Capitulo 3

Pedro Pablo Villegas

19/9/2019

VON NEUMANN

- Semilla con 5 dígitos
- Elevar al cuadrado con 10 dígitos (poner ceros iniciales si se necesita)
- Seleccionar los 5 números centrales
- Comenzar nuevamente

```
vonNeumann <- function(n, semilla) {
  resultados <- numeric(length = n)
  resultados[1] <- semilla
  i <- 2
    while (i <= n) {
      resultados[i] <-
        as.numeric(substr(
          resultados[i - 1]^2,
          nchar(resultados[i - 1]^2) - 7,
          nchar(resultados[i - 1]^2) - 3))
      i <- i + 1
    }
  return(resultados)
}

res <- vonNeumann(n = 100, semilla = 3001)

plot(x = res[1:length(res)-1], y = res[-1], type = "p")
barplot(res[1:50])
ks.test(res[1:50], "punif")$p.value
pacf(res[1:50])
```

GENERADOR LINEAL CONGRUENCIAL

Es un algoritmo para generar números cuasialeatorios

$$x_i = ax_{i-1} + c \quad Mod \quad M, \quad para \quad i \geq 1$$

donde:

- a es el multiplicador
- c es el incremento
- M es el módulo
- x_0 es la semilla con $x_0 < M$

Para obtener la secuencia de números pseudo aleatorios, usamos:

$$y_i = \frac{x_i}{M}$$

El algorito empleado es de la forma:

```
lcg <- function(x0, a, c, M, n){
    x <- numeric(n)
    y <- numeric(n)
    x[1] <- (a*x0 + c) %% M
    y[1] <- x[1]/M
    i <- 2
    while (i <= n) {
        x[i] <- (a*x[i - 1] + c) %% M
        y[i] <- x[i]/M
        i <- i + 1
    }
    return(list(x = x, y = y))
}
```

El período de la función `lcg` es a lo máximo **M** y para algunas elecciones de los factores, mucho menos que eso.

