Laboratorio 4

R básico: Aplicación de R a las probabilidades elementales

- Operaciones básicas
- Funciones de `R` para probabilidades básicas. Muestreo aleatorio.
- Flujo de control y bucles.

Mostramos los eventos de un espacio muestral

El paquete `sets` realiza operaciones básicas de conjuntos y ciertas generalizaciones

```
install.packages("sets"")
```

```
library(sets)
Omega = set("C", "S")
# Muestra un conjunto de todos los posibles eventos de un experimento de
# un espacio muestral Omega
2^Omega
```

```
{{}, {"C"}, {"S"}, {"C", "S"}}
```

```
Omega = set("a", "b", "c")
2^Omega
```

```
{{}, {"a"}, {"b"}, {"c"}, {"a", "b"}, {"a", "c"}, {"b", "c"},
 {"a", "b", "c"}}
```

Función de probabilidad

```
# Espacio muestral
Omega = c(1, 2, 3, 4)
# probabilidad de 4  eventos elementales
p = c(1/2, 1/4, 1/8, 1/8)
# ellos suman
sum(p)
```

```
[1] 1
```

Generamos todas las posibles 3-tuplas de $S_1 = \{1, 2\}$, $S_2 = \{1, 2, 3\}$ y $S_3 = \{1, 2\}$

```
help("expand.grid")
expand.grid(S1 = 1:2, S2 = 1:3, S3 = 1:2)
```

```
   S1 S2 S3
1   1  1  1
2   2  1  1
3   1  2  1
4   2  2  1
5   1  3  1
6   2  3  1
```

```
7   1  1  2
8   2  1  2
9   1  2  2
10  2  2  2
11  1  3  2
12  2  3  2
```

Contando el número de combinaciones

Para calcular el número de combinaciones de `n` ítems tomando `k` ítems a la vez, usamos la función `choose(n,k)`. El número dado es $n!/(n-k)!k!$.

```
help("choose")
choose(5, 3)
```

```
[1] 10
```

```
choose(50, 13)
```

```
[1] 354860518600
```

```
choose(50, 30)
```

```
[1] 4.712921e+13
```

Generando combinaciones

Generalizamos todas las combinaciones de `n` ítems tomando `k` ítems a la vez, usando la función `combn(items, k)`.

```
help("combn")
combn(1:5, 3)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    1    1    1    1    1    2    2    2     3
[2,]    2    2    2    3    3    4    3    3    4     4
[3,]    3    4    5    4    5    5    4    5    5     5
```

```
combn(c("T1","T2","T3","T4","T5"), 3)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,] "T1" "T1" "T1" "T1" "T1" "T1" "T2" "T2" "T2" "T3"
[2,] "T2" "T2" "T2" "T3" "T3" "T4" "T3" "T3" "T4" "T4"
[3,] "T3" "T4" "T5" "T4" "T5" "T5" "T4" "T5" "T5" "T5"
```

Generando números aleatorios

Si tu quieres generar números aleatorios, usa `r?` donde ? es una de las distribuciones listada en la tabla de arriba.

```
help("runif")
runif(10)
```

```
 [1] 0.1355620 0.7353156 0.4361521 0.8527455 0.9642488 0.6209955 0.9459451
 [8] 0.9546677 0.9979769 0.4718876
```

```r
runif(10, min = -2, max = 2)
```

```
 [1]  1.0595369 -0.5210195 -0.8780805 -0.6104707  0.3884907  1.4806550
 [7] -1.9724230  1.6267267 -1.1545126  1.9358282
```

```r
help("rnorm")
rnorm(10)
```

```
 [1]  2.250796371 -0.260158812 -0.398523409 -0.002550842 -0.065360944
 [6]  2.189469573 -1.526580785 -0.516160600  0.206442350  0.105948113
```

```r
rnorm(10, mean=100, sd=15)
```

```
 [1]  88.45684  92.29203  74.98489  93.78910 108.58478 118.46686  91.49897
 [8] 101.96406 124.49773  96.20696
```

```r
help("rbinom")
rbinom(10, size=10, prob=0.5)
```

```
 [1] 6 4 4 2 5 5 3 4 5 6
```

```r
help("rpois")
rpois(10, lambda=10)
```

```
 [1]  8  7 13 13  8 13 15  7 11  9
```

```r
help("rexp")
rexp(9, rate =0.1)
```

```
[1]  6.586729  6.370307  2.414540 26.914640  7.715508  4.149266  3.422283
[8]  7.498030  4.019696
```

```r
help("rgamma")
rgamma(9, shape=2, rate=0.1)
```

```
[1]  7.100811 14.262869 30.378040 12.692750 19.582137 14.074613 20.681636
[8]  5.763604 20.258187
```

```r
rnorm(3, mean = c(-10, 0, 10), sd = 1)
```

```
[1] -8.9331924  0.4140941  9.4769635
```

