

TAREA # 8

Modelo de urnas

Natalia Berenice Pérez López

20 de octubre de 2021

1. Objetivo

Suponiendo que cúmulos con c o más partículas (haciendo referencia al tamaño crítico c) son suficientemente grandes para filtrar, el objetivo de esta práctica es graficar para $n = 100000$ con $k \in \{100, 200, 400\}$ en cada iteración t el porcentaje de las partículas que se lograría filtrar si el filtrado se realiza finalizando esa iteración.

2. Desarrollo

Para generar el código de esta práctica se realizaron algunas ideas y pruebas iniciales, las cuales se encuentran en [mi repositorio](#) en GitHub. Se inició tomando como base el código revisado en clase que combina los fenómenos de agregación y rotura [2]. Las modificaciones que se le realizaron al código fueron: mantener constante la cantidad de partículas n con un valor de 100000, agregar un ciclo `for` para variar el tamaño de los cúmulos $k \in \{100, 200, 400\}$, agregar un ciclo `for` para hacer 30 réplicas del experimento para cada valor de k y así poder analizar los resultados en un diagrama caja-bigote, agregar un condicional `if` para obtener los histogramas del tamaño de los cúmulos solamente en la primer réplica de cada valor k y agregar un `data.frame` para almacenar el porcentaje de filtrado en cada iteración.

A continuación se muestra el código objetivo de la práctica:

```
1 library(testit) # para pruebas, recuerda instalar antes de usar
2 tamas <- c(100, 200, 400)
3 n <- 100000
4 df = data.frame()
5
6 for (k in tamas){
7   for (replica in 1:30){
8     originales <- rnorm(k)
9     cumulos <- originales - min(originales) + 1
10    cumulos <- round(n * cumulos / sum(cumulos))
11    assert(min(cumulos) > 0)
12    diferencia <- n - sum(cumulos)
13    if (diferencia > 0) {
14      for (i in 1:diferencia) {
15        p <- sample(1:k, 1)
16        cumulos[p] <- cumulos[p] + 1
17      }
18    } else if (diferencia < 0) {
19      for (i in 1:-diferencia) {
20        p <- sample(1:k, 1)
21        if (cumulos[p] > 1) {
22          cumulos[p] <- cumulos[p] - 1
23        }
24      }
25    }
26
27    png("p8_init.png")
28    plot(hist(cumulos), main="Estado inicial",
29          xlab="Tama\u{00f1}o de c\u{00fa}mulos", ylab="Frecuencia absoluta")
30    graphics.off()
31  }
```

```

32 assert(length(cumulos[cumulos == 0]) == 0) # que no haya vacios
33 assert(sum(cumulos) == n)
34 c <- median(cumulos) # tama o critico de cumulos
35 d <- sd(cumulos) / 4 # factor arbitrario para suavizar la curva
36
37 primero <- as.data.frame(table(cumulos))
38 names(primerio) <- c("tam", "num")
39 primero$tam <- as.numeric(levels(primerio$tam))[primerio$tam]
40 assert(sum(primerio$num * primero$tam) == n)
41
42 filtrados1 = primero[primero$tam >= c,]
43 filtrados1$cont = filtrados1$tam * filtrados1$num
44 f1 = sum(filtrados1$cont) # particulas removidas
45 porcentaje1 = 100 * f1/n # porcentaje exitosamente filtrado
46 paso1 = 0
47 resultado1 = c(k, replica, paso1, porcentaje1, c)
48 df = rbind(df, resultado1)
49 names(df) = c("k", "Replica", "Iteracion", "filtrado", "c")
50 assert(sum(abs(cumulos)) == n)
51
52 rotura <- function(x) {
53   return (1 / (1 + exp((c - x) / d)))
54 }
55 union <- function(x) {
56   return (exp(-x / c))
57 }
58 romperse <- function(tam, cuantos) {
59   romper <- round(rotura(tam) * cuantos) # independientes
60   resultado <- rep(tam, cuantos - romper) # los demas
61   if (romper > 0) {
62     for (cumulo in 1:romper) { # agregar las rotas
63       t <- 1
64       if (tam > 2) { # sample no jala con un solo valor
65         t <- sample(1:(tam-1), 1)
66       }
67       resultado <- c(resultado, t, tam - t)
68     }
69   }
70   assert(sum(resultado) == tam * cuantos) # no hubo perdidas
71   return(resultado)
72 }
73 unirse <- function(tam, cuantos) {
74   unir <- round(union(tam) * cuantos) # independientes
75   if (unir > 0) {
76     division <- c(rep(-tam, unir), rep(tam, cuantos - unir))
77     assert(sum(abs(division)) == tam * cuantos)
78     return(division)
79   } else {
80     return(rep(tam, cuantos))
81   }
82 }
83 freq <- as.data.frame(table(cumulos))
84 names(freq) <- c("tam", "num")
85 freq$tam <- as.numeric(levels(freq$tam))[freq$tam]
86 duracion <- 50
87 digitos <- floor(log(duracion, 10)) + 1
88 for (paso in 1:duracion) {
89   assert(sum(cumulos) == n)
90   cumulos <- integer()
91   for (i in 1:dim(freq)[1]) { # fase de rotura
92     urna <- freq[i,]
93     if (urna$tam > 1) { # no tiene caso romper si no se puede
94       cumulos <- c(cumulos, romperse(urna$tam, urna$num))
95     } else {
96       cumulos <- c(cumulos, rep(1, urna$num))
97     }
98   }
99   assert(sum(cumulos) == n)
100   assert(length(cumulos[cumulos == 0]) == 0) # que no haya vacios
101   freq <- as.data.frame(table(cumulos)) # actualizar urnas
102   names(freq) <- c("tam", "num")

