Monte Carlo method

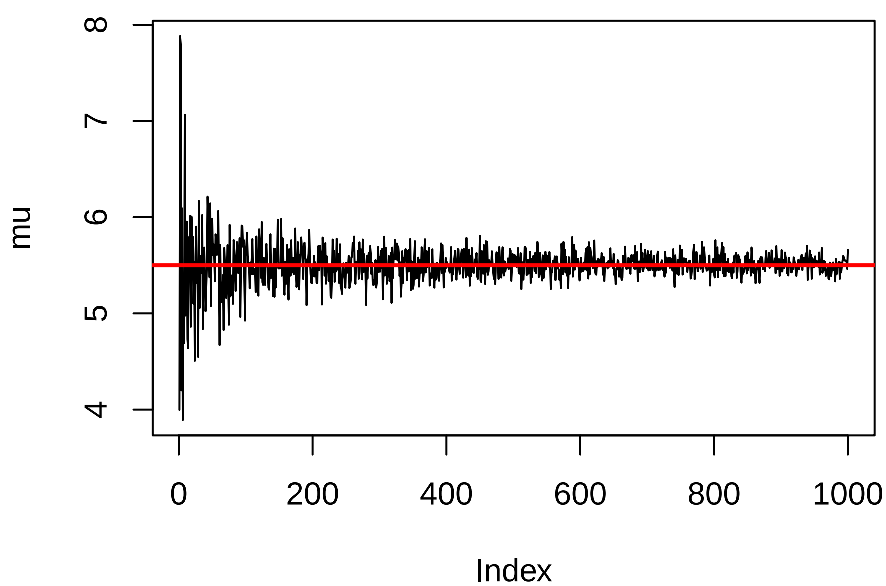
Introduction to Numerical Analysis

Joaquin Cavieres

Monte Carlo method

Example 1

```
n  = 1000                               # n iterations
mu = 0                                  # Mean vector
for(i in 1:n){
  mu[i] = mean(runif(i,1,10))           # sampling i from ~ Unif(1,10)
}
plot(mu, type ='l')                     # Graph the samples
abline(h = 5.5, col='red', lwd = 2) # Graph of the mean of the population
```



Example 2

Imagine a circle in a square of side 1, and the radius of the circle should initially have a value of $r = 0.5$. We know that the area of a circle is:

$$\pi * r^2$$

and the area of a square (in terms of the radius) should be:

$$(2 * r)^2$$

Finally, the radius of the area is:

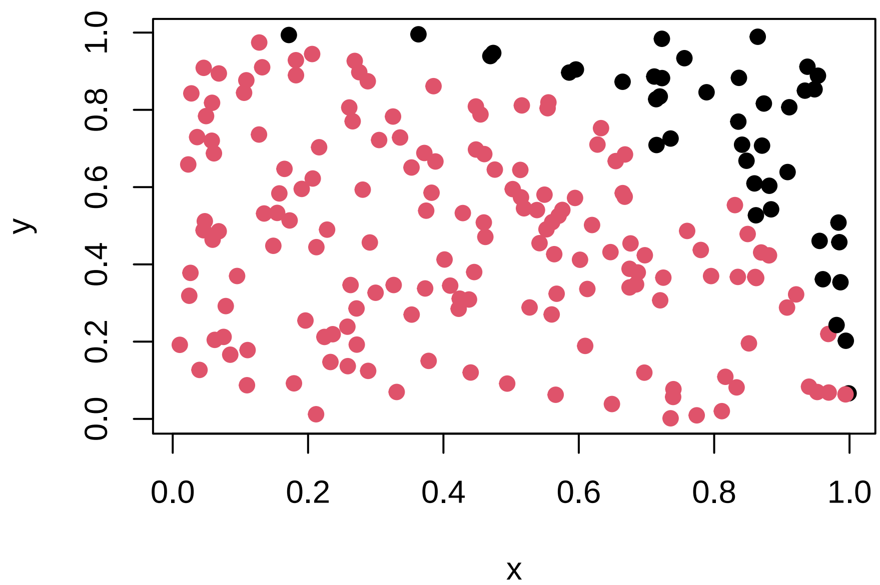
$$\rho = \frac{\text{Circle area}}{\text{Square area}} = \frac{\pi * r^2}{(2 * r)^2} = \frac{\pi}{4}$$

therefore,

$$\pi = 4 * \rho$$

To aproximate ρ we must generate random points in $[0, 1]$, and in this way we can calculate the ratio of points that

```
n<- 200
x<- runif(n,0,1)
y<- runif(n,0,1)
d <- (x^2+y^2<1)
plot(x,y, col= d+1, pch=19)
```



