Statistical Computation and Simulation

Generating empirical cdf from some data (using distr package) using it to generate new random samples

```
# install.packages('distr')

n <- 1e3
xs <- rnorm(n,.5,.2)    # a random sample

library(distr)
```

```
## Warning: package 'distr' was built under R version 3.5.1
```

```
## Loading required package: startupmsg
```

```
## Warning: package 'startupmsg' was built under R version 3.5.1
```

```
## Utilities for Start-Up Messages (version 0.9.5)
```

```
## For more information see ?"startupmsg", NEWS("startupmsg")
```

```
## Loading required package: sfsmisc
```

```
## Warning: package 'sfsmisc' was built under R version 3.5.1
```

```
## Object Oriented Implementation of Distributions (version 2.7.0)
```

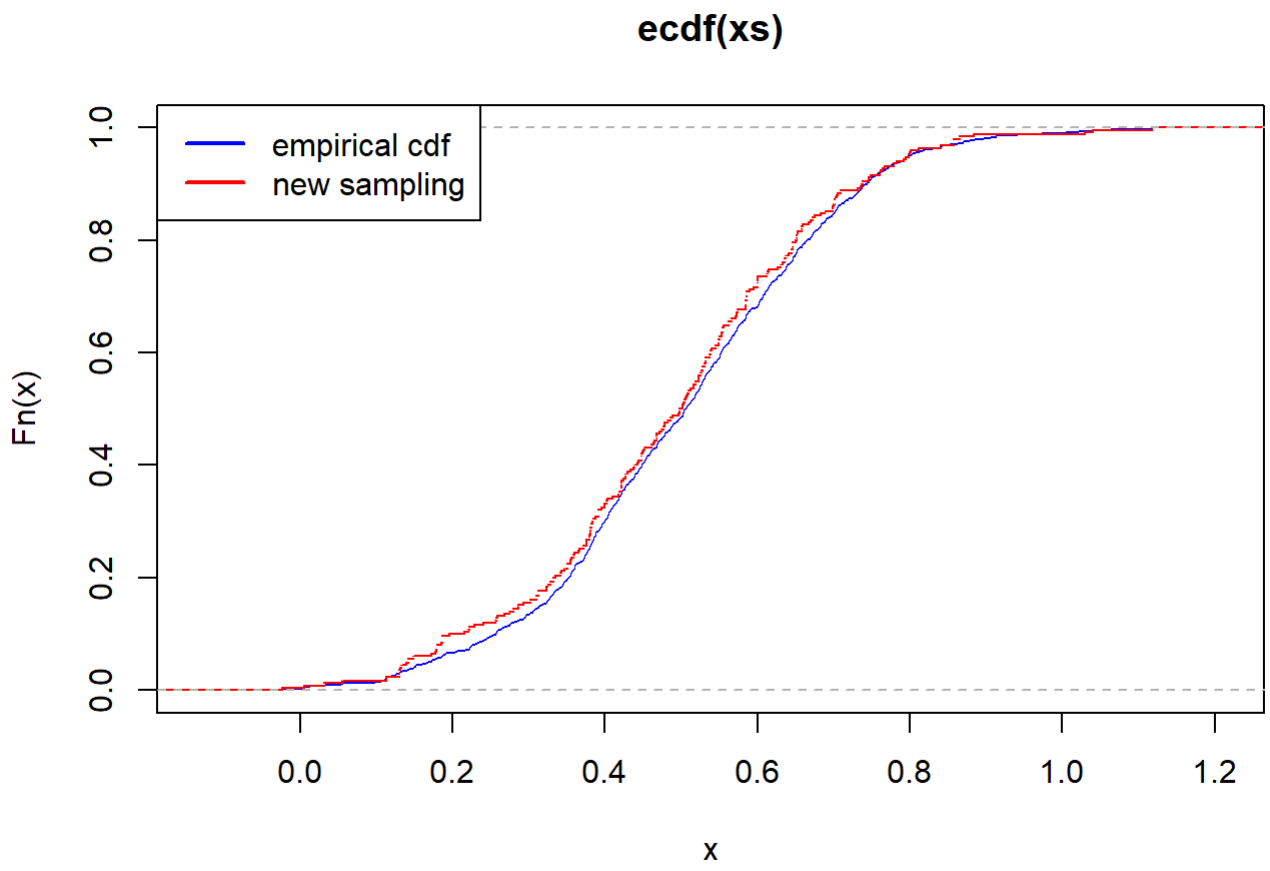
```
## Attention: Arithmetics on distribution objects are understood as operations on corresponding
random variables (r.v.s); see distrARITH().
## Some functions from package 'stats' are intentionally masked ---see distrMASK().
## Note that global options are controlled by distroptions() ---c.f. ?"distroptions".
```

```
## For more information see ?"distr", NEWS("distr"), as well as
##   http://distr.r-forge.r-project.org/
## Package "distrDoc" provides a vignette to this package as well as to several extension packag
es; try vignette("distr").
```

```
##
## Attaching package: 'distr'
```