```

```

103 freq$tam <- as.numeric(levels(freq$tam))[freq$tam]
104 assert(sum(freq$num * freq$tam) == n)
105 cumulos <- integer()
106 for (i in 1:dim(freq)[1]) { # fase de union
107   urna <- freq[i,]
108   cumulos <- c(cumulos, unirse(urna$tam, urna$num))
109 }
110 assert(sum(abs(cumulos)) == n)
111 assert(length(cumulos[cumulos == 0]) == 0) # que no haya vacios
112 juntarse <- -cumulos[cumulos < 0]
113 cumulos <- cumulos[cumulos > 0]
114 assert(sum(cumulos) + sum(juntarse) == n)
115 nt <- length(juntarse)
116 if (nt > 0) {
117   if (nt > 1) {
118     juntarse <- sample(juntarse)
119     for (i in 1:floor(nt / 2)) {
120       cumulos <- c(cumulos, juntarse[2*i-1] + juntarse[2*i])
121     }
122   }
123   if (nt %% 2 == 1) {
124     cumulos <- c(cumulos, juntarse[nt])
125   }
126 }
127 assert(sum(cumulos) == n)
128 freq <- as.data.frame(table(cumulos))
129 names(freq) <- c("tam", "num")
130 freq$tam <- as.numeric(levels(freq$tam))[freq$tam]
131 assert(sum(freq$num * freq$tam) == n)
132 tl <- paste(paso, "", sep="")
133 while (nchar(tl) < digitos) {
134   tl <- paste("0", tl, sep="")
135 }
136 if (replica == 1){
137   png(paste("p8_ct", tl, "k=", k, "rep=", replica, ".png", sep=""), width=300, height=300)
138   tope <- 50 * ceiling(max(cumulos) / 50)
139   hist(cumulos, breaks=seq(0, tope, 50),
140        main=paste("Paso", paso, "con ambos fen\u{00f3}menos"), freq=FALSE,
141        ylim=c(0, 0.02), xlab="Tama\u{00f1}o", ylab="Frecuencia relativa")
142   graphics.off()
143 }
144 freq
145 filtrados = freq[freq$tam >= c,]
146 filtrados$cont = filtrados$tam * filtrados$num
147 f = sum(filtrados$cont) # particulas removidas
148 porcentaje = 100 * f/n # porcentaje exitosamente filtrado
149 resultado = c(k, replica, paso, porcentaje, c)
150 df = rbind(df, resultado)
151
152 assert(sum(abs(cumulos)) == n)
153 }
154 }
155 }
156
157 library(ggplot2)
158 df$Iteracion = as.factor(df$Iteracion)
159 dfs = split.data.frame(df, f = df$k)
160 ggplot(dfs$'100', aes(x= Iteracion, y= filtrado)) +
161   geom_boxplot(fill = "#F8766D")+
162   labs(x = "Iteracion", y = "% filtrado", title = 'k = 100')
163
164 ggplot(dfs$'200', aes(x= Iteracion, y= filtrado)) +
165   geom_boxplot(fill = "#8800FF")+
166   labs(x = "Iteracion", y = "% filtrado", title = 'k = 200')
167
168 ggplot(dfs$'400', aes(x= Iteracion, y= filtrado)) +
169   geom_boxplot(fill = "#FF8800")+
170   labs(x = "Iteracion", y = "% filtrado", title = 'k = 400')

```

Listing 1: Código para graficar el porcentaje de filtrado en cada iteración.

La figura 1 muestra los diagramas caja-bigote del porcentaje de filtrado en cada iteración para cada valor de k . Se puede observar que el mayor porcentaje de filtrado se obtiene al inicio del experimento, cuando se crean los cúmulos y aún no comienzan las iteraciones, por lo que se podría denominar iteración 0.

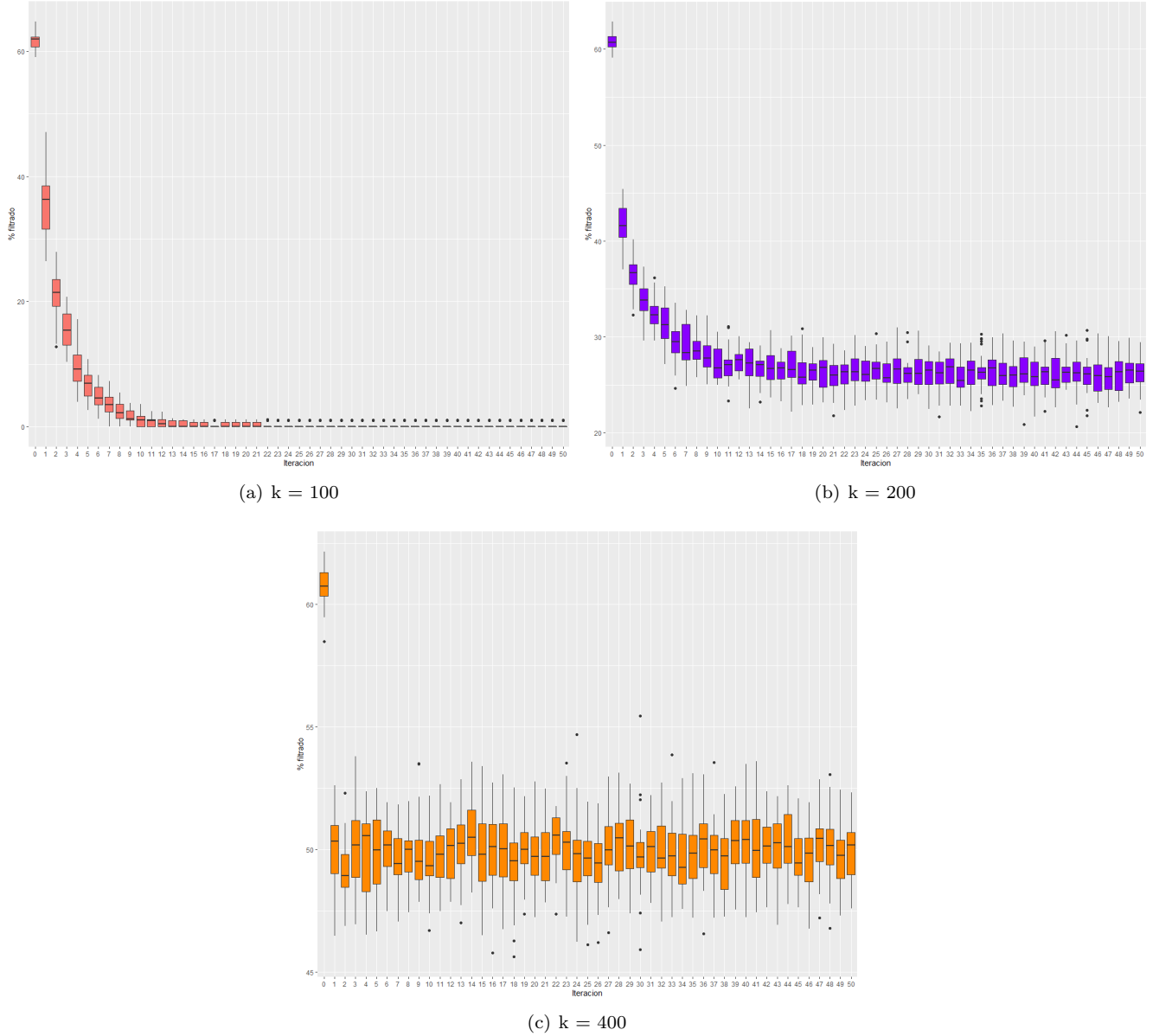


Figura 1: Porcentaje de filtrado en cada iteración.

Los gifs [1] de los histogramas de frecuencia relativa para cada paso simulando los fenómenos de rotura y agregación se encuentran en [mi repositorio](#) en GitHub.

