Ejercicio de programación VI: Estructuras de Control

[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.a-loop-for.R`
- `1.b-extra-apply.R`
- `2.a-zenon-recargado.R`
- `2.b-guardar-valores.R`
- `3.a-limites.R`
- `3.b-no-suben.R`
- `3.c-extra-heterogeneidad.R`
- `3.d-extremos.R`

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

- `datos`
- `evaluar.R`
- `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')

Para comprobar la fecha de su archivo datos ejecute:
>> fecha.datos()
```

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-X", 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. 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[sample(25, 1)] <- 46  # Alguno > 45 al menos
datos
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   49   35   44   54   45
## [2,]   35   54   41   37   41
## [3,]   50   41   38   43   48
## [4,]   46   42   34   45   47
## [5,]   46   41   39   46   45
```

Entonces el vector `out` será así:

```
out
```

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

(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. Una vez más, Zenón

2.a Zenón recargado

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 (recomendamos que use en particular la segunda línea en su script, para evitar problemas de redondeo):

```
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 **estructuras de control** 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. **Nota**: no es trivial el orden en el que actualiza los objetos `n` y `Zn`.

La siguiente es la salida en la consola para el caso de `epsilon <- 5e-2` y empezando con `n <- 1` (aunque no lo muestre en la salida):

```
## 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 de bien para cualquier valor de `epsilon` elegido y
3. es lo mejor 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

Luego de hacer la parte 2.a, usted decide que es buena idea guardar los valores de `Zn` un único vector numérico, al cual llamará `Z`. Para esto es necesario modificar tanto la preparación como las instrucciones del loop, como recordará de las lecciones de la unidad 6.

La siguiente es la salida en la consola para el caso de `epsilon <- 5e-2` y empezando con `n <- 1` (aunque no lo muestre en la salida):

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

```
Z
```

```
## [1] 0.5000 0.7500 0.8750 0.9375 0.9688
```

La figura 1 se obtuvo con el siguiente código:

```
plot(Z, xlab = "n", ylab = expression(Z[n]), type = "o", pch = 19, ylim = c(0.5,
    1))
```

