# Exercise set 3

Bjørn Christian Weinbach

01.09.2020

## Exercise 3.3

The Pareto(a, b) distribution has cdf

$$F(x) = 1 - \frac{b^a}{x^a} \tag{1}$$

Derive the probability inverse transformation $F^{-1}(U)$ and use the inverse transform method to to simulate a random sample from the Pareto(2, 2) dist.

The function $F^{-1}$ is derived by setting $F(X) = U$ and solving for X which gives us.

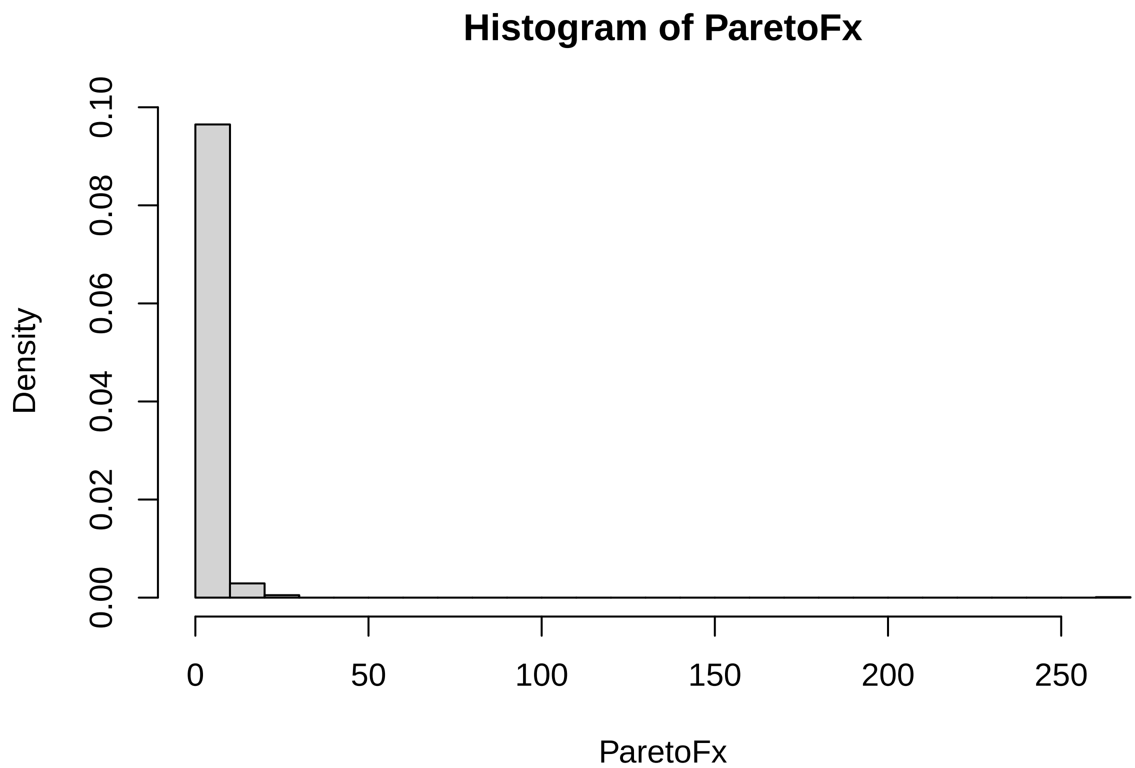
$$U = 1 - \frac{4}{x^2} \implies X = \frac{2}{\sqrt{(1-u)}} \tag{2}$$

