Sampling from probability distributions

Giri Iyengar

Sampling: When do we use it

- We want to simulate a complex system and need to know what can happen as the system operates
- Used in Economic forecasting, Financial modeling, Statistical Physics, Machine Learning
- Markov Chain Monte Carlo (MCMC) technique is a well-used simulation tool (You might have encountered it in your 401(k) modeling)

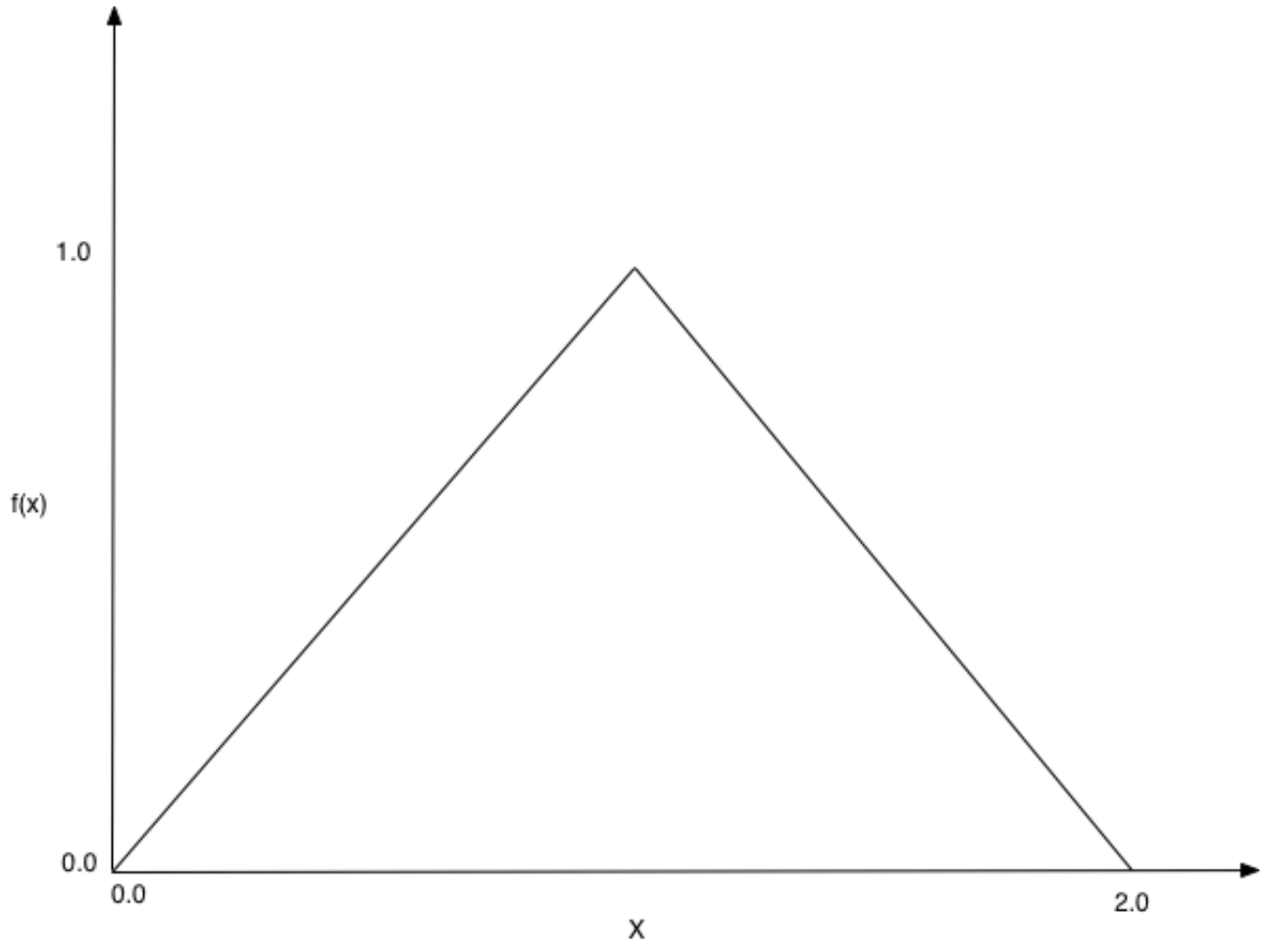
Some popular sampling techniques

- Inverse Transform Sampling
- Rejection Sampling
- Slice Sampling

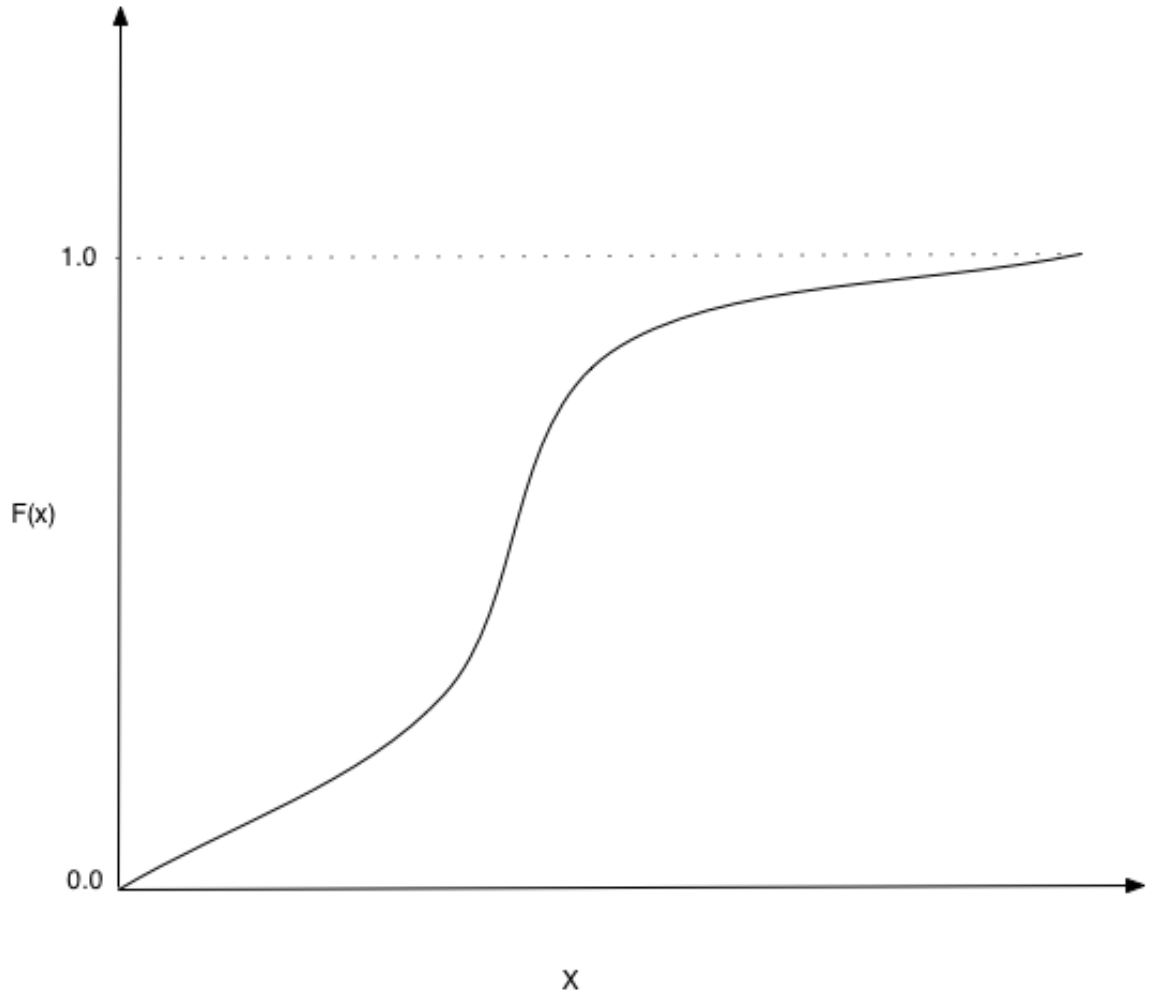
Inverse Transform Sampling

- Start with the CDF (cumulative distribution function)
  - Integral of the PDF
- Invert it
- Uniformly sample a value from [0,1]. Interpret as a probability
- Use the inverse to find the corresponding value of the random variable