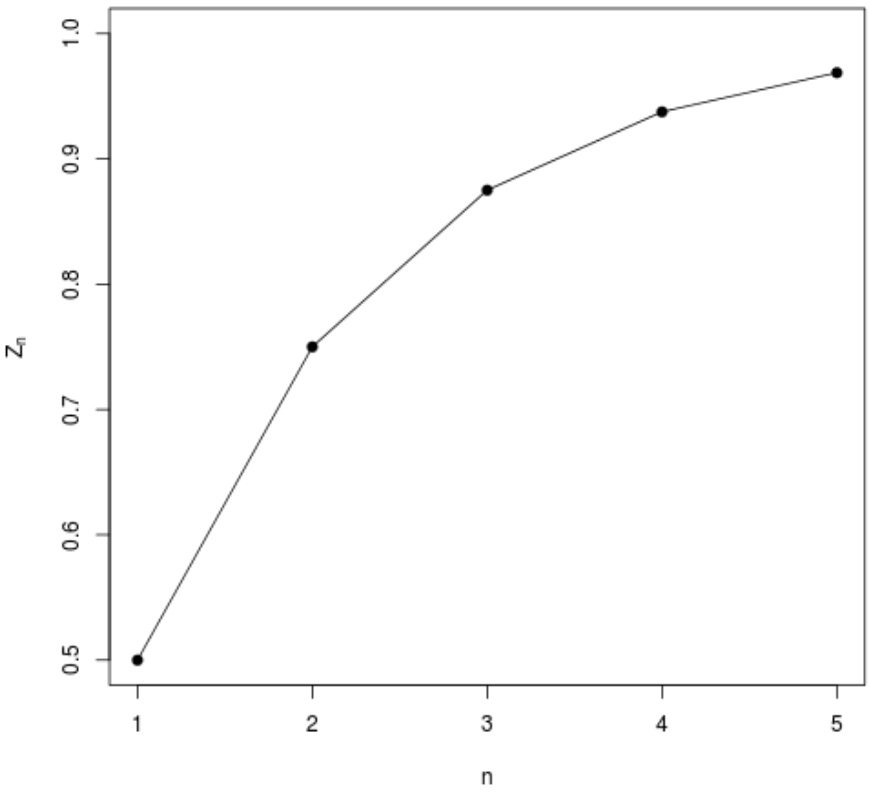

Figure 1: Serie de Zenón, con n (final) = 5

3. Línea urbana

Como proyecto del gobierno municipal local, usted va a simular el trayecto de un ómnibus a través de su recorrido. El objetivo es obtener una simulación razonablemente fiel a los datos observados para esa línea. Se supone que dicho gobierno va a utilizar los datos generados y complementarlo con análisis matemáticos rigurosos, a fin de diseñar un sistema de transporte óptimo que tenga un buen balance entre el costo de flota y eficiencia de transporte.

Nota: en este ejercicio, particularmente en las partes c y d, puede facilitar mucho la tarea hacer un diagrama de flujo sencillo antes de empezar a escribir código.

3.a Poniendo límites

Usted ya ha hecho un modelo que, si bien es muy crudo aún, tiene los componentes básicos necesarios. El código es el siguiente:

```
set.seed(0)
paradas <- 50
maximo <- 60
pasajeros <- rpois(1, 10)
registro <- numeric(paradas)
registro[1] <- pasajeros
for (i in 2:paradas) {
    # Bajan:
    pasajeros <- pasajeros - rpois(1, 2)

    # Suben:
    pasajeros <- pasajeros + rpois(1, 3)

    registro[i] <- pasajeros  # Actualiza registro
}
plot(registro, type = "o", pch = 19, xlab = "Parada", ylab = "No. de pasajeros",
    ylim = c(0, max(registro)))
abline(h = c(0, maximo), lty = 3)
```

Como puede ver en el gráfico generado (fig. 2), la cantidad de pasajeros sobrepasa en ocasiones al menos uno de los límites posibles para un bus real (en este caso, el máximo es 60 y el mínimo es 0, naturalmente).

El objetivo de este ejercicio tiene que ver con este último: **modificar el código de forma tal que se impongan límites a los valores posibles de `pasajeros`**. En particular, siempre que la cantidad de pasajeros sea mayor

