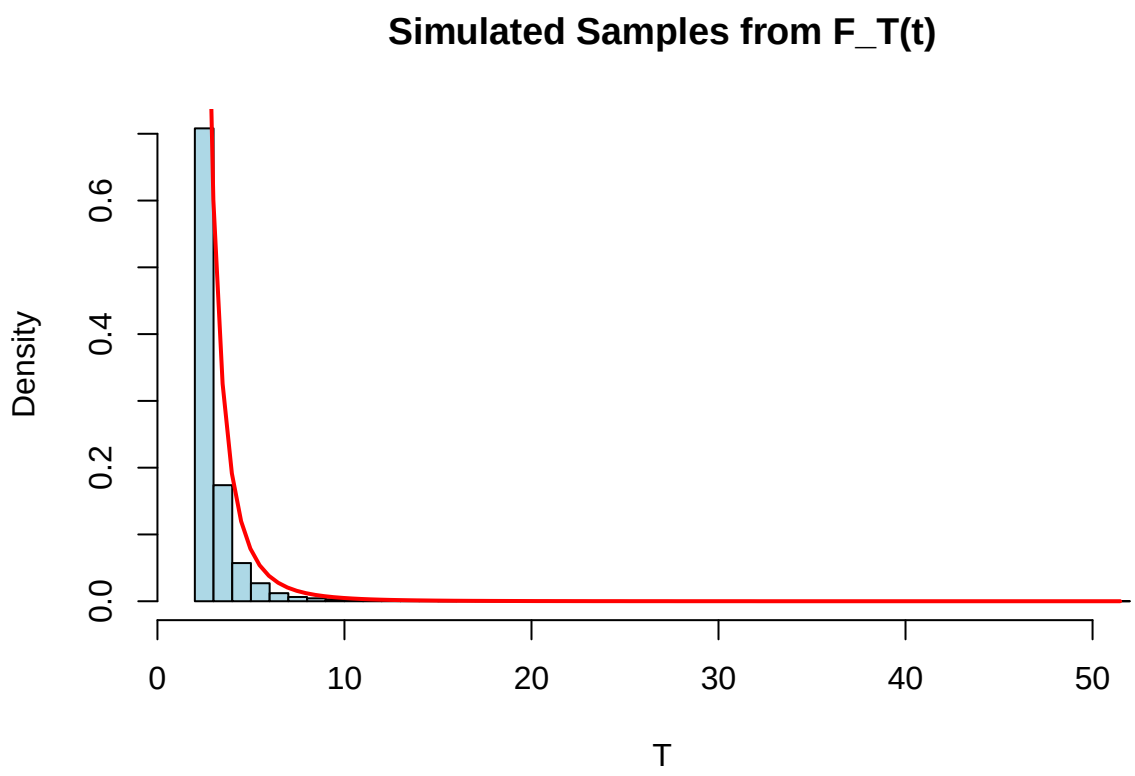
Daniel Losada and Sergio Quintanilla

Exercise 1

we simulate random samples from a given cumulative distribution function (CDF) using the inverse transform method. This method allows us to generate random values from a specified distribution by applying the inverse of its CDF to uniform random numbers.