and this we have $F^{-1}(U) = \frac{2}{\sqrt{(1-u)}}$

and in general terms $F^{-1}(U) = \frac{2}{\sqrt{(1-u)}}$

### Calculate using R

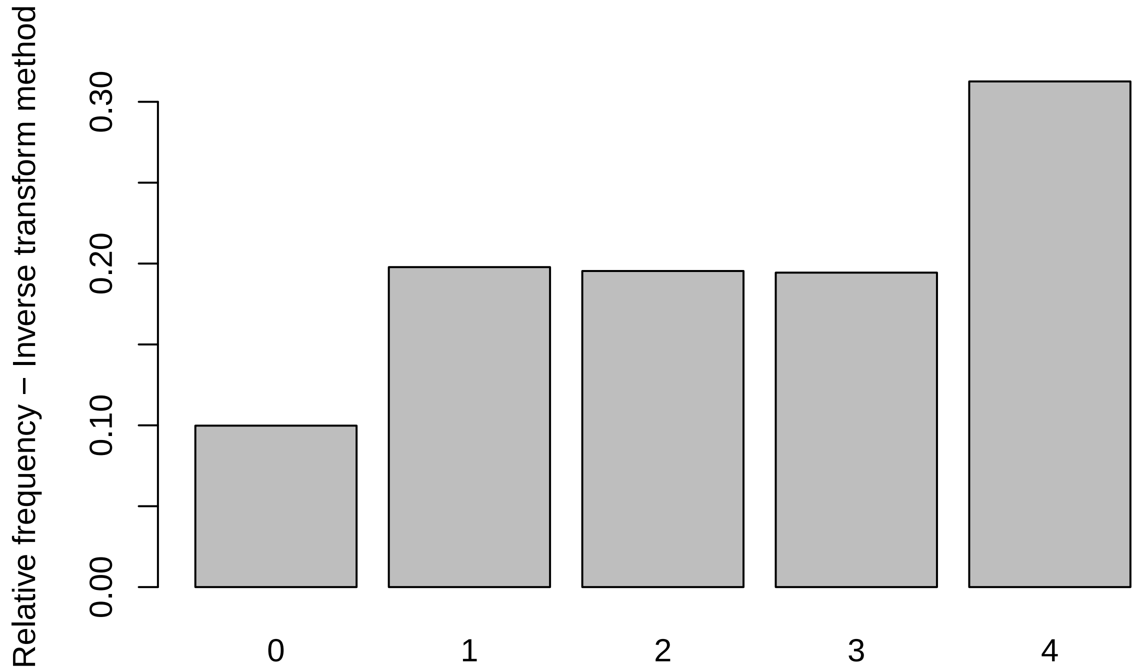
```r
simPareto <- function(nSim, a, b) {
  u <- runif(nSim)
  return(b*(1-u)^(-1/a))
}

nSim = 1000
ParetoFx <- simPareto(nSim = nSim, a = 2, b = 2)
hist(ParetoFx, breaks=30, prob=TRUE)
```



