Universidad Rey Juan Carlos

Grado en Matemáticas

Práctica 7

Geometría Computacional

Autores: Guillermo Grande Santi y

Alejandro López Adrados

Abril, 2022

Índice

1 Objetivos

El objetivo de esta práctica es la utilización de los algoritmos vistos en clase de la envolvente convexa en R. Para ello hemos seleccionado dos ejercicios:

- Ejercicio 1: Este ejercicio consiste en implementar en R el Algoritmo Scan de Graham de envolventes convexas. Para ello, basta con transformar el algoritmo visto al lenguaje de R.
- Ejercicio 2: Este ejercicio consiste en implementar en R un algoritmo basado en la ecuación de la recta cuya entrada sean tres puntos distintos del plano (A, B y C) que determine si C está a la derecha o a la izquierda de la recta que une A con B orientada por el vector B-A. Además, se pide si dichos puntos hacen un giro a la derecha o no. Este último apartado será uno de los pasos claves en el algoritmo solicitado anteriormente.

2 Metodología

2.1 Ejercicio 1

Para este ejercicio, por cada ejecución se crean un número de elementos aleatorios (puede cambiarse el número con la variable elementos) y se conseguirá dibujar la envolvente convexa de dicho conjunto. El primer paso de este algoritmo será encontrar el punto con la menor posición en el eje de coordenadas de todo el conjunto (menor posición Y).

Posteriormente, se llevará a cabo una ordenación de todos los puntos por los ángulos que forman con el punto inicial. Para ello se calcularán los ángulos que se forman entre cada punto con la horizontal que pasa por dicho punto y la recta entre este y el punto inicial. Entonces se ordenarán todos los ángulos usando el algoritmo quicksort que fue implementado en la práctica Algoritmos2, y con ellos, se ordenarán los puntos asociados a dichos ángulos.

A continuación, se realizará una selección de los puntos que pertenecerán a la envolvente convexa con un algoritmo que hemos creado basándonos en el giro

de cada tres puntos. Si este giro es negativo, el punto se incluye en la lista de puntos de la envolvente convexa, y si es mayor o igual que cero no sólo no se incluirá en la lista sino que también se eliminará de ella si fue incluido anteriormente.
Finalmente, todos estos puntos que han sido incluidos en el array de puntos pertenecientes a la envolvente convexa se pintarán por pantalla en un gráfico en el que podremos observar el correcto funcionamiento del algoritmo.

2.2 Ejercicio 2

Para este ejercicio comenzamos definiendo tres puntos al principio del programa, que podrán cambiar sus valores para ver los diferentes resultados. En el apartado A, primero creamos la recta que une A y B orientada por el vector B-A y posteriormente utilizamos un algoritmo para saber si el punto C está a la izquierda o a la derecha de la recta. Este algoritmo se basa en la pendiente de dicha recta y se evalúa según estos casos:

- Si la pendiente es positiva y el punto se encuentra por encima del gráfico, el punto está a la izquierda.
- Si la pendiente es positiva y el punto está por debajo, se encuentra a la derecha.
- Si la pendiente es negativa y el punto se encuentra por arriba entonces está a la derecha.
- Si la pendiente de la recta es negativa pero el punto está por debajo, está a la izquierda.

En cuanto al apartado B, se utiliza una parte del ejercicio 1 de está práctica, pues se pide conocer el giro entre estos puntos, que es justamente una de las piezas fundamentales de el algoritmo Scan de Sraham que diseñamos. Básicamente, se usará la misma fórmula usando los tres puntos, y si el giro es menor que cero, entonces el giro es a la derecha. En caso contrario, el giro será a la izquierda.

3 Conclusiones

En la parte teórica de esta asignatura hemos visto el temario relacionado con la envolvente convexa, pero verlo en esta práctica e implementarlo por nuestra parte ha sido una manera mucho mejor de entender algún algoritmo que se usa para sacar dicha envolvente convexa y mejorar nuestro conocimiento sobre el tema. Cabe destacar también la utilización de prácticas anteriores como la aplicación del algoritmo quicksort, que deja visible su gran utilidad en una gran cantidad de problemas.