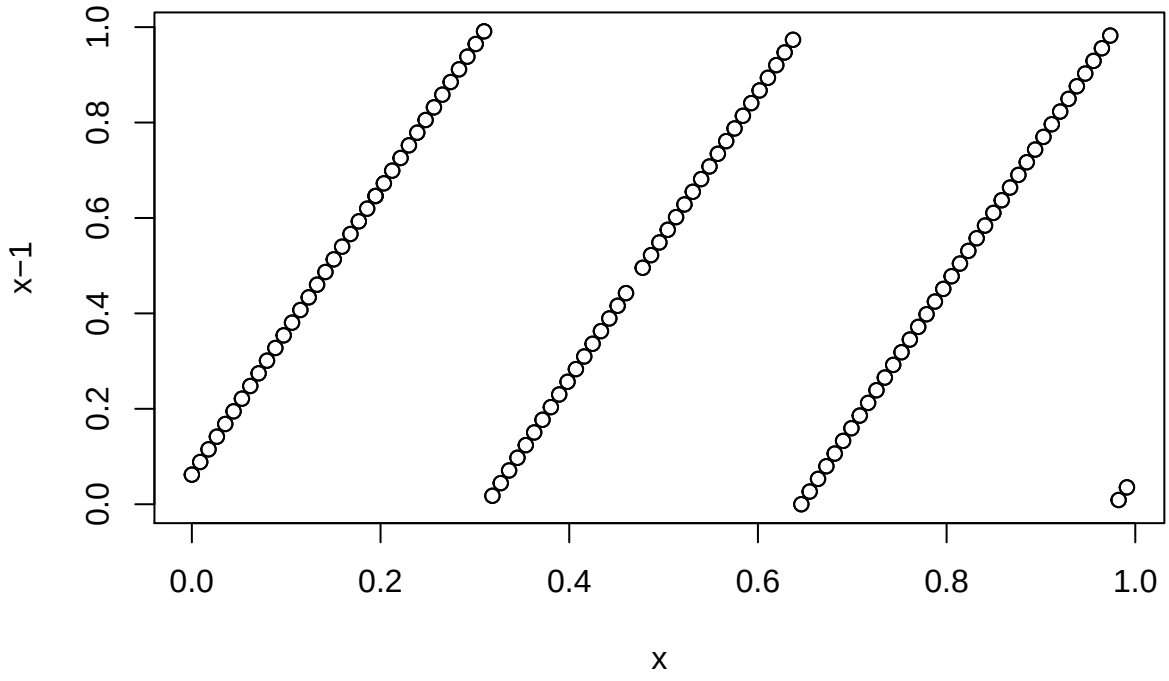
PERFORMANCE DEL GENERADOR DE NÚMEROS ALEATORIOS

Para evaluar el rendimiento del generador de números aleatorios, generalmente se chequea:

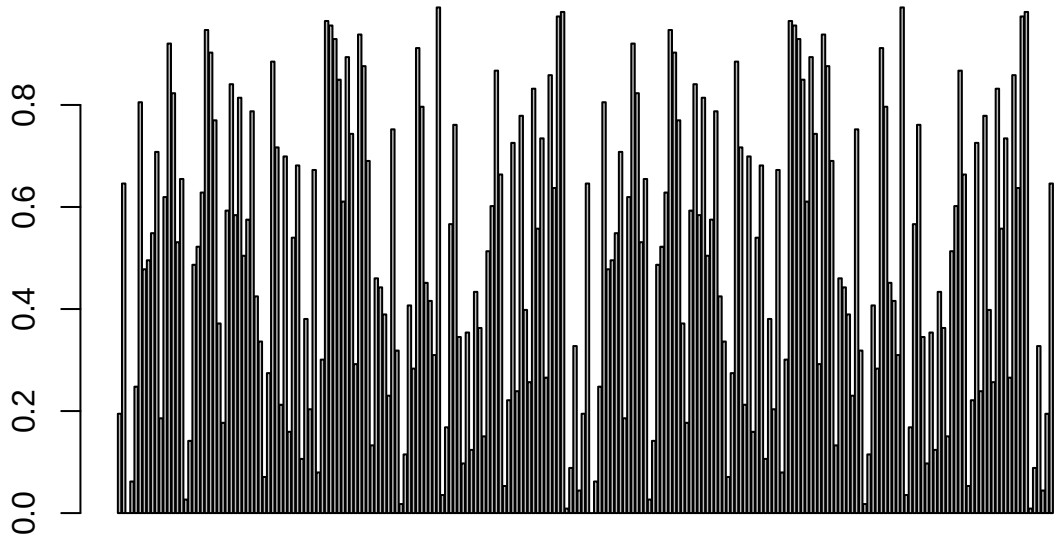
- Diagrama de dispersión de x_{n+1} vs. x_n
- Prueba de Kolmogorov-Smirnov para U(0, 1) vía `ks.test`
- Explorando la función de autocorrelación parcial *PACF* con `pacf`

Generador Lineal Congruencial `lcg()`

```
lcg.res <- lcg(5, 3, 7, 113, 226)

plot(x = lcg.res$y[1:225], y = lcg.res$y[-1], xlab = "x", ylab = "x-1")
```




```
barplot(lcg.res$y)
```

