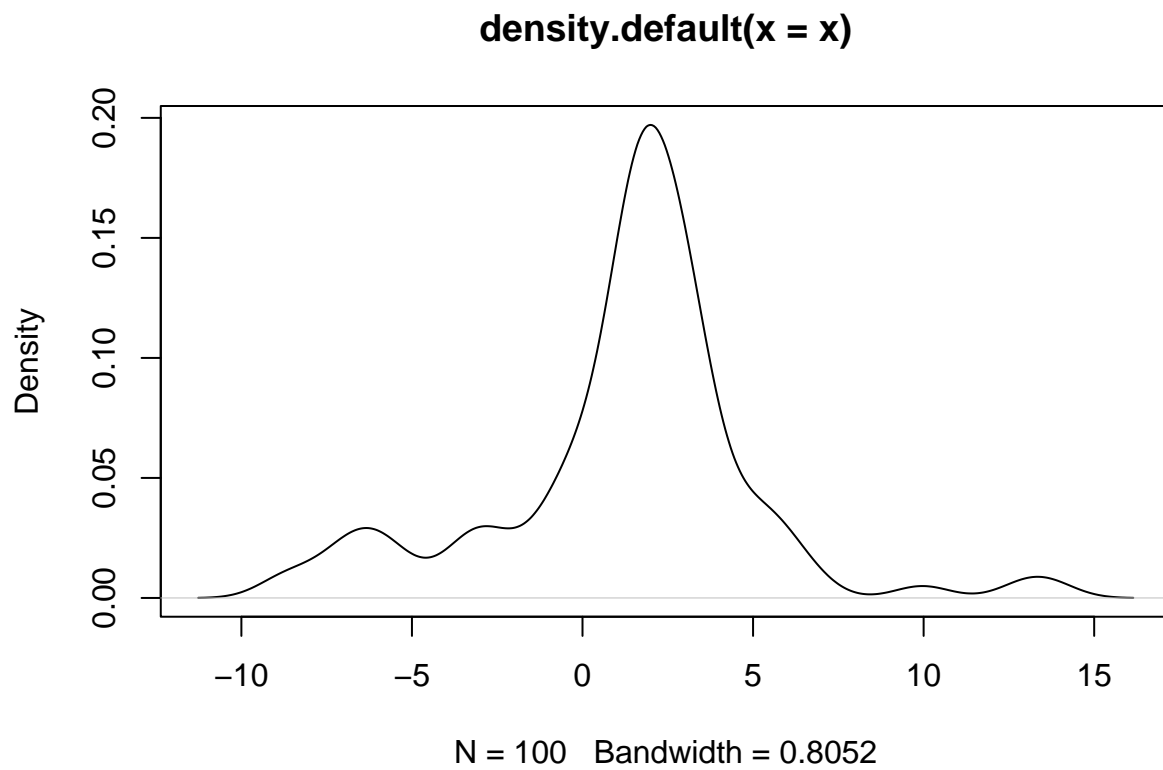
When running my samples, the estimate using the ratio function was closer to the true theta. This makes sense since a ratio function that is close to a constant yields a small variance.

```
#2a
gen.data <- function (n) {
  y <- rbinom(n, 2, 0.3)
  rnorm(n, 2, 1+3*y)
}
```