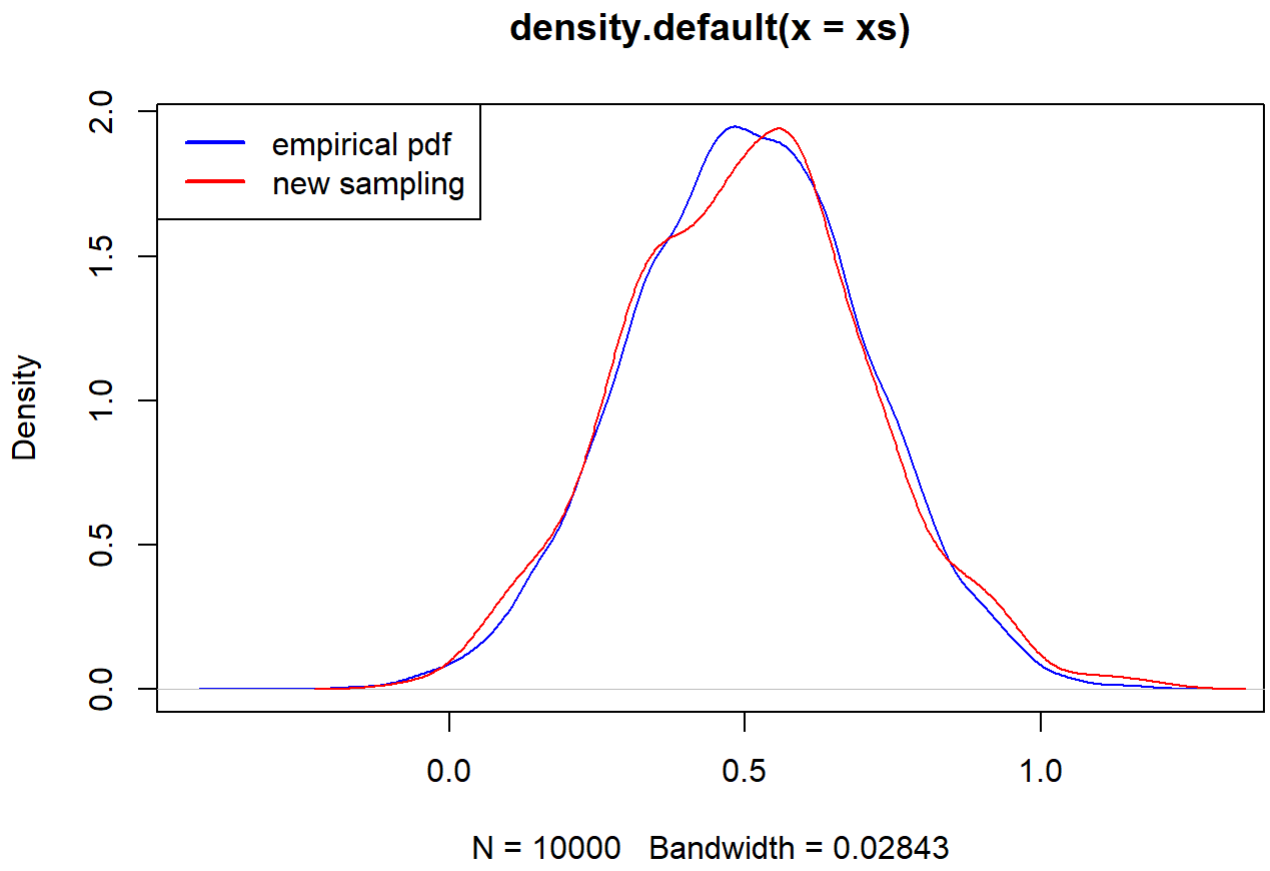
```
## The following objects are masked from 'package:stats':
##
##     df, qqplot, sd
```

```
emp.cdf <- DiscreteDistribution(xs)         # fit an empirical cdf
new.xs <- emp.cdf@r(250)                    # generate new samples


# plot original samples vs new samples
plot(ecdf(xs), col="blue")
plot(ecdf(new.xs), col="red", pch=".", add=T)
legend("topleft",c("empirical cdf","new sampling"), col=c("blue","red"), lwd=2)
```