We see those who fall into the circle

```
in_circle <-sum(d)
in_circle
```

```
## [1] 159
```

and out of the circle:

```
out_circle <-(n - in_circle)
out_circle
```

```
## [1] 41
```

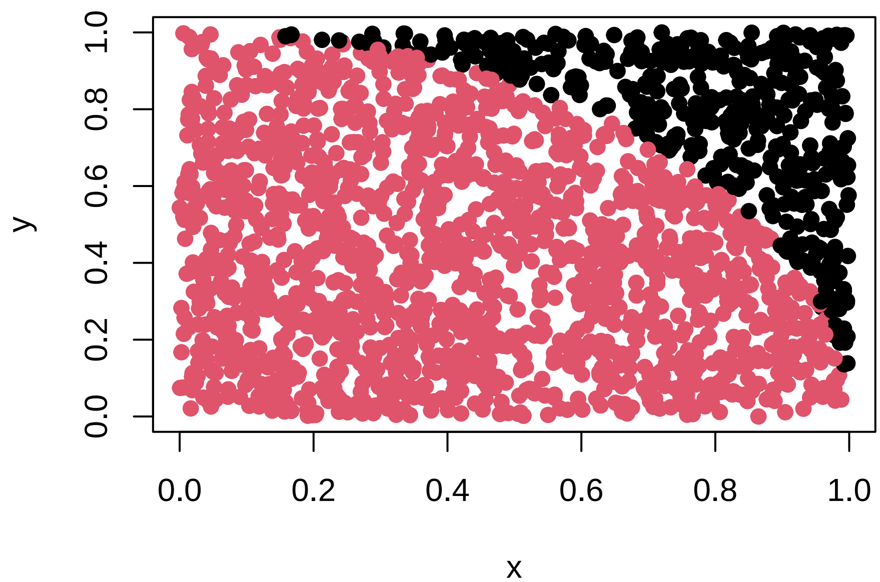
Calculates π

```
pi <- (in_circle / n)*4
pi
```

```
## [1] 3.18
```

Repeat the exercise for 2000 points:

```
n<-2000
x<-runif(n,0,1)
y<-runif(n,0,1)
d<-(x^2+y^2<1)
plot(x,y, col= d+1, pch=19)
```



Again, calculate the ratio of points that do not "fall" inside the circumference and the points that fall inside the circumference.

```
in_circle <-sum(d)
in_circle
```

```
## [1] 1563
```

outside:

```
out_circle <-(n - in_circle)
out_circle
```

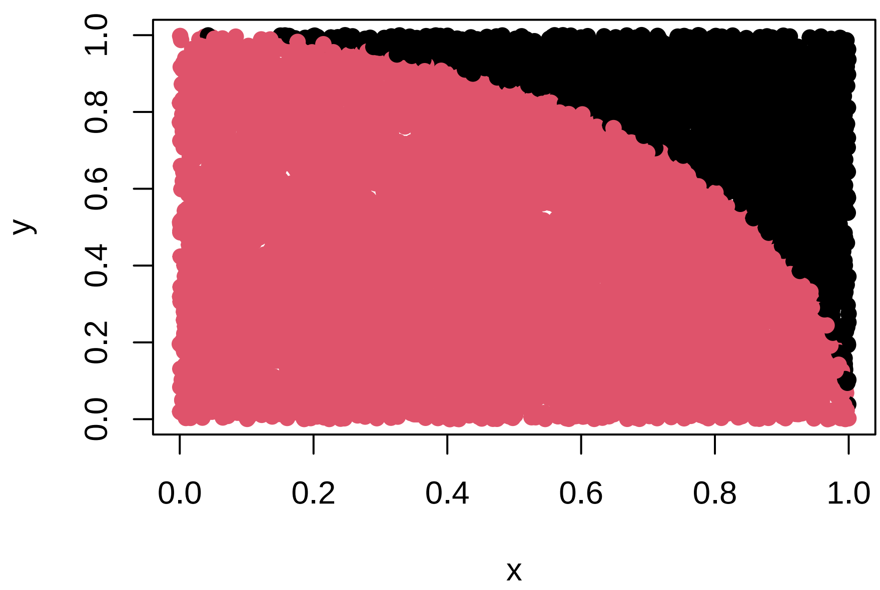
```
## [1] 437
```

We calculate (again) π

```
pi2 <- (in_circle / n)*4
pi2
```

```
## [1] 3.126
```

```
n <- 10000
x <- runif(n,0,1)
y <- runif(n,0,1)
d <- (x^2+y^2<1)
plot(x, y, col = d+1, pch=19)
```