Generando una muestra aleatoria

Si se desea una muestra aleatoria, podemos utilizar la función `sample(vec, n)`

```r
help("sample")
sample(airquality$Wind, 10)
```

```
 [1]  7.4 11.5  8.0  9.2  8.0 11.5 14.9  6.3  7.4  7.4
```

La función `sample` normalmente reliza muestras sin reemplazo, lo que significa que no seleccionará el mismo elemento dos veces. Algunos procedimientos estadísticos (especialmente el bootstrap) requieren muestreo con reemplazo, lo que significa que un elemento puede aparecer varias veces en la muestra. Especificando `replace = TRUE` en `sample` con reemplazo.

Es fácil implementar un bootstrap simple mediante el muestreo con reemplazo. Este fragmento de código muestra repetidamente un conjunto de datos xy calcula la mediana de la muestra:

```
medianas <- numeric(1000)
for (i in 1:1000) {
medianas[i] <- median(sample(x, replace=TRUE))
}
```

A partir de las estimaciones de bootstrap, podemos estimar el intervalo de confianza para la mediana:

```
ci <- quantile(medians, c(0.025, 0.975))
cat("El intervalo de confianza 95% es (", ci, ")\n")
```

Generación de números aleatorios reproducibles

Si se desea generar una secuencia de números aleatorios, pero desea reproducir la misma secuencia cada vez que se ejecuta el programa.

Antes de ejecutar su código R, llame a la función `set.seed` para inicializar el generador de números aleatorios a un estado conocido:

```
set.seed(1978)
```

Generando secuencias aleatorias

Puedes generar secuencias aleatorias, tales como la simulación del lanzamiento de una moneda o otro ensayo de Bernoulli.

Usamos `sample(set, n, replace=TRUE)`

```
sample(c("H","T"), 10, replace=TRUE)
```

```
 [1] "H" "H" "H" "T" "T" "H" "T" "H" "H" "H"
```

```
 sample(c(FALSE,TRUE), 20, replace=TRUE)
```

```
 [1]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
[12] FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE
```

```
sample(c(FALSE,TRUE), 20, replace=TRUE, prob=c(0.2,0.8))
```

```
 [1]  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE
[12] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
```

Permutar aleatoriamente un vector

Si se desea generar una permutación aleatoria de un vector. Si `v` es un vector, entonces la `sample(v)` devuelve una permutación aleatoria. La función `sample(v)` es equivalente a `sample(v, size = length (v), replace = FALSE)`

```
sample(1:12)
```

```
 [1] 12  6  4  5  2  9  7  8 10  3 11  1
```

```
sample(letters[1:10])
```

```
##  [1] "c" "j" "g" "e" "h" "a" "i" "d" "f" "b"
```

Estructuras básicas de programación

Flujo de control

If-else

```
if(FALSE)
{
    message("Esto no se ejecuta....")
} else
{
    message("esto deberia ejecutarse.")
}
```

ifelse

```
(r <- round(rnorm(2), 1))

(x <- r[1] / r[2])

if(is.nan(x))
{
    message("x es un NA")
} else if(is.infinite(x))
{
    message("x es infinito")
} else if(x > 0)
{
    message("x es positivo")
} else if(x < 0)
{
    message("x es negativo")
} else
{
    message("x es cerp")
}
```

Bucles

while

```
accion <- sample(
    c(
        "Aprender R",
        "Estudiar CM-274",
        "Leer el manga de One Piece",
        "Salir con Jessica"
    ),
    1
)

while(accion != "Salir con Jessica"){

    message("Hoy es un buen dia")
```

```
     accion <- sample(
        c(
            "Aprender R",
            "Estudiar CM-274",
            "Leer el manga de One Piece",
            "Salir con Jessica"
        ),
        1
    )
    message("accion = ", accion)
}
```

Un ejemplo numérico del uso de while con los caminos aleatorios unidimensionales **https://es.wikipedia.org/wiki/Camino_aleatorio**.

```
# Un camino aleatorio con While

x=0
n=0
set.seed(333)
while (x <= 10) {
  n=n+1
  x=x+rnorm(1,mean=.5,sd=1)
  }

print(paste ("n = ", n, ", x = ",round(x,2) ))
```

For

El bucle `for` acepta una variable de iteración y un vector. La sintaxis para el bucle `for` es

```
  for (nombre in valores ) expresion
```