Para analizar si existe una relación entre la iteración en que se realiza el filtrado y el porcentaje de filtrado que se obtiene se realizó una prueba estadística. Debido a los resultados obtenidos al revisar la normalidad de los datos, se eligió realizar la prueba estadística **Kruskal Wallis**.

En el cuadro 1 se resumen los resultados de la revisión de los supuestos para poder aplicar la prueba y los resultados al aplicar la prueba estadística **Kruskal Wallis**. El supuesto outliers se refiere a la cantidad de valores atípicos que existen en los grupos y la normalidad se obtuvo con la prueba de **Shapiro Wilk**. En los resultados del cuadro 1 se observa que para todos los valores de k no existe normalidad en los datos, ya que el valor de p es menor a 0,05, además al aplicar la prueba estadística también se obtienen valores de p menores a 0,05.

Cuadro 1: Resultados de los supuestos y de la prueba estadística Kruskal Wallis.

| k | Outliers | Normalidad | Chi cuadrada | Valor de p |
|-----|----------|------------------------|--------------|-----------------------|
| 100 | 244 | $2,02 \times 10^{-58}$ | 6824.4 | $2,2 \times 10^{-16}$ |
| 200 | 139 | $1,68 \times 10^{-51}$ | 6330.6 | $2,2 \times 10^{-16}$ |
| 400 | 38 | $2,89 \times 10^{-41}$ | 7035.8 | $2,2 \times 10^{-16}$ |

Hipótesis nula : Las medias son iguales en todos los grupos.

Hipótesis alternativa: Debido a que $p < 0,05$ se rechaza la hipótesis nula, es decir que si existen diferencias significativas entre las medias de los grupos.

Se entiende entonces que la iteración en la que se realiza el filtrado si tiene un efecto significativo en el porcentaje de cúmulos que se logran filtrar.

A continuación se muestra el código utilizado para realizar la prueba estadística Kruskal Wallis:

```

1 library(tidyverse)
2 library(ggpubr)
3 library(car)
4 library(rstatix)
5 library(rapportools)
6 library(readr)
7 library(gridExtra)
8
9 #PRUEBA ESTADISTICA...
10 #Estadísticas descriptivas
11 #k = 100
12 dfs$`100` %>%
13   group_by(k) %>%
14   get_summary_stats(filtrado, type = "mean_sd")
15 #k = 200
16 dfs$`200` %>%
17   group_by(k) %>%
18   get_summary_stats(filtrado, type = "mean_sd")
19 #k = 400
20 dfs$`400` %>%
21   group_by(k) %>%
22   get_summary_stats(filtrado, type = "mean_sd")
23 #SUPUESTOS PARA ANOVA
24 #1:Outliers
25 #k = 100
26 dfs$`100` %>%
27   group_by(k) %>%
28   identify_outliers(filtrado)
29 #k = 200
30 dfs$`200` %>%
31   group_by(k) %>%
32   identify_outliers(filtrado)
33 #k = 400
34 dfs$`400` %>%
35   group_by(k) %>%
36   identify_outliers(filtrado)
37 #2:Normalidad por Shapiro
38 #k = 100
39 dfs$`100` %>%
40   group_by(k) %>%
41   shapiro_test(filtrado)
42 #k = 200
43 dfs$`200` %>%
44   group_by(k) %>%
45   shapiro_test(filtrado)
46 #k = 400
47 dfs$`400` %>%
48   group_by(k) %>%
49   shapiro_test(filtrado)
50 #3:Homogeneidad de varianza con prueba Levene
51 #k = 100
52 dfs$`100` %>%
53   levene_test(filtrado$k)

```

```

54 #k = 200
55 dfs$'200' %>%
56   levene_test(filtrado~k)
57 #k = 400
58 dfs$'400' %>%
59   levene_test(filtrado~k)
60 #PRUEBA ESTADISTICA KRUSKAL WALLIS
61 #k = 100
62 dfs$'100' %>%
63   kruskal.test(filtrado ~ k)
64 #k = 200
65 dfs$'200' %>%
66   kruskal.test(filtrado ~ k)
67 #k = 400
68 dfs$'400' %>%
69   kruskal.test(filtrado ~ k)
70 #PRUEBA WILCOXON
71 pairwise.wilcox.test((dfs$'100')$filtrado, (dfs$'100')$Iteracion)
72 pairwise.wilcox.test((dfs$'200')$filtrado, (dfs$'200')$Iteracion)
73 pairwise.wilcox.test((dfs$'400')$filtrado, (dfs$'200')$Iteracion)

```

Listing 2: Código para la prueba estadística Kruskal Wallis.

3. Reto 1

Como el primer reto, determina si existe algún intervalo de iteraciones en el que el filtrado alcance un óptimo. Realiza réplicas para determinar si el momento en el cual se alcanza el máximo tiene un comportamiento sistemático. Incluye visualizaciones para justificar las conclusiones.

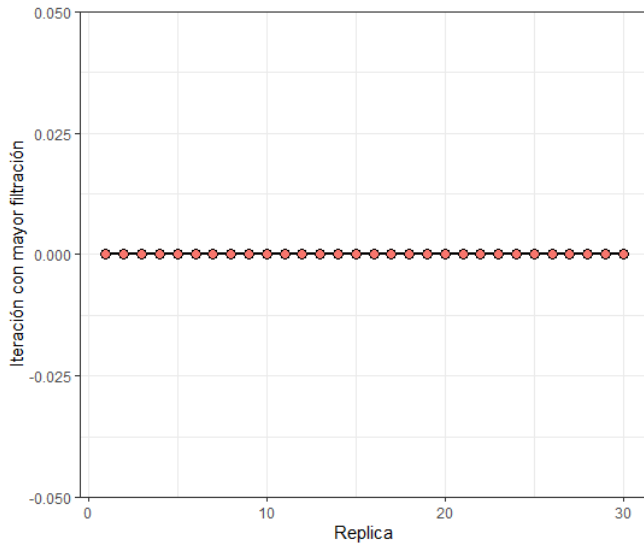
En la figura 1 de la tarea base podemos notar que la mejor iteración para obtener el máximo porcentaje filtrado de cúmulos es la iteración 0. En el cuadro 2 se muestra la iteración ideal para cada valor de k , el promedio de filtración que se obtiene y el promedio de tamaño crítico c en cada valor de k .

Cuadro 2: Iteración ideal para filtrar.