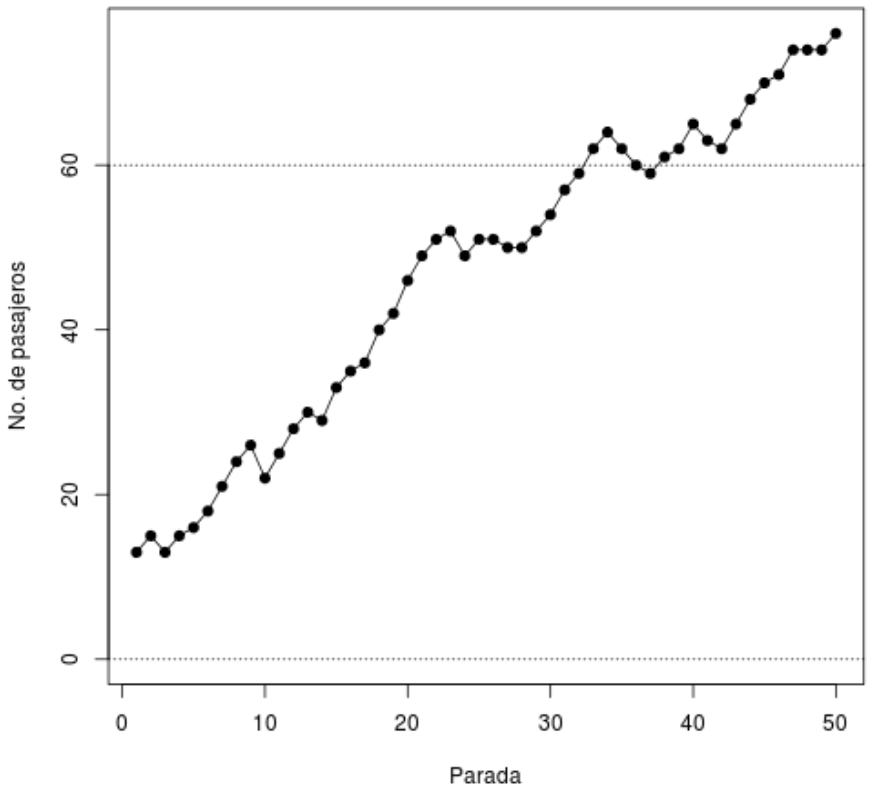

Figure 2: Modelo 'crudo'; las líneas indican los límites que debería tener el nro. de pasajeros

que el máximo estipulado (siendo que ya subieron y bajaron los pasajeros correspondientes a ese turno), debe corregirse el valor de `pasajeros`, dejándolo en el `maximo`. El caso análogo debe ocurrir si `pasajeros` toma valor negativo, cambiándolo por 0.

(En la figura 3 se muestra una simulación en la que se cumplen estas condiciones.)

3.b ¿Cuántos se quedan afuera?

Uno de los principales objetivos del gobierno municipal es el de saber cuántos pasajeros quedan sin poder subir al bus en cada parada. Para esto es necesario registrar, en su simulación, el número de pasajeros que sobrepasan el límite máximo de la capacidad del vehículo.

Usted decide crear un vector numérico, el cual tendrá tantos valores como paradas hay en el recorrido y registrará para cada una el la cantidad de personas que no han podido subir al vehículo. Dicho vector será `nosuben` y deberá tener la longitud correcta *antes* de iniciar el loop. La figura 3 muestra la progresión del número de pasajeros (negro) y personas que no pueden subir (verde) a través de una simulación.

Note que el vector `nosuben` no puede tener valores negativos, su valor mínimo será 0.

3.c Heterogeneidad

El modelo creado no está nada mal, pero desde las oficinas de la municipalidad llegan reportes de que la cantidad de personas que suben y bajan varía mucho según la altura del recorrido en la que se encuentre el bus. Luego de mirar cuidadosamente los datos, usted concluye que:

1. La cantidad de pasajeros que suben aumenta mucho en la la mitad del recorrido. Usted considera razonable decretar que a partir de la parada 15 y hasta la parada 35, el número promedio de gente que sube es de 8, en lugar de 3 como en el resto del recorrido. En otras palabras, en ese tramo, la cantida de pasajeros que deberá subir será `rpois(1, 8)`, mentras que en el resto del recorrido serán `rpois(1, 3)`.

2. Con la gente que baja el patrón es diferente: aumenta bastante la bajada sobre todo en la segunda mitad del trayecto. En particular, a partir de la parada 33 usted considera razonable determinar que bajan 5 personas en promedio por parada. En otras palabras, antes de la parada 33 bajarán `rpois(1, 2)` por turno, luego serán `rpois(1, 5)`.