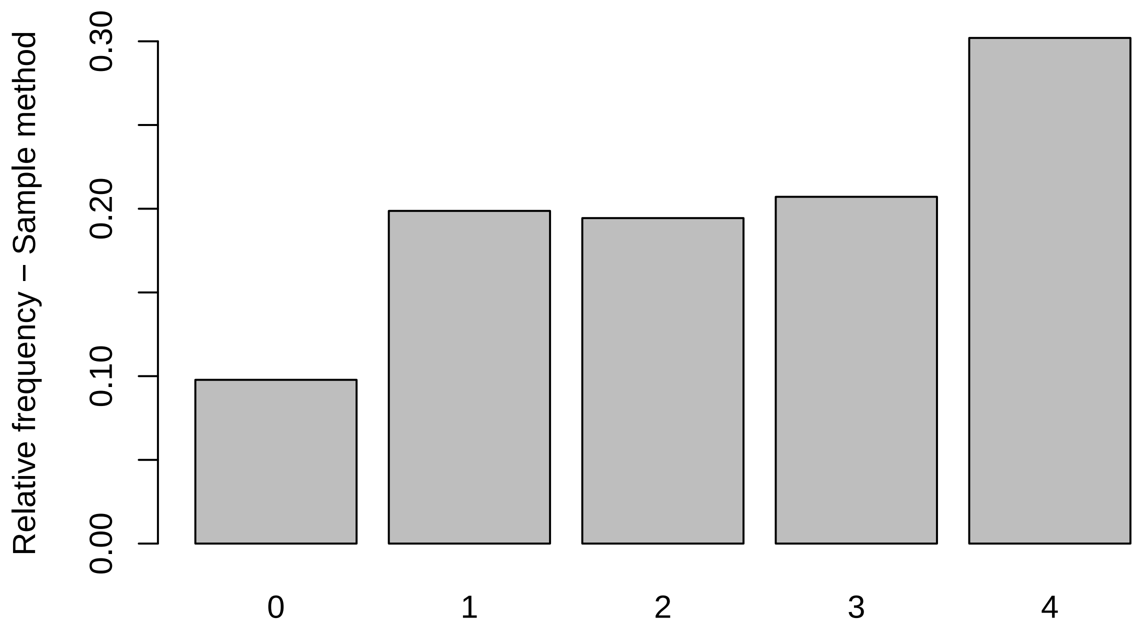

## Exercise 3.5

Use the inverse transform method to generate a random sample of size from distribution

```
# Alternatively the built in sample function can be used
## Generating data from the number of heads example where
## f(0)=1/8, f(1)=3/8, f(2)=3/8 and f(3)=1/8

# Function for simulating number of heads according to the distribution above
discrv <- function(Nsim){
   U <- runif(Nsim)
   X <- rep(0,Nsim)
   X[(U>0.1) & (U<=0.3)] <- 1
   X[(U>0.3) & (U<=0.5)] <- 2
   X[(U>0.5) & (U<=0.7)] <- 3
   X[U>0.7] <- 4
   return(X)
}

Nsim <- 10000

dfx <- discrv(Nsim)
relfreq <- table(dfx)/Nsim
barplot(relfreq,ylab="Relative frequency - Inverse transform method")
```




```
# Alternatively the built in sample function can be used
dfx2 <- sample(0:4,size=Nsim,replace=TRUE,prob=c(0.1,0.2,0.2,0.2,0.3))
relfreq <- table(dfx2)/Nsim
barplot(relfreq,ylab="Relative frequency - Sample method")
```