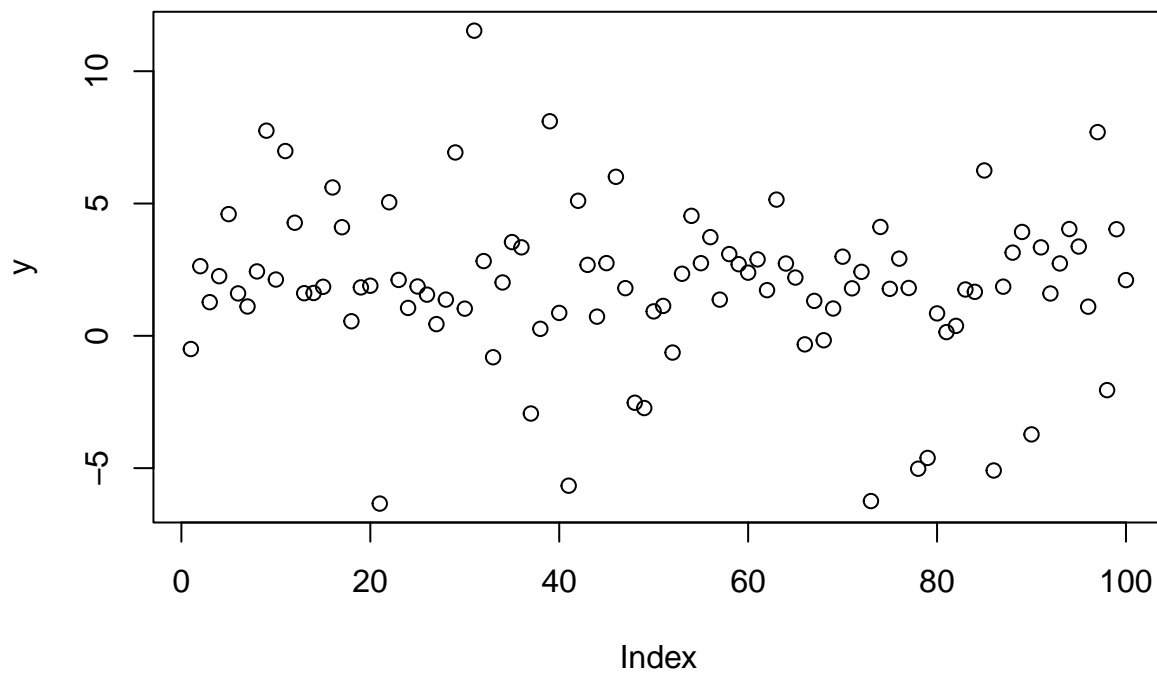
```
x <- (gen.data(100))
plot(x)
```



```
plot(density(x))
```