Another alternative is using the density function which uses (by default gaussian) kernels to estimate the density of the sample

```
n <- 1e4
xs <- rnorm(n,.5,.2)

d <- density(xs)
new_xs <- sample(d$x, replace = TRUE, prob=d$y)

#plot original sample vs new sample

plot(density(xs), col= "blue")
lines(density(new_xs), col = "red")
legend("topleft",c("empirical pdf", "new sampling"), col = c("blue","red"), lwd =2)
```

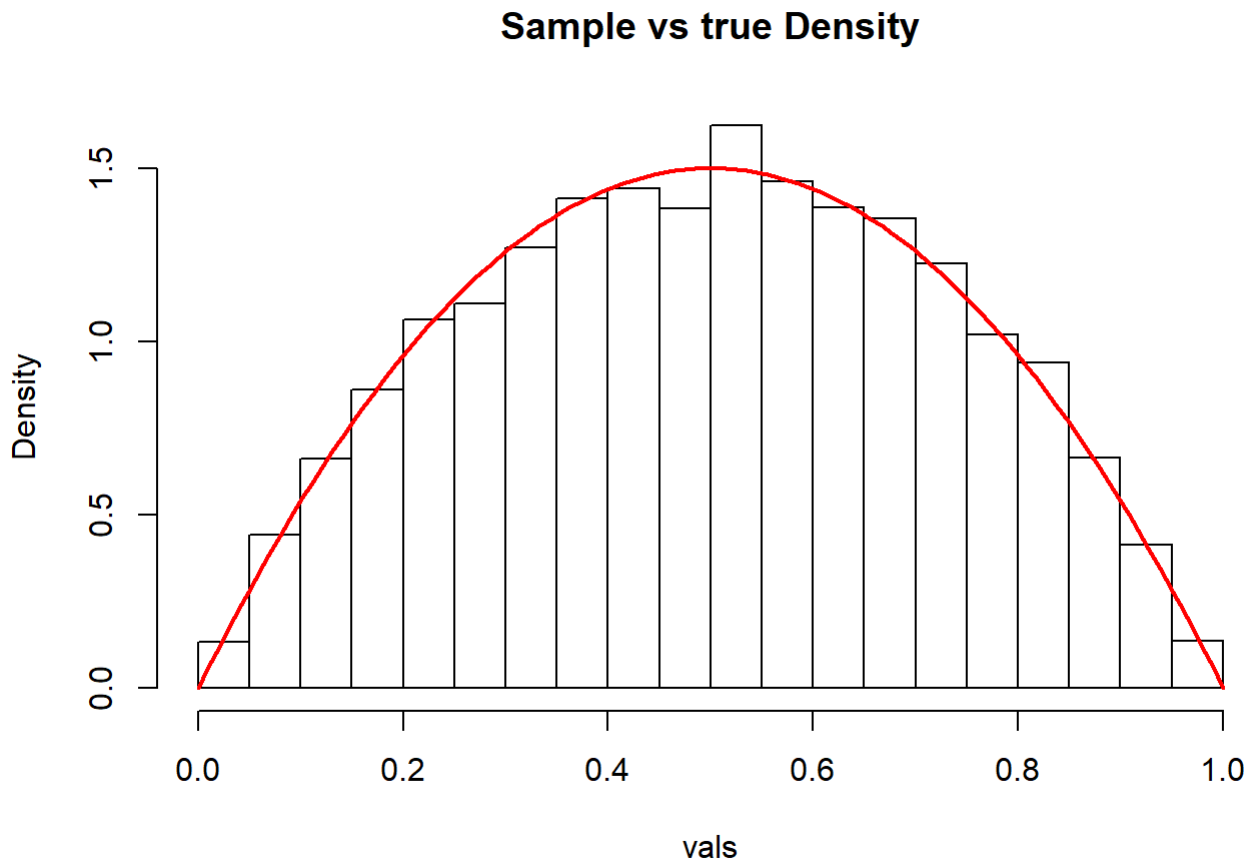


Acceptance - Rejection method

Generate n samples from f using rejection sampling with g (rg samples from g)

