Ejercicio de programación VI: Estructuras de Control

[IMSER 2013]

De otros años:

- Cambiar parte d del útimo parcial: si el bus tiene el máximo de pasajeros no sube a más nadie y si tiene el mínimo (0) no baja nadie.
- Crecimiento exponencial . . . ?
- Caminata del borracho?
- Símil función preg?
- Loops con errores: un while que es infinito, un for que tiene mal puesto el rango, . . .

---

1. Conteos por fila

1.a Loop for

Suponga que usted debe analizar regularmente matrices de datos, obtenidos en muestreos sucesivos, con cantidades de filas variables. Una de las tareas que se le pide realizar cotidianamente es hacer un conteo de la cantidad de valores mayores a 45 *por cada fila* de la matriz. Para no tener que hacerlo manualmente, usted decide que lo mejor es crear un script de R con el cual hacer este conteo automáticamente.

Para hacer el script lo mejor es usar una matriz de juguete con la cual hacer pruebas. Para esto sirven las siguientes líneas (que también se encuentran en el script del ejercicio):

```
# Generación de la matriz datos:
datos <- matrix(rpois(rpois(1, 125) * 15, 43), ncol = 15)
```

Para lograr su objetivo, usted deberá usar un loop `for`, con el cual completará el vector `out`, el cual contendrá las sumas de valores mayores a 45 por filas. Como un mini ejemplo, si su matriz `datos` es la siguiente:

```
datos <- datos[1:5, 1:5]
datos
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   33   35   47   48   41
## [2,]   49   34   43   56   52
## [3,]   38   53   54   30   44
## [4,]   50   51   31   31   38
## [5,]   34   47   44   44   44
```

Entonces el vector `out` será así:

```
out
```

```
## [1] 2 3 2 2 1
```

(Este es un ejemplo creado con números al azar, por supuesto.)

Nótese que el sistema de corrección espera que haya un `for` en el script (naturalmente) y que la *variable de iteración* (ver lección 6.2) tome como valores **números enteros positivos**.

1.b Extra: apply

Luego de hacer el script del ejercicio 1.a, usted se da cuenta que lo mismo se puede hacer y de forma "más elegante" con una función `apply`. Para esto usted deduce que debe crear una función propia capaz de contar la cantidad de valores mayores a 45 (o a un valor variable, si usted lo prefiere) de un vector numérico cualquiera y luego aplicar esta función con `apply` a todas las filas de `datos` (note que cada fila es un vector cuando se las trata por separado).

Para completar el ejercicio deberá simplemente usar `apply` para lograr el mismo vector `out` que en el ejercicio 1.a.

---

2. todavía no se bien que

2.a Zenón recargado

Hacer un while para encontrar el n para un valor arbitrario de ε, siguiendo las directivas del ejercicio de zenon.R

Como seguramente recordará, en el Repartido I se propuso calcular la serie que representa a la paradoja de Zenón, cuyo valor para el enésimo elemento se definió como:

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

En R, el valor de Z_n se puede obtener con las siguientes líneas de código:

```
n <- 20  # El n puede ser cualquiera en verdad...
Zn <- sum(1/(2^(1:n)))
```

En ocasión de aquel primer repartido, el objetivo era encontrar el mínimo `n` que cumpliera la desigualdad $1 - Z_n < \varepsilon$, siendo $\varepsilon = 10^{-6}$. El único método con que contaba en ese momento era manualmente cambiar el valor de `n` aumentando en una unidad, ejecutar el código y determinar manualmente si cumplía tal condición (para repetir el proceso en caso de que de no hacerlo). Sin embargo con las herramientas que usted a aprendido en esta unidad es posible ver que se pueden automatizar estos procedimientos, e incluso generalizarlo para cualquier ε. Este es el objetivo de este ejercicio.

Para esto usted deberá usar el loop `while`, ya que es (en principio) el más adecuado para esta tarea, pues no sabemos de antemano cual va a ser el `n` "correcto". Además recuerde que el loop `while` necesita que se cumpla una condición para continuar su ejecución, tal como el procedimiento de encontrar el `n` correcto.

La siguiente es la salida en la consola para el caso de `epsilon <- 5e-2`:

```
## n = 2 - Zn = 0.75
## n = 3 - Zn = 0.875
## n = 4 - Zn = 0.9375
## n = 5 - Zn = 0.9688
```

Recuerde que: **1.** en cada iteración `n` debe aumentar en una unidad, **2.** el script debe funcionar igual debien para cualquier valor de `epsilon` elegido y **3.** debe utilizar el código indicado anteriormente para obtener `Zn` o la corrección automática tendrá problemas con el redondeo de los valores.