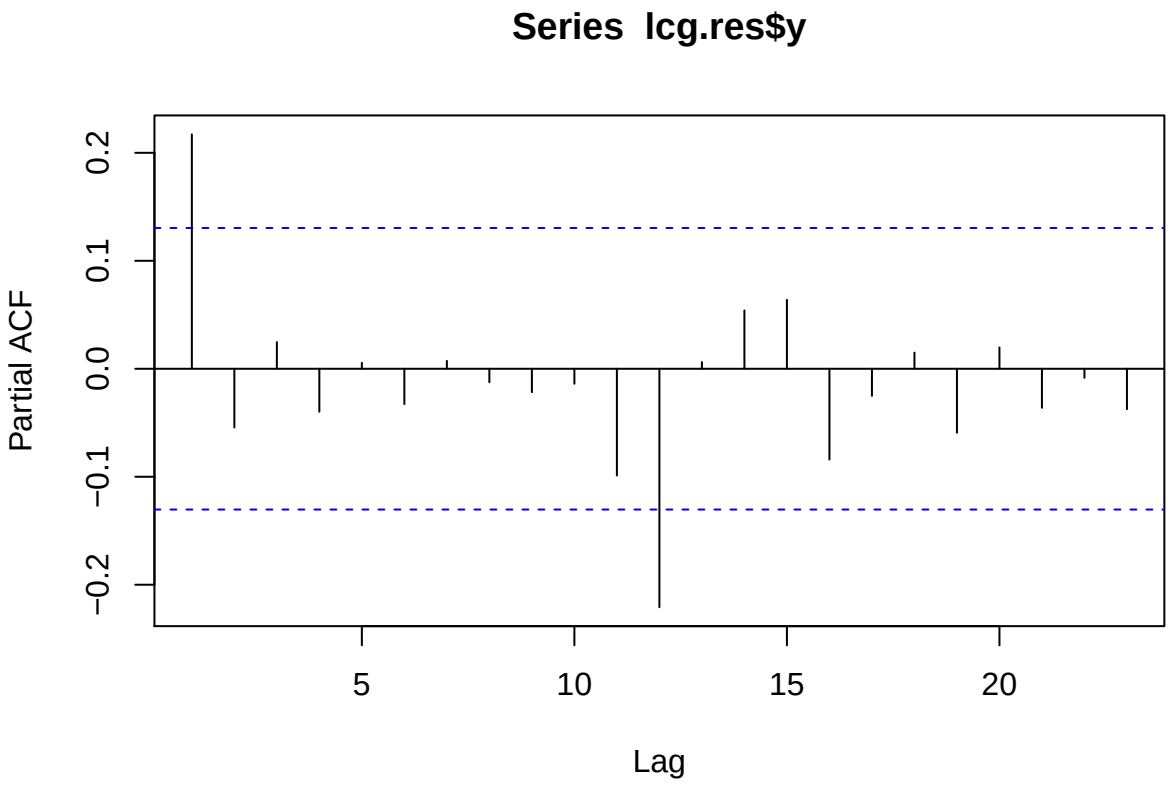



```
ks.test(lcg.res$y, 'punif')$p.value
```

```
## Warning in ks.test(lcg.res$y, "punif"): ties should not be present for the
## Kolmogorov-Smirnov test
```

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

```
pacf(lcg.res$y)
```



NÚMEROS PSEUDO-ALEATORIOS Y QUASI-ALEATORIOS

Los únicos cosas que son realmente aleatorias, son las mediciones de fenómenos físicos. La única forma de simular aleatoriedad en los computadores es a través de algoritmos determinísticos. Existen dos formas para generar aleatoriedad: pseudo y quasi.

Números pseudo-aleatorios

Los métodos pseudo-aleatorios buscan simular *aleatoriedad*

Linear congruential generators

$$x_n = (ax_{n-1} + c) \quad mod \quad m$$

Multiple recursive generators

$$x_n = (a_1 x_{n-1} + \ldots + a_k x_{n-k} c) \quad mod \quad m$$

`runif (RNG)`

Mersenne-Twister

`runif`

Well equidistributed long-period linear generators

SIMD-oriented Fast Mersenne Twister algorithms

Números quasi-aleatorios

Los métodos quasi-aleatorios son determinísticos pero buscan estar equidistribuidos

Monte Carlo para integrar, no buscan la aleatoriedad sino una buena quidistribución, son secuencias de números.

