Ejercicios de programación I: Introducción

[IMSER 2013]

Archivos incluidos

Archivos de ejercicios:

```
## 1-varianza.R, 2-zenon.R, 3-extra-dist.R
```

Archivos complementarios (¡no modificar!):

```
## evaluar.R, datos, letra.pdf, plotTriang.R
```

---

(1) Cálculo de la varianza de una muestra

Preámbulo: sumatorias

Crear el código necesario para calcular sumatorias en R u otros lenguajes no es intiutivo cuando no se tiene experiencia en programación. En R la función `sum` sirve para este propósito. Si tenemos un vector `s` con una cantidad arbitraria de elementos, entonces la suma de todos ellos se realiza con el comando `sum(s)`. Por ejemplo:

```
s <- 1:100
sum(s)
```

```
## [1] 5050
```

> Nota: el vector `s` es simplemente la secuencia de los números enteros del 1 al 100. El operador `:` es uno de las más usados en R.

Habrá notado que se hizo el cálculo en dos partes: (1) Crear el vector `s`, el cual contiene todos los términos de la sumatoria y (2) ejecutar la función `sum`. Esta separación es recomendable cuando queremos hacer sumatorias más sofisticadas, a fin de mantener el código más simple.

Un ejemplo es la fórmula para calcular la varianza no sesgada de una muestra de `n` datos:

$$\sigma^2 = \frac{1}{n-1} \cdot \sum_{i=1}^{i=n} (x_i - \overline{x})^2$$

En donde $\overline{x}$ denota el valor promedio de la muestra de datos, calculado como $\frac{1}{n} \cdot \sum_i x_i$. Si quisiéramos usar un vector `s` para calcular esta sumatoria, entonces el iésimo elemento de este (i.e.: `s[i]`) será obtenido con la fórmula $(x_i - \overline{x})^2$. Por suerte en R no hace falta calcular los `s[i]` uno por uno, ya que es posible *vectorizar*: realizar operaciones matemáticas sobre vectores como si se trataran de valóres únicos.

Un ejemplo concreto de vectorización es el siguiente:

```
v <- 1:5
v * 2
```

```
## [1]  2  4  6  8 10
```

Como puede ver, en la segunda línea se aplica la operación de multiplicación a todo el vector `v`, en lugar de ir elemento por elemento. En lenguajes de programación de *bajo nivel* (i.e.: menor abstracción respecto a los detalles de la computadora) no es posible vectorizar así, por lo que es necesario usar *loops* para hacer este tipo de operaciones. Veremos los loops en la unidad "Estructuras de control".

Tarea

En el archivo "1-varianza.R" usted deberá escribir los pasos necesarios para calcular la varianza del vector de ejemplo `x` que se muestra en el archivo (siga las instrucciones incluidas en los comentarios).

Si su solución es correcta, lo cual implica que es genérica, entonces el valor de `out` obtenido coincidirá con la salida de la función `var`. Puede correr las siguientes líneas varias veces para determinar si esto es así:

```
source("1-varianza.R")
out == var(x)
```

Esto producirá un `TRUE` o un `FALSE` en caso de que `out` esté bien o mal calculado, respectivamente.

(2) Paradoja de Zenón

Según la clásica paradoja de la dicotomía de Zenón es imposible caminar de un punto A a un punto B, debido a que primero debemos movernos la mitad del camino, posteriormente avanzar la mitad de la mitad del camino y así ad infinitum, sin llegar jamás a B. Esto se traduce en avanzar primero 1/2 de camino, luego 1/4, 1/8, 1/16 y así sucesivamente. Entonces, tras n pasos de este tipo se alcanza una fracción de camino equivalente a:

$$Z_n = \sum_{i=1}^{i=n} \frac{1}{2^i} = \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + ... + \frac{1}{2^n} = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + ... + \frac{1}{2^n}$$

(Nótese que el primer valor de i es 1 y no 0.)

Si es cierta el enunciado de Zenón, entonces no importa que tan grande sea n, esta sumatoria siempre será menor a 1, a pesar ser este el límite de la misma cuando $n \to \infty$. Podemos verificar numéricamente esta afirmación usando R: si elegimos un valor ε arbitrariamente pequeño, entonces podemos encontrar un n lo bastante grande para que $1 - Z_n < \varepsilon$.