2.b Guardar los valores

Puede usar for o while, da lo mismo

```
## n = 2 - Zn = 0.75
## n = 3 - Zn = 0.875
## n = 4 - Zn = 0.9375
## n = 5 - Zn = 0.9688
```

```
## [1] 5
```

```
## [1] 0.03125
```

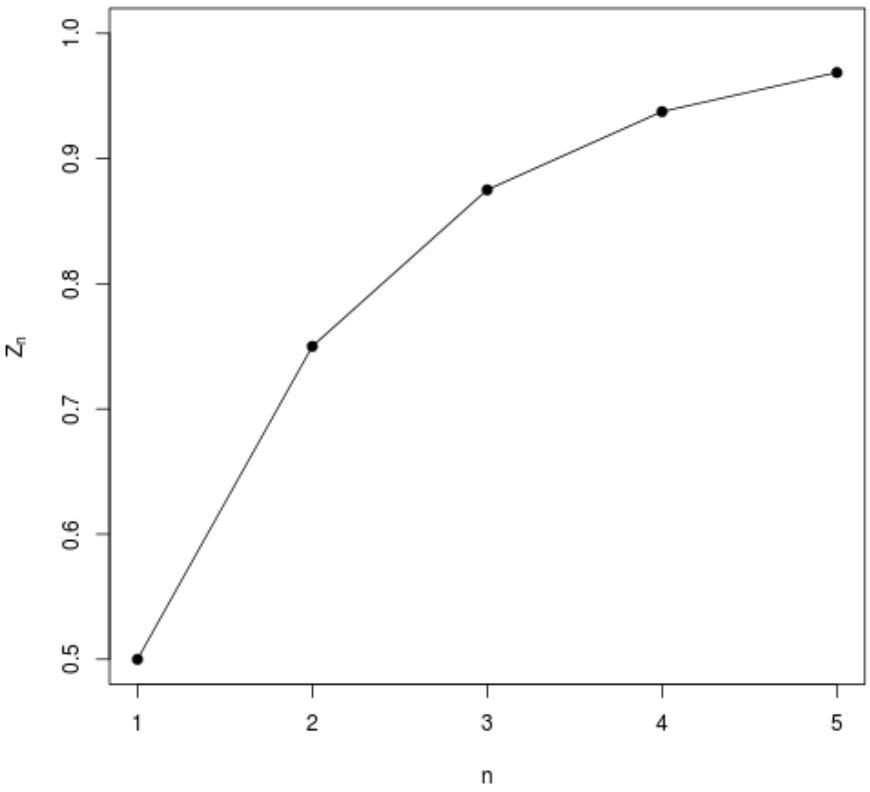

Figure 1: Serie de Zenón, con n = 5

Nota: en el ejemplo de

Parcial II

Curso IMSER 2012

Instrucciones:

En el archivo "parcial-II.R" ud. tiene un script en el cual deberá guardar todos los comandos del ejercicio, siguiendo la demarcación que se muestra en el mismo.

Nota: los ejercicios del parcial son dependientes de los anteriores en el sentido de que utilizan objetos creados, pero no implica que no se puedan tratar de resolver independientemente.

Los ejercicios con (∗) presentan un puntaje de 1.5, mientras que los que no tienen (∗) equivalen a 1 punto.

Una vez terminado el parcial usted deberá subir a la página del EVA el archivo "parcial-II.R" con su código.

Letra

Para simular la cantidad de pasajeros de un ómnibus urbano se ha creado el código que aparece sobre el final del ejercicio (y que también se encuentra en el script parcial-II.R). El criterio es el siguiente: el bus recorre 25 paradas, empezando el trayecto sin pasajeros. En cada parada se subirá una cantidad aleatoria de entre 0 a 6 personas (siendo todas las cantidades equiprobables), pero debido a que existe un máximo estipulado de 44 pasajeros, a partir del momento en que se alcanza ese valor el vehículo deja de subir gente.

(∗) a. Código incompleto

Completar el código: las líneas en blanco que se encuentran dentro de los límites del código indican en dónde debe cambiarse. **El resto de las líneas están correctas**.