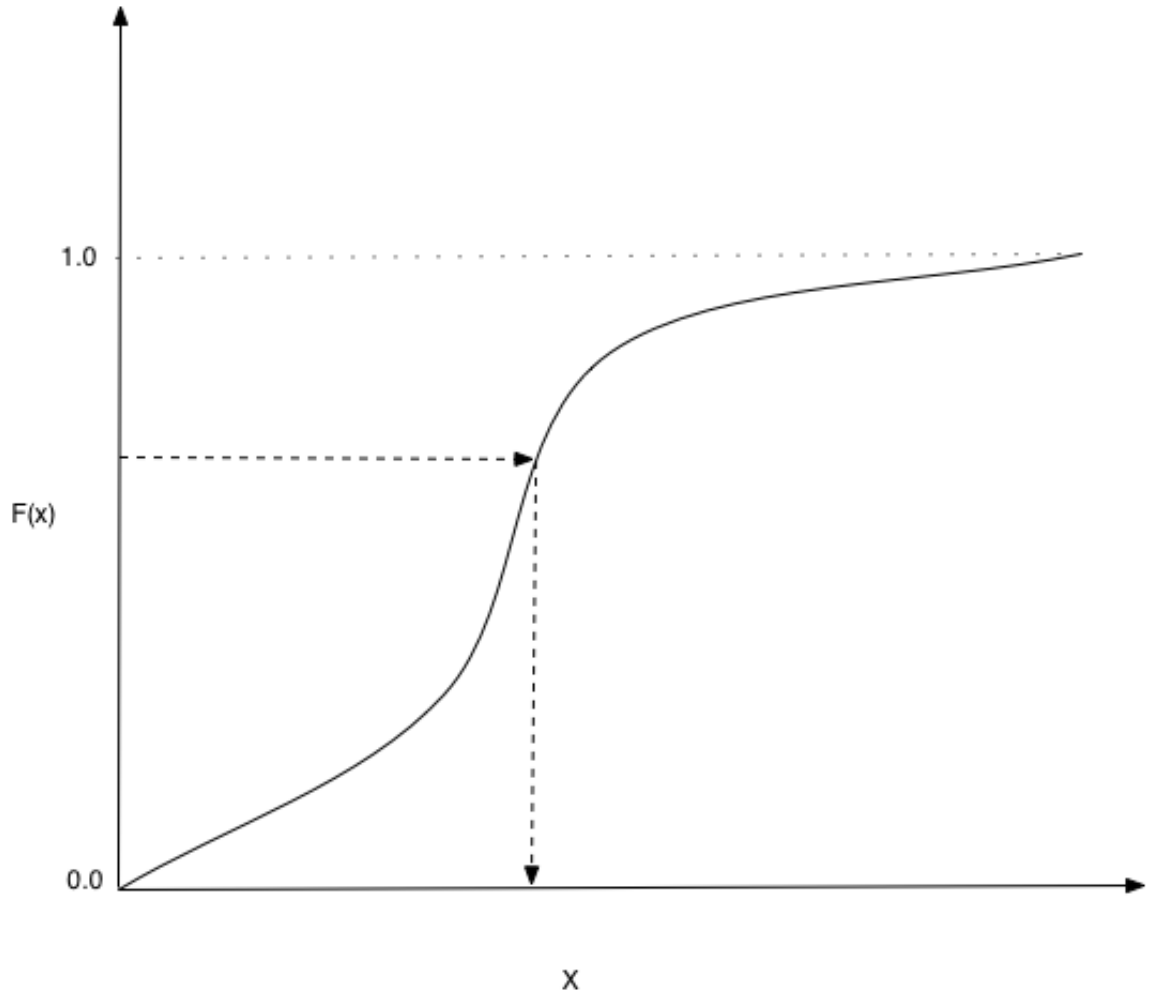
Triangular PDF



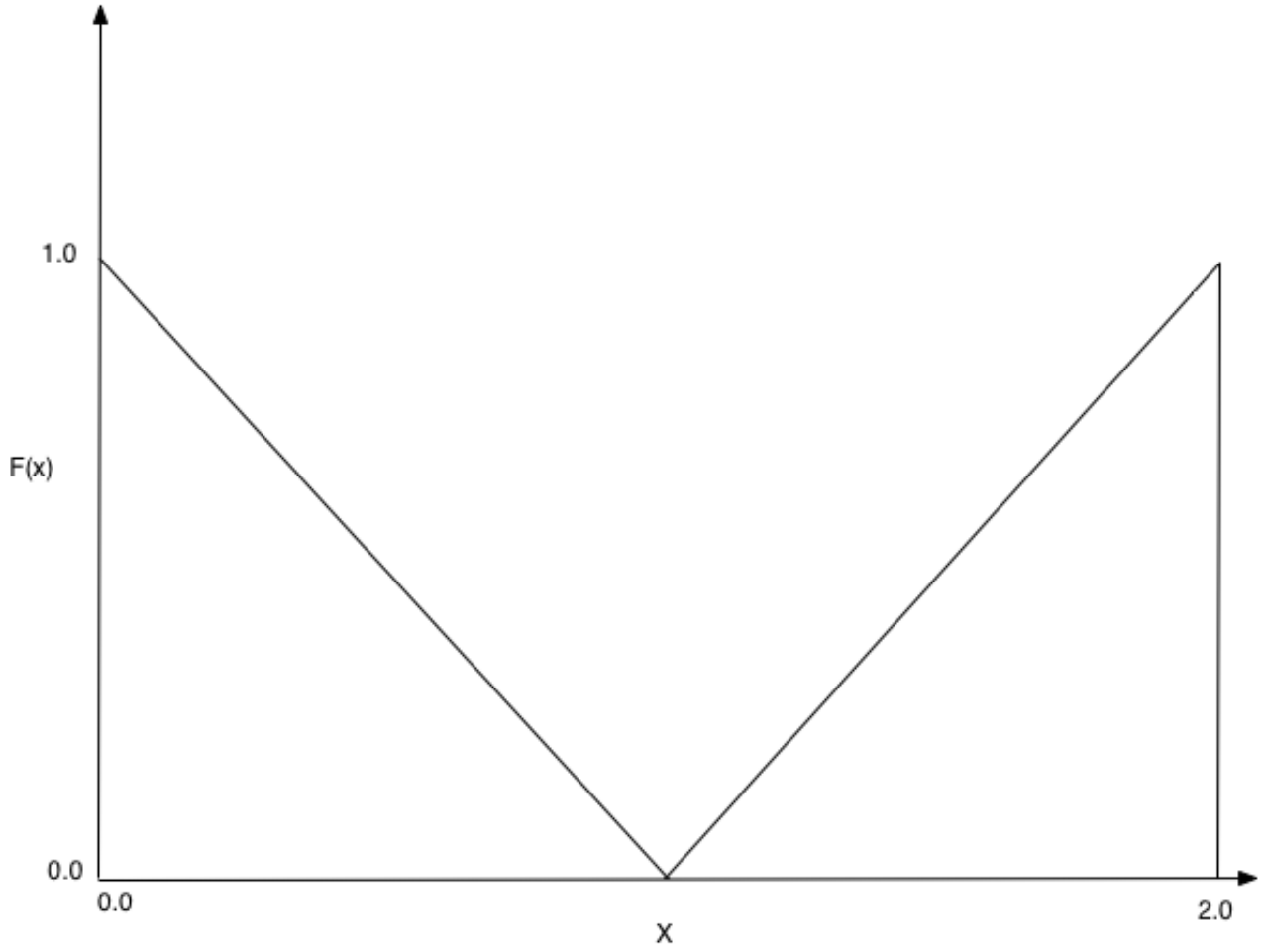
Corresponding CDF for Triangular PDF



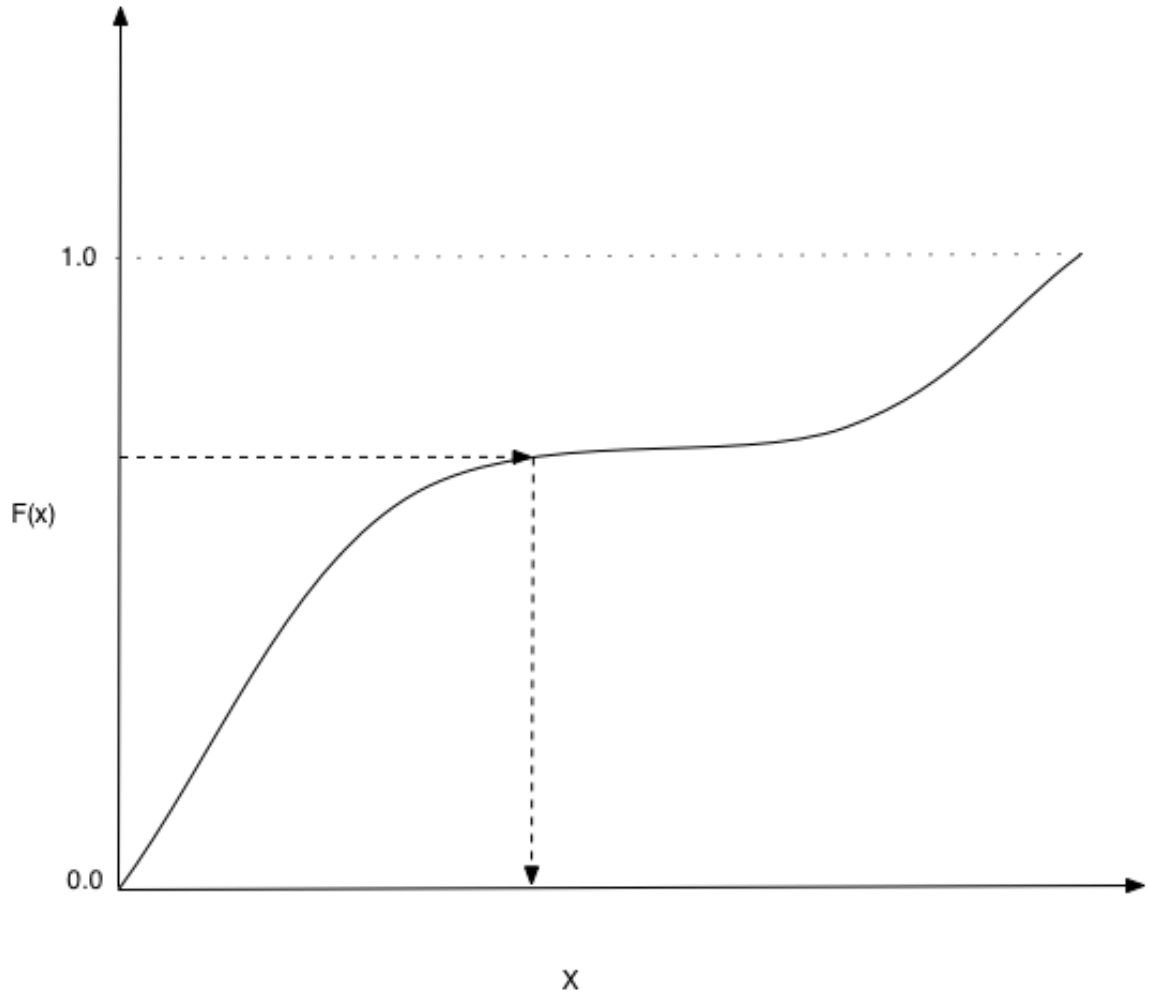
Inverse Sampling in Action



Inverse Triangular PDF



CDF for Inverse Triangular PDF



Inverse Transform Sampling

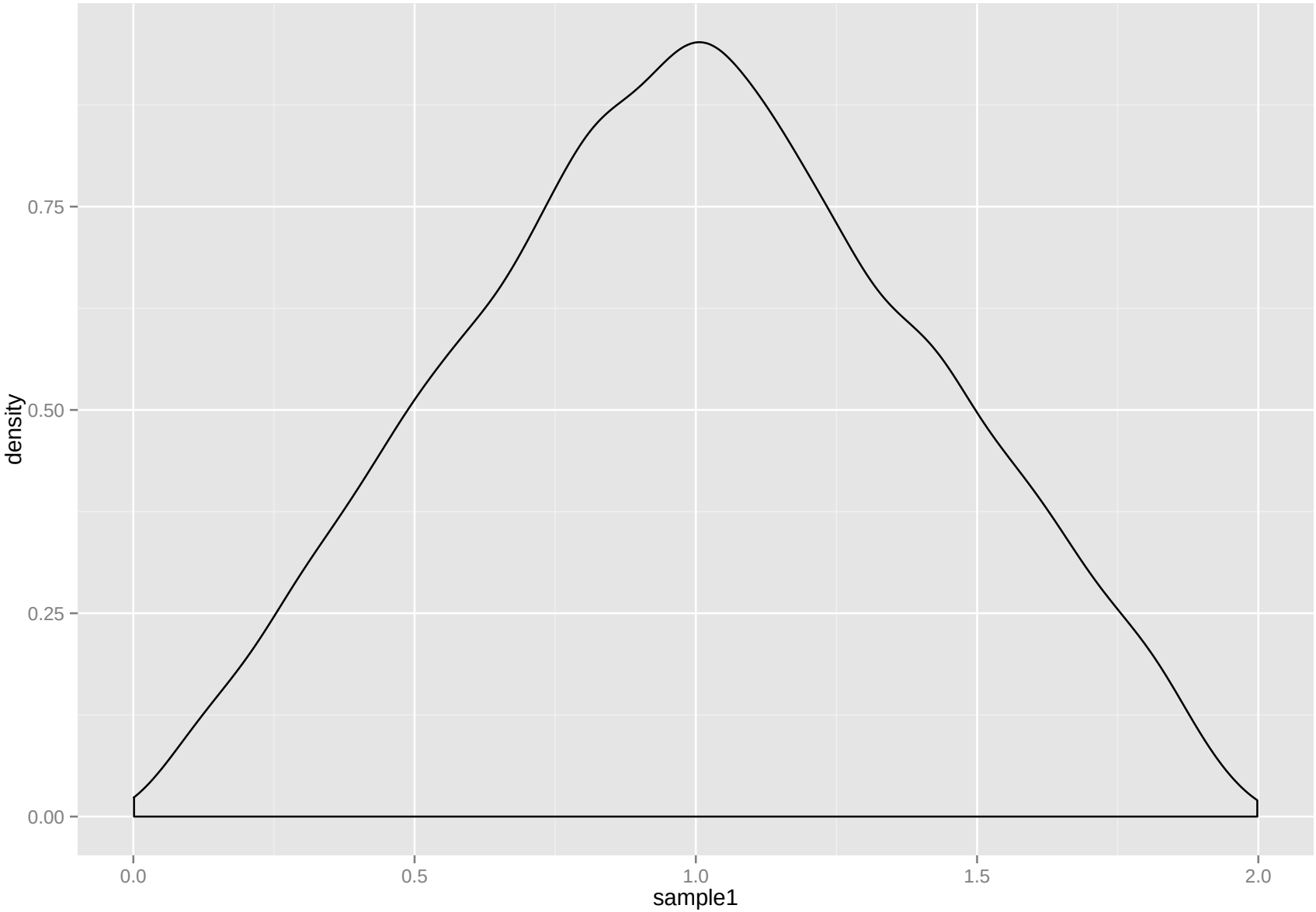
- Given $f(x) = x, x \in [0, 1]$ and $f(x) = 2 - x, x \in (1, 2]$
- We get F(x), the CDF as
  - $F(x) = \frac{x^2}{2}, x \in [0, 1]$ and $F(x) = 2x - \frac{x^2}{2} - 1, x \in (1, 2]$
- From this, we can compute an inverse function of F(x), $F^{-1}(x)$ such that $F^{-1}F(x) = x$
- A little bit of Algebra shows that we get the inverse function of the CDF as
  - $F^{-1}(y) = \sqrt{2y}, y \in [0, 0.5]$ and $F^{-1}(y) = 2 - \sqrt{2(1-y)}, y \in (0.5, 1]$.