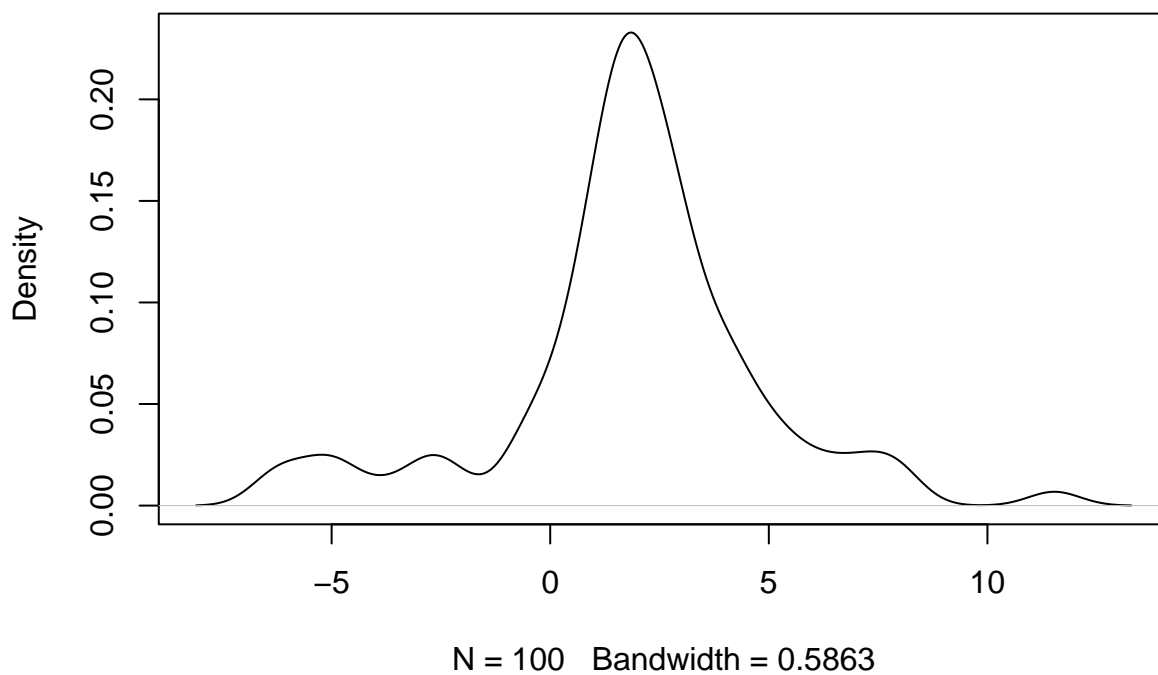


```
y <- (gen.data(100))
plot(y)
```

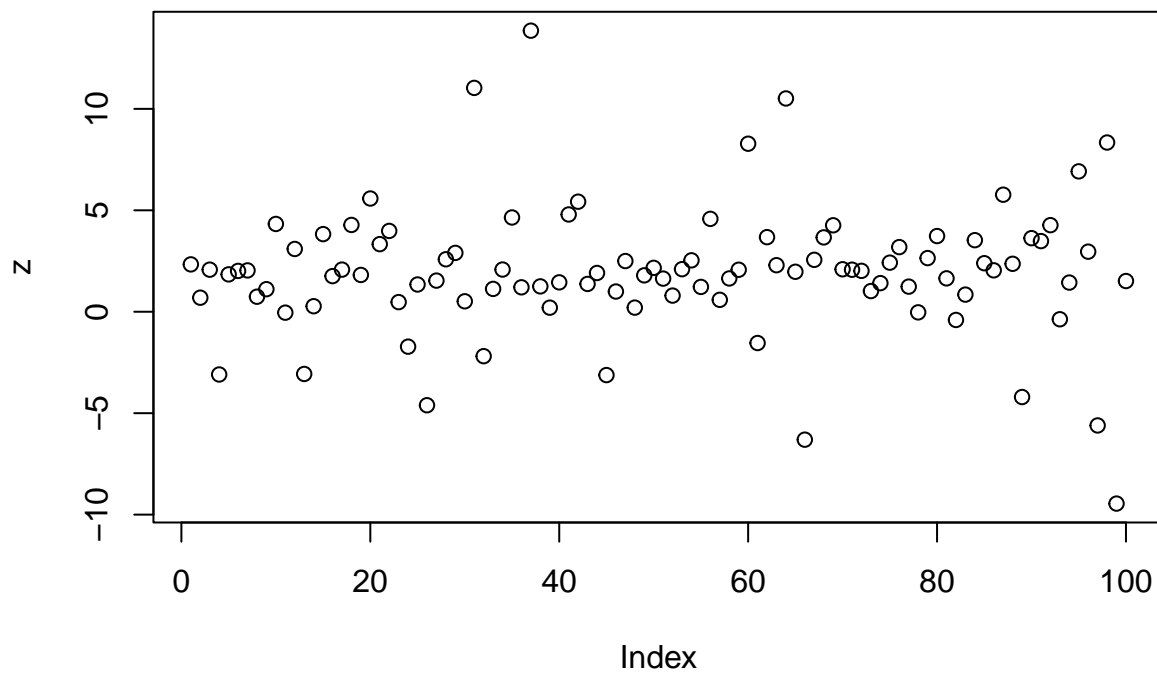


```