4 Anexos

4.1 Código del ejercicio 1

```
#Implementar en R el Algoritmo Scan de Graham.de envolventes convexas.



quicksort <- function(x)
{
  #Caso base
  if (length(x) <= 1) {
    return(x)
  }
  #Inicializamos un array que guardará todos los valores menos la mediana
  resto <- c()
  #Elegimos la mediana del vector
  mediana <- length(x) %/% 2
  #Se elige el pivote
  pivote <- x[mediana]
  #Bucle que inserta en resto hasta la posición mediana-1
  for (i in 1:mediana - 1) {
```

```
    resto[i] <- x[i]
  }
  longitud <- length(x)
  siguiente_mediana <- mediana + 1
  #Bucle que inserta en resto desde la posición mediana+1
  for (i in siguiente_mediana:longitud) {
    resto[i - 1] <- x[i]
  }
  #Recursividad sobre el vector resto:
  #Parte izquierda del árbol (menores)
  L1 <- quicksort(resto[resto < pivote])
  #Parte derecha del árbol (mayores)
  L3 <- quicksort(resto[resto >= pivote])
  #Se devuelve la concatenación de parte izquierda-pivote-parte derecha
  return(c(L1, pivote, L3))
}


scanGraham <- function(puntos) {
  colnames(puntos) <- c("x", "y")
  plot(
    puntos[, 1],
    puntos[, 2],
    xlim = c(-limite, limite),
    ylim = c(-limite, limite),
    type = "p",
    col = "green"
  )

  i <- 1
```