in the circle,

```
in_circle <-sum(d)
in_circle
```

```
## [1] 7826
```

outside,

```
out_circle <- (n - in_circle)
out_circle
```

```
## [1] 2174
```

π for 10000 points

```
pi3 <- (in_circle/n)*4
pi3
```

```
## [1] 3.1304
```

Now for 500000 points

```
n <- 500000
x <- runif(n,0,1)
y <- runif(n,0,1)
d <-(x^2+y^2<1)

in_circle <-sum(d)
in_circle
```

```
## [1] 392619
```

```
out_circle <-(n - in_circle)
out_circle
```

```
## [1] 107381
```

```
pi4 <- (in_circle/n)*4
pi4
```

```
## [1] 3.140952
```

To avoid this "step by step", we build a function:

```
montecarlo <- function (f, a, b, m = 1000) {
x <- runif (m, min = a, max = b)
y.hat <- f(x)
pi <- (b - a) * sum(y.hat) / m
return (pi)
}
```

The evaluation of our function

```r
f <- function(x) { sqrt(1 - x^2) }
montecarlo(f, 0, 1, m = 1e3) * 4
```

```
## [1] 3.138602
```

```r
# Increasing the number of iterations
montecarlo(f, 0, 1, m = 1e6) * 4
```

```
## [1] 3.14207
```

Example 3

Approximate the following integral:

$$g(x) = \sin(x), \tag{1}$$

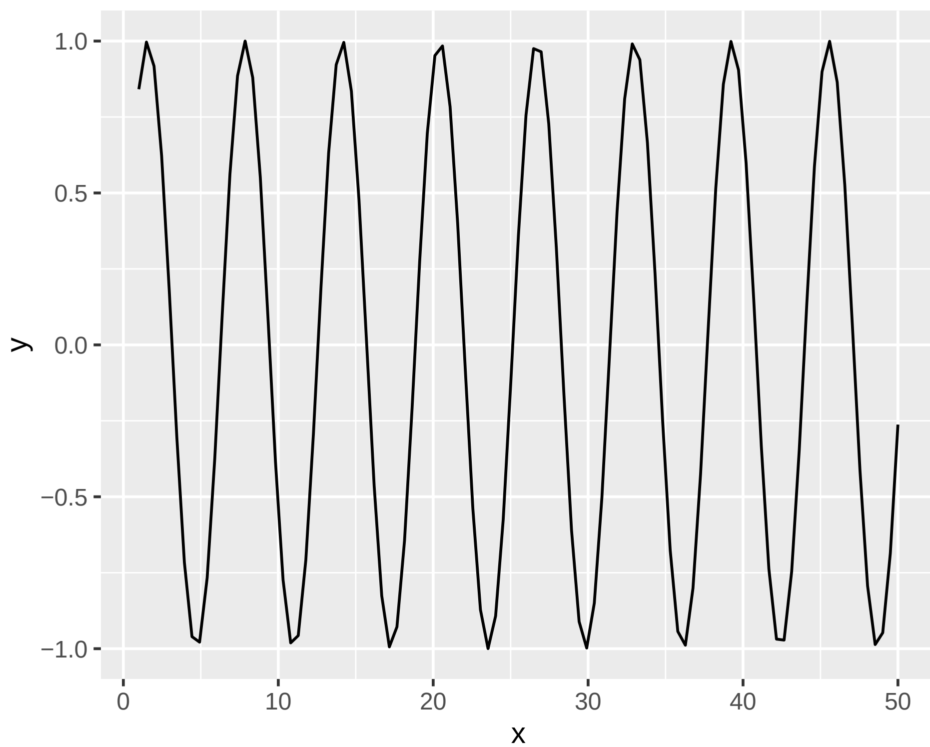
from 0 to 1.