En este ejercicio, usted deberá completar el código del archivo "2-zenon.R". En primera instancia, usted debería preocuparse de que el código sirva para cualcular la sumatoria Z_n, dado un `n` cualquiera. Como referencia, considere que los Z_n para $n = 3, 4, 5, 6, 7, 8$ deberían ser:

```
0.875
0.9375
0.96875
0.984375
0.9921875
0.9960938
```

A continuación sólo queda encontrar el n exacto tal que $1 - Z_n < \varepsilon$. El ε elegido es 10^{-6}, lo que en R se puede escribir `1e-6`. Cambie el valor de `n`, aumentando de a una unidad por vez, hasta encontrar el valor para el que $1 - Z_n < 10^{-6}$. El siguiente código le permitirá confirmar si se cumple la condición:

```
source("2-zenon.R")
1 - Zn < 1e-06
```

> Nota: como habrá notado, en este ejercicio y *a diferencia de la mayoría de los casos* de este curso, la respuesta para el valor de `n` no es general, si no que debe ser un número entero específico.

Habiendo creado correctamente el vector `s`, puede visualizar la convergencia de la serie usando el comando:

```r
plot(cumsum(s), type = "o", xlab = "n", ylab = expression(Z[n]))
```

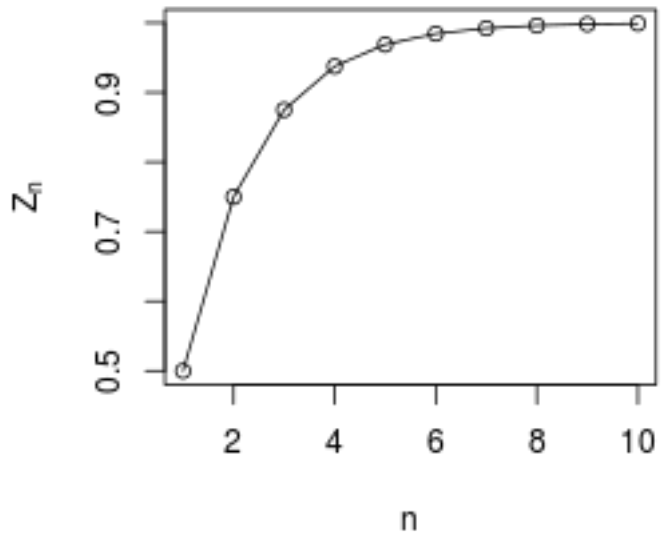

Figure 1: serie de Zenón

---

(3) Extra: Distancias entre puntos

Preámbulo: distancia euclidiana

La definición de distancia euclidiana se basa en el teorema de pitágoras. Para dos puntos en un espacio bidimensional (un plano) la distancia entre ambos se puede calcular a través de las coordenadas cartesianas de los mismos. El teoréma de pitágoras sirve para calcular la hipotenusa h de un triángulo rectánculo, con la fórmula:

$$h = \sqrt{C_{ad}^2 + C_{op}^2}$$

Aquí C_{ad} y C_{op} son los catetos *adyacente* y *opuesto* del triángulo (la magnitud de los dos lados menores del triángulo rectángulo; el orden no importa realmente).

Siguiendo la Figura 2, para calcular la distancia entre A y B, d_{AB}, esta fórmula se traduce en:

$$d_{AB} = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$$

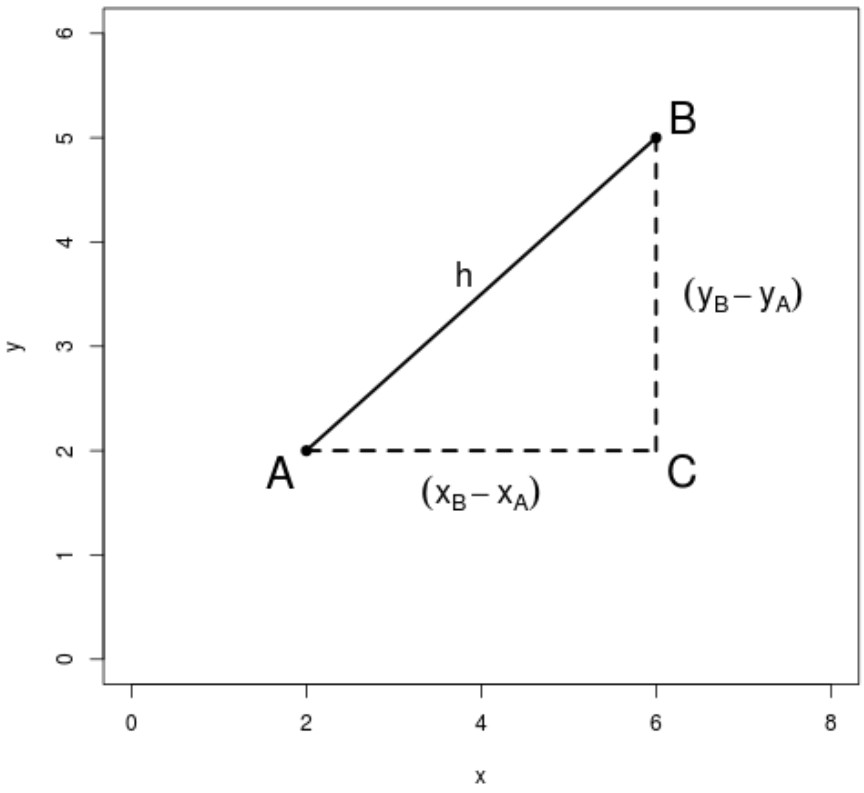

Figure 2: triángulo ABC

Figura 2: Los puntos A, B y C de coordenadas (x_A, y_A), (x_B, y_B) y (x_B, y_A) pueden formar un triángulo rectángulo como muestra la figura, cuyos catetos tienen los valores indicados entre paréntesis. Por lo tanto, la distancia entre A y B es la hipotenusa del triángulo y se puede hallar con la fórmula de Pitágoras.
Nota: puede reproducir este gráfico con la función `plotTriang` contenida en el script "plotTriang.R"; además lo invitamos a que examine dicho código si le interesa entender cómo se crea tal figura.

Como práctica previa al ejercicio, calcule la distancia entre los puntos A y B, de coordenadas (1, 6) y (5, 2) (es aproximadamente 8.94).

Tarea

La Figura 3 muestra un mapa de arbustos frutales (puntos en blanco), que se encuentran florecidos durante buena parte del año. El punto central (negro) es un panal de abejas que utilizan estos arbustos para alimentar a sus larvas. El objetivo de este ejercicio es encontrar dos arbustos, el más cercano y el más lejano respecto al panal, siguiendo un vuelo directo panal-arbusto.

Como ya puede adivinar, para esto usaremos el teorema de pitágoras, ya que necesitamos saber las distancias panal-arbusto para todos los puntos. Las coordenadas de los arbustos son generadas de forma aleatoria, con un vector para las X (longitud) y otro para las Y (latitud). Dichos vectores son `arb.x` y `arb.y` respectivamente.

Para aplicar la idea de la Figura 2, debemos primero determinar cuáles son los puntos y los triángulos asociados que nos interesan. De hecho, siguiendo la analogía de la figura, el punto C es irrelevante, ya que nos alcanza con las medidas de los catetos, las cuales se pueden obtener a partir de las coordenadas de A y B. En este caso "A" es siempre el mismo: el panal, de coordenadas (0.431, 0.587). El resto de los puntos son los arbustos. Dado que tenemos las coordenadas de todos, podemos hallar los catetos de los triángulos, siguiendo el esquema de la Figura 2. Esto no es más que una resta de coordenadas. Recuerde que no tiene que hacerlo de a uno por vez, ya que en R es posible *vectorizar* operaciones.

> Nota: en el script se espera (o mejor dicho, recomienda) que usted construya los vectores `cat.ad` y `cat.op`, cuyos valores serán los de los catetos adyacentes y opuestos respectivamente. En verdad no importa si los adyacentes son los catetos horizontales o verticales (lo mismo para los opuestos), porque no afecta al cálculo de las distancias. Tampoco afecta el orden en que resta las coordenadas, ya que los resultados se elevan al cuadrado al calcular la hipotenusa.