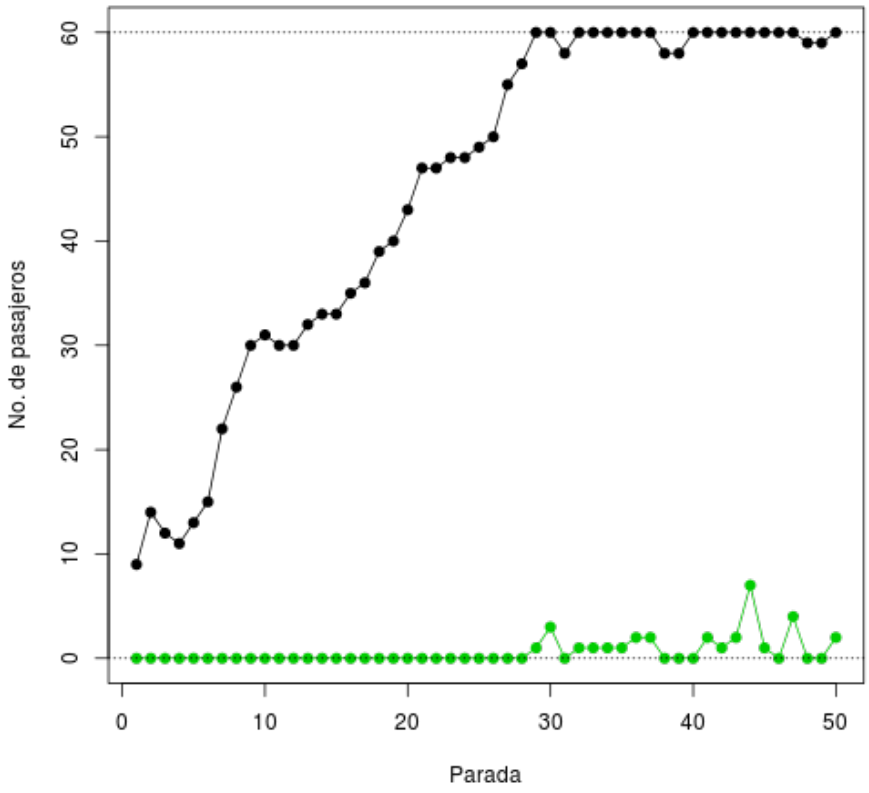

Figure 3: Simulación con límites y el vector nosuben en verde

Nota: el sistema de corrección del ejercicio espera que usted use **siempre** líneas como estas para simular subidas o bajadas de pasajeros:

```
# Usando lambdas 2 y 3 como ejemplos:
pasajeros <- pasajeros - rpois(1, 2)  # Bajan
pasajeros <- pasajeros + rpois(1, 3)  # Suben
```

(Aquí figuran 2 y 3 como promedio, pero en cada caso usted deberá cambiar el valor según lo que dice la letra.)