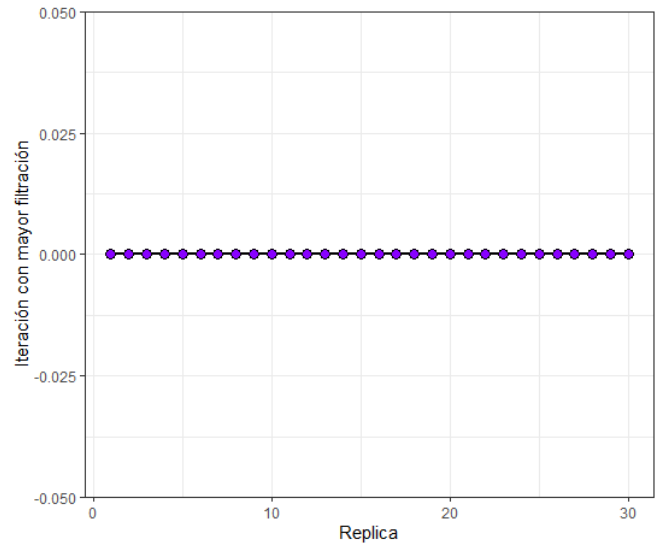
| k | Iteración | Filtración promedio (%) | c promedio |
|-----|-----------|-------------------------|--------------|
| 100 | 0 | 61,6 | 966,5 |
| 200 | 0 | 60,9 | 499 |
| 400 | 0 | 60,2 | 249 |

En la figura 2 se muestra la iteración ideal para filtrar y obtener el mayor porcentaje de cúmulos filtrados en cada una de las 30 réplicas para cada valor de k . Podemos apreciar que tanto para $k = 100$, $k = 200$ y $k = 400$ el mejor momento para filtrar es antes de comenzar las iteraciones debido a que existe un mayor porcentaje de cúmulos con un tamaño c mayor o igual a la mediana de los cúmulos iniciales.

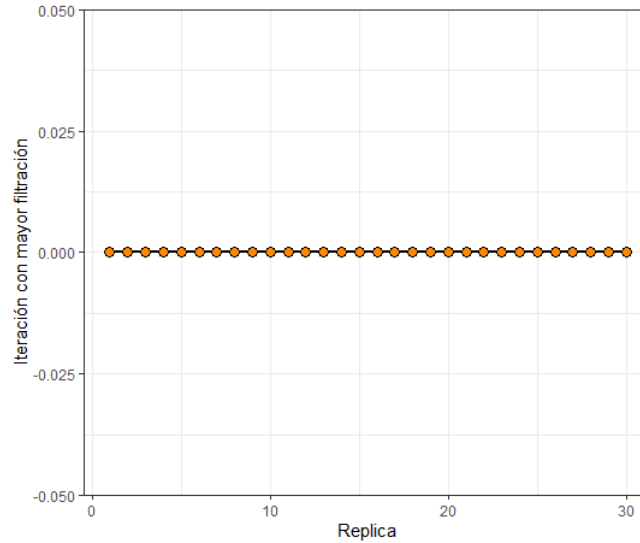
En [mi repositorio](#) de GitHub se encuentra el código realizado para este reto al cual solo se le agreron las instrucciones para realizar las gráficas de la figura 2 y para obtener los valores promedios de *filtración* y c mostrados en el cuadro 2.



(a) $k = 100$



(b) $k = 200$



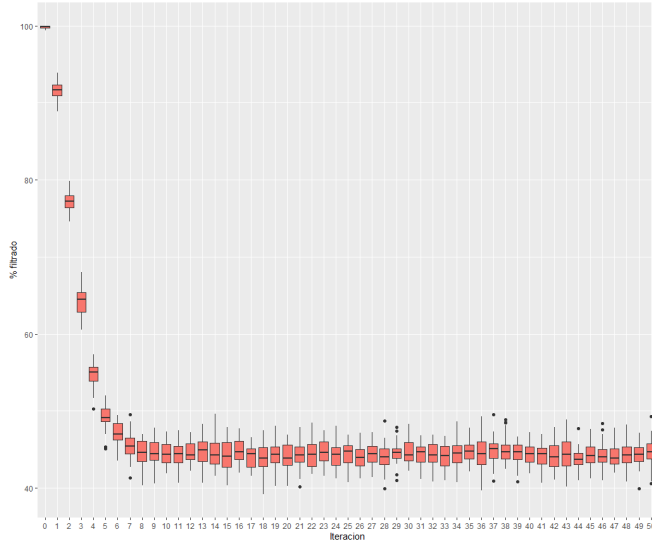
(c) $k = 400$

Figura 2: Iteración con mayor porcentaje de filtración en cada réplica para cada valor k .

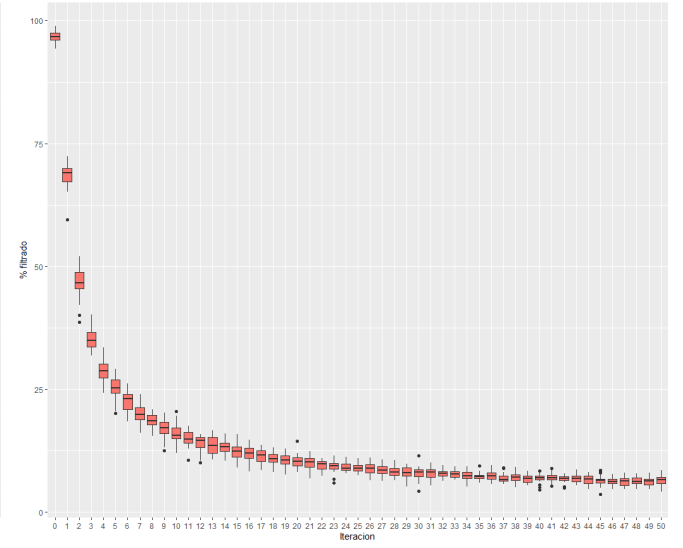
4. Reto 2

Como un segundo reto, determina cómo los resultados de la tarea y del primer reto dependen del valor de c . ¿Qué cambia y cómo si c ya no se asigna como la mediana inicial sino como un valor menor o mayor?.

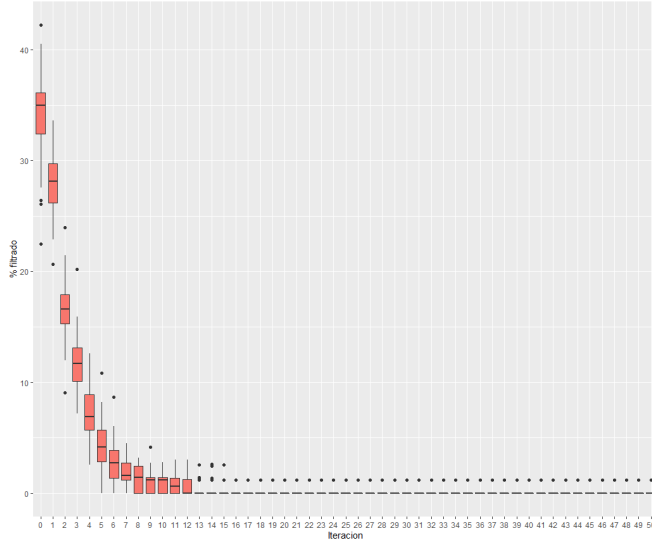
Para este reto se utilizaron los siguientes valores de c : 300, 600, 1200 y 1800, el valor de c representa el tamaño crítico de los cúmulos por lo que también se podría denominar *filtro* ya que en el código se establece que se deben filtrar los tamaños de cúmulos mayores o iguales a c . Al código objetivo de la tarea base se le agregó un ciclo `for` para variar el tamaño crítico de los cúmulos c en los valores previamente mencionados y se realizaron diagramas caja-bigote para cada valor de c con su respectivo valor k . En las figuras 3, 4 y 5 se muestran los resultados obtenidos y en los cuadros 3, 4 y 5 se indica la iteración ideal para cada tamaño crítico así como el promedio de filtración que se consigue en la respectiva iteración.