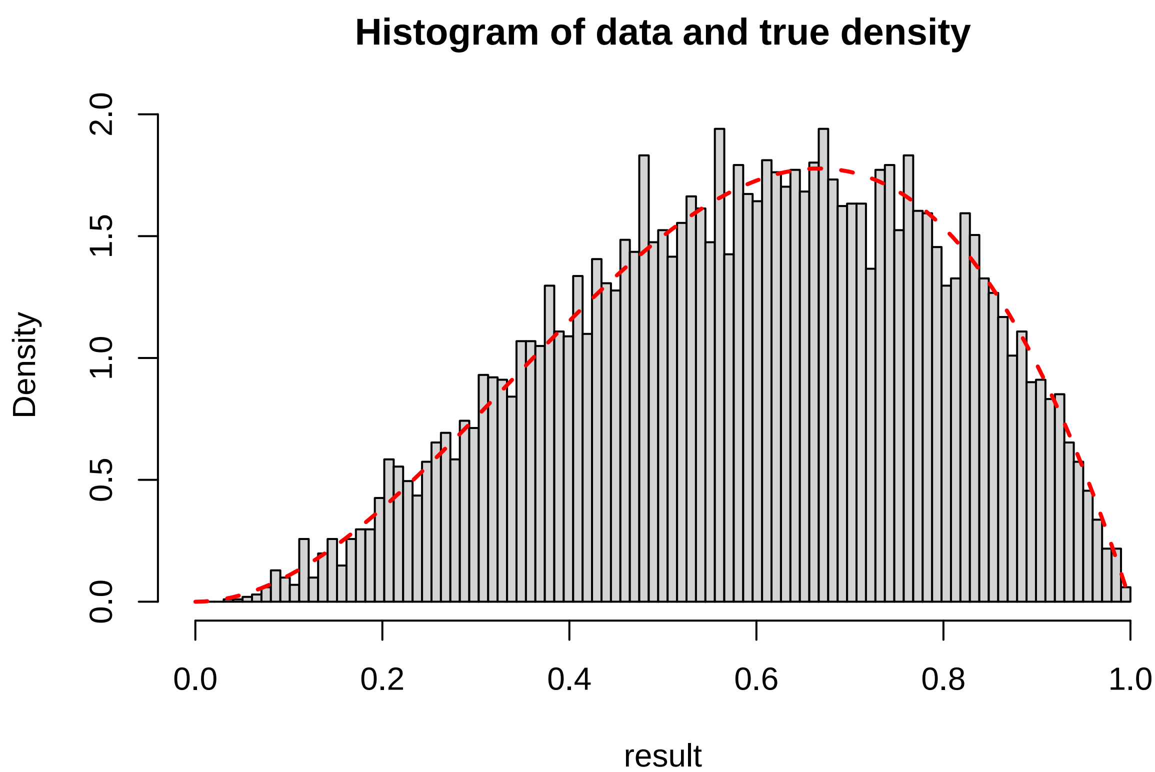



## Exercise 3.7

Write a function to generate a random sample of size n from the beta(a,b) distribution by the acceptance-rejection method. Sample size should be 1000 from the beta(3,2) distribution. Graph the sample with the theoretical Beta(3,2) density.

```r
## Function for acceptance rejection generation
rejectionBeta <- function(a, b, x, n) {
   f <- function(x) (x^(a-1)*(1-x)^(b-1) / beta(a, b))
   g <- function(x) 1
   C <- max(f(x)/g(x))

   naccepts <- 0
   result.sample <- rep(NA, n)

   while (naccepts < n) {
    y <- runif(1)
    u <- runif(1)

    if ( u <= f(y) / (C*g(y)) ) {
      naccepts <- naccepts + 1
      result.sample[naccepts] = y
    }
   }

   result.sample
```

```
}


# Sequence for x-axis
x <- seq(0, 1, 0.01)
# Sample from distribution
result <- rejectionBeta(3, 2, x, 10000)
# The exact function for plotting
f <- function(x) (x^(3-1)*(1-x)^(2-1) / beta(3, 2))

# Histogram of simulated data with true density on top as red line
hist(result, prob = TRUE,breaks=seq(0,1,length.out=max(10,sqrt(Nsim))),
     main="Histogram of data and true density")
curve(f(x), col = "red", lty = 2, lwd = 2, add = TRUE,xlim=c(0,1))
```




## Exercise 3.17

### a = 3, b = 2

```
Niter <- 1000
Nsim <- 5000

system.time(for (i in 1:Niter)
   rejectionBeta(3, 2, x, Nsim))
```

```
##    user  system elapsed
##   38.22    0.01   38.23
```

```
system.time(for (i in 1:Niter)
   rbeta(Nsim, shape1=3, shape2=2))
```

```
##    user  system elapsed
##    0.83    0.00    0.83
```

### a = 1, b = 1

```
Niter <- 1000
Nsim <- 5000

system.time(for (i in 1:Niter)
   rejectionBeta(1, 1, x, Nsim))
```

```
##    user  system elapsed
##   21.00    0.02   21.03
```

```
system.time(for (i in 1:Niter)
   rbeta(Nsim, shape1=1, shape2=1))
```

```
##    user  system elapsed
##    0.84    0.00    0.84
```

## Exercise 1

a. Simulate the probability that component 1 and component 2 is working.

```
library("Rlab")                                  # Load Rlab package
```

```
## Rlab 2.15.1 attached.
```

```
##
## Attaching package: 'Rlab'
```

```
## The following objects are masked from 'package:stats':
##
##     dexp, dgamma, dweibull, pexp, pgamma, pweibull, qexp, qgamma,
##     qweibull, rexp, rgamma, rweibull
```

```
## The following object is masked from 'package:datasets':
##
##     precip
```

```
n <- 10
s <- 10000
mat <- matrix(rbern(n*s, c(0.7,0.95,0.95,0.95,0.99,0.99,0.92,0.92,0.92,0.7)), n, s)
p <- numeric(4)
p[1] <- (sum(mat[1,]) / s) * (sum(mat[2,]) / s)
```

b. The system works if and only if at least one of components 1 or 2 works.

```
p[2] <- sum(mat[1,] / s) + sum(mat[2,] / s) - p[1]
```

c. The system works if and only if both component 1 and component 2 and at least one of components 3 or 4 work.

```r
p[3] <- p[1] * (sum(mat[3,] / s) + sum(mat[4,] / s) - (sum(mat[3,]) / s) * (sum(mat[4,]) / s))
```

d. The system works if and only if at least 7 of the 10 components work.

```r
p[4] <- sum(colSums(mat) < 7) / s
```