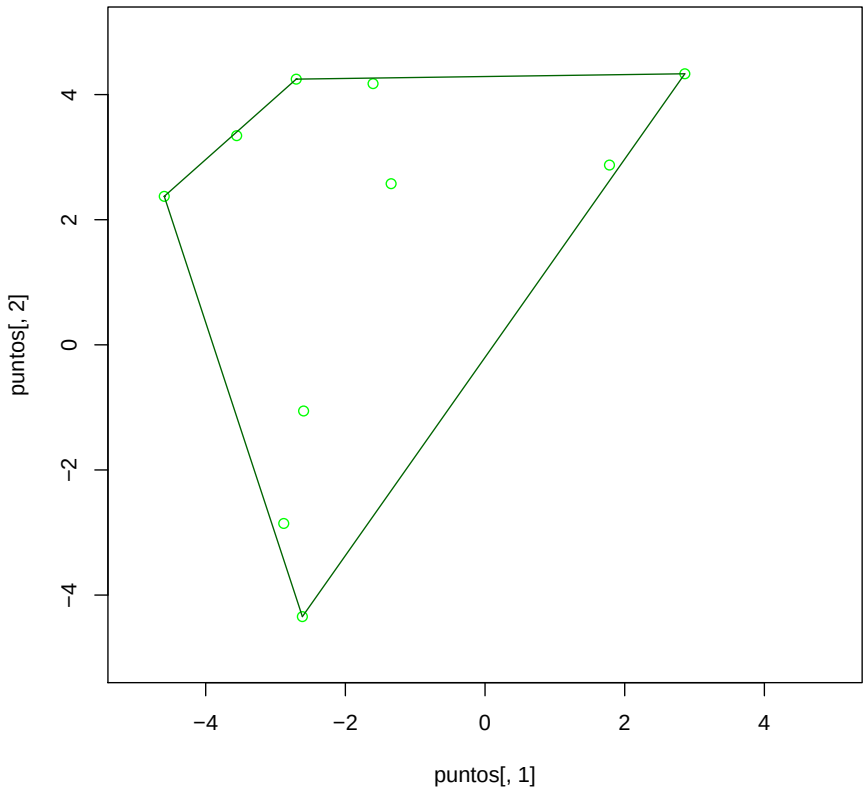
```
aux <- c(0, 0)
dimension <- dim(puntos)
for (i in 1:dimension[1]) {
  if (puntos[i, 2] < puntos[1, 2]) {
    aux <- puntos[i, ]
    puntos[i, ] <- puntos[1, ]
    puntos[1, ] <- aux
  }
}
puntos[1, ]

#Recta abscisas
horizontal <- c(1, 0)
j <- 2
matriz <- matrix(nrow = dimension[1], ncol = 1)
matriz[1] = 0
for (j in 2:dimension[1]) {
  recta <- puntos[1, ] - puntos[j, ]
  matriz[j] <-
    acos((recta %*% horizontal) / sqrt((recta[1] ^ 2) + (recta[2] ^ 2)) *
           sqrt((horizontal[1] ^ 2) + (horizontal[2] ^ 2)))
}
puntos <- cbind(puntos, matriz)
angOrdenados <- quicksort(puntos[, 3])

h <- 1
k <- 1
dimension2 <- dim(t(angOrdenados))
for (h in 1:dimension2[2]) {
  while (angOrdenados[h] != puntos[k, 3]) {
```

```
    k = k + 1
  }
  aux <- puntos[h, ]
  puntos[h, 1] <- puntos[k, 1]
  puntos[h, 2] <- puntos[k, 2]
  puntos[h, 3] <- puntos[k, 3]
  puntos[k, ] <- aux
  k <- 2
}

puntosOrdenados <- cbind(puntos[, 1], puntos[, 2])

l <- 2
cont <- 2
puntosEnvolventeConvexa <- matrix(nrow = dimension[1], ncol = 2)
puntosEnvolventeConvexa[1, ] <- puntosOrdenados[1, ]
dimensionActual <- dim(puntosOrdenados)
while (l <= dimensionActual[1]) {
  dimensionActual <- dim(puntosOrdenados)
  puntosAntes <- puntosOrdenados[l - 1, ]
  puntosActual <- puntosOrdenados[l, ]
  if ((l + 1) > dimensionActual[1]) {
    puntosDespues <- puntosOrdenados[1, ]
  } else {
    puntosDespues <- puntosOrdenados[(l + 1), ]
  }
  giro <-
    (puntosActual[1] - puntosAntes[1]) * (puntosDespues[2] - puntosAntes[2]) -
    (puntosActual[2] - puntosAntes[2]) * (puntosDespues[1] - puntosAntes[1])
  if (giro < 0) {
```

```
    puntosEnvolventeConvexa[cont, ] <- puntosActual
    cont = cont + 1
  } else {
    puntosOrdenados <- puntosOrdenados[-l, ]
    l = l - 2

    z <- 1
    while (z < cont - 1) {
      if ((puntosActual[1] == puntosEnvolventeConvexa[z, 1]) &
          (puntosActual[2] == puntosEnvolventeConvexa[z, 2])) {
        puntosEnvolventeConvexa <- puntosEnvolventeConvexa[-z, ]
        cont = cont - 1
      } else {
        z = z + 1
      }
    }

    if ((puntosActual[1] == puntosEnvolventeConvexa[z, 1]) &
        (puntosActual[2] == puntosEnvolventeConvexa[z, 2])) {
      puntosEnvolventeConvexa <- puntosEnvolventeConvexa[-z, ]
      cont = cont - 1
    }

  }
  l = l + 1

}

puntosEnvolventeConvexa <-
  puntosEnvolventeConvexa[!rowSums(!is.finite(puntosEnvolventeConvexa)), ]
```