Usando este conocimiento entonces se pueden calcular las distancias, que no son otra cosa que las hipotenusas de triángulos imaginarios. Estas distancias se encontrarán en el vector `dst`. Una vez obtenido `dst`, es necesario encontrar los valores máximo y mínimo. Pero no necesitamos los valores en si, tenemos que sabér en qué posición, dentro del vector `dst`, estos se encuentran.

Para esto son muy útiles las funciones `which`. Acuda a la ayuda de R y aprenda sobre las funciones `which`, `which.max`, y `which.min`. Básicamente, devuelven las posiciones de un vector (u otro objeto, como una matriz) que cumplen con una condición particular. Utilizando `which.min` y `which.max` el script debe

crear los objetos `i` y `j` respectivamente (o si quiere puede usar `which`, pero es más complicado).

Finalmente sólo queda crear los vectores `arb.cerca` y `arb.lejos`, con las coordenadas de los arbustos. Para esto debe extraer las coordenadas apropiadas de los vectores `arb.x` y `arb.y`, usando los corchetes, para luego concatenarlas con la función `c`.

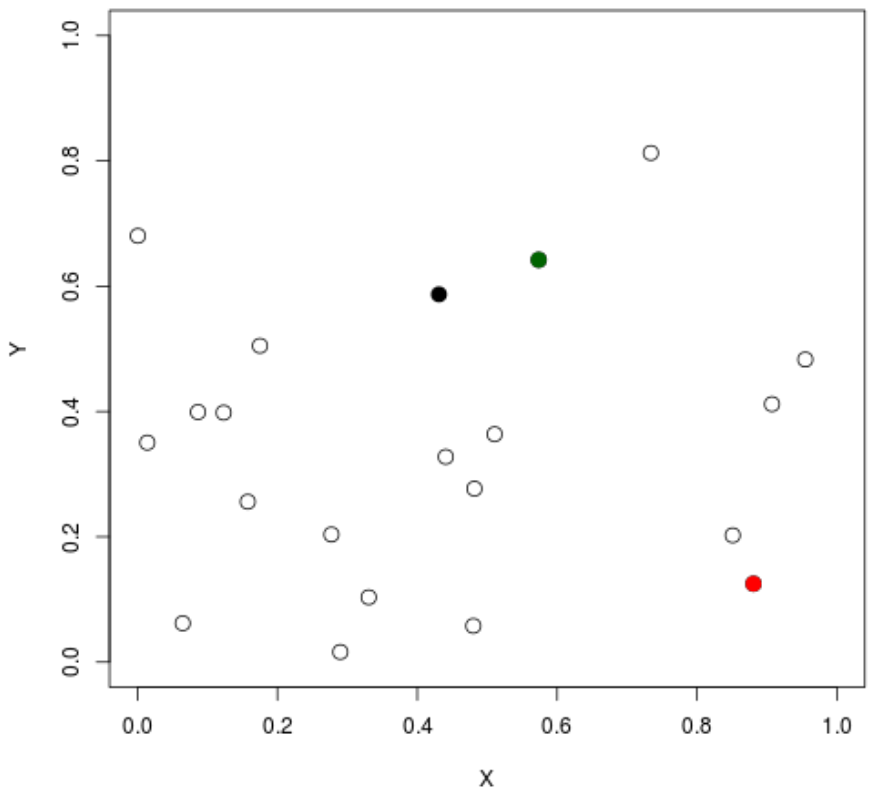

Figure 3: arbustos frutales (círculos abiertos) y un panal (círculo cerrado). Los puntos son generados con valores aleatorios, siguiendo los mismos comandos que se encuentran en el archivo del ejercicio. El arbusto más cercano al panal está marcado en verde y el más lejano en rojo.

Para comprobar que su resultado es correcto, puede utilizar el siguiente código:

```
plot(arb.x, arb.y, xlim = 0:1, ylim = 0:1)
points(0.431, 0.587, pch = 19)
```

```
points(arb.x[i], arb.y[i], pch = 19, col = "darkgreen")
points(arb.x[j], arb.y[j], pch = 19, col = "red")
```