Inverse Transform Sampling

Now, to sample from this inverse CDF, we can do the following:

```
invcdf <- function(y) {
    if (y >= 0 && y <= 1) {
        ret <- ifelse(y < 0.5,sqrt(2*y),2-sqrt(2*(1-y)))
    }
}

sample1 <- sapply(runif(20000),invcdf)
sdf = data.frame(sample1)
```

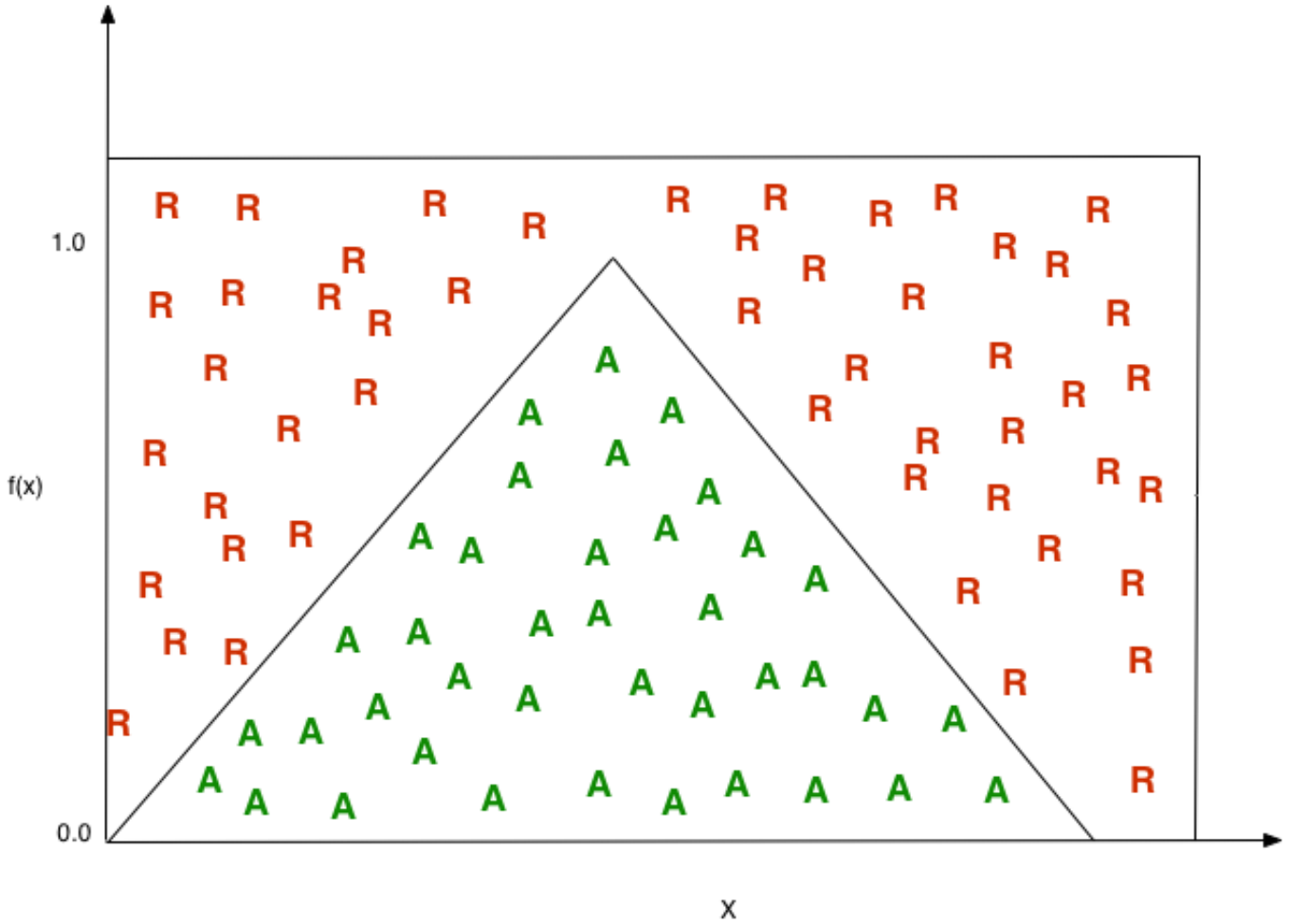
Inverse Transform Sampling



Rejection Sampling

- Choose a function M(x) that is strictly larger than f(x) over the range
- Uniformly sample (x,y) over the range of x and y
- Accept all samples that are under the curve of f(x) and reject all samples that are above f(x)