Van der Corput sequences

Halton sequences

USAR NÚMEROS ALEATORIOS PARA EVALUAR INTEGRALES

Sea $g(x)$ una función y queremos encontrar θ tal que:

$$\theta = \int_0^1 g(x)dx$$

Si U es uniformemente distribuído sobre el mismo soporte que $g(x)$, entonces podemos expresar θ como $E[g(U)]$

Para resolver integrales entre a y b

Si $\theta = \int_a^b g(x)dx$ con $a < b$, hacemos la siguiente sustitución:

$$y = \frac{x-a}{b-a}$$
$$dy = \frac{dx}{b-a}$$

La nueva integral es:

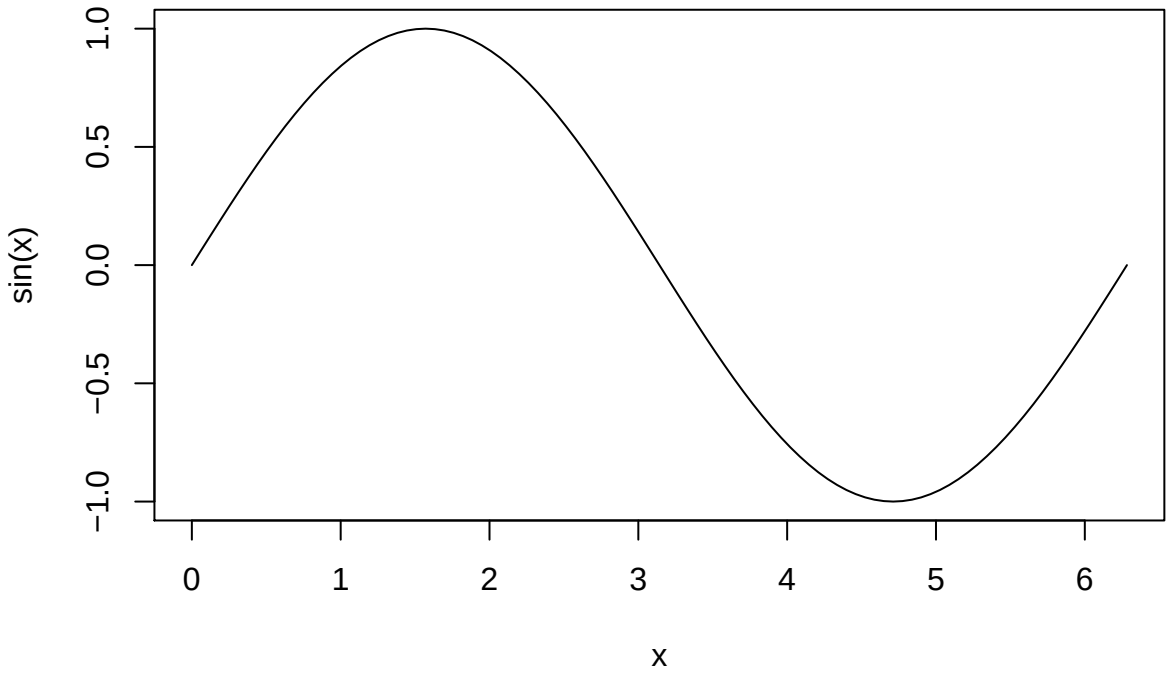
$$\theta = \int_a^b g(x)dx = \int_0^1 g((b-a)y+a)(b-a)d(y) = \int_0^1 h(y)dy$$