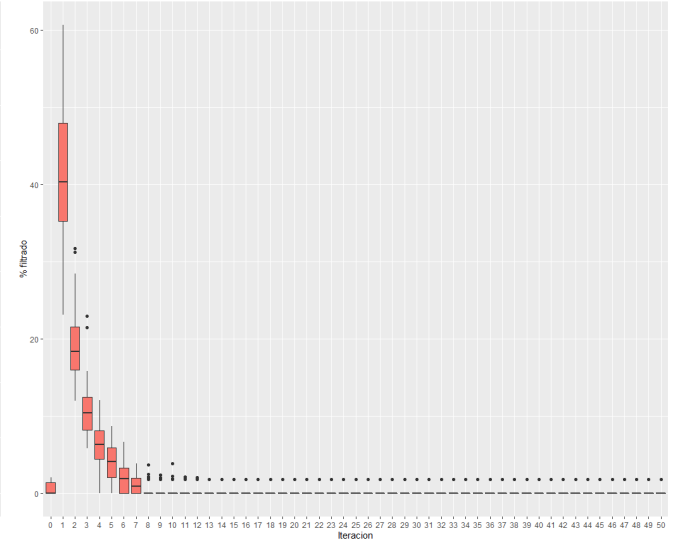
(a) $k = 100$ y $c = 300$



(b) $k = 100$ y $c = 600$



(c) $k = 100$ y $c = 1200$

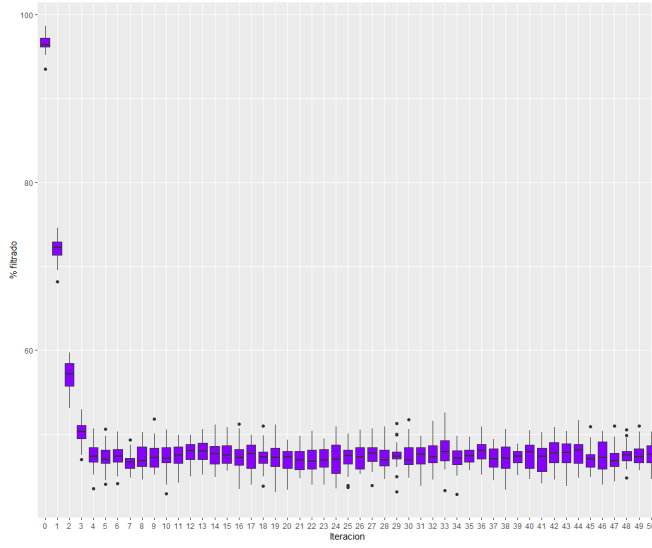


(d) $k = 100$ y $c = 1800$

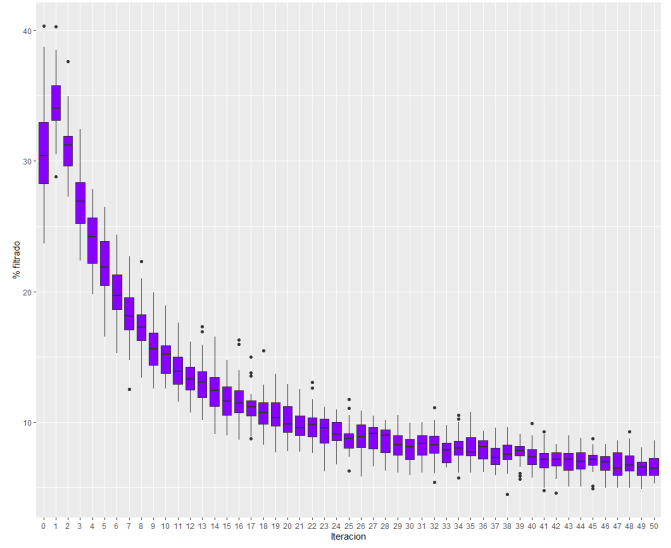
Figura 3: Porcentaje filtrado en cada iteración para cada tamaño crítico con $k = 100$.

Cuadro 3: Iteración ideal para filtrar.

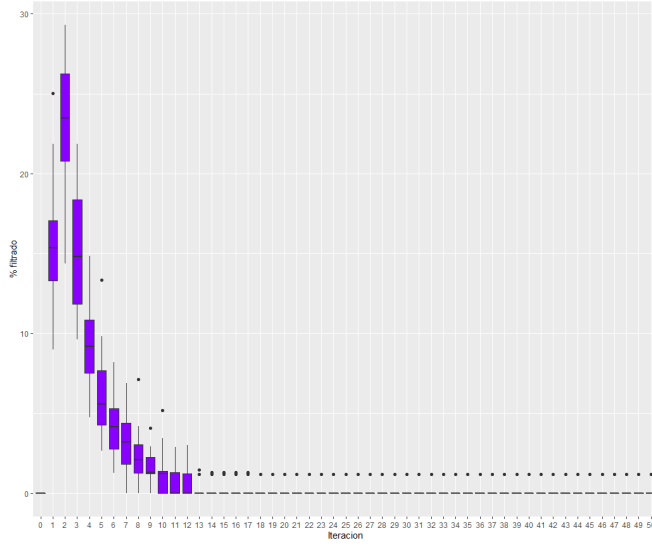
| k | c | Iteración | Filtración promedio (%) |
|-----|------|-----------|-------------------------|
| 100 | 300 | 0 | 99,9 |
| 100 | 600 | 0 | 96,6 |
| 100 | 1200 | 0 | 33,9 |
| 100 | 1800 | 1 | 41,9 |