Rejection Sampling in Action

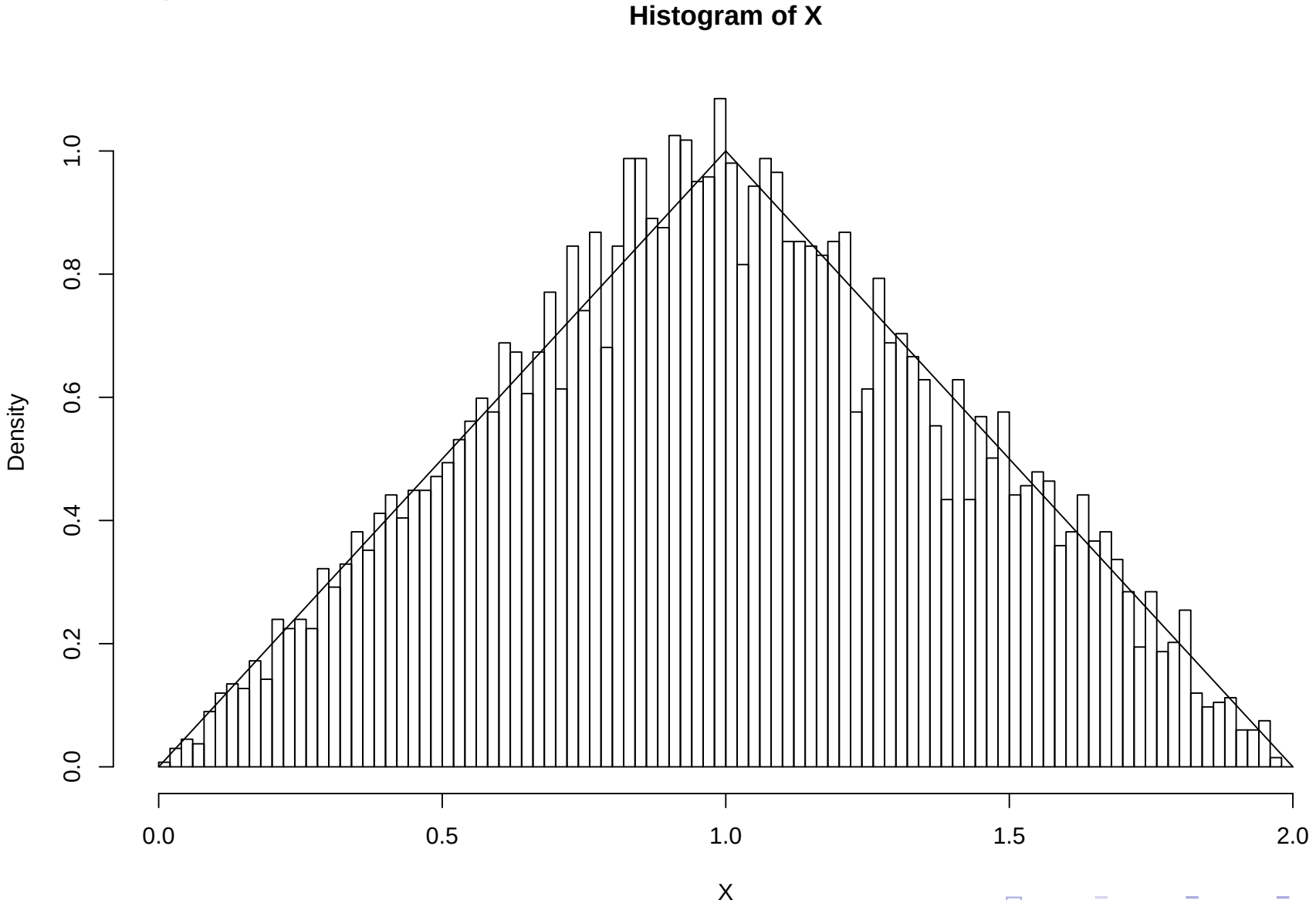


Rejection Sampling

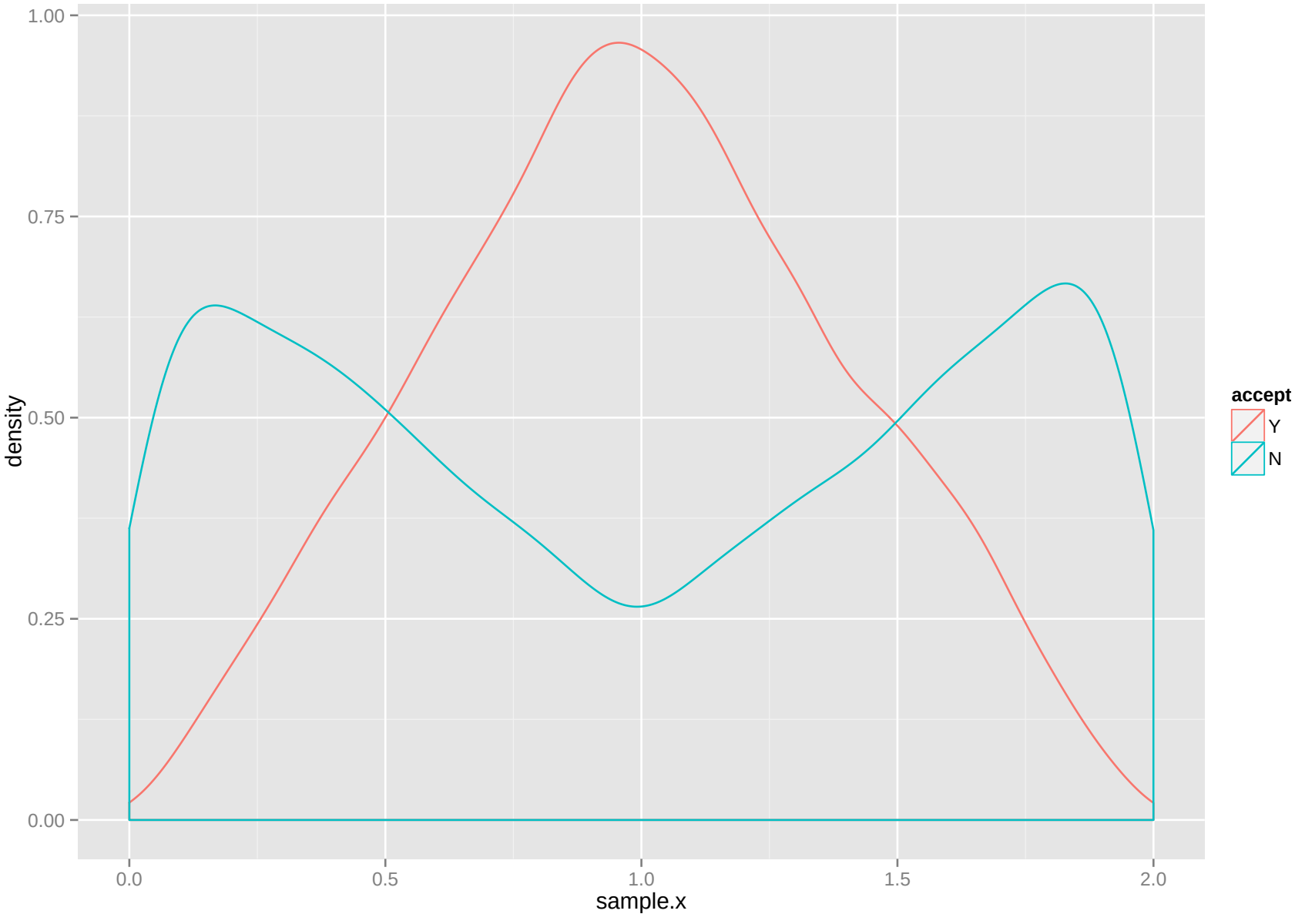
```
sample.x = runif(20000,0,2)
accept = c()
fx <- function(x) {if (x >= 0 && x <= 2) {y <- ifelse(x <=1

# dnorm(sample.x[i],0.5,0.175)
for(i in 1:length(sample.x)){
 U = runif(1, 0, 1)
 if(dunif(sample.x[i], 0, 2)*3*U <= fx(sample.x[i])) {
   accept[i] = 'Y'
 }
 else if(dunif(sample.x[i], 0, 2)*3*U > fx(sample.x[i])) {
  accept[i] = 'N'
 }
}

T = data.frame(sample.x, accept = factor(accept, levels= c(
```

