Ejercicios de programación V: Funciones

[IMSER 2013]

Archivos incluidos:

El archivo con los ejercicios del práctico debe bajarse y descomprimirse en disco duro, creando la carpeta **`rep-X`** (nota: no debe dentro de ningún disco, partición o carpeta protegida a la escritura, como puede ser un disco duro externo de backup). Usted deberá abrir el RStudio y seleccionar dicha carpeta como su directorio de trabajo con `setwd` o en RStudio la combinación **Ctrl + Shift + K**. En esta carpeta se encuentran algunos archivos que usted deberá modificar:

- **`1-triangulo.R`**
- **`2.a-filtroc.R`**
- **`2.b-extra-aplicar.R`**
- **`2.c-educacion.R`**
- **`3.a-cambia.pares.R`**
- **`3.b-radio.R`**
- **`3.c-extra-distancias.R`**

Adicionalmente los siguientes archivos son necesarios, pero **no deben ser modificados** para que el método de calificación automático funcione correctamente:

- `datos`
- `evaluar.R`
- `edu.data.RData`
- `HandbookSpanish.pdf`
- `INSTRUCCIONES.pdf`

Mecanismo de corrección:

Nota: más recomendaciones **importantes** se hacen en el documento Dinámica de los repartidos.

Lo primero que debe hacer es cargar el archivo evaluar.R con la función `source` y la codificación de caracteres "UTF-8" (lo cual afecta a la función `evaluar` en particular), de la siguiente manera:

```
source("evaluar.R", encoding = "UTF-8")
```

Nótese que hemos dejado de usar la función `options`, de forma que de ahora en más **no ejecute el comando**:

```
options(encoding = "utf-8")  # No me ejecuten!
```

Este cambio se debe a que hemos detectado que esta elección trae más problemas que soluciones.

Si usted ha ejecutado todos los pasos anteriores correctamente, al usar el comando `ls()` verá que `"evaluar"` figura en su sesión y además en la consola debería ver lo siguiente:

```
Archivo de código fuente cargado correctamente

Chequeo de encoding:
  Los siguientes caracteres deben ser vocales con tilde:
    á - é - í - ó - ú
  Si *no se ven correctamente* corra el siguiente comando:
    source('evaluar.R', encoding = 'UTF-8')
```

Usted trabajará modificando los contenidos de los archivos de los ejercicios con RStudio (u otro programa de su preferencia) según las consignas que se describen a continuación. Luego de terminar cada ejercicio y **guardando el archivo** correspondiente en el disco duro, usted podrá verificar rápidamente si su respuesta es correcta ejecutando el comando:

```
evaluar()
```

y además podrá en todo momento verificar su puntaje con la función `verNotas()`. Tenga siempre en cuenta que, a **menos que sea indicado** por la letra del ejercicio, las soluciones deben ser genéricas y por lo tanto deben servir aún si se modifican los datos originales (i.e.: no use valores fijos si no comandos). Usualmente se utilizan valores generados de forma aleatoria para las correcciones automáticas. Los objetos que son evaluados en la corrección automática estarán

indicados con un asterísco en las instrucciones de cada script. Nótese además que en los archivos **se indica claramente en dónde se inicia y dónde finaliza su código** y que debe respetar esta organización para que la corrección de los ejercicios funcione bien.

NOTA: se agregó la función `fecha.datos` para facilitar el acceso al a información de 1. cuál es la versión que usted tiene en su pc y 2. cómo encontrar la fecha de la última versión.

Al finalizar

Una vez terminados y guardados los archivos de los ejercicios del repartido, usted deberá ejecutar `evaluar()` y seleccionar la última opción ("Todos") y luego subir el archivo "datos" (sin extensión), incluido en la carpeta "rep-1", a la sección de entregas de la portada del curso en la plataforma EVA. Este archivo se podrá reemplazar con uno más nuevo, en caso de que desee corregir algún error; en caso de querer que el archivo sea corregido antes de la fecha de entrega, puede cambiarle el nombre a "datos-finalizado", pero en ese caso la nota no se cambiará de ahí en adelante.