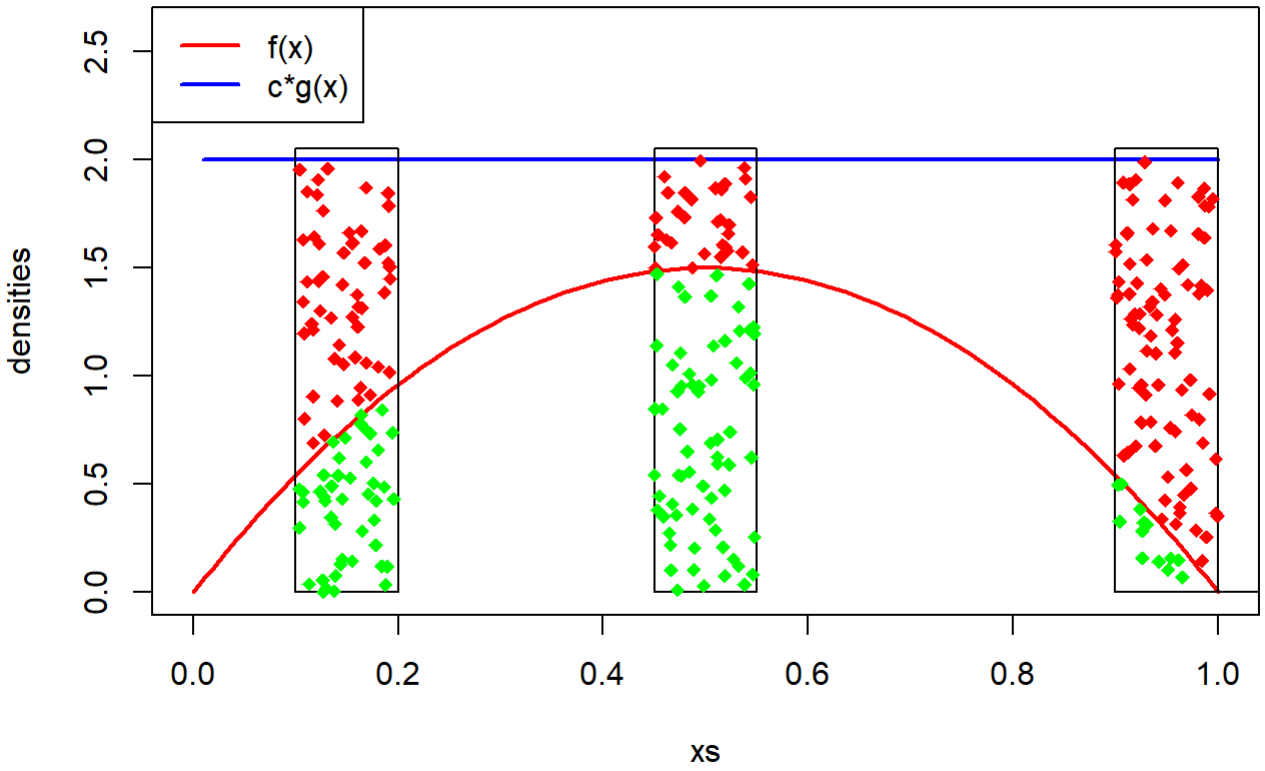
```
accept.reject <- function(f, c, g, rg, n) {
  n.accepts     <- 0
  result.sample <- rep(NA, n)

  while (n.accepts < n) {
    y <- rg(1)               # step 1
    u <- runif(1,0,1)        # step 2
    if (u < f(y)/(c*g(y))) { # step 3 (accept)
      n.accepts <- n.accepts+1
      result.sample[n.accepts] = y
    }
  }

  result.sample
}
```

```
f  <- function(x) 6*x*(1-x)     # pdf of Beta(2,2), maximum density is 1.5
g  <- function(x) x/x           # g(x) = 1 but in vectorized version
rg <- function(n) runif(n,0,1)  # uniform, in this case
c  <- 2                         # c=2 since f(x) <= 2 g(x)

vals <- accept.reject(f, c, g, rg, 10000)       # generating sample values

# Checking if it went well
hist(vals, breaks=30, freq=FALSE, main="Sample vs true Density")
xs <- seq(0, 1, len=100)
lines(xs, dbeta(xs,2,2), col="red", lwd=2)   # fitting beta distribution
```