Rejection Sampling

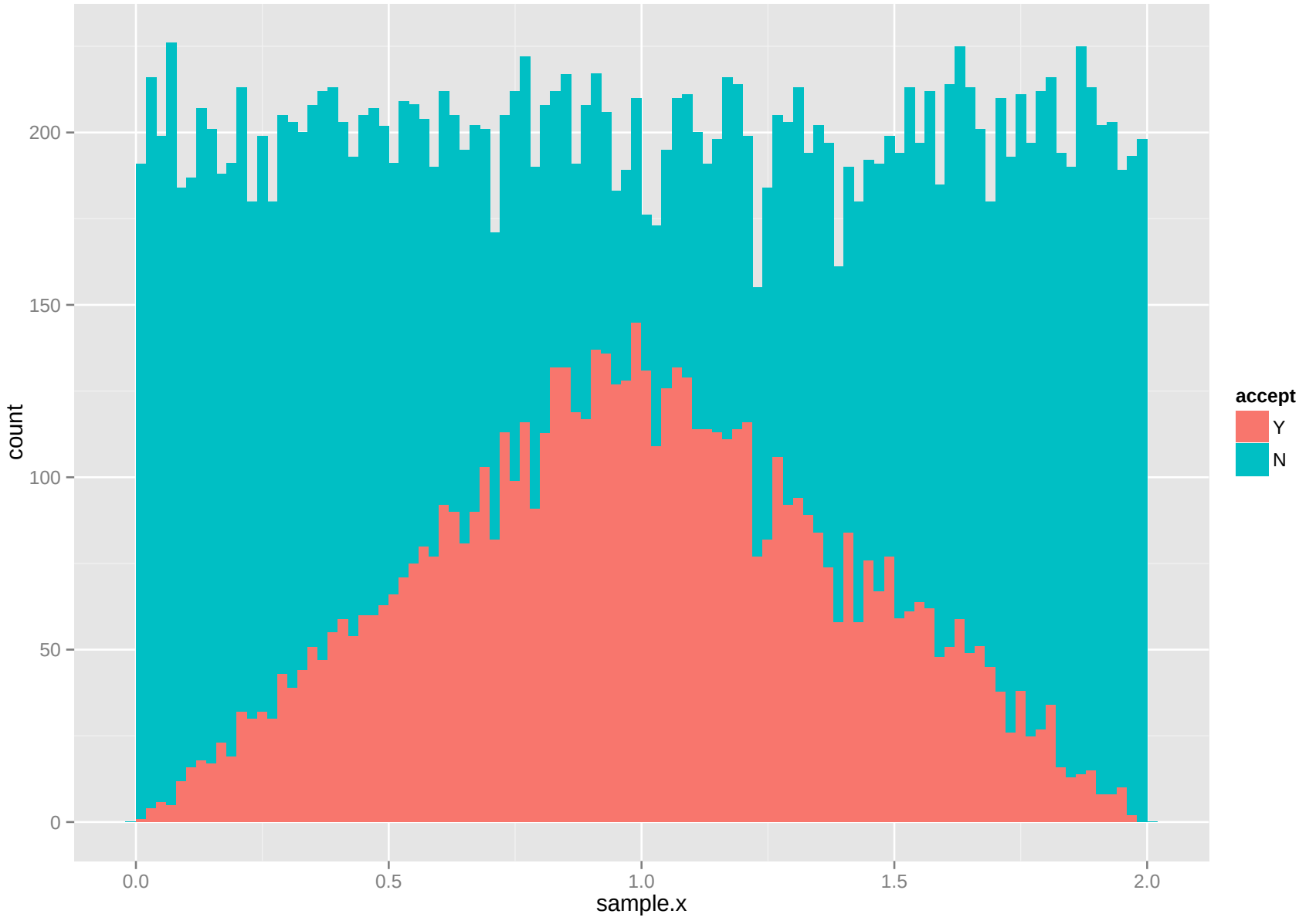
We can plot the results along with the true distribution with the following code.



Rejection Sampling



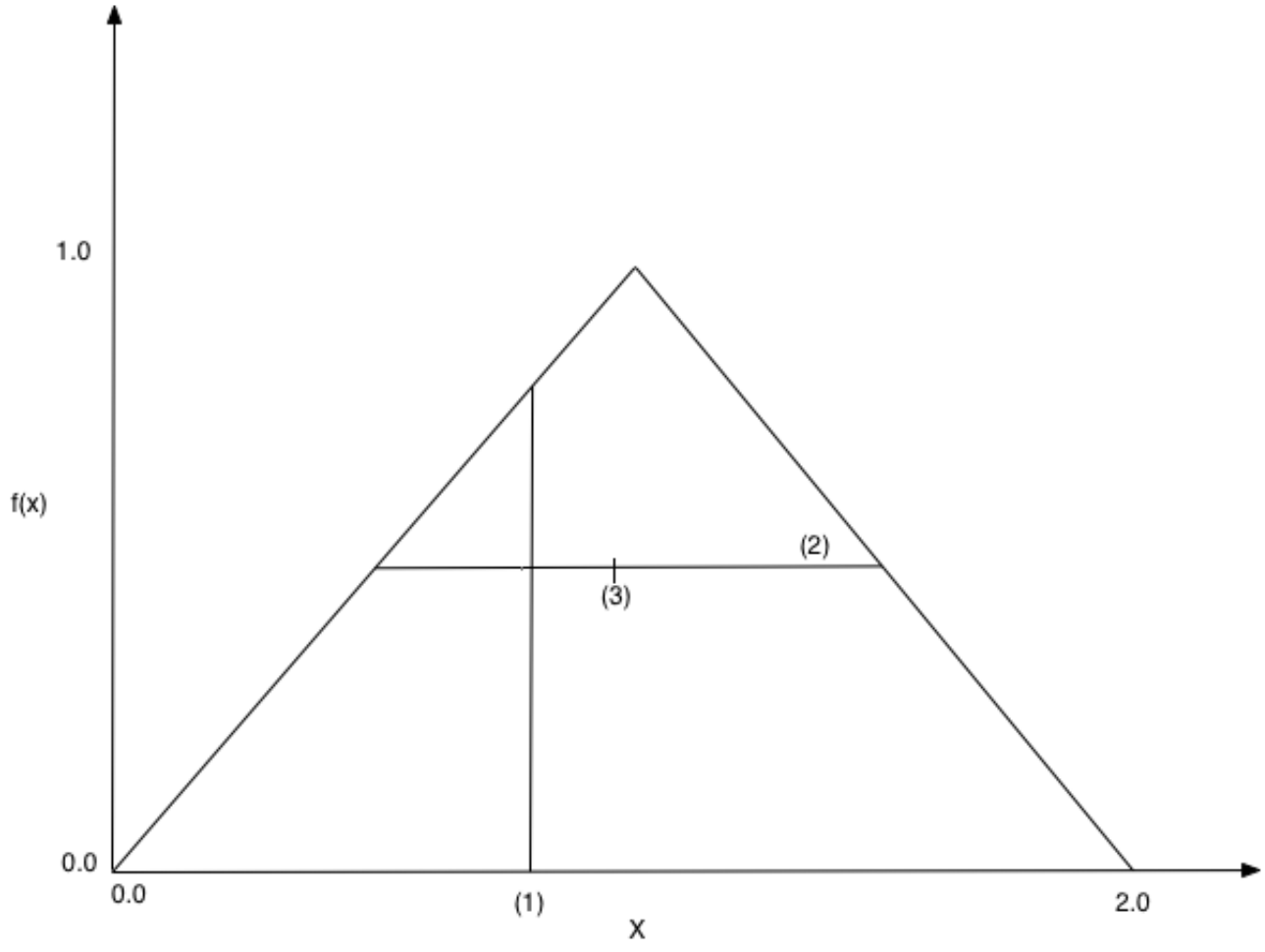
Rejection Sampling



Slice Sampling

- First select an $\hat{x}$ from the range of f(x)
- Then, vertically choose a $\hat{y}$ that is between 0 and $f(\hat{x})$
- Draw a horizontal slice at this $\hat{y}$
- Uniformly sample between $\hat{x}_{min}$ and $\hat{x}_{max}$ at this $\hat{y}$.
- Repeat with this as the new $\hat{x}$