Código de Honor

Si bien animamos a que trabaje en equipos y que haya un intercambio fluido en los foros del curso, es fundamental que las respuestas a los cuestionarios y ejercicios de programación sean fruto del trabajo individual. En particular, consideramos necesario que no utilice el código creado por sus compañeros, si no que debe programar sus propias instrucciones, ya que de lo contrario supone un sabotaje a su propio proceso de aprendizaje. Esto implica también evitar, en la medida de lo posible, exponer el código propio a sus colegas. Como profesores estamos comprometidos a dar nuestro mayor esfuerzo para dar las herramientas y explicaciones adecuadas a fin de que pueda encontrar su propio camino para resolver los ejercicios.

En casos de planteos de dudas a través del foro, en los que considere que es imposible expresar un problema sin exponer su própio código, entonces es aceptable hacerlo. De todas formas en estos casos es preferible que envíe su código por correo electrónico directamente a un profesor, explicando la problemática.

---

1. Triángulos, volumen II

Si hacemos un poco de memoria recordaremos que en el ejercicio 1 del repartido I, tomábamos los catetos de un triángulo rectángulo y escribíamos el código

necesario para calcular el valor de la hipotenusa y el área del mismo. Como vimos entonces, el valor de la hipotenusa se calcula como:

$$hip = \sqrt{cat.op^2 + cat.ad^2}$$

mientras que el área del triángulo rectángulo es:

$$A = \frac{cat.op \cdot cat.ad}{2}$$

A partir de los valores de los catetos y de la hipotenusa es posible calcular el valor de los restantes ángulos del triángulo, siendo el ángulo adyacente

$$\alpha_{ad} = arccos\left(\frac{cat.ad}{hip}\right)$$

y el ángulo opuesto

$$\alpha_{op} = arccos\left(\frac{cat.op}{hip}\right)$$

En el presente ejercicio debemos escribir el código de una función llamada `triangulo` que, a partir de los mismos argumentos de las funciones `area` e `hipot` de aquella ocasión, o sea de los catetos de un triángulo rectángulo, calcule la hipotenusa, el área y los ángulos adyacente y opuesto del triángulo. La salida de la función deberá ser una lista con los objetos denominados de la siguiente manera: `hipotenusa`, `area`, `angulo.adyacente` y `angulo.opuesto`, en ese orden. Los ángulos deberán estar expresados en grados. En R, la salida de las funciones trigonométricas, como `asin` y `acos` están expresadas en radianes, por lo que se deberá hacer la transformación correspondiente, teniendo en cuenta que π radianes equivalen a 180^o, y por lo tanto:

$$\alpha_{gr} = 180 \cdot \frac{\alpha_{rad}}{\pi}$$

siendo α_{gr} y α_{rad} los equivalentes en grados y radianes de un ángulo α cualquiera. A modo de ejemplo, la salida deseada al evaluar esta función con el cateto adyacente y el opuesto valiendo respectivamente 4 y 3, sería la siguiente:

```
triangulo(4, 3)
$hipotenusa
[1] 5

$area
[1] 6
```

```
$angulo.adyacente
[1] 36.8699

$angulo.opuesto
[1] 53.1301
```

(Recuerde que la función triángulo debe dar el resultado correcto para cualquier par de valores positivos que sean asignados a los argumentos.)

---

2. Educación

La Organización de la Naciones Unidas (ONU) en el año 2011 presentó un informe basado en indicadores utilizados para medir y supervisar los Objetivos de Desarrollo del Milenio (ODM).

> "La Declaración del Milenio de las Naciones Unidas de 2000, basada en las conferencias mundiales de las Naciones Unidas durante el decenio de 1990, representó un fuerte compromiso con el derecho al desarrollo, la paz y la seguridad, la igualdad de género, la erradicación de las numerosas dimensiones de la pobreza y el desarrollo humano sostenible. En la Declaración, adoptada por 147 jefes de Estado y 189 Estados, se incorporaban lo que ha llegado a conocerse con el nombre de 'ocho objetivos de desarrollo del milenio', incluidas 18 metas con plazos cronológicos delimitados" (Fuente).

Para este ejercicio se utilizarán tres de los indicadores que han sido propuestos para cumplir con el objetivo 2 de los ODM Lograr la enseñanza primaria universal:

- Tasa Neta de Matriculación en la Enseñanza Primaria (`TM`).
- Porcentaje de alumnos que comienzan el primer grado y llegan al quinto grado (`PA`).
- Tasa de alfabetización de personas entre 15 y 24 años (`TA`).