A deterministic approximation to this integral is:

```
g <- function(x){
     sin(x)
}
integrate(g, 0, 1)
```

```
## 0.4596977 with absolute error < 5.1e-15
```

```
library(ggplot2)
ggplot(data.frame(x=c(1, 50)), aes(x=x)) +
       stat_function(fun = g)
```

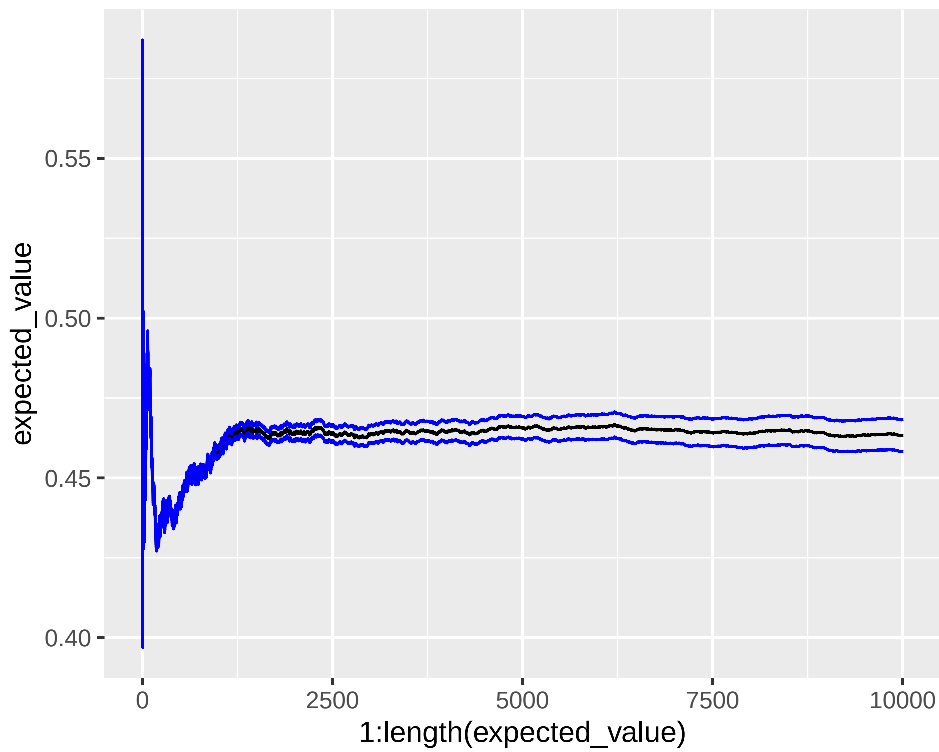


Mean and credible intervals:

```
iter <- 10^4
 applyfun <- g(runif(iter))
 expected_value <- cumsum(applyfun) / seq_len(iter)
 error_est <- sqrt(cumsum((applyfun - expected_value)^2))/(iter)

 data_plot <- data.frame(expected_value, error_est)

 library(tidyverse)
 data_plot %>%
   ggplot(aes(x = 1:length(expected_value))) +
   geom_line(aes(y = expected_value), color = "black") +
   geom_line(aes(y = expected_value  + 2*error_est), color = "blue") +
   geom_line(aes(y = expected_value  - 2*error_est), color = "blue")
```



In that function we are not explicitly given a density function. But, if we see,

$$\int_0^1 g(x)dx = \int_0^1 g(x)f(x)dx, \tag{2}$$

$f(x) = 1$ is a uniform density. Thus, we can we view the integral above as the expectation $E[g(X)]$, where $X \sim \mathrm{U}(0,1)$, so the Monte Carlo approximation can then be computed in a stochastic way as:

```
n <- 500
# step 1: generate n i.i.d samples from f (in this case uniform(0,1))
x_sim <- runif(n, 0, 1)

# compute the MC approximation
mu_mc <- sum(sapply(x_sim, g))/n
mu_mc
```

```
## [1] 0.4726064
```

What happen if we increase the `n`?

```
n <- 5000
# step 1: generate n i.i.d samples from f (in this case uniform(0,1))
x_sim <- runif(n, 0, 1)

# compute the MC approximation
mu_mc <- sum(sapply(x_sim, g))/n
mu_mc
```

```
## [1] 0.4599399
```