Slice Sampling in Action



Slice Sampling in Action

Extracted from the mcmc tutorial (part of the diversitree package)

```
library(diversitree)
```

```
## Loading required package: deSolve
##
## Attaching package: 'deSolve'
##
## The following object is masked from 'package:graphics':
##
##     matplot
##
## Loading required package: ape
## Loading required package: subplex
## Loading required package: Rcpp
```

```
make.mvn <- function(mean, vcv) {
    logdet <- as.numeric(determinant(vcv, TRUE)$modulus)
```

Slice Sampling in Action

Our target distribution has mean 0, and a VCV with positive covariance between the two parameters.

```
vcv <- matrix(c(1, 0.25, 0.25, 0.75), 2, 2)
lik <- make.mvn(c(0, 0), vcv)
```

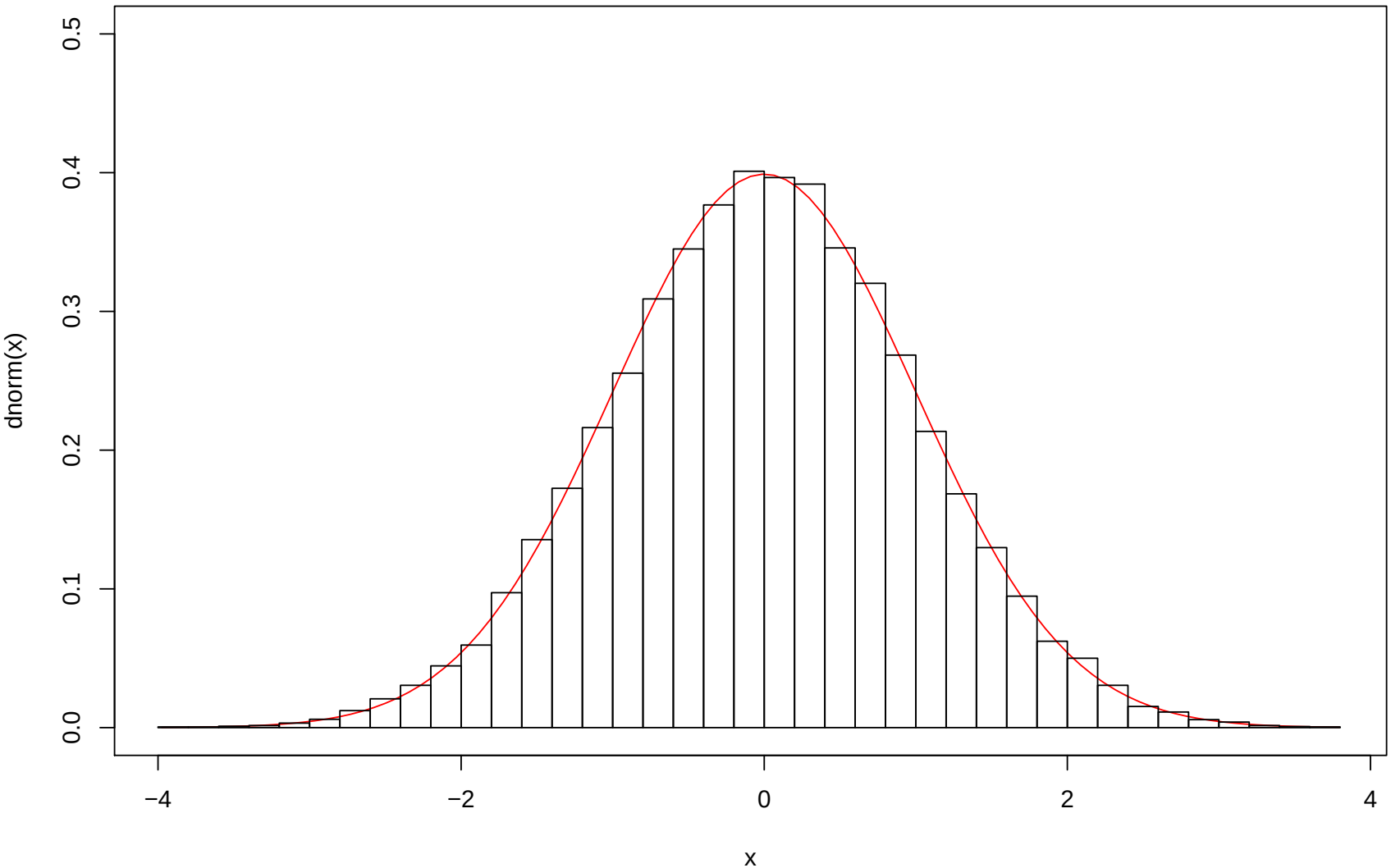
Sample 10,000 points from the distribution, starting at c(0, 0).

```
set.seed(1)
samples <- mcmc(lik, c(0, 0), 20000, 1, print.every = 1000
```

```
## 10000: {-0.2465, 1.6853} -> -3.90028
## 20000: {-0.6881, 0.0283} -> -1.91646
```

Slice Sampling in Action

The marginal distribution of V1 (the first axis of the distribution) should be a normal distribution with mean 0 and variance 1:



Slice Sampling with Triangular PDF

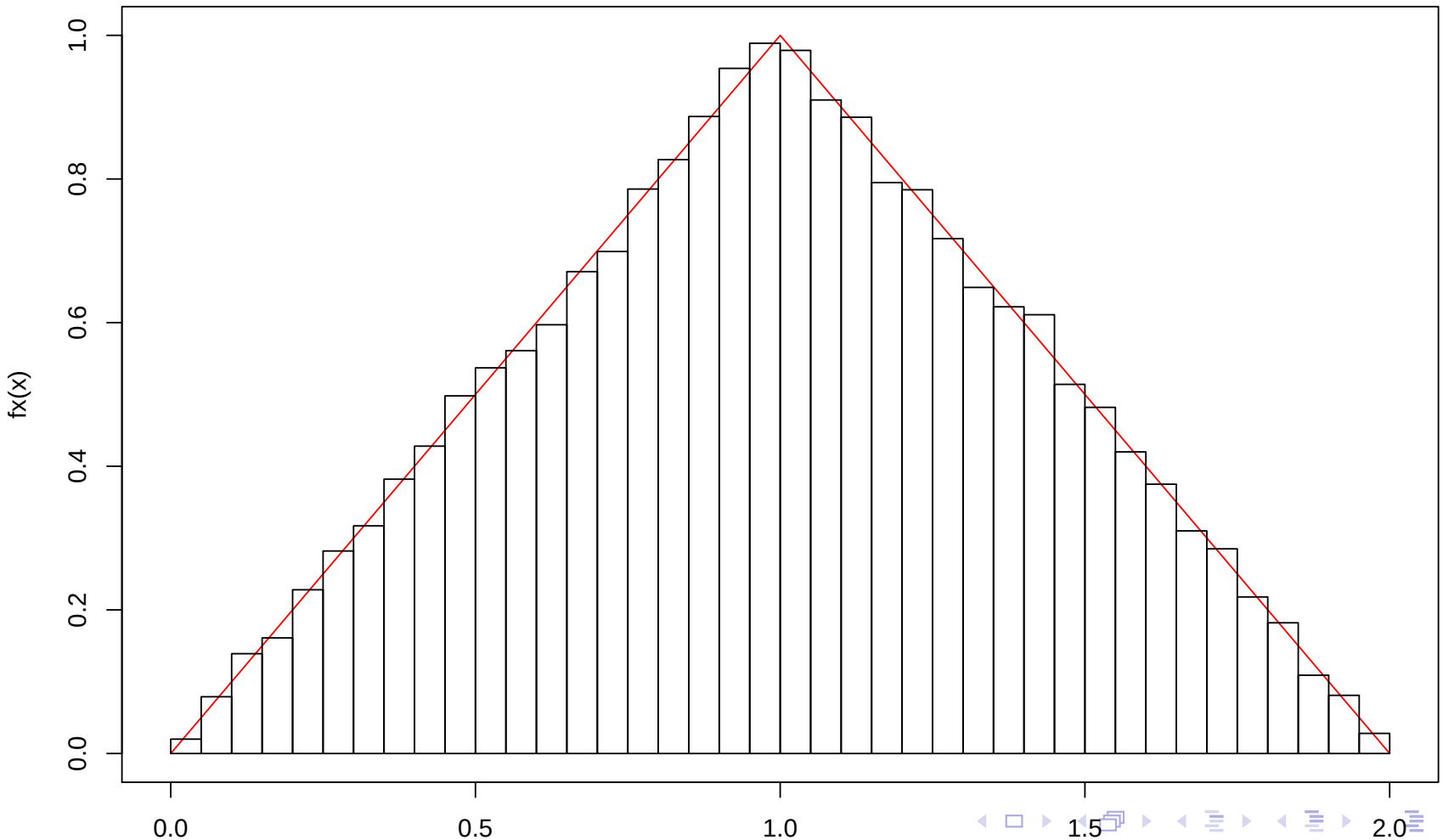
Try with Triangular PDF

```
set.seed(1)
lik <- function(x) { if (x >= 0 && x < 1) log(x) else log(2
samples <- mcmc(lik, 0.0001, 20000, 1, lower=0.0001, upper=
```

```
## 20000: {1.8357} -> -1.80636
```