Las tres variables están expresadas en porcentaje. El objetivo es realizar un cálculo grueso del porcentaje de niños que completan la educación primaria (`PC`) para ciertos paises, a partir de `TM` y `PA`, usando la fórmula: $PC = (TM \cdot PA)/100$. Finalmente se hará una regresión entre la tasa de alfabetización y el porcentaje de finalización de primaria calculado (i.e.: `TA` en función de `PC`).

El archivo "edu.data.RData" contiene una data.frame llamada `edu.data` con datos de varios años para ciertos países, usando las abreviaciones de arriba para cada indicador (importar a R con `load` o los botones de RStudio). Los datos fueron extraidos de la portada oficial de la ONU para los indicadores de los OMD.

Sin embargo no vamos a ir directamente a este cálculo, si no a desarrollar funciones que permitan hacerlo rápidamente con cualquier tabla de datos con una estructura similar a la que usamos como ejemplo.

2.a Filtro por columnas

Ya vimos que la función subset sirve para filtrar observaciones, las filas, en un data.frame con el fin de eliminar aquellas que no sirven a nuestros propósitos. En este ejercicio usted hará una función que haga algo similar, pero con las columnas. Considere la función `grep`. Esta sirve para reconocer patrones en vectores character, es decir, puede identificar palabras en un texto. Por ejemplo:

```
v <- c("glucosa", "citosina", "timina", "ribosa", "adenina", "sacarosa", "guanina")
i <- grep("osa", v)
v[i]
```

```
## [1] "glucosa"  "ribosa"   "sacarosa"
```

Como se puede ver aquí, `grep` devuelve un vector numérico con las posiciones de `v` tales que el string "osa" es parte de la palabra (en este casos, nombres de monosacáridos). Esto por supuesto puede usarse con nombres de columnas y es lo que se pide que usted haga en este ejercicio.

Vamos a usar esta capacidad para filtrar data.frames según sus nombres de columnas. En el script de este ejercicio, usted deberá codificar la función `filtroc`.

Argumentos:

- `x`: una data.frame con nombres de columnas útiles para hacer el filtrado.
- `clave`: string utilizado para seleccionar columnas (i.e.: un vector character de longitud 1). **Importante**: no debe importar si la `clave` está en mayúsculas o minúsculas (i.e.: da lo mismo usar `"TM"`, `"tm"` o `"tM"`). Para esto puede usar las funciones `tolower` o `toupper` para manipular vectores character.

Acciones:

1. Con alguna de las funciones `grep` determinar cuáles son las columnas de `x` que contienen el string `clave`.
2. Devolver un data.frame con las columnas seleccionadas en punto anterior. Esto puede ser un problema cuando se selecciona una sola columna (e.g.: la columna `TA` en `edu.data`). Para evitarlos indice la data.frame `x` sin usar coma (vea la diferencia entre escribir `edu.data[,11]` y `edu.data[11]` como ejemplo ilustrador).

El siguiente ejemplo debería ilustrar mejor el funcionamiento esperado de `filtroc`:

```
load("edu.data.RData")
filtroc(edu.data, "Tm")  # Se usan mayúsculas y minúsculas
```

```
##                   TM_2005 TM_2006 TM_2007 TM_2008 TM_2009
## Aruba                97.5    98.3    98.7    97.5    94.7
## Colombia             96.3    95.3    94.2    93.7    93.2
## Cuba                 95.2    96.1    98.8    99.5    99.8
## El Salvador          95.1    95.0    94.7    94.6    94.6
## Guatemala            95.1    96.4    97.2    96.8      NA
## México               99.6    99.6    99.4    99.5    99.6
## Nicaragua            94.6    92.9    93.1    94.4      NA
## Panamá               99.1    99.0    99.0    98.9    97.1
## Paraguay             95.1    93.2    90.8    88.2    85.7
## Perú                 99.6    99.7    99.7    98.2    97.2
## Trinidad y Tobago    91.3      NA    96.9    95.0    96.0
## Uruguay              97.6    99.9      NA    99.3    99.5
## Venezuela            92.8    93.4    94.4      NA    94.2
```

Nótese que esto permite elegir columnas con datos de `TM` sin importar el año en que fueron tomados.