Código fuente:

```
# Preparación:
paradas <- 25
pasajeros <- 0
## <<
registro[1] <- pasajeros
for (i in 1:paradas) {
    ## << A ver si no se llenó:
    if (pasajeros >= 44)
```

```
        {
            # Ajuste por si llega a 44 antes de terminar:
            registro[i:paradas] <- 44
            cat("Bus lleno!\n")  # Mensaje de aviso...
            break
        }  # <- este no estaba
    ## << Si no se corta el loop, se agrega un nuevo registro:
    registro[i] <- pasajeros
    # Para ir viendo cuánto hay:
    cat("Parada", i, "hay", pasajeros, "pasajeros\n")
}
plot(registro, xlab = "Parada", ylab = "No. de pasajeros")
```

Sugerencias: (1) la función `sample` puede ser útil para simular la subida de pasajeros. (2) Tanto en este como en los siguientes ejercicios, en caso de no estar seguro/a de cómo proceder, puede facilitar mucho la tarea hacer un diagrama de flujo sencillo antes de empezar a escribir código.

La siguiente imagen se obtuvo haciendo la simulación, ejecutando previamente `set.seed(11)` y luego el comando gráfico:

```
plot(registro, xlab = "Parada", ylab = "No. de pasajeros")
```

---

b. Cambio de loop

Modifique el código de la parte anterior de forma tal que haga lo mismo, pero utilizando un loop `while`.

Sugerencias: (1) agregue manualmente una variable (p.ej.: `i`) que sirva para indexar los distintos objetos y recuerde actualizarla en la linea correcta del código y (2) usar un `if` posterior al loop puede ser útil para sustituir los ceros del final por 44 en el vector registro (aunque no es de ninguna manera el único método).

Nota: el gráfico que se dió en la parte anterior tambiéen aplica para este ejercicio.

(*) c. Función `bus`

Modifique el código reparado en la parte **a** para crear una función llamada `bus` que ejecute la misma simulación, en la que el número de paradas y capacidad máxima del bus sean los argumentos de la misma. El nombre de estos argumentos son a su elección.

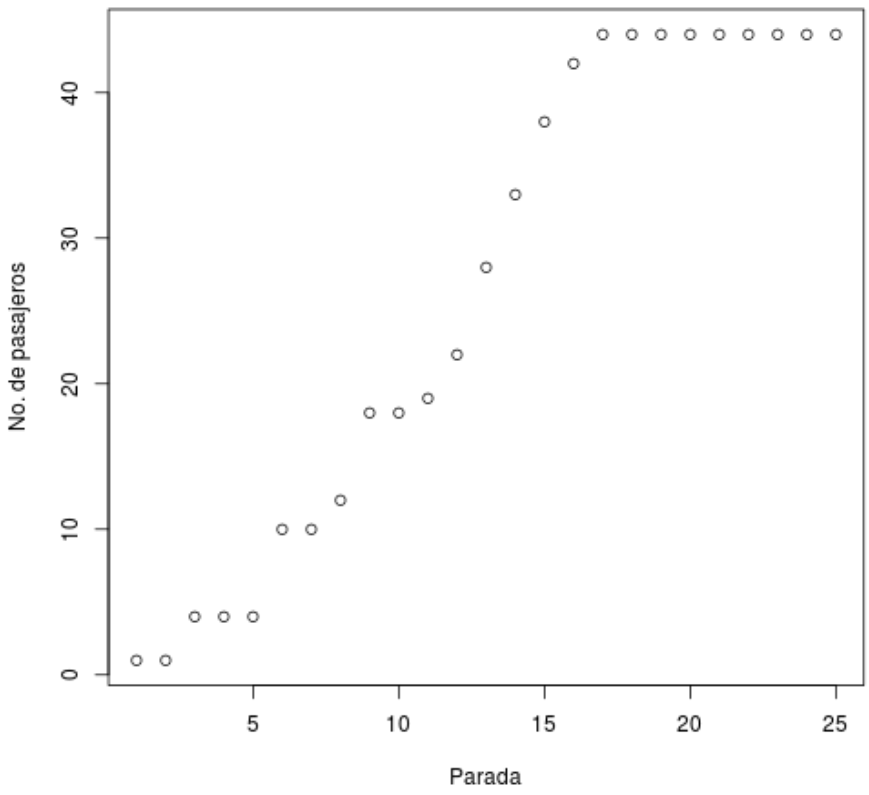

Figure 2: plot of chunk unnamed-chunk-10

Como salida la función debe devolver simplemente el vector `registro`.