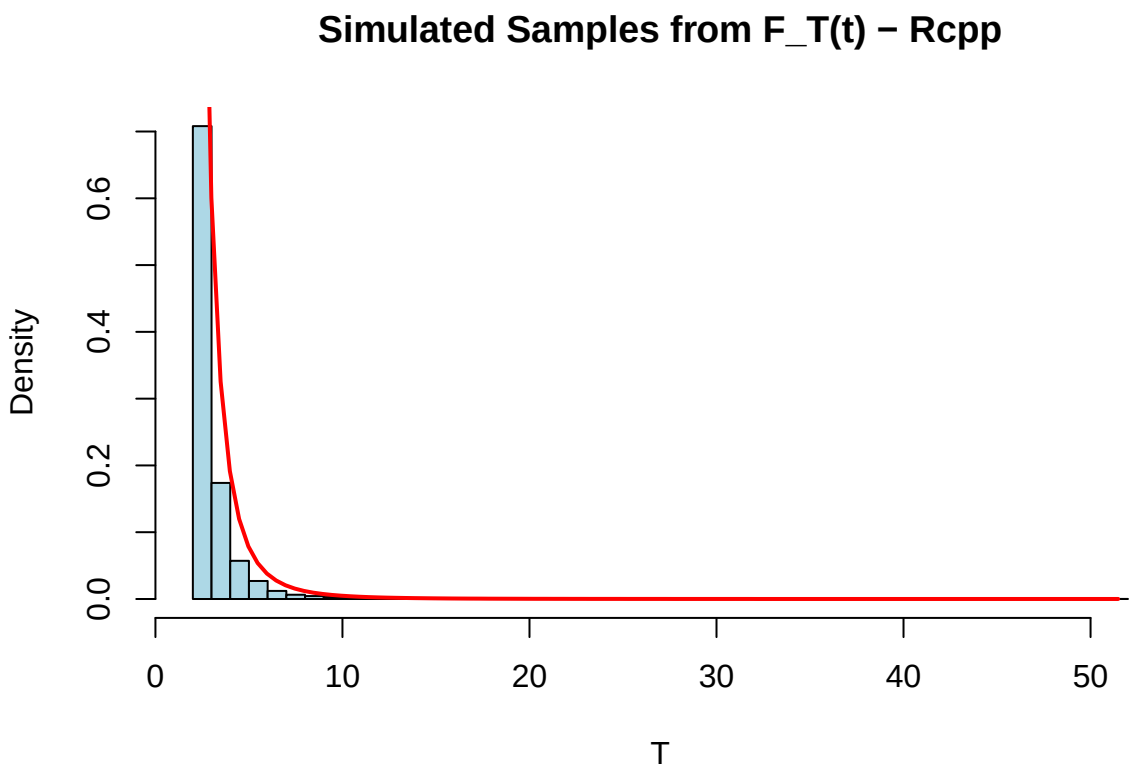
By inverting this function, we derive a formula to transform uniform random numbers into samples that follow this distribution. We then generate and visualize these samples, comparing them with the theoretical probability density function (PDF).

```r
simulate_T <- function(n) {
  U <- runif(n)  # Generate n uniform random numbers
  T <- 2 / (1 - U)^(1/3)  # Apply inverse transform
  return(T)
}

set.seed(123)
samples <- simulate_T(10000)

# Plot histogram
hist(samples, probability = TRUE, col = "lightblue", breaks = 50, main = "Simulated Samples from F_T(t)

# Overlay theoretical density (Derivative of CDF)
curve(6 * (2^3) / x^4, from = 2, to = max(samples), add = TRUE, col = "red", lwd = 2)
```



The histogram of simulated samples aligns well with the theoretical density function, confirming that the inverse transform method correctly generates values following the given distribution. The majority of values are concentrated near t=2, with a long right tail, as expected from the shape of the density function.

```
library(Rcpp)

cppFunction('
NumericVector simulate_T_rcpp(int n) {
  NumericVector U = runif(n);  // Generate uniform samples
  NumericVector T = 2 / pow(1 - U, 1.0 / 3.0);  // Apply inverse transform
  return T;
}
')

# Example usage
set.seed(123)
samples_rcpp <- simulate_T_rcpp(10000)

# Verify by histogram
hist(samples_rcpp, probability = TRUE, col = "lightblue", breaks = 50, main = "Simulated Samples from F_
curve(6 * (2^3) / x^4, from = 2, to = max(samples_rcpp), add = TRUE, col = "red", lwd = 2)
```



Exercise 2

We use the acceptance-rejection method to simulate samples from a Pareto distribution. Since its tail is not as heavy, we choose a Pareto(2,3) distribution as an instrumental density function to generate candidate samples efficiently.

First, we determine a constant M such that the ratio of the target and instrumental density functions satisfies fS(t) <= M*fT(t) for all t > 2.

```
library(ggplot2)

f_S <- function(t) { 8 / t^5 }  # PDF of S ~ Pareto(2,4)
f_T <- function(t) { 6 / t^4 }  # PDF of T ~ Pareto(2,3)

# Compute M
M <- max((f_S(2) / f_T(2)))  # Maximized at t = 2
cat("(a) M =", M, "\n")
```

```
## (a) M = 0.6666667
```

```
# Function to sample from Pareto(2,3) using inverse transform sampling
rpareto_T <- function(n) {
  U <- runif(n)
  return(2 / (U^(1/3)))  # Inverse CDF
}

# Acceptance-rejection sampling for S ~ Pareto(2,4)
rpareto_S <- function(n) {
  samples <- numeric(n)
  count <- 0
  while (count < n) {
    T <- rpareto_T(1)  # Sample T ~ Pareto(2,3)
    U <- runif(1)
    if (U <= 2 / T) {  # Acceptance condition
      count <- count + 1
      samples[count] <- T
    }
  }
  return(samples)
}

#Show a 99% confidence interval algorithm based on MC = 10000 observations.
set.seed(123)
MC <- 10000
samples_S <- rpareto_S(MC)

# Compute expected value
E_S <- mean(samples_S)
cat("(c) Estimated E[S]:", E_S, "\n")
```