Sugerencia: Antes de utilizar `grep` convertir las letras de la `clave` a mayúsculas/minúsculas según su preferencia; lo mismo para un vector con los nombres de columnas de `x` (pero no para los nombres de la data.frame en sí). Luego utilice estos vectores como entrada de `grep`.

2.b Extra: aplicar

El objetivo es hacer una función que combine las funciones `filtroc` (ej. 2.a) y `apply`. Esta función se llamará `aplicar`. Durante la corrección el sistema utilizará su propia versión de `filtroc`.

Argumentos:

- `x`: una data.frame cualquiera.
- `clave`: clave para seleccionar columnas (ver `filtroc`, ej. 2.a).
- `FUN`: función a aplicar a las filas de los datos de `x`, una vez que se seleccionan las columnas.
- `...`: argumento especial, utilizado para enviar argumentos nombrados a la función `FUN`, cualquiera sea esta (puede ser una cantidad arbitraria de argumentos; `?dotsMethods` o el el manual introductorio para ver la ayuda de R respecto a este argumento especial).

Acciones:

1. Seleccionará las columnas de `x` según la `clave` dispuesta por el usuario (tal como lo debería hacer la función `filtro`), resultando en un data.frame.
2. Aplicará la función `FUN` a todas las *filas* de la data.frame resultante del paso anterior (se sugiere usar `apply` aquí).
3. Devolverá un vector con los valores obtenidos en el paso anterior.

```
load("edu.data.RData")
aplicar(edu.data, "pA", mean, na.rm = TRUE)
```

```
##             Aruba          Colombia              Cuba       El Salvador
##             90.43             83.50             96.30             74.90
##         Guatemala            México         Nicaragua            Panamá
##             63.53             92.82             47.50             87.84
##          Paraguay              Perú Trinidad y Tobago           Uruguay
##             78.15             87.42             91.40             93.88
##         Venezuela
##             90.36
```

Nota: en este ejemplo el argumento especial `...` sirve para enviar el argumento nombrado `na.rm = TRUE` a la función `mean` (a través del argumento `...` de la propia función `apply`). Los demás argumentos son: `x = edu.data`, `clave = "Tm"` y `FUN = mean`. En la corrección automática se utilizarán otras funciones y argumentos para poner a prueba el ejercicio.

2.c Función educación

La función `educacion` será la que calcule el `PC` ("porcentaje de colnclusión" o de "finalización" de la primaria) y además haga la regresión entre `TA` y `PC`. La función supone de antemano que todas las columnas tienen nombres con los prefijos `TM`, `PA` o `TA`, y además que hay una única columna `TA`. Si a usted le interesa puede agregar más argumentos para que su función no parta de este supuesto, pero es una opción no contemplada por el ejercicio.

Note que en este ejercicio es posible utilizar las funciones que se piden en 2.a y 2.b, si bien no es obligatorio. De todas formas, durante la corrección el sistema usará su propia versión de dichas funciones.

Argumentos:

- `x`: data.frame con los datos de entrada (como `edu.data`), en la que deben estar los TM, PA y TA, tal como se explica anteriormente.

Acciones:

1. Calcular, por país/fila, el promedio de los valores de TM y de PA. Para esto pueden ser útiles las funciones `filtroc` del ejercicio 2.a y `aplicar` del 2.b.
2. Calcular con estos el valor de PC para cada país (lo que debe hacerse con los promedios de TM y PA obtenidos en el punto anterior).
3. Agregar estos tres valores calculados como columnas a la data.frame de entrada (`x`).
4. Realizar una regresión lineal de TA en función de PC.
5. Devolver un objeto de clase list con 1. la regresión y 2. la data.frame modificada

Nota: los cálculos de promedios deben hacerse dejando de lado los valores `NA`.

Salida: `educacion` debe devolver un objeto de clase `list`, el cual debe contener los siguientes objetos (y en el mismo orden):

- `reg`: el objeto `lm` obtenido al hacer la regresión lineal.
- `datos`: data.frame con los datos originales más las columnas correspondientes a los promedios de `TM`, `PA` y `PC`, en ese orden.