plot(density(y))
```

**density.default(x = y)**

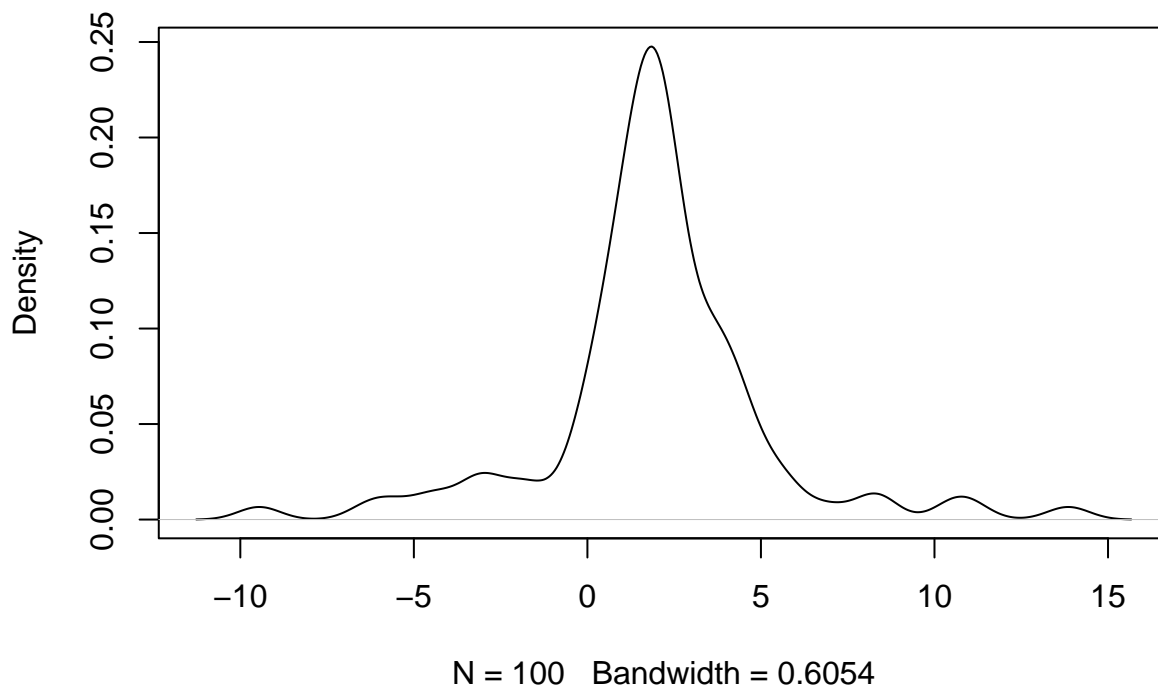


```
z <- (gen.data(100))  
plot(z)
```