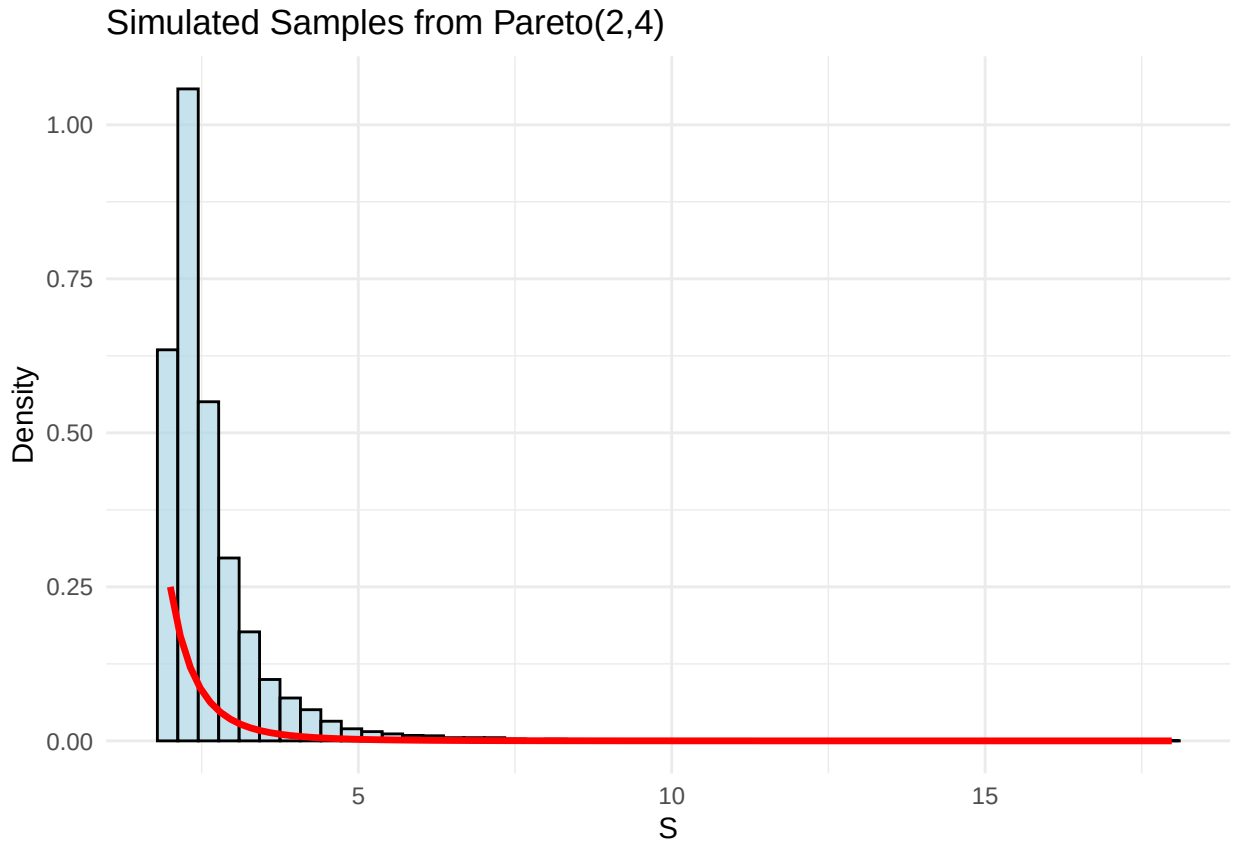
```
## (c) Estimated E[S]: 2.671081
```

```r
# Compute 99% confidence interval
alpha <- 0.01
se <- sd(samples_S) / sqrt(MC)
CI <- c(E_S - qnorm(1 - alpha/2) * se, E_S + qnorm(1 - alpha/2) * se)
cat("(c) 99% Confidence Interval: [", CI[1], ",", CI[2], "]\n")
```

```
## (c) 99% Confidence Interval: [ 2.646214 , 2.695948 ]
```

```r
# Plot histogram with theoretical density
ggplot(data.frame(S = samples_S), aes(x = S)) +
  geom_histogram(aes(y = ..density..), bins = 50, fill = "lightblue", color = "black", alpha = 0.7) +
  stat_function(fun = f_S, color = "red", lwd = 1.2) +
  labs(title = "Simulated Samples from Pareto(2,4)", x = "S", y = "Density") +