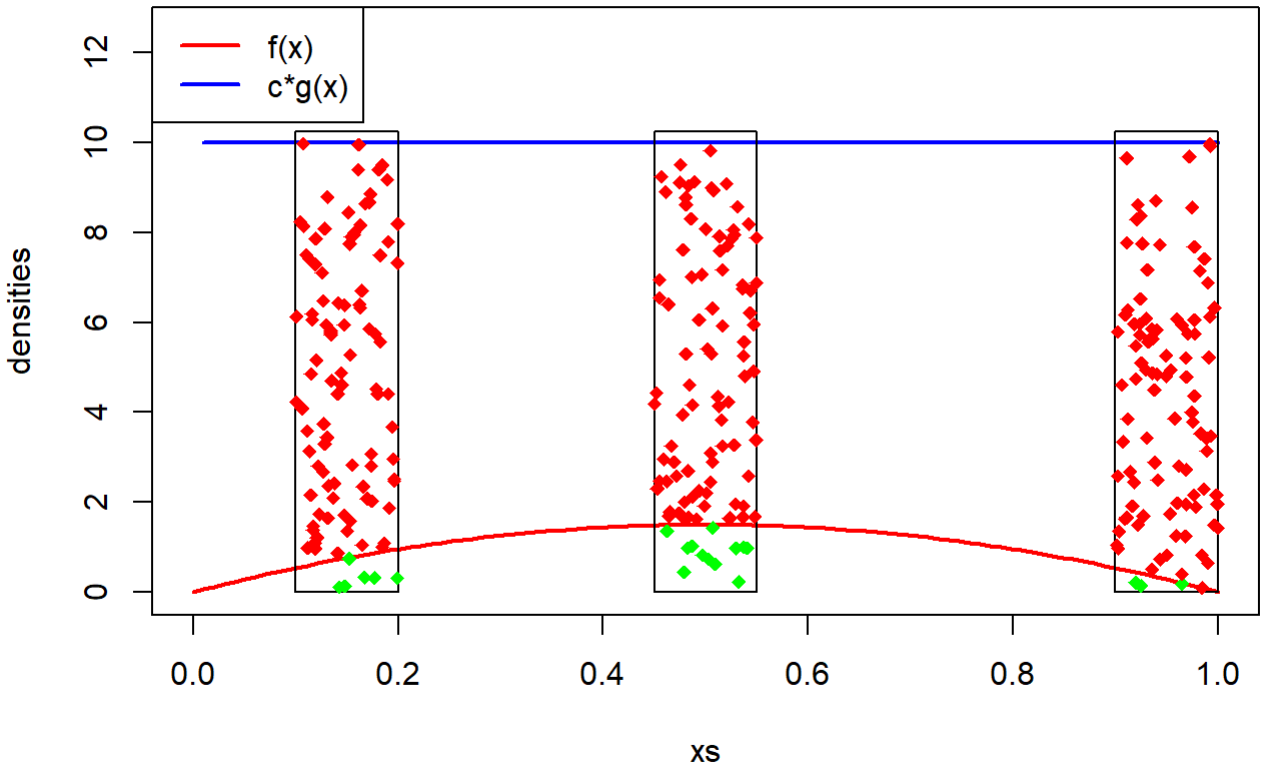
Visualizing the method of accepting (green dots) and rejecting (red dots) at some specified segments:

```
xs <- seq(0,1, len =100)
#plot(xs)
#plot(dbeta(xs,2,2))
#plot(xs,dbeta(xs,2,2))          # Plotting y vs x ; xs sets the x range from 0 -1
plot(xs, dbeta(xs,2,2), ylim=c(0,c*1.3), type="l", col="red", lwd=2, ylab="densities")

lines(xs)
lines(xs, c*g(xs), type="l", col="blue", lwd=2)
legend("topleft",c("f(x)","c*g(x)"), col=c("red","blue"), lwd=2)


draw.segment <- function(begin.segment, end.segment) {
  segments(c(begin.segment,end.segment,end.segment,begin.segment), c(0,0,c*1.025,c*1.025),
           c(end.segment,end.segment,begin.segment,begin.segment), c(0,c*1.025,c*1.025,0))
  n.pts <- 100
  us <- runif(n.pts, 0, 1)
  ys <- begin.segment + rg(n.pts)*(end.segment-begin.segment)
  accepted <- us < f(ys)/(c*g(ys))
  points(ys, c*us, col=ifelse(accepted,"green","red"), pch=18)
}

draw.segment(0.10, 0.20)
draw.segment(0.45, 0.55)
draw.segment(0.90, 1.00)
```