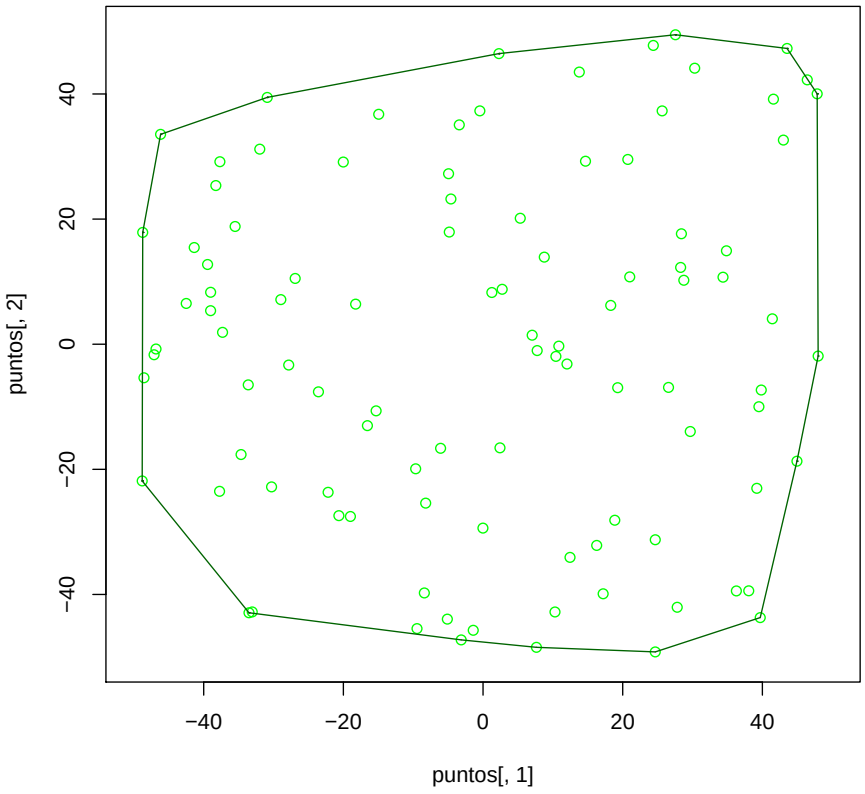
```
  n <- 1
  dimension3 <- dim(puntosEnvolventeConvexa)
  for (n in 1:dimension3[1]) {
    if (n + 1 > dimension3[1]) {
      segments(
        x0 = puntosEnvolventeConvexa[n, 1],
        y0 = puntosEnvolventeConvexa[n, 2],
        x1 = puntosEnvolventeConvexa[1, 1],
        y1 = puntosEnvolventeConvexa[1, 2],
        col = "darkgreen"
      )
    } else {
      segments(
        x0 = puntosEnvolventeConvexa[n, 1],
        y0 = puntosEnvolventeConvexa[n, 2],
        x1 = puntosEnvolventeConvexa[n + 1, 1],
        y1 = puntosEnvolventeConvexa[n + 1, 2],
        col = "darkgreen"
      )
    }
  }
}

#Ejemplo con 10 elementos de -5 a 5
elementos <- 10
limite <- 5
vec1 <- runif(elementos, -limite, limite)
vec2 <- runif(elementos, -limite, limite)
vec <- matrix(nrow = elementos, ncol = 2)
vec <- cbind(vec1, vec2)
```

```
puntos <- vec

scanGraham(puntos)
```