  theme_minimal()
```

```
## Warning: The dot-dot notation (`..density..`) was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(density)` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



The estimated expectation from our simulation closely matches the theoretical value E[s]=2.6667, with a 99% confidence interval of approximately [2.646, 2.696], confirming the accuracy of our method. The histogram of the simulated values aligns well with the theoretical density.

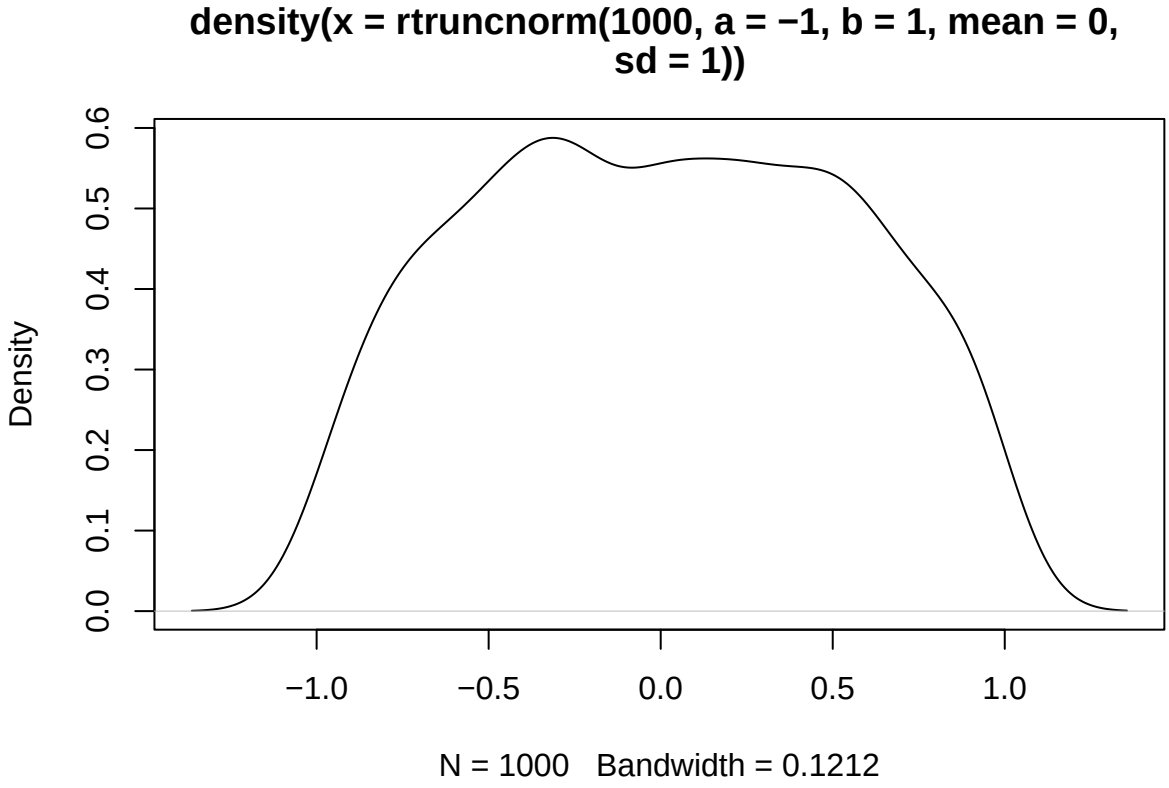
Excercise 3

First, let's observe what we expect to see using the truncnorm library's random sampling distribution.