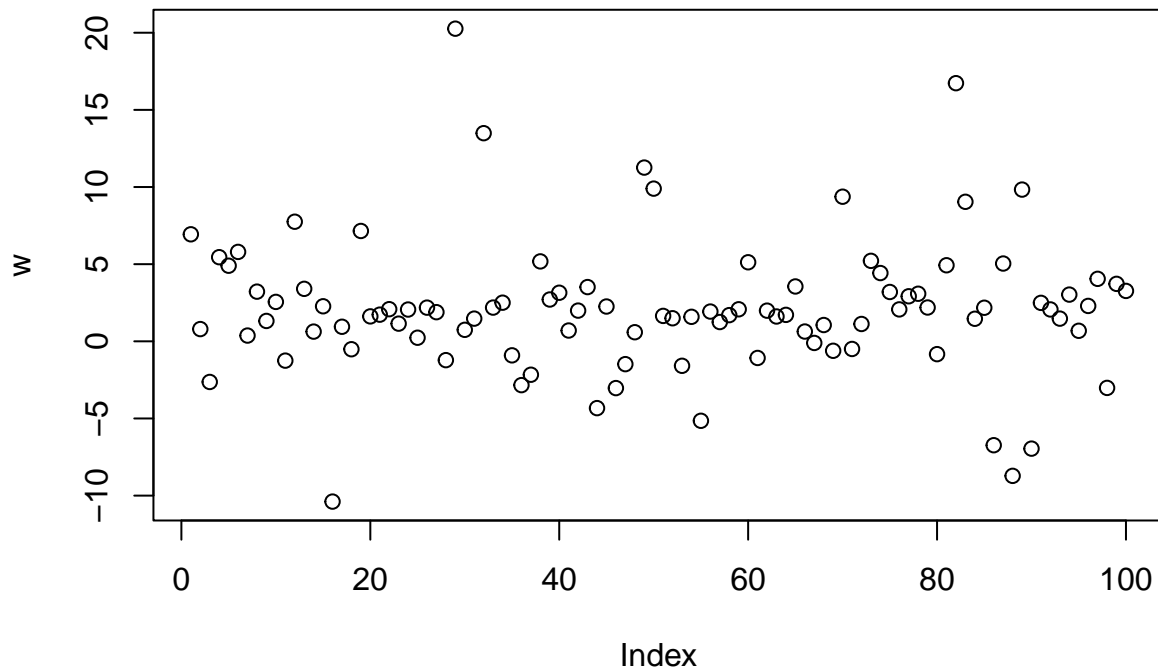


```
plot(density(z))
```

**density.default(x = z)**

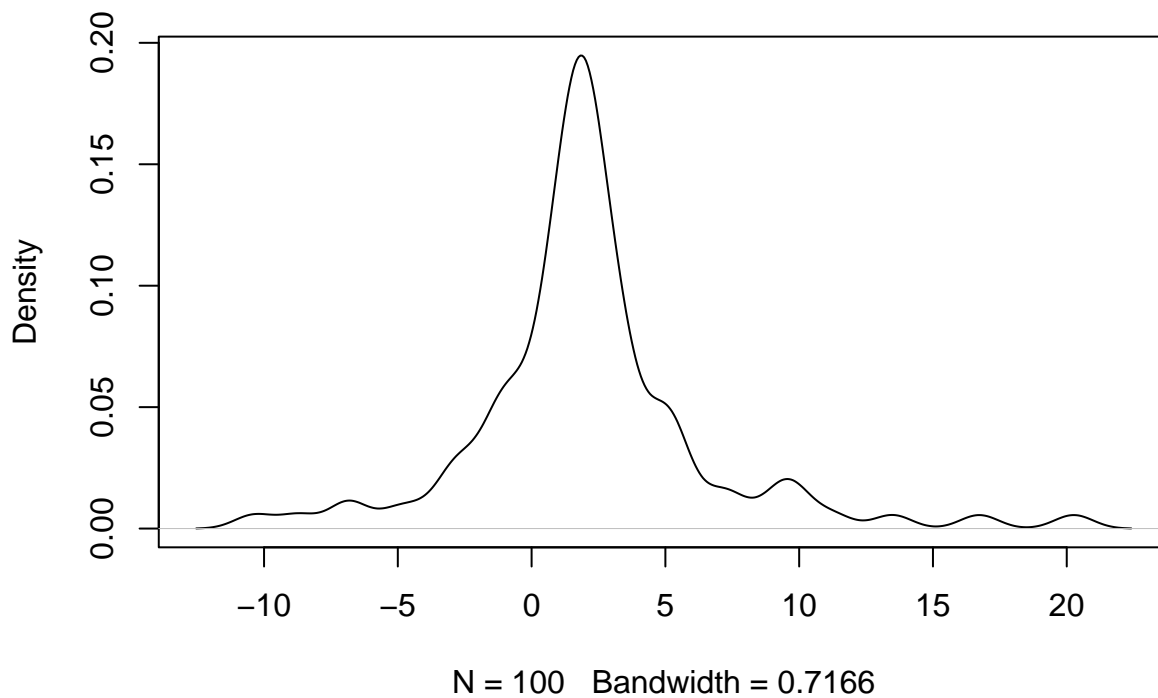


```
w <- (gen.data(100))
plot(w)
```



```
plot(density(w))
```

**density.default(x = w)**



In the function, first we can choose the parameter  $n$ . The function generates  $n$  binomial values with success rate 0.3, and stores all 100 observations as a variable. Then a set of  $n$  normal values is generated with each normal value coming from a normal distribution with mean 2 and variance  $1 + 3$  times whatever the binomial generated number was for that placement. The resulting distribution is a normal distribution with mean two, which has occasionally long tails.

```

one.sim <- function (n, conf.level=0.95) {
  y <- rbinom(n, 2, 0.3)
  f <- rnorm(n, 2, 1+3*y)
  mean <- mean(f)
  alpha <- 1 - conf.level
  z.quantile <- qnorm(1 - alpha / 2)
  SE <- sd(f)/sqrt(n)

  bounds <- z.quantile * SE

  lower <- mean - z.quantile * SE
  upper <- mean + z.quantile * SE
  CI <- cbind(lower,
              upper )

  if((CI[, "lower"] <= 2) & (2 <= CI[, "upper"])){
    d <- "CI contains u"
  }
  else{
    d <- "CI does not contain u"
  }

  list(mean=mean, lower = lower, upper = upper, d = d)
}
one.sim(100)

```

```

## $mean
## [1] 1.903634
##
## $lower
## [1] 1.206286
##
## $upper
## [1] 2.600981
##
## $d
## [1] "CI contains u"