Slice Sampling with Triangular PDF

```
## Warning in Ops.factor(left): '!' not meaningful for fact

## Warning in if (!add) {: the condition has length > 1 and
## element will be used
```

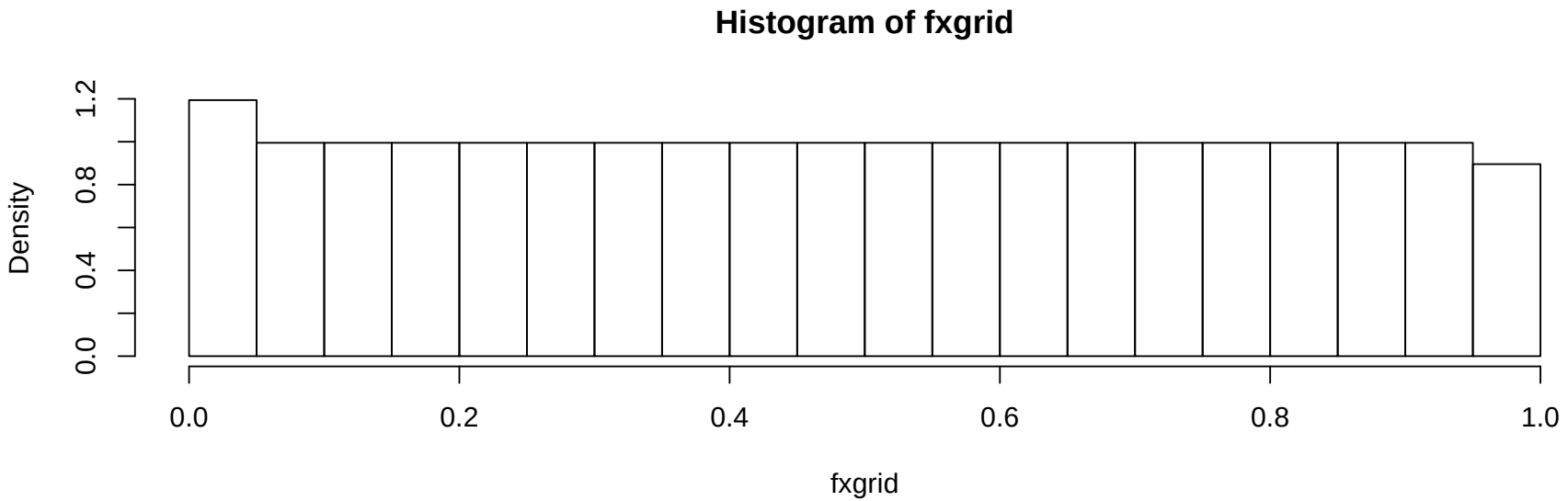


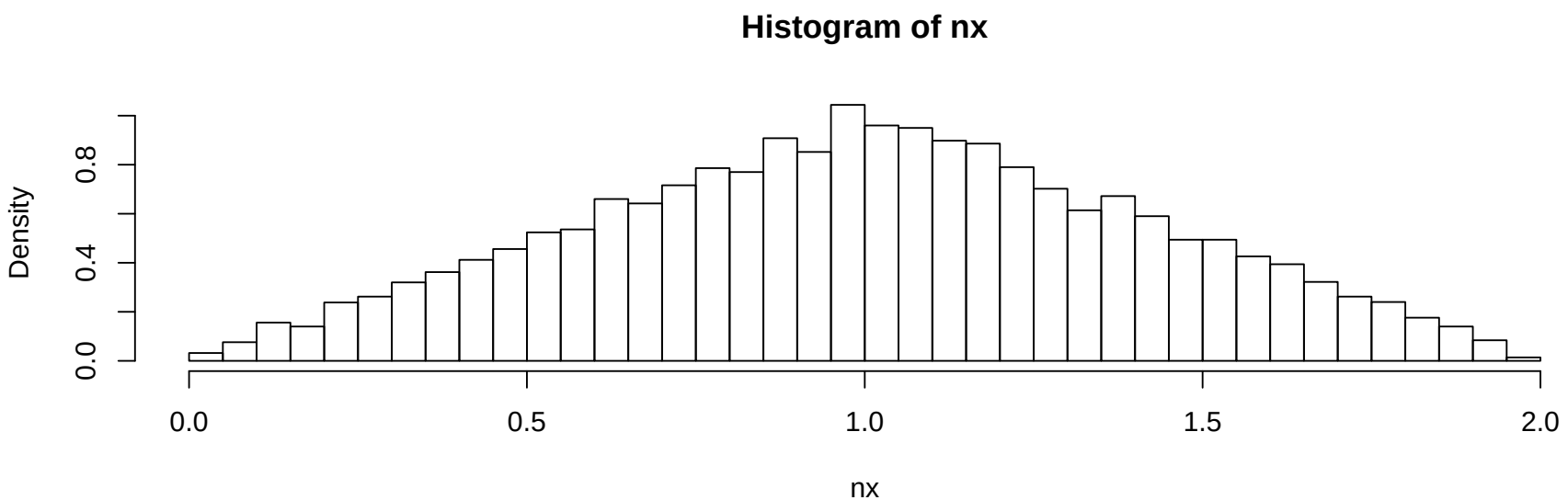
Using built-in R Sample function to do this without knowing any sampling

If you don't want to know how to do these sampling yourself, you can always just use the samp function in R

```
xgrid=seq(0,2,by=0.01)
fxgrid <- sapply(xgrid,fx)
nx <- sample(xgrid,10000,replace=TRUE,prob=fxgrid)
```

Compare sampling from a function and sampling from a PDF





R one liner

```
par(mfrow=c(1,1))
nx <- sample(seq(0,2,by=0.01),10000,replace=TRUE,prob=sappl
hist(nx,30,freq=F)
```