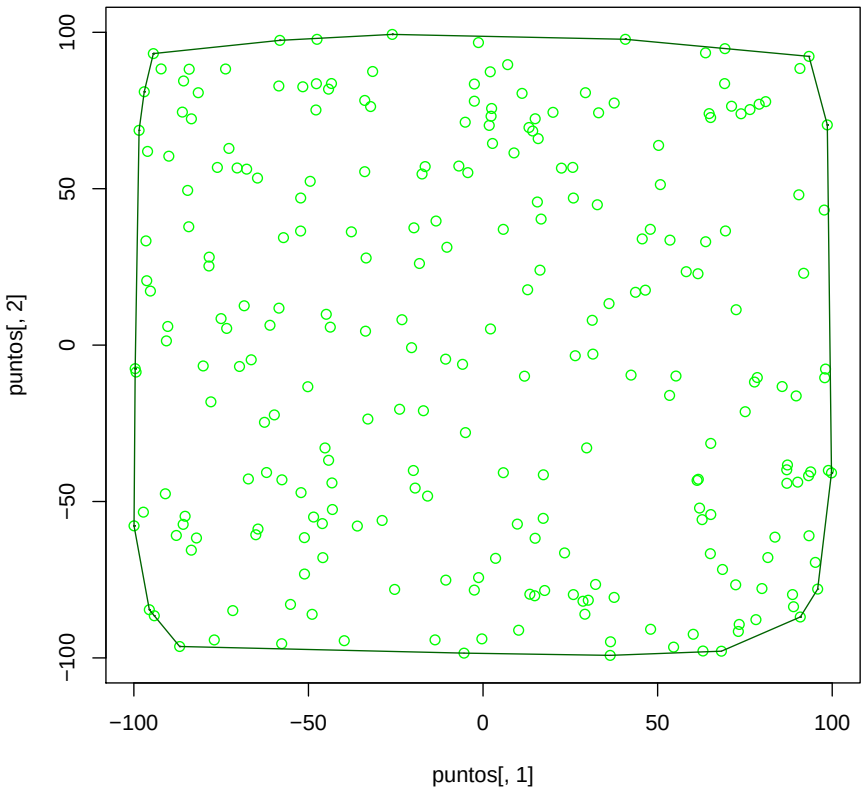

```
#Ejemplo con 100 elementos de -50 a 50
elementos <- 100
limite <- 50
vec1 <- runif(elementos, -limite, limite)
vec2 <- runif(elementos, -limite, limite)
vec <- matrix(nrow = elementos, ncol = 2)
```

```
vec <- cbind(vec1, vec2)

puntos <- vec

scanGraham(puntos)
```




```
#Ejemplo con 250 elementos de -100 a 100
elementos<-250
limite <- 100
vec1 <- runif(elementos, -limite, limite)
vec2 <- runif(elementos, -limite, limite)
```

```
vec <- matrix(nrow = elementos, ncol = 2)
vec <- cbind(vec1, vec2)
puntos <- vec


scanGraham(puntos)
```