donde $h(y) = g((b-a)y+a)(b-a)$

- Evaluar la integral $\int_0^{2\pi} \sin(x)dx$ usando simulación Monte Carlo

$$\theta = \int$$

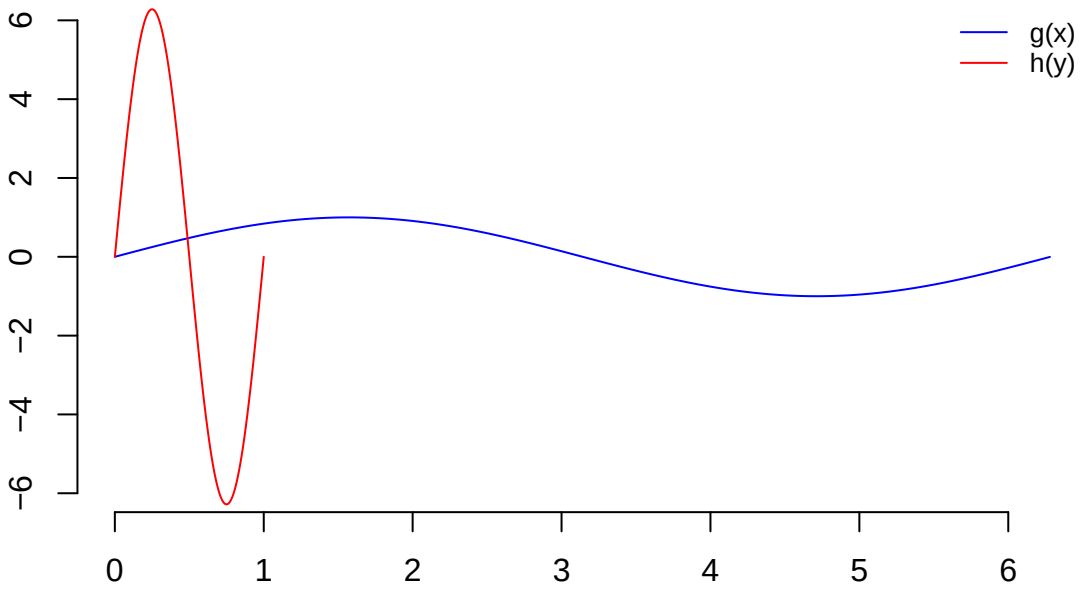
```
curve(sin(x), 0, 2*pi)
```




```
a <- 0
b <- 2*pi
n <- 10000
gx <- function(x) sin(x)
hy <- function(y) gx(a + (b - a)*y)*(b - a)
u <- runif(n)
mean(hy(u))
```

```
## [1] 0.02029545
```

```
vab <- seq(a, b, 0.01)
v01 <- seq(0, 1, 0.01)

plot(x = vab, y = gx(vab), type = "l", col = "blue", xlab = "", ylab = "",
     ylim = c(-6, 6), frame = FALSE)
lines(x = v01, y = hy(v01), type = "l", col = "red")
legend("topright", legend = c("g(x)", "h(y)"), lty = c(1, 1),
       col = c("blue", "red"), bg = "transparent", bty = "n", cex = 0.8)
```