```

```

#one.sim for just obtaining mean
one.sim <- function (n, conf.level=0.95) {
  y <- rbinom(n, 2, 0.3)
  f <- rnorm(n, 2, 1+3*y)
  mean <- mean(f)
  alpha <- 1 - conf.level
  z.quantile <- qnorm(1 - alpha / 2)
  SE <- sd(f)/sqrt(n)

  bounds <- z.quantile * SE

  lower <- mean - z.quantile * SE
  upper <- mean + z.quantile * SE
  CI <- cbind(lower,

```

```

upper )

if((CI[, "lower"] <= 2) & (2 <= CI[, "upper"])){
  d <- "CI contains u"
}
else{
  d <- "CI does not contain u"
}

plus.or.minus <- qnorm(1-alpha/2) * SE
c(lower=mean(f)-plus.or.minus, upper=mean(f)+plus.or.minus)

}

# How have replicates?
K <- 10000
## Carry out the simulation for n=10
results.n10 <- replicate(K, one.sim(n=100))
## Which intervals contain the true value of mu?
CIs.contains.mu.n10 <- (results.n10[,1] <= 2) & (2 <= results.n10[,2])
## Estimate the coverage using the sample proportion
coverage.n10 <- mean(CIs.contains.mu.n10)
round(coverage.n10, 5)

## [1] 0.9482
## [1] 0.923
## Calculate the standard error for this estimated coverage
SE.coverage.n10 <- sqrt( coverage.n10 * (1-coverage.n10) / K )
round(SE.coverage.n10, 5)

## [1] 0.00222
## [1] 0.003
## And build a 95% CI for the coverage
CI.coverage.n10 <- c(lower=coverage.n10-1.96*SE.coverage.n10,
upper=coverage.n10+1.96*SE.coverage.n10)
round(CI.coverage.n10, 5)

## lower upper
## 0.94386 0.95254

```

Here we learn the one sample t-based confidence procedure works in cases with a normal distribution generated from values of another distribution.

```

#2d
#one.sim for just obtaining mean
one.sim <- function (n, conf.level=0.95) {
  y <- rbinom(n, 2, 0.3)
  f <- rnorm(n, 2, 1+3*y)
  mean <- mean(f)
  alpha <- 1 - conf.level
  z.quantile <- qnorm(1 - alpha / 2)
  SE <- sd(f)/sqrt(n)
}

```

```

bounds <- z.quantile * SE

lower <- mean - z.quantile * SE
upper <- mean + z.quantile * SE
CI <- cbind(lower,
             upper )

if((CI[, "lower"] <= 2) & (2 <= CI[, "upper"])){
  d <- "CI contains u"
}
else{
  d <- "CI does not contain u"
}

plus.or.minus <- qnorm(1-alpha/2) * SE
c(lower=mean(f)-plus.or.minus, upper=mean(f)+plus.or.minus)

}

# How have replicates?
K <- 10000
## Carry out the simulation for n=10
results.n10 <- replicate(K, one.sim(n=20))
## Which intervals contain the true value of mu?
CIs.contains.mu.n10 <- (results.n10[,1] <= 2) & (2 <= results.n10[,2])
## Estimate the coverage using the sample proportion
coverage.n10 <- mean(CIs.contains.mu.n10)
round(coverage.n10, 5)

## [1] 0.9375

## [1] 0.923
## Calculate the standard error for this estimated coverage
SE.coverage.n10 <- sqrt( coverage.n10 * (1-coverage.n10) / K )
round(SE.coverage.n10, 5)

## [1] 0.00242

## [1] 0.003
## And build a 95% CI for the coverage
CI.coverage.n10 <- c(lower=coverage.n10-1.96*SE.coverage.n10,
                    upper=coverage.n10+1.96*SE.coverage.n10)
round(CI.coverage.n10, 5)

##      lower      upper
## 0.93276 0.94224

```

When the sample size is decreased to  $n = 20$ , the coverage also decreases slightly. This makes sense as when the sample size goes down, the variance increases, which would make it more likely for the estimated  $u$  value would miss the confidence interval and not be covered.