#### Calculated probabilities

```r
p
```

```
## [1] 0.6634800 0.9849200 0.6618118 0.0069000
```

# Exercise 2

a. Find conditional and unconditional probability of find larger than 8

Simulate $P(X > 8 | field found) using inverse transform method

```r
triangle <- function(n, a, b, c) {
  f <- function (x) {
    return((c-a)/(b-a))
  }
  u <- runif(n)
  sample <- ifelse(u < f(c),
         a + sqrt(u*(b-a)*(c-a)),
         b - sqrt((1-u)*(b-a)*(b-c))
        )
  return(sample)
}


sims <- 10000
a <- 2
b <- 10
c <- 6

vals <- triangle(sims, 2, 10, 6)
p1 <- sum(vals > 8) / sims
p2 <- p1 * 0.4
```

b. By simulation and the risk-weighted and the non risk-weighted expectation for the total resource R

```r
tenTriangles <- function(Nsim) {
   n1 <- sum(triangle(Nsim, 2, 6, 4) / sims)
   n2 <- sum(triangle(Nsim, 3, 11, 7) / sims)
   n3 <- sum(triangle(Nsim, 2, 6, 4) / sims)
   n4 <- sum(triangle(Nsim, 1, 9, 5) / sims)
   n5 <- sum(triangle(Nsim, 8, 10, 9) / sims)
   n6 <- sum(triangle(Nsim, 5, 9, 7) / sims)
   n7 <- sum(triangle(Nsim, 2, 6, 4) / sims)
   n8 <- sum(triangle(Nsim, 3, 5, 4) / sims)
   n9 <- sum(triangle(Nsim, 8, 12, 10) / sims)
   n10 <- sum(triangle(Nsim, 3, 7, 5) / sims)

   mat <- matrix(0, 10, 2)
```

```
  mat[1,] <- c(n1, n1*0.8)
  mat[2,] <- c(n2, n2*0.3)
  mat[3,] <- c(n3, n3*0.6)
  mat[4,] <- c(n4, n4*0.6)
  mat[5,] <- c(n5, n5*0.5)
  mat[6,] <- c(n6, n6*0.9)
  mat[7,] <- c(n7, n7*0.5)
  mat[8,] <- c(n8, n8*0.8)
  mat[9,] <- c(n9, n9*0.4)
  mat[10,] <- c(n10, n10*0.4)

  return(c(sum(mat[,1]), sum(mat[,2])))
}

tenTriangles(10000)
```

```
## [1] 58.99372 32.70854
```

d. By simulation and an estimate of the accompanying standard deviation for the two expectations. Find how many replications are necessary to be 95% certain that the errors in the estimates of the means are not exceeding 0.2.

```
sample <- 100
nsim <- 10000
matrix <- matrix(nrow = 0, ncol = 2)
for (i in 1:sample) {
  matrix <- rbind(matrix, tenTriangles(nsim))
}
sdr1 <- sd(matrix[,1])
sdr2 <- sd(matrix[,2])
n1 <- (4*sdr1^2)/(0.2^2)
n2 <- (4*sdr2^2)/(0.2^2)
n1
```

```
## [1] 0.1088601
```