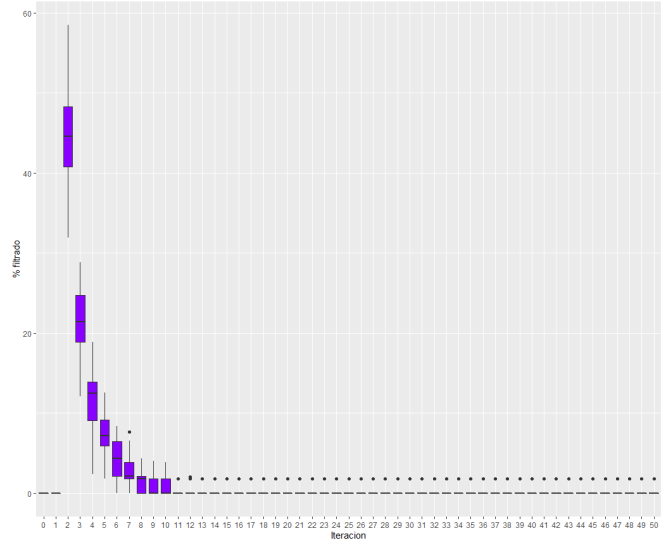
(a) $k = 200$ y $c = 300$



(b) $k = 200$ y $c = 600$



(c) $k = 200$ y $c = 1200$

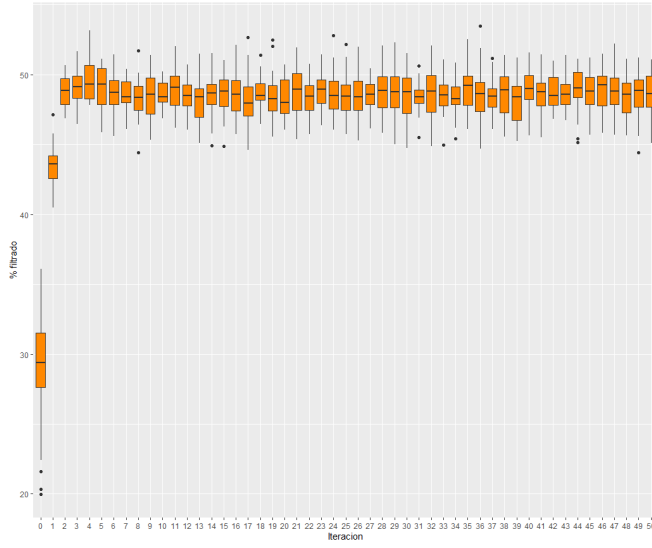


(d) $k = 200$ y $c = 1800$

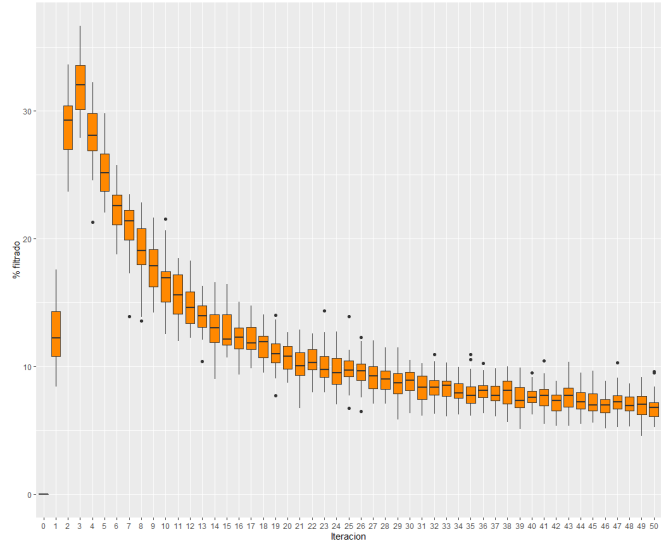
Figura 4: Porcentaje filtrado en cada iteración para cada tamaño crítico con $k = 200$.

Cuadro 4: Iteración ideal para filtrar.

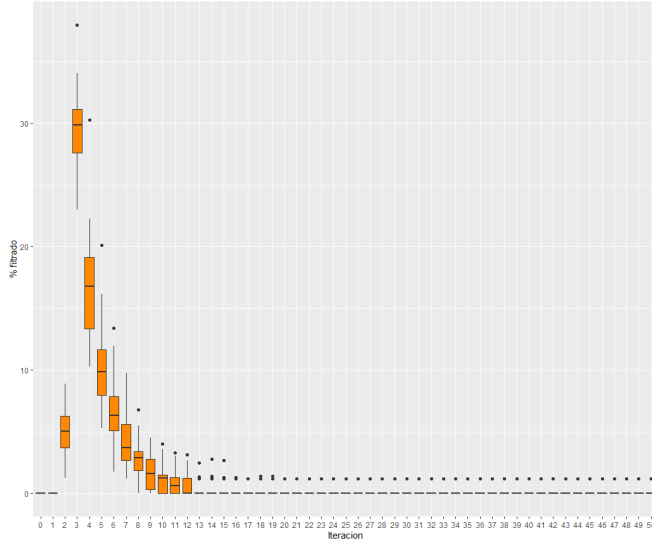
| k | c | Iteración | Filtración promedio (%) |
|-----|------|-----------|-------------------------|
| 200 | 300 | 0 | 96,5 |
| 200 | 600 | 1 | 34,4 |
| 200 | 1200 | 2 | 23,2 |
| 200 | 1800 | 2 | 44,4 |



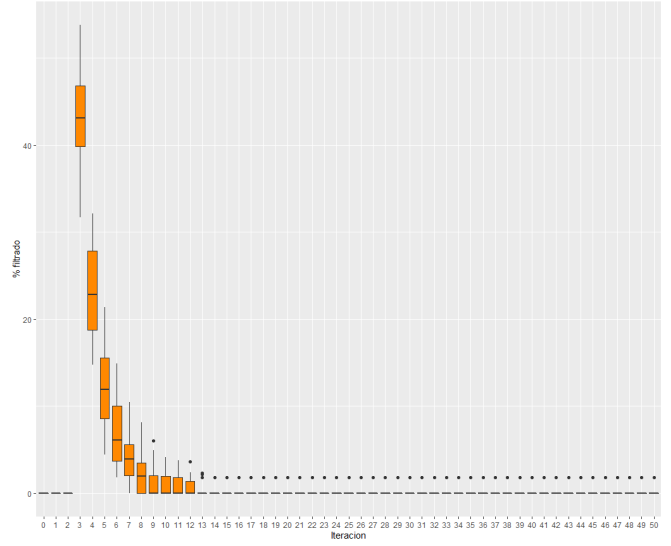
(a) $k = 400$ y $c = 300$



(b) $k = 400$ y $c = 600$



(c) $k = 400$ y $c = 1200$



(d) $k = 400$ y $c = 1800$

Figura 5: Porcentaje filtrado en cada iteración para cada tamaño crítico con $k = 400$.