```
library(truncnorm)
```

```
## Warning: package 'truncnorm' was built under R version 4.4.2
```

```
library(AR)
plot(density(rtruncnorm(1000,a=-1,b=1,mean=0,sd=1)))
```



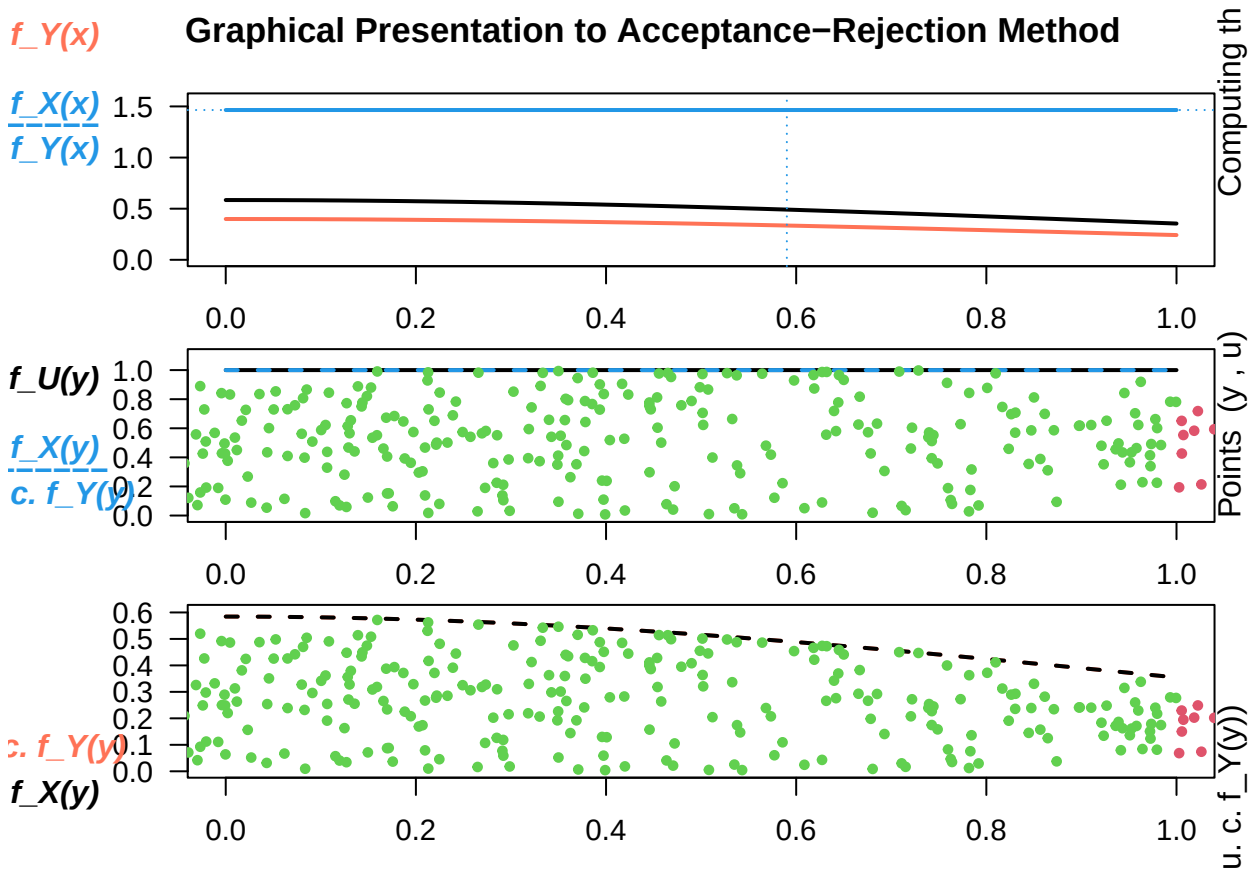
Now let's test the different sampling methods.

a)

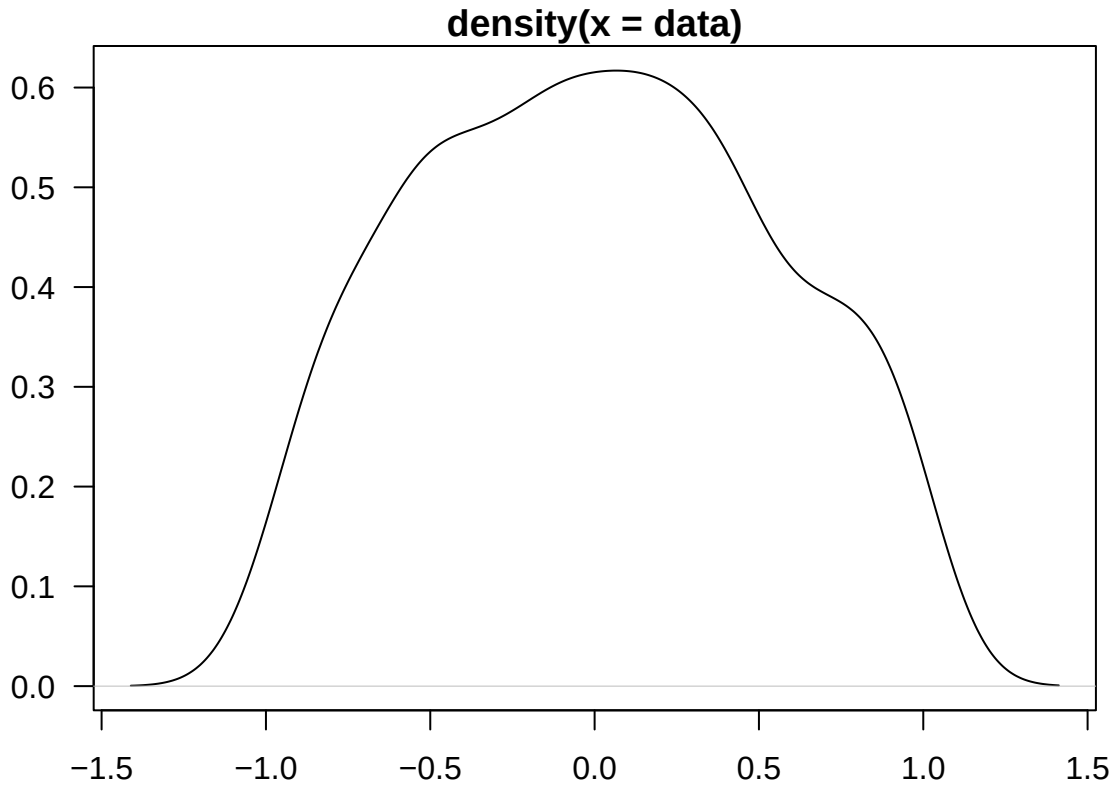
```
AR_norm = function(n, off=F){
  data = AR.Sim ( n = n ,
    f_X = function (y){ dtruncnorm(y,-1,1,0,1)} ,
    Y.dist = "norm" , Y.dist.par = c (0,1) ,
    Rej.Num = TRUE , Rej.Rate = TRUE , Acc.Rate = TRUE )
  if (off == T) {dev.off()} #to block out plots in efficiency test
  plot(density(data))
  if (off == T) {dev.off()}
```