El bucle `for`, iteraciona a través de los componentes nombre de valores uno a la vez. En el ejemplo anterior `nombre` toma el valor de cada elemento sucesivo de `valores`, hasta que se complete sus componentes.

```
lenguajes <- c("Python", "JS", "C", "C++", "R", "Bash")
for(l in lenguajes){
  print(l)
}
```

```
[1] "Python"
[1] "JS"
[1] "C"
[1] "C++"
[1] "R"
[1] "Bash"
```

Ejercicios

1 . Usa R, para calcular las respuesta numéricas de lo siguiente:

- $1 + 2(3 + 4)$
- $4^3 + 3^{2+1}$
- $\sqrt{(4 + 3)(2 + 1)}$

- $\left(\frac{1+2}{3+4}\right)^2$

2 . La función `sd` calcula la desviación estándar. Calcula la desviación estándar desde el 0 al 100.

3 . Vea la demostración de símbolos matemáticos, usando `demo(plotmath)`.

4 . Genera aleatoriamente 1.000 mascotas , de las opciones `perro`, `gato`, `pollo` y `pez dorado`, con la misma probabilidad de que cada uno sea elegido. Muestra los primeros valores de la variable resultante y cuente el número de cada tipo de mascota.

5 . La *Conjetura de Collatz* señala que para todo número natural n, si se realiza la siguiente recursión:

$$f(n) = \begin{cases} 3n+1 & n = 2k+1 \\ \frac{n}{2} & n = 2k \end{cases}$$

Siempre se llegará a 1 luego de cierta cantidad de iteraciones. Para hallar la cantidad de pasos de un número se usa la siguiente iteración:

```
n <- 100
pasos <- 1
while(n!=1){
    if(n %% 2 == 0){
        n <- n/2
    } else {
        n <- 3*n + 1
    }
    pasos <- pasos + 1
}
print(pasos)
```

Diseña un programa que halle la secuencia de menor longitud de entre los números en el rango $[100, 200]$ y además determine cuál es esa secuencia.

6 . Jessica estaba estudiando teoría de números y aprendió el algoritmo de Euclides, pero en la clase estaba tan concentrada que no llegó a apuntar correctamente el algoritmo dado por su profesor. A pesar de todo, ella recuerda exactamente todas las líneas, pero no el orden correcto. Dadas las siguientes líneas de código, reconstruya el algoritmo de Euclides iterativo y use $a = 10^5 + 3$ y $b = 10^8 + 9$:

```
a <- 1001
b <- 7
while(b!=0){
    b <- carry
    a <- b
    carry <- a %% b
}
print(a)
```

7 . Usando la función `sample` obtenga un muestreo de 10 números en el rango $[1, 1000]$ (con reemplazo) y determine la relación entre la cantidad de primos encontrados y el tamaño de la muestra. Según la teoría de primos, una cota superior para la cantidad de primos menores o iguales a n es $\frac{n}{ln(n)}$, analice cuán preciso es esto con este caso y un muestreo de 20 números en el rango de $[1, 2000]$

8 . Supongamos que x es un vector numérico. Explica en detalle, como las siguientes expresiones son evaluadas y que valores toman

```
sum(!is.na(x))
c(x,x[-(1:length(x))])
x[length(x) + 1]/length(x)
sum(x > mean(x))
```

9 .Usando la función `cumprod` o otra relacionada, calcula

$$1+\frac{2}{3}+\left(\frac{2}{3}\frac{4}{5}\right)+\left(\frac{2}{3}\frac{4}{5}\frac{6}{7}\right)+\cdots+\left(\frac{2}{3}\frac{4}{5}\cdots\frac{38}{39}\right).$$

10 . Sea X el número de `unos` obtenidos en 12 lanzamientos de un dado. Entonces X tiene una distribución Binomial $(n=12, p=1/3)$. Calcule una tabla de probabilidades binomiales para $x=0,1,\ldots,12$ por dos métodos:

- Usando la fórmula para la densidad: $P(X=K)=\binom{n}{k}p^k(1-p)^{n-k}$ y aritm'etica en R. Usa `0:12` para la secuencia de x valores y la función `choose` para calcular los coeficientes binomiales $\binom{n}{k}$.
- Usando la función `dbinom` de R y comparar tus resultados con ambos métodos.

11 . Sea X el número de `unos` obtenidos en 12 lanzamientos de un dado. Entonces X tiene una distribución Binomial $(n=12, p=1/3)$. Calcula el CDF para $x=0,1,\ldots,12$ por dos métodos:

- Usando la función `cumsum` y el resultado del ejercicio anterior.
- Con el uso de la función `pbinom`. ¿ Qué es $P(X>7)$?.