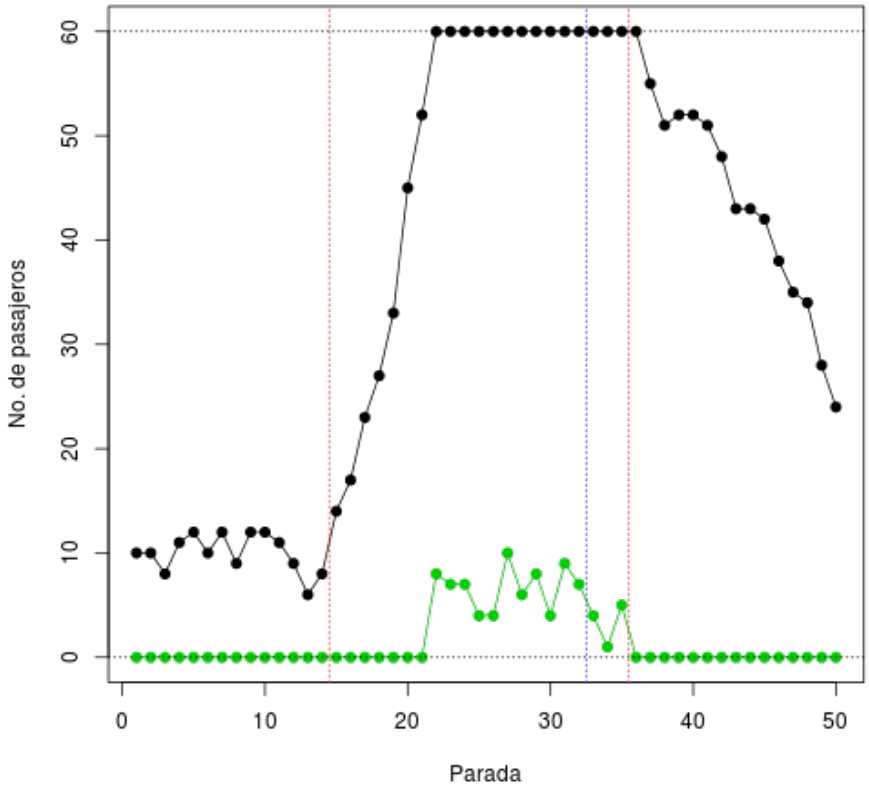

Figure 4: La subida y bajada de pasajeros varía; las líneas verticales muestran los cambios en subidas (rojo) y bajadas (azul)

3.d Extra: extremos

Estando ya bastante conforme con la performance de su modelo, usted se da cuenta de un detalle: nadie, o como mucho una cantidad insignificante de personas, toma un ómnibus para recorrer menos de 5 paradas. Por esta razón decide modificar nuevamente su código para dar cuenta de este detalle. Las modificaciones son las siguientes:

1. Si el bus se encuentra en una parada *anterior* a la quinta, nadie se bajará.
2. Si el bus se encuentra en una parada *posterior* a la 45, nadie se subirá.

La figura 5 muestra un ejemplo hecho incluyendo estas dos modificaciones. Como ya se ha mencionado anteriormente, es muy recomendable utilizar diagramas de flujo (como el mostrado en la lección 6.1) para entender la dinámica y poder crear el código de forma más ordenada y eficiente.

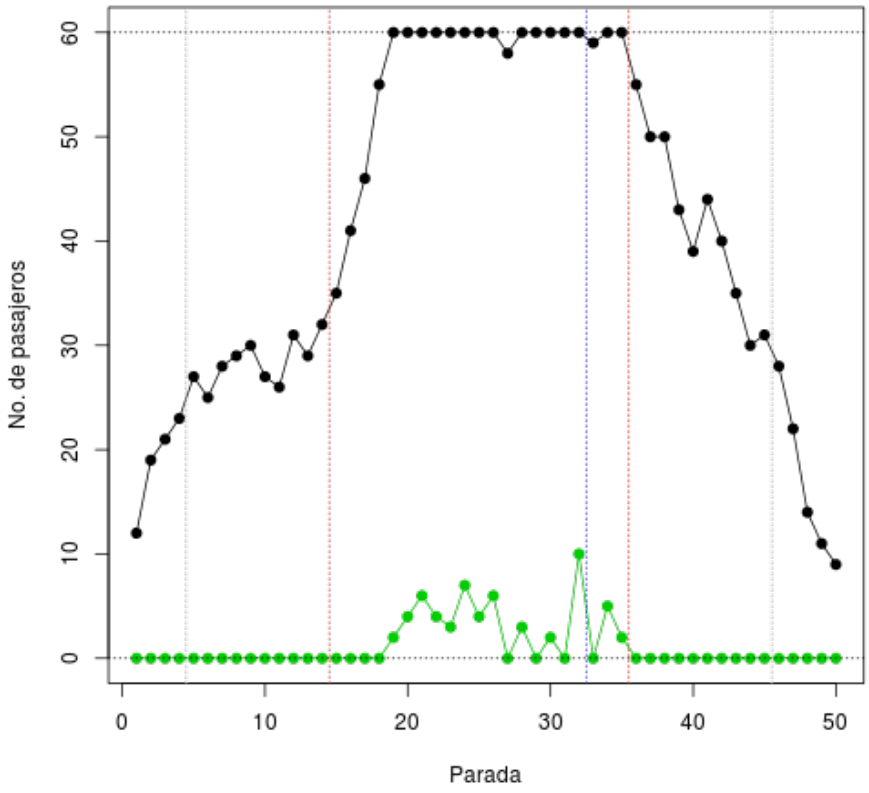

Figure 5: Modelo final; líneas grises indican los extremos