Cuadro 5: Iteración ideal para filtrar.

| k | c | Iteración | Filtración promedio (%) |
|-----|------|-----------|-------------------------|
| 400 | 300 | 4 | 49,6 |
| 400 | 600 | 3 | 31,9 |
| 400 | 1200 | 3 | 29,6 |
| 400 | 1800 | 2 | 43,1 |

A continuación se muestra el código generado para el reto 2:

```
1 library(testit) # para pruebas, recuerda instalar antes de usar
2 tamas <- c(100, 200, 400)
3 n <- 100000
4 df = data.frame()
5 se = c(300, 600, 1200, 1800)
6
7 for (k in tamas){
```

```

8   for (c in se){
9     for (replica in 1:30){
10      originales <- rnorm(k)
11      cumulos <- originales - min(originales) + 1
12      cumulos <- round(n * cumulos / sum(cumulos))
13      assert(min(cumulos) > 0)
14      diferencia <- n - sum(cumulos)
15      if (diferencia > 0) {
16        for (i in 1:diferencia) {
17          p <- sample(1:k, 1)
18          cumulos[p] <- cumulos[p] + 1
19        }
20      } else if (diferencia < 0) {
21        for (i in 1:-diferencia) {
22          p <- sample(1:k, 1)
23          if (cumulos[p] > 1) {
24            cumulos[p] <- cumulos[p] - 1
25          }
26        }
27      }
28
29      assert(length(cumulos[cumulos == 0]) == 0) # que no haya vacios
30      assert(sum(cumulos) == n)
31
32      d <- sd(cumulos) / 4 # factor arbitrario para suavizar la curva
33
34      primero <- as.data.frame(table(cumulos))
35      names(primeros) <- c("tam", "num")
36      primero$tam <- as.numeric(levels(primeros$tam))[primeros$tam]
37      assert(sum(primeros$num * primeros$tam) == n)
38
39      filtrados1 = primero[primeros$tam >= c,]
40      filtrados1$cont = filtrados1$tam * filtrados1$num
41      f1 = sum(filtrados1$cont) # particulas removidas
42      porcentaje1 = 100 * f1/n # porcentaje exitosamente filtrado
43      paso1 = 0
44      resultado1 = c(k, replica, paso1, porcentaje1, c)
45      df = rbind(df, resultado1)
46      names(df) = c("k", "Replica", "Iteracion", "filtrado", "c")
47      assert(sum(abs(cumulos)) == n)
48
49      rotura <- function(x) {
50        return (1 / (1 + exp((c - x) / d)))
51      }
52      union <- function(x) {
53        return (exp(-x / c))
54      }
55      romperse <- function(tam, cuantos) {
56        romper <- round(rotura(tam) * cuantos) # independientes
57        resultado <- rep(tam, cuantos - romper) # los demas
58        if (romper > 0) {
59          for (cumulo in 1:romper) { # agregar las rotas
60            t <- 1
61            if (tam > 2) { # sample no jala con un solo valor
62              t <- sample(1:(tam-1), 1)
63            }
64            resultado <- c(resultado, t, tam - t)
65          }
66        }
67        assert(sum(resultado) == tam * cuantos) # no hubo perdidas
68        return(resultado)
69      }
70      unir <- function(tam, cuantos) {
71        unir <- round(union(tam) * cuantos) # independientes
72        if (unir > 0) {
73          division <- c(rep(-tam, unir), rep(tam, cuantos - unir))
74          assert(sum(abs(division)) == tam * cuantos)
75          return(division)
76        } else {
77          return(rep(tam, cuantos))
78        }

```

```

79 }
80 freq <- as.data.frame(table(cumulos))
81 names(freq) <- c("tam", "num")
82 freq$tam <- as.numeric(levels(freq$tam))[freq$tam]
83 duracion <- 50
84 digitos <- floor(log(duracion, 10)) + 1
85 for (paso in 1:duracion) {
86   assert(sum(cumulos) == n)
87   cumulos <- integer()
88   for (i in 1:dim(freq)[1]) { # fase de rotura
89     urna <- freq[i,]
90     if (urna$tam > 1) { # no tiene caso romper si no se puede
91       cumulos <- c(cumulos, romperse(urna$tam, urna$num))
92     } else {
93       cumulos <- c(cumulos, rep(1, urna$num))
94     }
95   }
96   assert(sum(cumulos) == n)
97   assert(length(cumulos[cumulos == 0]) == 0) # que no haya vacios
98   freq <- as.data.frame(table(cumulos)) # actualizar urnas
99   names(freq) <- c("tam", "num")
100  freq$tam <- as.numeric(levels(freq$tam))[freq$tam]
101  assert(sum(freq$num * freq$tam) == n)
102  cumulos <- integer()
103  for (i in 1:dim(freq)[1]) { # fase de union
104    urna <- freq[i,]
105    cumulos <- c(cumulos, unir(urna$tam, urna$num))
106  }
107  assert(sum(abs(cumulos)) == n)
108  assert(length(cumulos[cumulos == 0]) == 0) # que no haya vacios
109  juntarse <- -cumulos[cumulos < 0]
110  cumulos <- cumulos[cumulos > 0]
111  assert(sum(cumulos) + sum(juntarse) == n)
112  nt <- length(juntarse)
113  if (nt > 0) {
114    if (nt > 1) {
115      juntarse <- sample(juntarse)
116      for (i in 1:floor(nt / 2)) {
117        cumulos <- c(cumulos, juntarse[2*i-1] + juntarse[2*i])
118      }
119    }
120    if (nt %% 2 == 1) {
121      cumulos <- c(cumulos, juntarse[nt])
122    }
123  }
124  assert(sum(cumulos) == n)
125  freq <- as.data.frame(table(cumulos))
126  names(freq) <- c("tam", "num")
127  freq$tam <- as.numeric(levels(freq$tam))[freq$tam]
128  assert(sum(freq$num * freq$tam) == n)
129  t1 <- paste(paso, "", sep="")
130  while (nchar(t1) < digitos) {
131    t1 <- paste("0", t1, sep="")
132  }
133
134  freq
135  filtrados = freq[freq$tam >= c,]
136  filtrados$cont = filtrados$tam * filtrados$num
137  f = sum(filtrados$cont) # particulas removidas
138  porcentaje = 100 * f/n # porcentaje exitosamente filtrado
139  resultado = c(k, replica, paso, porcentaje, c)
140  df = rbind(df, resultado)
141
142  assert(sum(abs(cumulos)) == n)
143 }
144 }
145 }
146 }
147
148
149 a1 = df[df$k == 100 & df$c == 300, ]