Nota: la función `summary` además de imprimir en la consola datos útiles de una regresión necesarios para este ejercicio, también genera una **salida invisible** (ver lección 5.3).

El siguiente es un ejemplo obtenido con los datos y una función `educacion` completada:

```
load("edu.data.RData")
e <- educacion(edu.data)
e$reg
```

```
##
## Call:
## lm(formula = TA ~ PC, data = x)
##
## Coefficients:
## (Intercept)           PC
##      75.095        0.268
```

```
str(e$datos)
```

```
## 'data.frame':    13 obs. of  14 variables:
##  $ TM_2005: num  97.5 96.3 95.2 95.1 95.1 99.6 94.6 99.1 95.1 99.6 ...
##  $ TM_2006: num  98.3 95.3 96.1 95 96.4 99.6 92.9 99 93.2 99.7 ...
##  $ TM_2007: num  98.7 94.2 98.8 94.7 97.2 99.4 93.1 99 90.8 99.7 ...
##  $ TM_2008: num  97.5 93.7 99.5 94.6 96.8 99.5 94.4 98.9 88.2 98.2 ...
##  $ TM_2009: num  94.7 93.2 99.8 94.6 NA 99.6 NA 97.1 85.7 97.2 ...
##  $ PA_2005: num  NA 80.8 96.9 67.3 63.3 92.1 50.2 85.2 76.2 NA ...
##  $ PA_2006: num  93.4 85.2 96.8 68.9 62.5 92.3 43.9 88.2 79.1 89.7 ...
##  $ PA_2007: num  86.5 NA 96.1 75.7 64.8 91.5 48.4 85.2 79.2 82 ...
##  $ PA_2008: num  91.4 NA 95.5 76.1 NA 94 NA 86.8 78.1 87.6 ...
##  $ PA_2009: num  NA 84.5 96.2 86.5 NA 94.2 NA 93.8 NA 90.4 ...
##  $ TA     : num  99.1 98 100 94.9 87 ...
##  $ TM     : num  97.3 94.5 97.9 94.8 96.4 ...
##  $ PA     : num  90.4 83.5 96.3 74.9 63.5 ...
##  $ PC     : num  88 78.9 94.3 71 61.2 ...
```

---

3. Funciones con problemas

En este ejercicio se propone arreglar el código de algunas funciones simples (y otra no tan simple). Para esto la idea es utilizar los métodos y conceptos vertidos en la lección 5.4: **Depuración de funciones**.

Recuerde que hay errores más comunes que otros. Por ejemplo funciones, operadores u objetos **mal escritos**, **paréntesis**, **llaves**, o **corchetes** mal cerrados y **errores de indización** (como poner 1 sólo número para indizar una matriz), son de los más frecuentes. Otros también "populares" son el utilizar de forma inconsistente las comillas (i.e.: mezclar `"` con `'` descuidadamente) u olvidar poner una coma entre argumentos. Recomendamos tengan cerca la lección 1.3 extra: **errores y afines**.

3.a Cambiador de valores en subíndices pares...

Esta sencilla función toma un vector `x` y cambia aquellos valores que se encuentran en las posiciones pares del mismo (`x[2]`, `x[4]`, etc.), usando como sustitutos los elementos del vector `subs` (el segundo argumento). Un ejemplo de salida de esta función puede ser el siguiente:

```
cambia.pares(1:6, NA)
```

```
## [1]  1 NA  3 NA  5 NA
```

El objetivo de este ejercicio es arreglar el código de la función contenida en el script asociado de forma tal que ejecute correctamente su tarea. El código tiene cuatro errores,

3.b Radios

La función radio toma como argumento el valor `r`, un número cualquiera, y calcula tres valores asociados con circunferencias y esferas: el perímetro (P), área (A; de la circunferencia) y volumen (V; de la esfera), utilizando las fórmulas:

$$P = 2 \cdot \pi \cdot r$$
$$A = \pi \cdot r^2$$
$$V = \frac{4 \cdot \pi \cdot r^3}{3}$$

La función está pensada para generar una salida invisible, al mismo tiempo que imprimir en la consola los resultados obtenidos (nótese el uso de la función `cat` para este cometido). El siguiente es un ejemplo de salida de la función:

```
x <- radio(5)
```

```
## Perímetro: 31.42
## Área:      78.54
## Volumen:   523.6
```

```
x
```

```
##      P      A      V
##  31.42  78.54 523.60
```

El objetivo de este ejercicio es arreglar el código de la función contenida en el script asociado de forma tal que ejecute correctamente su tarea. En total hay 5 errores (aunque 1 de ellos incluye 2 cambios en una misma línea).