The higher the density of f the more points are accepted however if cg(x) >> f(x) we will have lots of rejections,which will decrease the quality of simulation for same amount of points, Checking for c = 10 :

```
c <- 10

xs <- seq(0,1, len =100)
plot(xs, dbeta(xs,2,2), ylim=c(0,c*1.25), type="l", col="red", lwd=2, ylab="densities")
lines(xs, c*g(xs), type="l", col="blue", lwd=2)
legend("topleft",c("f(x)","c*g(x)"), col=c("red","blue"), lwd=2)

draw.segment(0.10, 0.20)
draw.segment(0.45, 0.55)
draw.segment(0.90, 1.00)
```



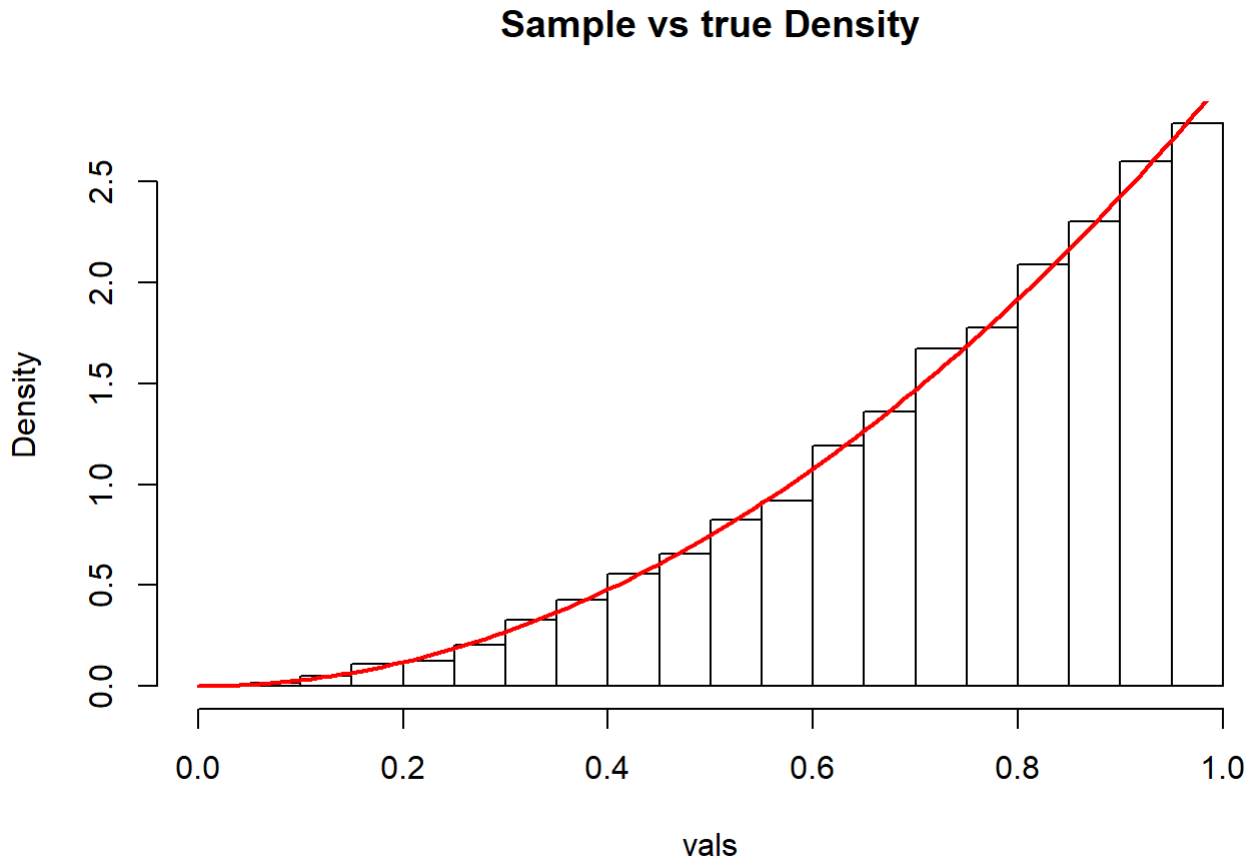
Adaptive Rejection Sampling

This number of pints is not enough to get an estimate with the quality of previous example.

```
#install.packages("ars")

library(ars)
```

```
## Warning: package 'ars' was built under R version 3.5.1
```

```
log.f <- function(x) log(3*x^2) # log(f(x))
log.f.dx <- function(x) 2/x     # d/dx log(f(x))

vals <- ars(1e4,                # how many points are required
            log.f, log.f.dx,    # the needed functions
            lb=TRUE, ub=TRUE,   # there are a lower and upper bounds for the pdf
            xlb=0, xub=1,       # and which are those bounds
            x=c(.1,.5,.9))      # some initial points inside the pdf

# Checking if it went well
hist(vals, breaks=30, freq=FALSE, main="Sample vs true Density")
xs <- seq(0, 1, len=100)
curve(3*x^2, 0, 1, col="red", lwd=2, add=T)
```