```

```

150 a2 = df[df$k == 100 & df$c == 600, ]
151 a3 = df[df$k == 100 & df$c == 1200, ]
152 a4 = df[df$k == 100 & df$c == 1800, ]
153
154 b1 = df[df$k == 200 & df$c == 300, ]
155 b2 = df[df$k == 200 & df$c == 600, ]
156 b3 = df[df$k == 200 & df$c == 1200, ]
157 b4 = df[df$k == 200 & df$c == 1800, ]
158
159 c1 = df[df$k == 400 & df$c == 300, ]
160 c2 = df[df$k == 400 & df$c == 600, ]
161 c3 = df[df$k == 400 & df$c == 1200, ]
162 c4 = df[df$k == 400 & df$c == 1800, ]
163
164 library(ggplot2)
165 a1$Iteracion = as.factor(a1$Iteracion)
166 ggplot(a1, aes(x= Iteracion, y= filtrado)) +
167   geom_boxplot(fill = "#F8766D")+
168   labs(x = "Iteracion", y = "% filtrado", title = 'k = 100 y c = 300')
169
170 a2$Iteracion = as.factor(a2$Iteracion)
171 ggplot(a2, aes(x= Iteracion, y= filtrado)) +
172   geom_boxplot(fill = "#F8766D")+
173   labs(x = "Iteracion", y = "% filtrado", title = 'k = 100 y c = 600')
174
175 a3$Iteracion = as.factor(a3$Iteracion)
176 ggplot(a3, aes(x= Iteracion, y= filtrado)) +
177   geom_boxplot(fill = "#F8766D")+
178   labs(x = "Iteracion", y = "% filtrado", title = 'k = 100 y c = 1200')
179
180 a4$Iteracion = as.factor(a4$Iteracion)
181 ggplot(a4, aes(x= Iteracion, y= filtrado)) +
182   geom_boxplot(fill = "#F8766D")+
183   labs(x = "Iteracion", y = "% filtrado", title = 'k = 100 y c = 1800')
184
185 b1$Iteracion = as.factor(b1$Iteracion)
186 ggplot(b1, aes(x= Iteracion, y= filtrado)) +
187   geom_boxplot(fill = "#8800FF")+
188   labs(x = "Iteracion", y = "% filtrado", title = 'k = 200 y c = 300')
189
190 b2$Iteracion = as.factor(b2$Iteracion)
191 ggplot(b2, aes(x= Iteracion, y= filtrado)) +
192   geom_boxplot(fill = "#8800FF")+
193   labs(x = "Iteracion", y = "% filtrado", title = 'k = 200 y c = 600')
194
195 b3$Iteracion = as.factor(b3$Iteracion)
196 ggplot(b3, aes(x= Iteracion, y= filtrado)) +
197   geom_boxplot(fill = "#8800FF")+
198   labs(x = "Iteracion", y = "% filtrado", title = 'k = 200 y c = 1200')
199
200 b4$Iteracion = as.factor(b4$Iteracion)
201 ggplot(b4, aes(x= Iteracion, y= filtrado)) +
202   geom_boxplot(fill = "#8800FF")+
203   labs(x = "Iteracion", y = "% filtrado", title = 'k = 200 y c = 1800')
204
205 c1$Iteracion = as.factor(c1$Iteracion)
206 ggplot(c1, aes(x= Iteracion, y= filtrado)) +
207   geom_boxplot(fill = "#FF8800")+
208   labs(x = "Iteracion", y = "% filtrado", title = 'k = 400 y c = 300')
209
210 c2$Iteracion = as.factor(c2$Iteracion)
211 ggplot(c2, aes(x= Iteracion, y= filtrado)) +
212   geom_boxplot(fill = "#FF8800")+
213   labs(x = "Iteracion", y = "% filtrado", title = 'k = 400 y c = 600')
214
215 c3$Iteracion = as.factor(c3$Iteracion)
216 ggplot(c3, aes(x= Iteracion, y= filtrado)) +
217   geom_boxplot(fill = "#FF8800")+
218   labs(x = "Iteracion", y = "% filtrado", title = 'k = 400 y c = 1200')
219
220 c4$Iteracion = as.factor(c4$Iteracion)

```

```

221 ggplot(c4, aes(x= Iteracion, y= filtrado)) +
222   geom_boxplot(fill = "#FF8800")+
223   labs(x = "Iteracion", y = "% filtrado")
224
225 a1[a1$Iteracion == 0,] %>%
226   get_summary_stats(filtrado, type = "mean_sd")
227
228 a2[a2$Iteracion == 0,] %>%
229   get_summary_stats(filtrado, type = "mean_sd")
230
231 a3[a3$Iteracion == 0,] %>%
232   get_summary_stats(filtrado, type = "mean_sd")
233
234 a4[a4$Iteracion == 1,] %>%
235   get_summary_stats(filtrado, type = "mean_sd")
236
237 b1[b1$Iteracion == 0,] %>%
238   get_summary_stats(filtrado, type = "mean_sd")
239
240 b2[b2$Iteracion == 1,] %>%
241   get_summary_stats(filtrado, type = "mean_sd")
242
243 b3[b3$Iteracion == 2,] %>%
244   get_summary_stats(filtrado, type = "mean_sd")
245
246 b4[b4$Iteracion == 2,] %>%
247   get_summary_stats(filtrado, type = "mean_sd")
248
249 c1[c1$Iteracion == 0,] %>%
250   get_summary_stats(filtrado, type = "mean_sd")
251
252 g1 = c1 %>%
253   group_by(Iteracion) %>%
254   get_summary_stats(filtrado, type = "mean_sd")
255
256 g1 %>%
257   get_summary_stats(mean, type = "max")
258
259 c2[c2$Iteracion == 3,] %>%
260   get_summary_stats(filtrado, type = "mean_sd")
261
262 c3[c3$Iteracion == 3,] %>%
263   get_summary_stats(filtrado, type = "mean_sd")
264
265 c4[c4$Iteracion == 3,] %>%
266   get_summary_stats(filtrado, type = "mean_sd")

```

Listing 3: Código para graficar el porcentaje de filtrado en cada iteración.

5. Conclusión

Con base en los diagramas caja-bigote y el resultado obtenido de la prueba estadística **Kruskal Wallis** para la tarea base, puedo concluir que la iteración en que se realiza el proceso de filtrado si afecta el porcentaje de cúmulos filtrados, y si el filtro que se utiliza es la mediana de los cúmulos el momento ideal para filtrar será antes de comenzar las iteraciones ya que existen más cúmulos mayores o iguales al tamaño del *filtro*. De igual manera si el tamaño del *filtro* es menor que la mediana de los cúmulos también se tendrá como momento ideal para filtrar la iteración 0, sin embargo si el tamaño del *filtro* es mayor que la mediana de los cúmulos la iteración ideal para filtrar no será precisamente la 0 pero si durante las primeras iteraciones.

En general ésta práctica fue mucho de mi agrado ya que no tuve grandes dificultades para realizarla y aprendí a manipular mejor los `data.frame` en RStudio.

Referencias

- [1] PHOTOGRAMIO. Crear gif animados. URL <https://photogramio.com/es/gif-maker#ezgif>.
- [2] Elisa Schaeffer. Urn model, 2021. URL <https://github.com/satuelisa/Simulation/blob/master/UrnModel/aggrFrag.R>.