```
}
AR_norm(500)
```

```
## Optimal c = 1.465
```



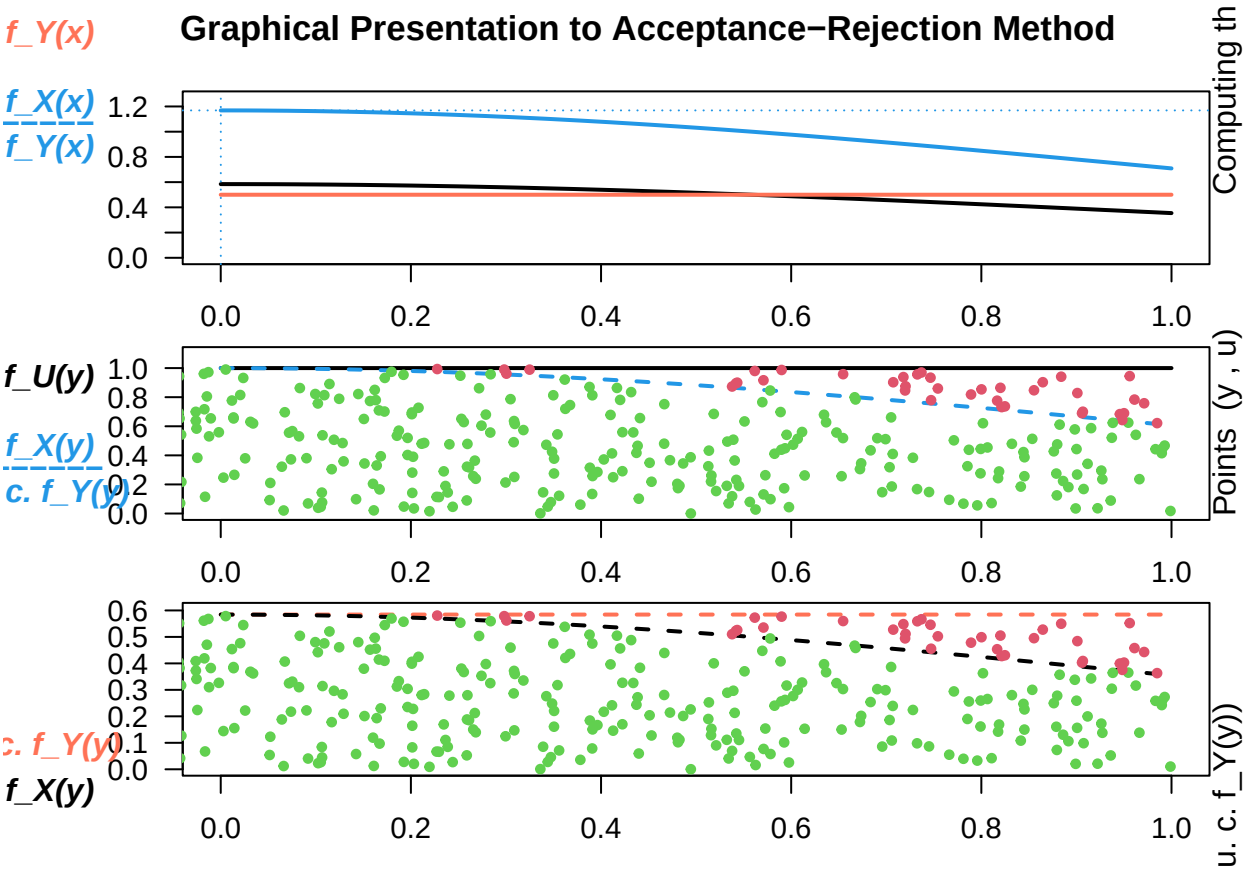

```
## The numbers of Rejections = 267
## Ratio of Rejections = 0.348
## Ratio of Acceptance = 0.652
```



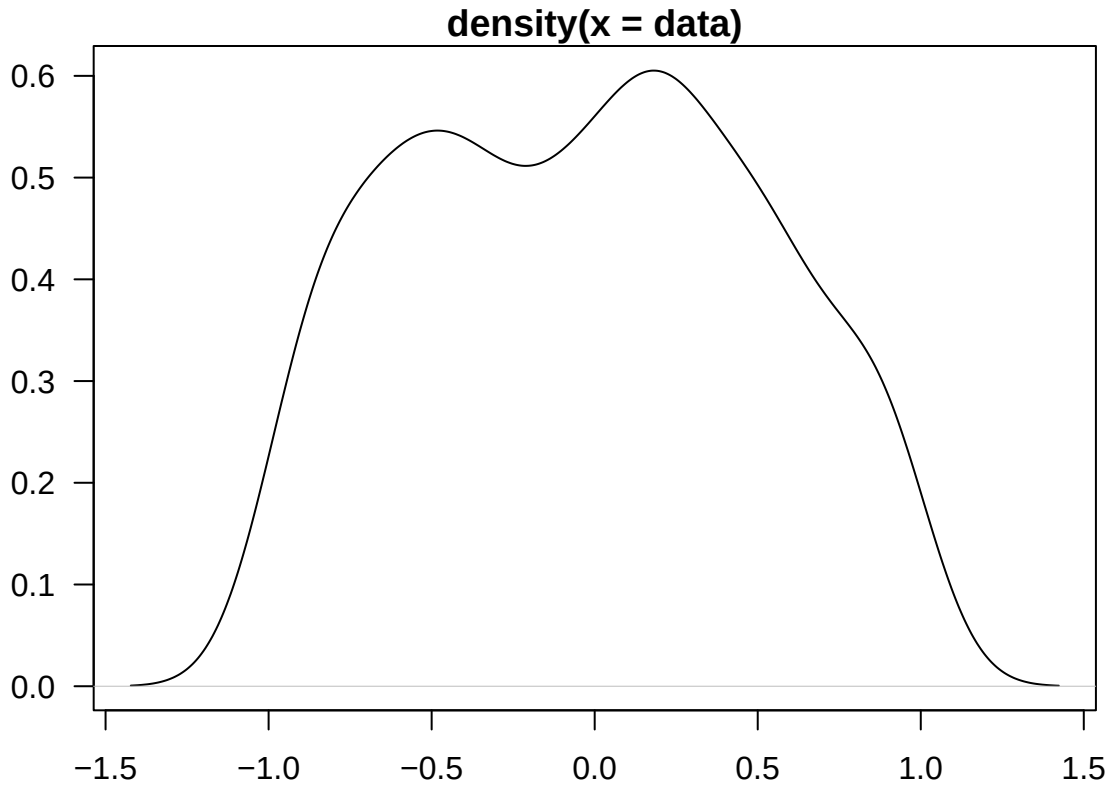
b)

```
AR_unif = function(n,off=F){
  data = AR.Sim ( n = n ,
    f_X = function (y){ dtruncnorm(y,-1,1,0,1)} ,
    Y.dist = "unif" , Y.dist.par = c (-1,1) ,
    Rej.Num = TRUE , Rej.Rate = TRUE , Acc.Rate = TRUE)
  if (off == T) {dev.off()} #to block out plots in efficiency test
  plot(density(data))
  if (off == T) {dev.off()}