Nota: recuerde que esta función debe trabajar correctamente para cualquier elección del número de paradas y máximo de pasajeros. En el siguiente ejemplo se muestra un caso que puede servir de referencia:

```
set.seed(11)
# Nro. de paradas = 40 Máximo de pasajeros = 80
x <- bus(40, 80)
```

```
## Parada 1 hay 1 pasajeros
## Parada 2 hay 1 pasajeros
## Parada 3 hay 4 pasajeros
## Parada 4 hay 4 pasajeros
## Parada 5 hay 4 pasajeros
## Parada 6 hay 10 pasajeros
## Parada 7 hay 10 pasajeros
## Parada 8 hay 12 pasajeros
## Parada 9 hay 18 pasajeros
## Parada 10 hay 18 pasajeros
## Parada 11 hay 19 pasajeros
## Parada 12 hay 22 pasajeros
## Parada 13 hay 28 pasajeros
## Parada 14 hay 33 pasajeros
## Parada 15 hay 38 pasajeros
## Parada 16 hay 42 pasajeros
## Parada 17 hay 45 pasajeros
## Parada 18 hay 47 pasajeros
## Parada 19 hay 48 pasajeros
## Parada 20 hay 51 pasajeros
## Parada 21 hay 52 pasajeros
## Parada 22 hay 56 pasajeros
## Parada 23 hay 58 pasajeros
## Parada 24 hay 60 pasajeros
## Parada 25 hay 60 pasajeros
## Parada 26 hay 63 pasajeros
## Parada 27 hay 65 pasajeros
## Parada 28 hay 65 pasajeros
## Parada 29 hay 65 pasajeros
## Parada 30 hay 67 pasajeros
## Parada 31 hay 70 pasajeros
## Parada 32 hay 72 pasajeros
## Parada 33 hay 74 pasajeros
## Parada 34 hay 75 pasajeros
## Bus lleno!
```

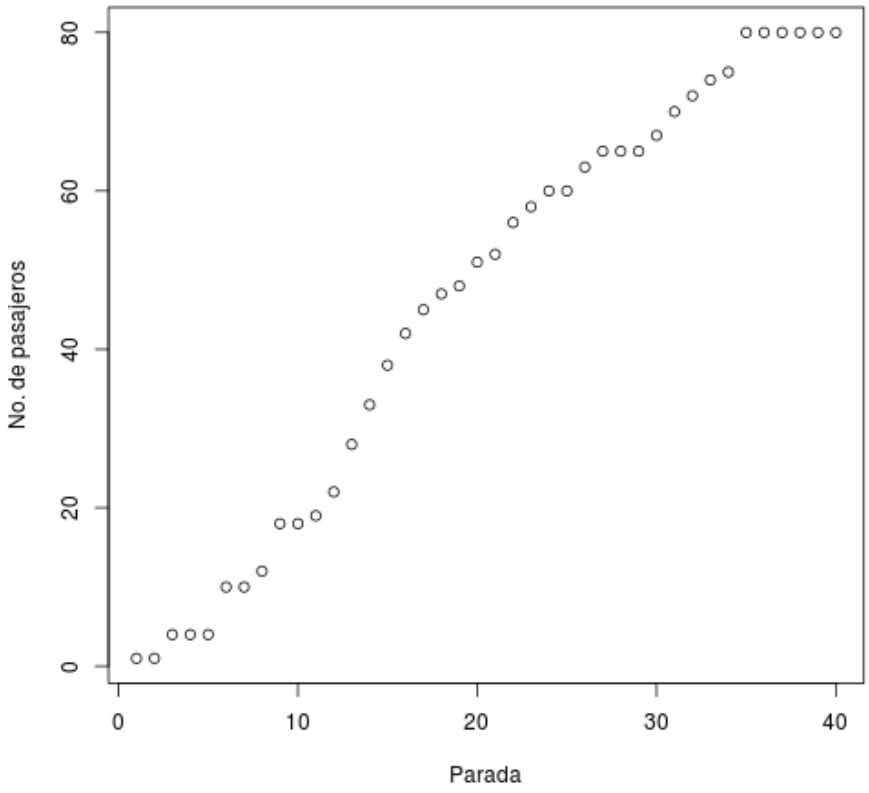

Figure 3: plot of chunk unnamed-chunk-12

---

d. Gente que también baja

Hacer una variante del código (de cualquiera de las partes anteriores) en la que además de subir personas, a partir de la parada 10 se bajen entre 1 y 5 pasajeros por parada. Tanto la subida y la bajada deben ejecutarse **antes** de determinar si se alcanzó el máximo estipulado de pasajeros y por lo tanto si debe dejar de detenerse el bus en las paradas.