3.c Extra: encuentra distancias

La función `distancias`, escrita en el archivo "distancias.R", busca realizar una tarea parecida a la que ya se hizo en el Repartido I del curso: calcular las distancias de un punto a un conjunto de coordenadas (ej. 1.c). En este caso se utiliza un algoritmo alternativo para calcular las distancias, el cual es más robusto dadas las limitaciones de las computadoras.

El significado de los argumentos de la función están explicados en el propio archivo, así como varios de los pasos internos, a través del uso de comentarios. Al igual que en ejercicios anteriores, el objetivo es corregir los errores que tiene el archivo para que la función `distancias` cumpla su tarea correctamente. El siguiente es un ejemplo de cómo debería ser la salida de la función (incluyendo la figura):

```
pts <- matrix(rnorm(20), ncol = 2)
x <- distancias(pts, p = c(0.3, -0.1))
```

```
## d.max = 2.33 - punto: -0.56 2.06
## d.min = 0.35 - punto: -0.02 -0.24
```

```
x
```

```
## $dists
##  d.max  d.min
## 2.3268 0.3486
##
## $posiciones
## i.max i.min
##     3    10
##
## $puntos
##               x       y
## d.max -0.56210  2.0612
```

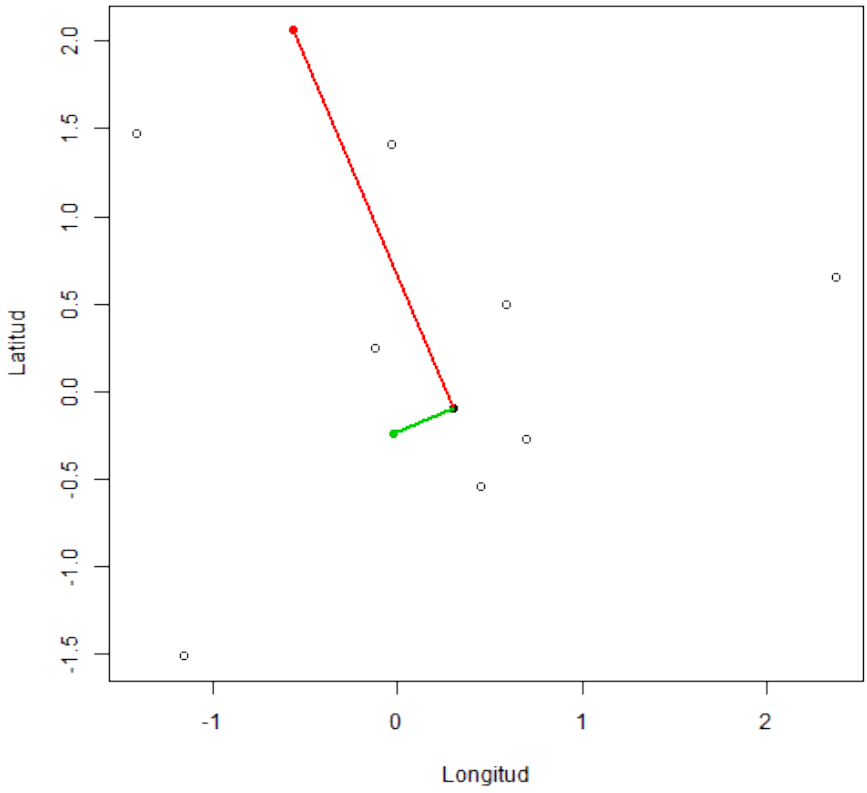

Figure 1: salida gráfica de la función distancias

```
## d.min -0.01781 -0.2432
##
## $centro
## [1]  0.3 -0.1
```

Hay 5 errores para corregir, pero dos de ellos no van a generar ningún mensaje de error ni de aviso, excepto al usar `evaluar`.