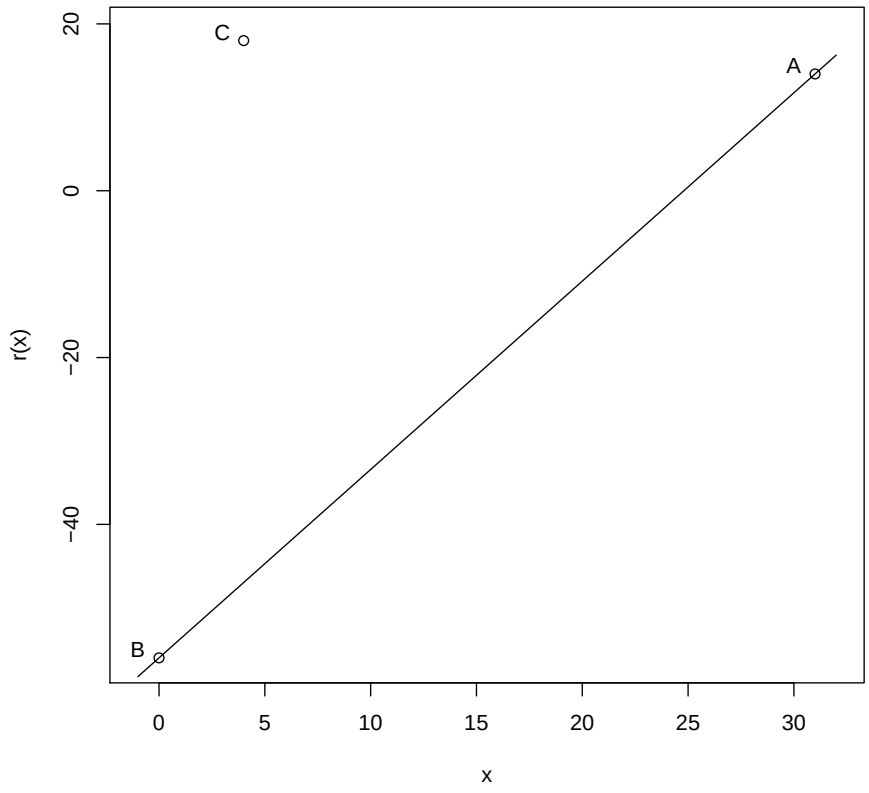


4.2 Código del ejercicio 2

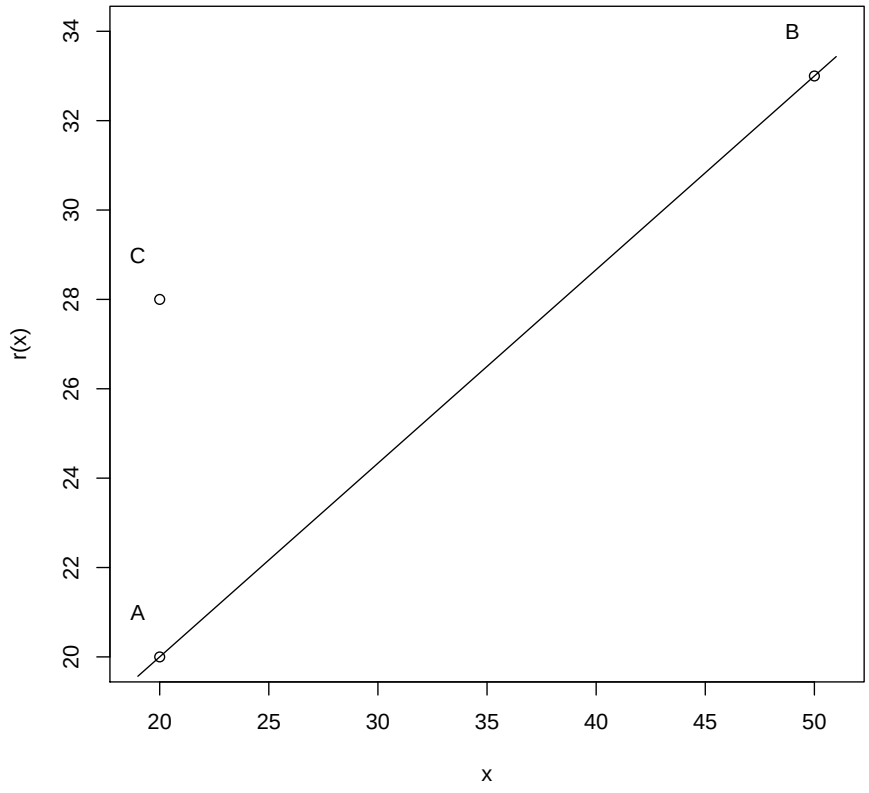
```r
#EJERCICIO 2
#Implementar en R un algoritmo basado en la ecuación
#de la recta cuya entrada sean tres puntos distintos del
#plano (A, B y C) que determine:
x <- c()
y <- c()

#a) Si C está a la derecha o a la izquierda de la recta
#que une A con B orientada por el vector B-A.



apartadoA <- function(A, B, C)
{
  r <- function(x) {
    y = (((B[2] - A[2]) / (B[1] - A[1])) * (x - A[1])) + A[2]
  }
  x <- seq(min(A[1], B[1], C[1])-1, max(A[1], B[1], C[1])+1)
  plot(x, r(x), type = "l",
  ylim = c(min(A[2], B[2], C[2])-1, max(A[2], B[2], C[2]))+1)
  if (B[1] == A[1]) {
    abline(v = A[1])
  }
  text(A[1]-1,A[2]+1,"A")
  text(B[1]-1,B[2]+1,"B")
  text(C[1]-1,C[2]+1,"C")
  points(A[1], A[2])
  points(B[1], B[2])
  points(C[1], C[2])
  x <- C[1]
  y <- r(C)
```

```
  pendiente <- (B[2] - A[2]) / (B[1] - A[1])
  if (A[2] == B[2]) {
    s <- 'Recta horizontal, ni a derecha ni a izquierda'
  }
  else{
    if (((y[1] < C[2]) & (pendiente > 0)) |
        (y[1] > C[2]) & (pendiente < 0))
      s <- 'El punto está a la izquierda de la recta'
    else if (((y[1] > C[2]) & (pendiente > 0)) |
             (y[1] < C[2]) & (pendiente < 0))
      s <- 'El punto está a la derecha de la recta'
    if (y[1] == C[2])
      s <- 'El punto está en la recta'
  }

  s
}

#b) Si A, B y C hacen un giro a la derecha o no.
apartadoB <- function(A, B, C)
{
  giro <-
    (B[1] - A[1]) * (C[2] - A[2]) - (B[2] - A[2]) * (C[1] - A[1])
  if (giro < 0)
    s2 <- "Giro a la derecha"
  else if (giro > 0)
    s2 <- "Giro a la izquierda"
  else
    "No hay giro"
  s2
```