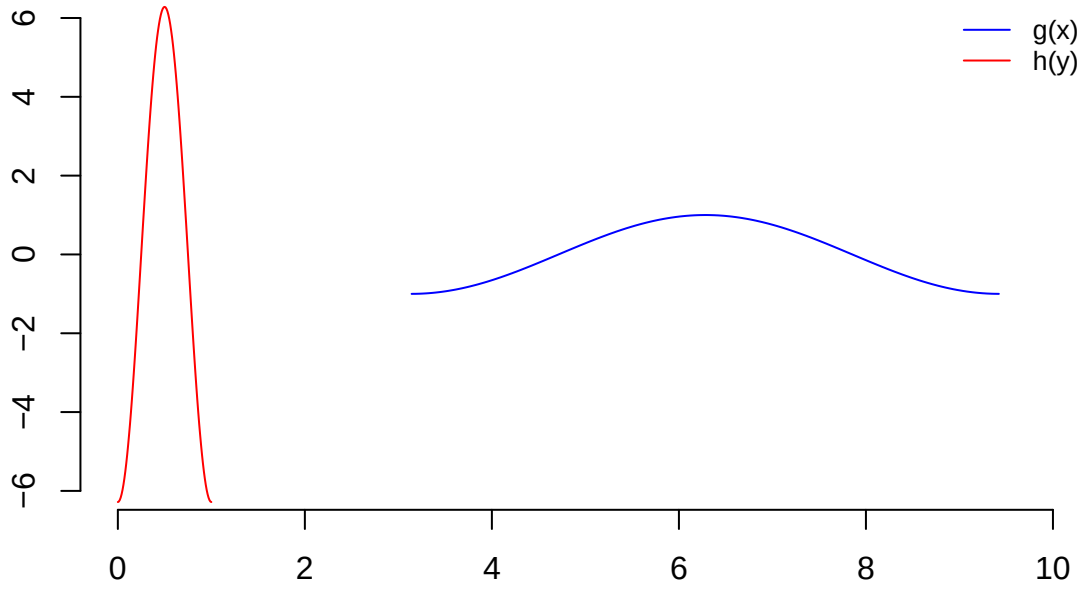
Evaluar la integral $\int_{\pi}^{3\pi} \cos(x)dx$ usando simulación Monte Carlo

```
a <- pi
b <- 3*pi
n <- 100000
gx <- function(x) {
    cos(x)
}

hy <- function(y) {
    gx(a + (b - a)*y)*(b - a)
}

u <- runif(n)

mean(hy(u))
```

```
## [1] -0.007454672
```

```
vab <- seq(a, b, 0.01)

v01 <- seq(0, 1, 0.01)

plot(x = vab, y = gx(vab), type = "l", col = "blue", xlab = "", ylab = "",
     ylim = c(-6, 6), xlim = c(0, 10), frame = FALSE)
lines(x = v01, y = hy(v01), type = "l", col = "red")
```

```r
legend("topright", legend = c("g(x)", "h(y)"), lty = c(1, 1),
       col = c("blue", "red"), bg = "transparent", bty = "n", cex = 0.8)
```



Otra forma de resolver la integral entre a y b

Generar un número aleatorio uniforme U_i de $U(a, b)$ y luego obtener la integral usando:

$$\theta = \int_a^b g(x)dx = (b-a) \times \sum_{i=1}^{k} \frac{g(U_i)}{k}$$

- Obtener la integral de $\theta = \int_{-1}^{1} e^{x+x^2} dx$ usando el método anterior

```r
a <- -1
b <- 1
n <- 10000
u <- runif(n, min = a, max = b)
f <- function(x){
    exp(x + x^2)
}
int <- (b - a)*mean(f(u))
```

Resolver integrales entre 0 e ∞

Si $\theta = \int_0^\infty g(x)dx$ podemos hacer el cambio de variable:

$$y = \frac{1}{x+1}$$
$$dy = \frac{-dx}{(x+1)^2} = -y^2 dx$$

Y obtenemos la nueva integral:

$$\theta = \int_0^\infty g(x)dx = \int_1^0 \frac{g(\frac{1}{y-1})}{-y^2}dy = \int_0^1 \frac{g(\frac{1}{y-1})}{y^2}dy = \int_0^1 h(y)dy$$