Sugerencia: puede ser muy útil hacer un diagrama de flujo sencillo para planificar el código antes de escribirlo.

El siguiente es un ejemplo del resultado de aplicar los cambios que se piden en la función `bus`:

```
set.seed(11)
x <- bus(30, 60)
```

```
## Parada 1 hay 1 pasajeros
## Parada 2 hay 1 pasajeros
## Parada 3 hay 4 pasajeros
## Parada 4 hay 4 pasajeros
## Parada 5 hay 4 pasajeros
## Parada 6 hay 10 pasajeros
## Parada 7 hay 10 pasajeros
## Parada 8 hay 12 pasajeros
## Parada 9 hay 18 pasajeros
## Parada 10 hay 17 pasajeros
## Parada 11 hay 15 pasajeros
## Parada 12 hay 16 pasajeros
## Parada 13 hay 17 pasajeros
## Parada 14 hay 18 pasajeros
## Parada 15 hay 19 pasajeros
## Parada 16 hay 21 pasajeros
## Parada 17 hay 22 pasajeros
## Parada 18 hay 23 pasajeros
## Parada 19 hay 22 pasajeros
## Parada 20 hay 21 pasajeros
## Parada 21 hay 20 pasajeros
## Parada 22 hay 16 pasajeros
## Parada 23 hay 18 pasajeros
## Parada 24 hay 16 pasajeros
## Parada 25 hay 14 pasajeros
## Parada 26 hay 12 pasajeros
## Parada 27 hay 14 pasajeros
```

```
## Parada 28 hay 16 pasajeros
## Parada 29 hay 13 pasajeros
## Parada 30 hay 15 pasajeros
```

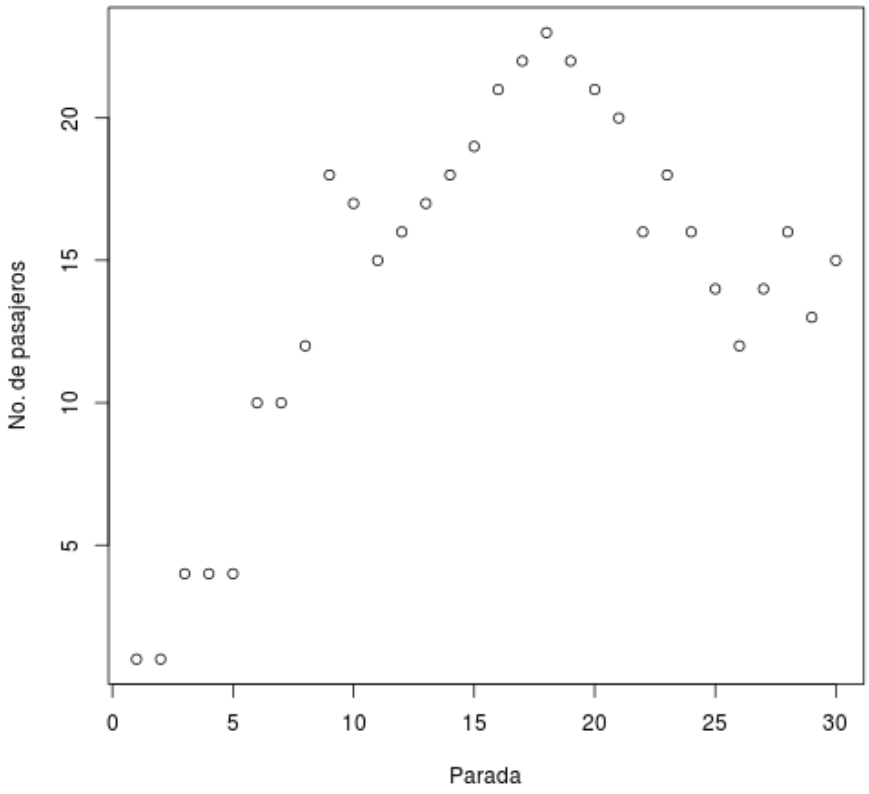

Figure 4: plot of chunk unnamed-chunk-14