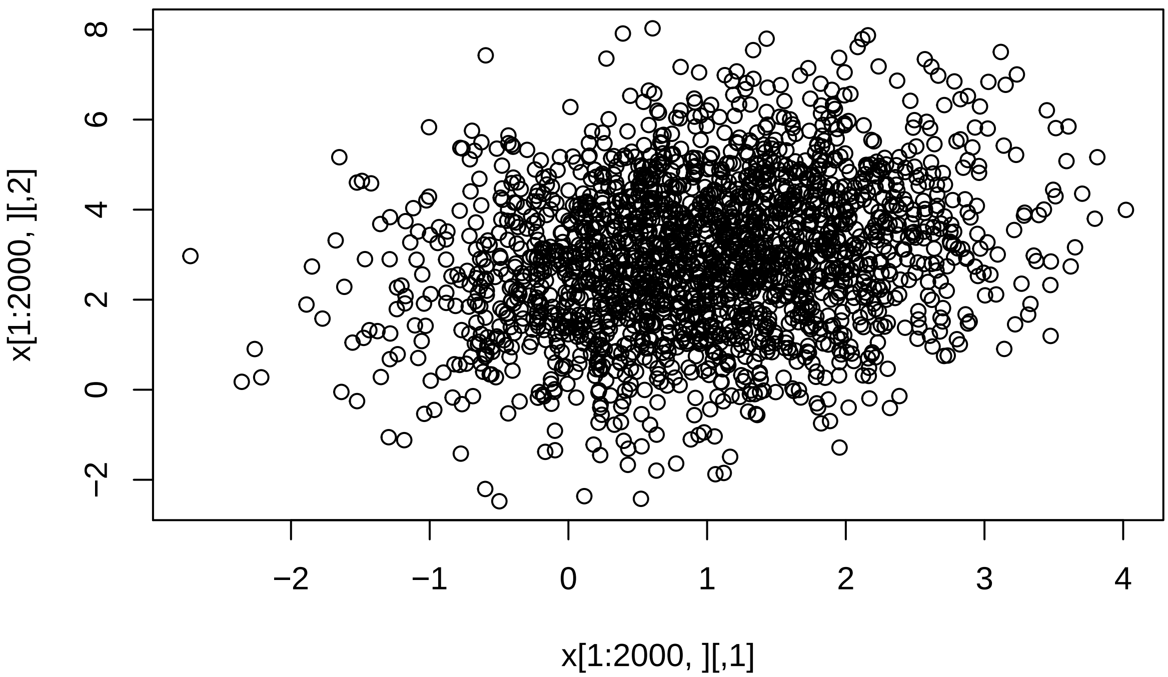
```
n2
```

```
## [1] 0.03579746
```

# Exercise 3

a.

```
library(mvtnorm)

sims <- 10000
sigma <- matrix(c(1,0.4041,0.4041,3), ncol=2)
x <- rmvnorm(n=sims, mean=c(1, 3), sigma=sigma)
p <- sum((x[,1] + x[,2]) >= 3) / sims
plot(x[1:2000,])
```

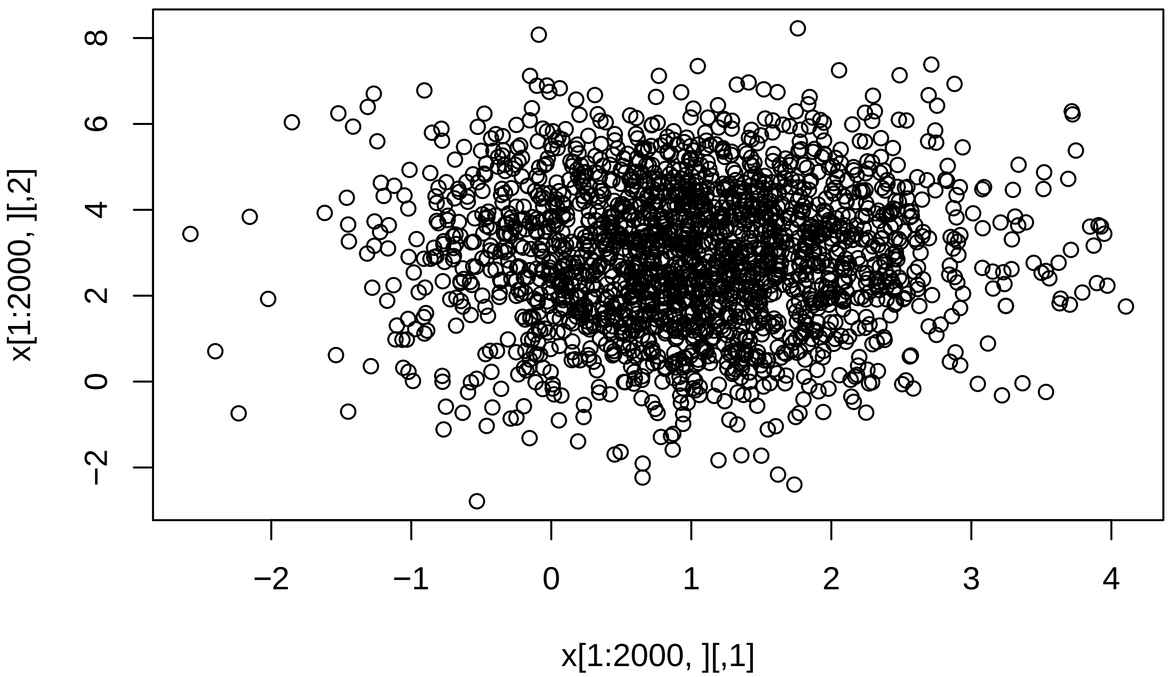



```
p
```

```
## [1] 0.6751
```

b.

```
library(mvtnorm)

sims <- 10000
sigma <- matrix(c(1,0,0,3), ncol=2)
x <- rmvnorm(n=sims, mean=c(1, 3), sigma=sigma)
p <- sum((x[,1] + x[,2]) >= 3) / sims
plot(x[1:2000,])
```




```
p
```

```
## [1] 0.6789
```