}

AR_unif(500)
```

```
## Optimal c = 1.169
```



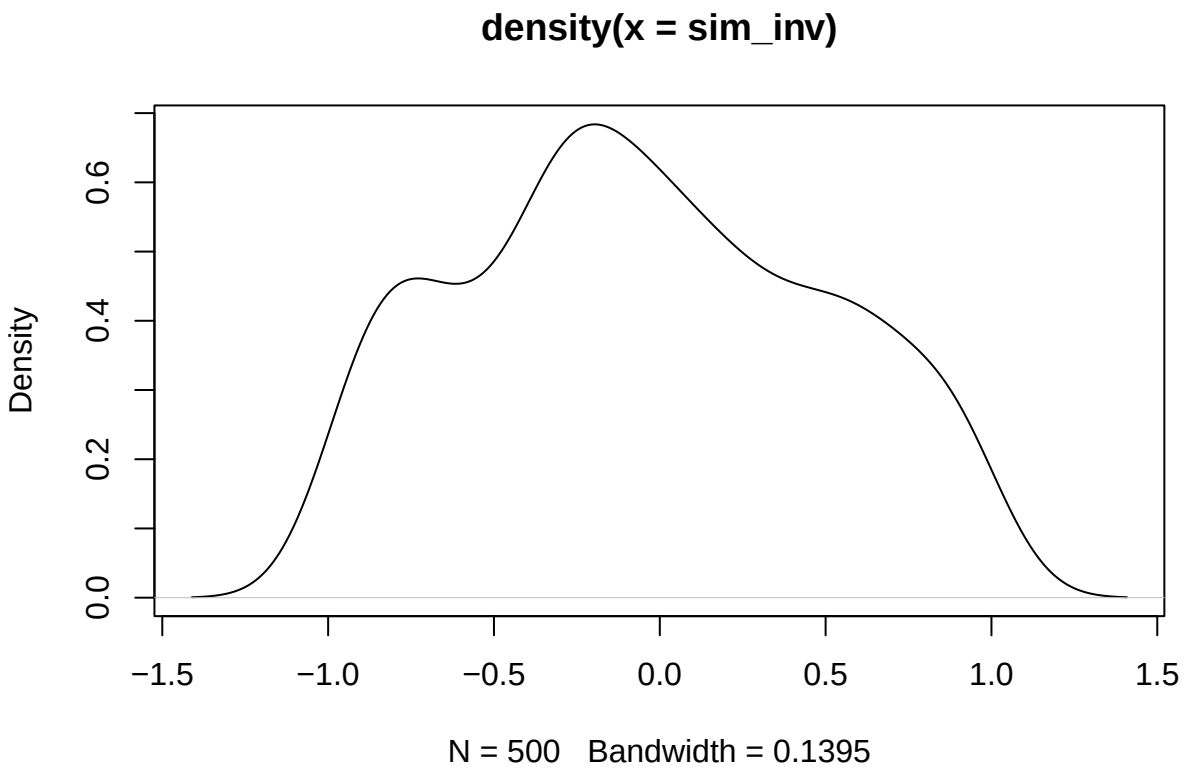

```
## The numbers of Rejections = 82
## Ratio of Rejections = 0.141
## Ratio of Acceptance = 0.859
```



Overall, a and b look quite similar, although not exactly, but both estimate a truncated distribution quite well. The uniform one appears quite similar to a common normal distribution, indicating it might be underestimating the variance around the mean. However, the normal version appears to have more skew, which would make sense given it has another layer of bias, depending on the candidate normal distribution's behavior.

c)

```
I_norm = function(n, off=F){
  #set our parameters for the truncated distribution
  mean <- 0
  sd <- 1
  a <- -1
  b <- 1
  u <- runif(n)

  #apply the inverse CDF to transform the uniform ru to a truncated normal
  sim_inv <- qtruncnorm(u, mean = mean, sd = sd, a = a, b = b)
  if (off == F) {plot(density(sim_inv))} #to block out plots in efficiency test
}
I_norm(500)
```



This one appears mostly like a regular normal distribution with some irregularities, but is not as close as the prior two sampling methods.

```
library(microbenchmark)
n=500
result=microbenchmark( #use microbenchmark to compare efficiencies
  AR_norm = AR_norm(n, off=T),
  AR_unif = AR_unif(n, off=T),
  I_norm = I_norm(n, off=T),
  times = 30
)
```

```
result
```

```
## Unit: milliseconds
##     expr      min       lq       mean   median       uq      max neval
##  AR_norm 143.3164 149.2271 156.326983 156.5535 162.6241 171.1853    30
##  AR_unif 113.9268 120.4012 127.941273 123.8279 127.1543 199.7756    30
##   I_norm   1.8920   1.9228   2.184883   1.9532   2.0049   8.1232    30
```

We can see that the inverse transform technique was by far the fastest, exponentially faster than either Accept-Reject algorithm. Within AR algorithms, the uniform candidate density took half the time the normal one did. That being said, looking at the graphs above, the distributions appear to be more accurately estimated by the normal AR algorithm, which makes sense given we expect a tradeoff between accuracy and speed.