Monte Carlo Integration

pre-condition: a < b

```
MC.simple.est <- function(g, a, b, n=1e4) {
  xi <- runif(n,a,b)      # step 1
  g.mean <- mean(g(xi))   # step 2
  (b-a)*g.mean            # step 3
}


g <- function(x)  exp(-x)

MC.simple.est(g,2,4)
```

```
## [1] 0.1169242
```

Importance Sampling

Two problems with the previous method: *does not apply to unbounded intervals* performs poorly if the pdf is not very uniform, namely at distribution tails

Eg: X~N(0,1), estimate P(X>4.5)

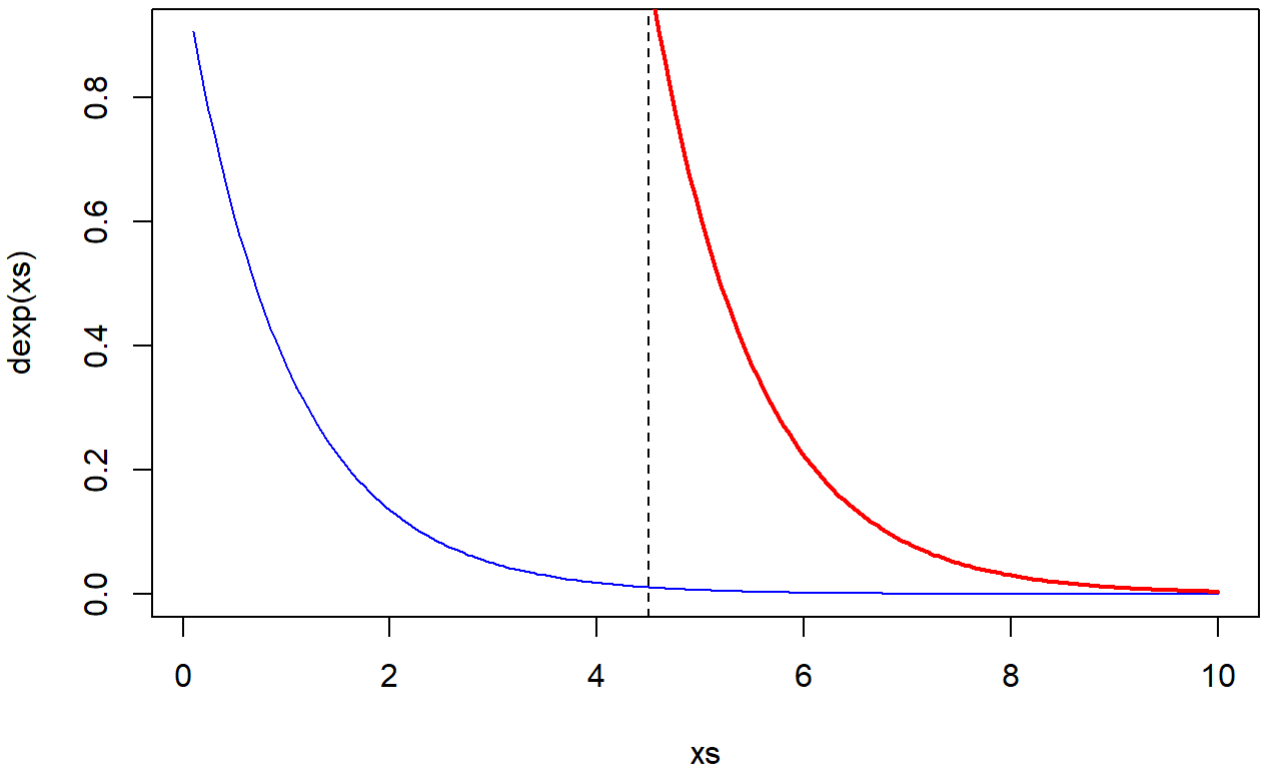
```
# True - value
pnorm(4.5, lower.tail=FALSE)          # theta (could also be computed by 1-pnorm(4.5))
```

```
## [1] 3.397673e-06
```