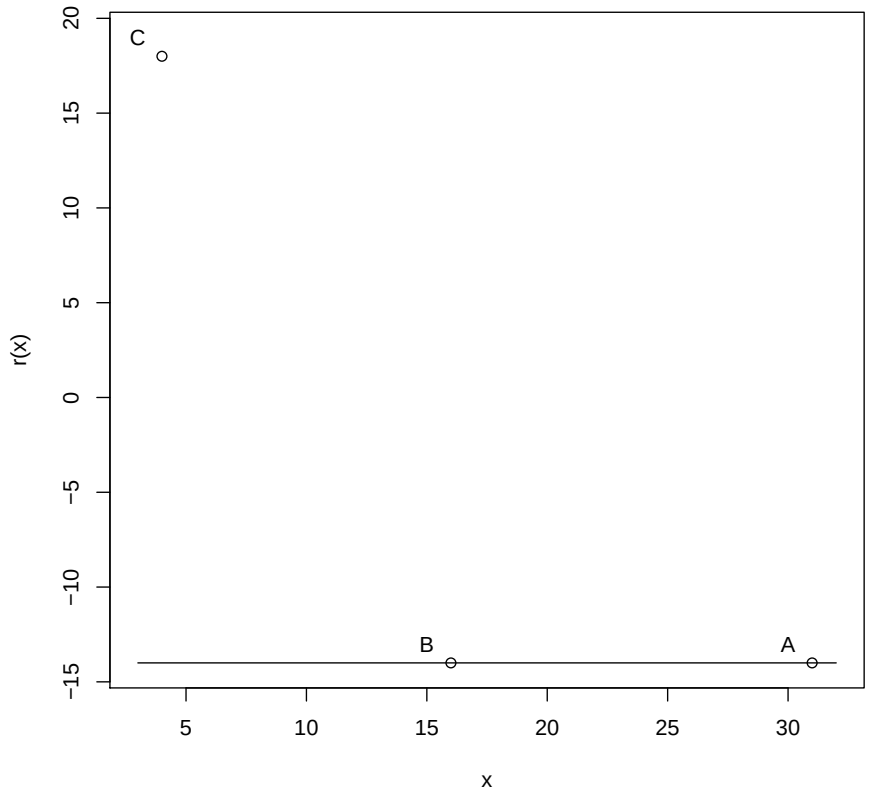
```
}

#Ejemplo con recta descendente, giro derecha
A <- c(31, 14)
B <- c(0, -56)
C <- c(4,18)

apartadoA(A, B, C)
```



```
## [1] "El punto está a la izquierda de la recta"

apartadoB(A, B, C)

## [1] "Giro a la derecha"

#Ejemplo con recta ascendente, giro izquierda
A <- c(20, 20)
B <- c(50, 33)
C <- c(20,28)


apartadoA(A, B, C)
```



```
## [1] "El punto está a la izquierda de la recta"

apartadoB(A, B, C)

## [1] "Giro a la izquierda"

#Ejemplo con recta horizontal, giro derecha
A <- c(31, -14)
B <- c(16, -14)
C <- c(4,18)


apartadoA(A, B, C)
```



```
## [1] "Recta horizontal, ni a derecha ni a izquierda"

apartadoB(A, B, C)

## [1] "Giro a la derecha"
```