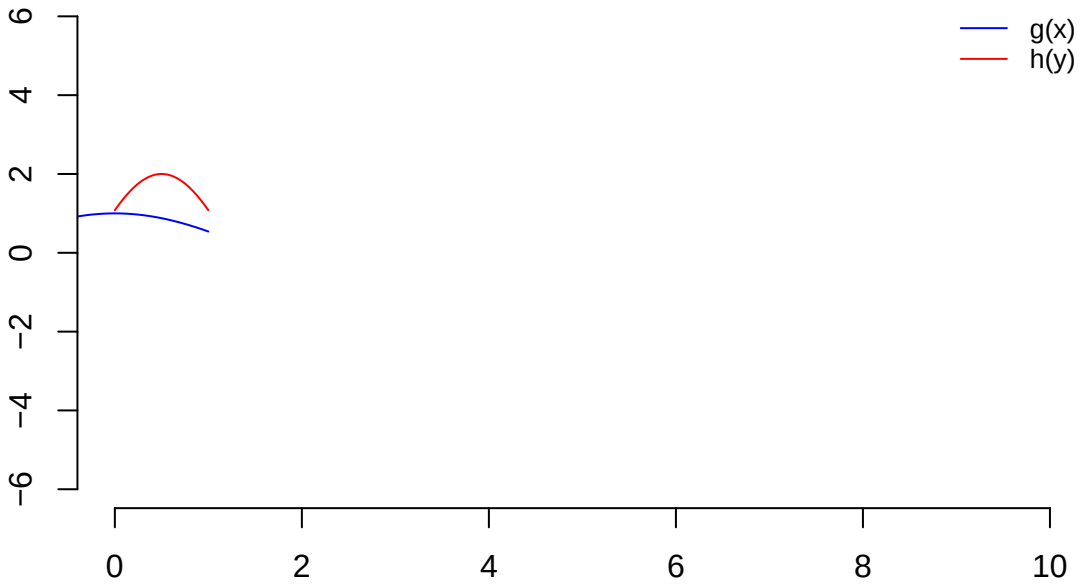
Donde $h(y) = \frac{g(\frac{1}{y-1})}{y^2}$

- $\int_0^\infty e^{-x}dx$

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

h <- function(y) g(1 / (y - 1))/(y^2)

u <- runif(n = 10000)

mean(h(u))
```

```
## [1] Inf
```

```
vab <- seq(a, b, 0.01)

v01 <- seq(0, 1, 0.01)

plot(x = vab, y = gx(vab), type = "l", col = "blue", xlab = "", ylab = "",
     ylim = c(-6, 6), xlim = c(0, 10), frame = FALSE)
lines(x = v01, y = hy(v01), type = "l", col = "red")
legend("topright", legend = c("g(x)", "h(y)"), lty = c(1, 1),
       col = c("blue", "red"), bg = "transparent", bty = "n", cex = 0.8)
```



Integrales multidimensionales

```
fun <- function(x) exp(-x[1] - x[2])
k <- 5
U <- matrix(runif(n = 2*k), ncol = 2)
mean(apply(X = U, MARGIN = 1, FUN = fun))

k <- c(10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000)

multi <- function(n){
    fun <- function(x) exp(-x[1] - x[2])
    i = 1
    for (i in 1:length(k)) {
        U <- matrix(runif(n = 2*k[i]), ncol = 2)
        resultados[i,1] <- k[i]
        resultados[i,2] <- mean(apply(X = U, MARGIN = 1, FUN = fun))
        i <- i + 1
    }
    return(resultados)
}

res <- multi(1000)

plot(x = res[,1], y = res[,2], type = "b", xlab = "k", ylab = "resultado")
```

```
?persp()
```

- $\int_0^1 \int_0^1 xy^2 dxdy$

```
fun <- function(x) exp(x[1] * x[2]^2)
k <- 20
U <- matrix(runif(n = 2*k), ncol = 2)
mean(apply(X = U, MARGIN = 1, FUN = fun))

k <- c(10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000)

multi <- function(n){
    fun <- function(x) exp(-x[1] - x[2])
    i = 1
    for (i in 1:length(k)) {
        U <- matrix(runif(n = 2*k[i]), ncol = 2)
        resultados[i,1] <- k[i]
        resultados[i,2] <- mean(apply(X = U, MARGIN = 1, FUN = fun))
        i <- i + 1
    }
    return(resultados)
}

res <- multi(1000)

plot(x = res[,1], y = res[,2], type = "b", xlab = "k", ylab = "resultado")

?persp()
```