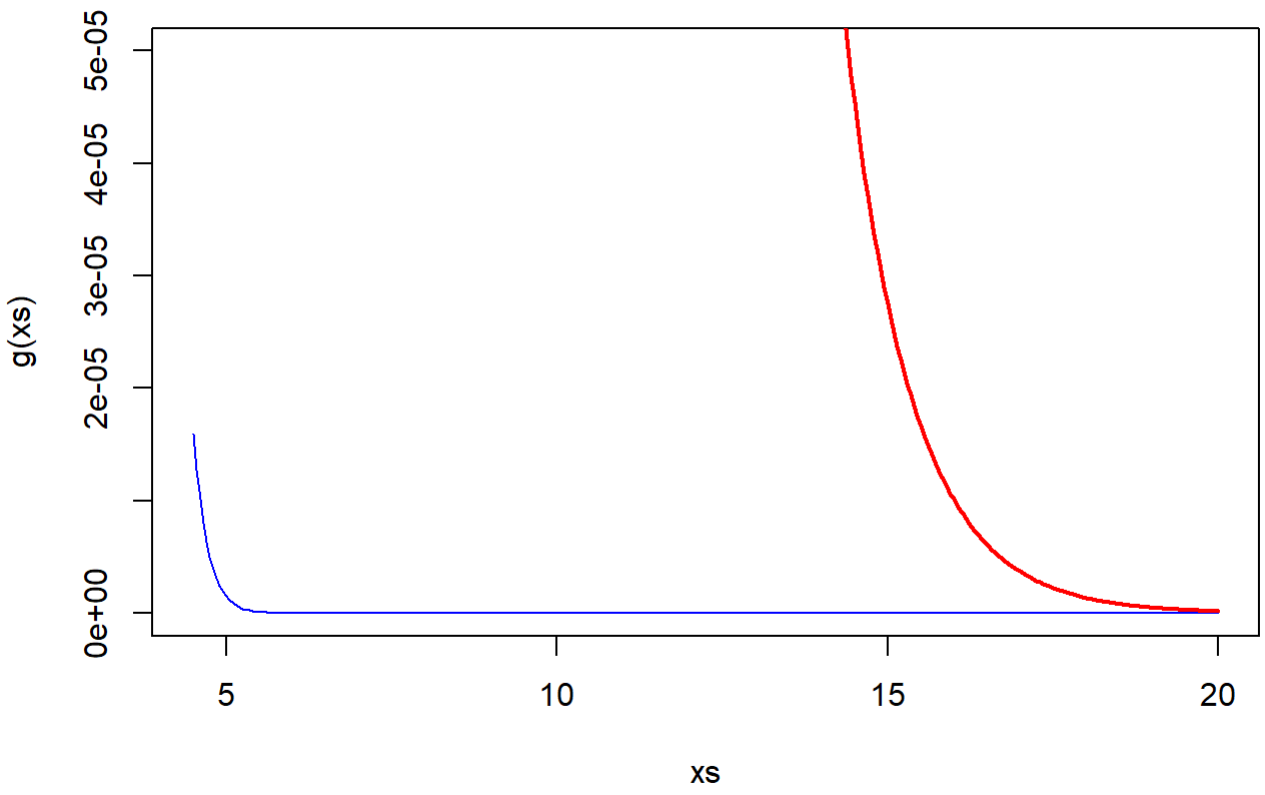
```
# MC estimation
n <- 1e4
indicators <- rnorm(n) > 4.5
sum(indicators) / n           # hat.theta
```

```
## [1] 0
```

```
# rh generates sample from candidate pdf

i.sampling <- function(f, g, h, rh, n=1e4) {
  ys <- rh(n)
  mean(g(ys)*f(ys)/h(ys))
}


xs <- seq(0.1,10,by=0.05)
plot(xs,dexp(xs),col="blue", type="l")   # the exponential pdf
lines(xs,exp(-(xs-4.5)),col="red",lwd=2) # the truncated pdf
abline(v=4.5,lty=2)
```




```
# plot of target pdf g (in blue) and the candidate pdf h (in red)

g <- dnorm
h <- function(x) exp(-(x-4.5))

xs <- seq(4.5,20,by=0.05)
plot(xs,g(xs),col="blue", type="l", ylim=c(0,0.5e-4))
lines(xs,h(xs),col="red",lwd=2)
```


g(xs)
0e+00
2e-05
3e-05
4e-05
5e-05
5
10
15
20
xs