Práctica 1

Anabel Gómez Ríos

Ejercicio de Generación y Visualización de datos.

1. Construir una función `lista = simula_unif(N, dim, rango)` que calcule una lista de longitud `N` de vectores de dimensión `dim` conteniendo números aleatorios uniformes en el intervalo `rango`.

```
set.seed(237)
```

```
simula_unif <- function(N, dim, rango) {
  lapply(1:N, function(x) runif(dim, min = rango[1], max = rango[2]))
}
```

Por ejemplo, vamos a obtener una lista de longitud 4 de vectores de dimensión 3 en el rango (0,1):

```
l <- simula_unif(4,3,c(0,1))
l
```

```
## [[1]]
## [1] 0.5921064 0.5576325 0.4955501
##
## [[2]]
## [1] 0.3277974 0.5416440 0.8203180
##
## [[3]]
## [1] 0.48786829 0.05446795 0.74735277
##
## [[4]]
## [1] 0.4940525 0.5267060 0.2230311
```

2. Construir una función `lista = simula_gauss(N, dim, sigma)` que calcule una lista de longitud `N` de vectores de dimensión `dim` conteniendo números aleatorios gaussianos de media 0 y varianzas dadas por el vector `sigma`.

```
simula_gauss <- function(N, dim, sigma) {
  lapply(1:N, function(x) rnorm(dim, mean = 0, sigma))
}
```

Vamos a obtener por ejemplo una lista de longitud 3 de vectores de dimensión 2 y vector de desviaciones (1,7):

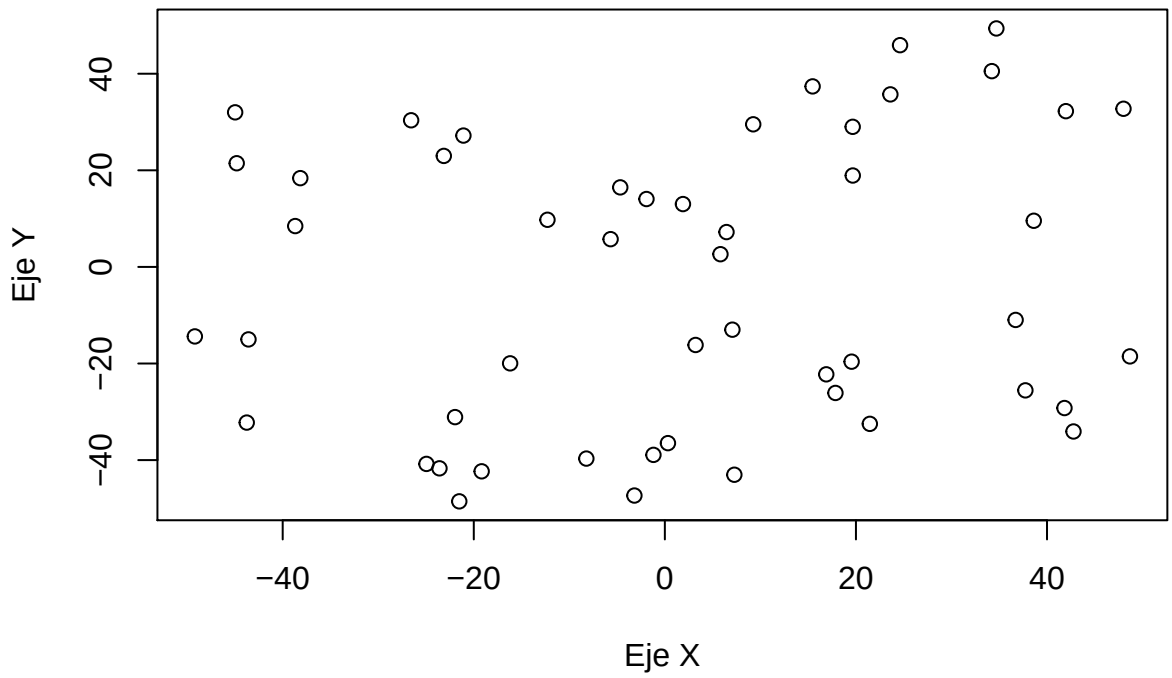
```
simula_gauss(3,2,c(1,7))
```

```
## [[1]]
## [1] -0.8273568 -3.3738051
##
## [[2]]
## [1]  -0.6827164 -15.3826366
##
## [[3]]
## [1] -1.094434 -3.823909
```

Como vemos, la varianza 1 se coge para la primera coordenada de los vectores y la varianza 7 para la segunda coordenada. Si ponemos por ejemplo c(1,7,3) como vector de desviaciones y la dimensión de los vectores es 2, ignora el número 3.

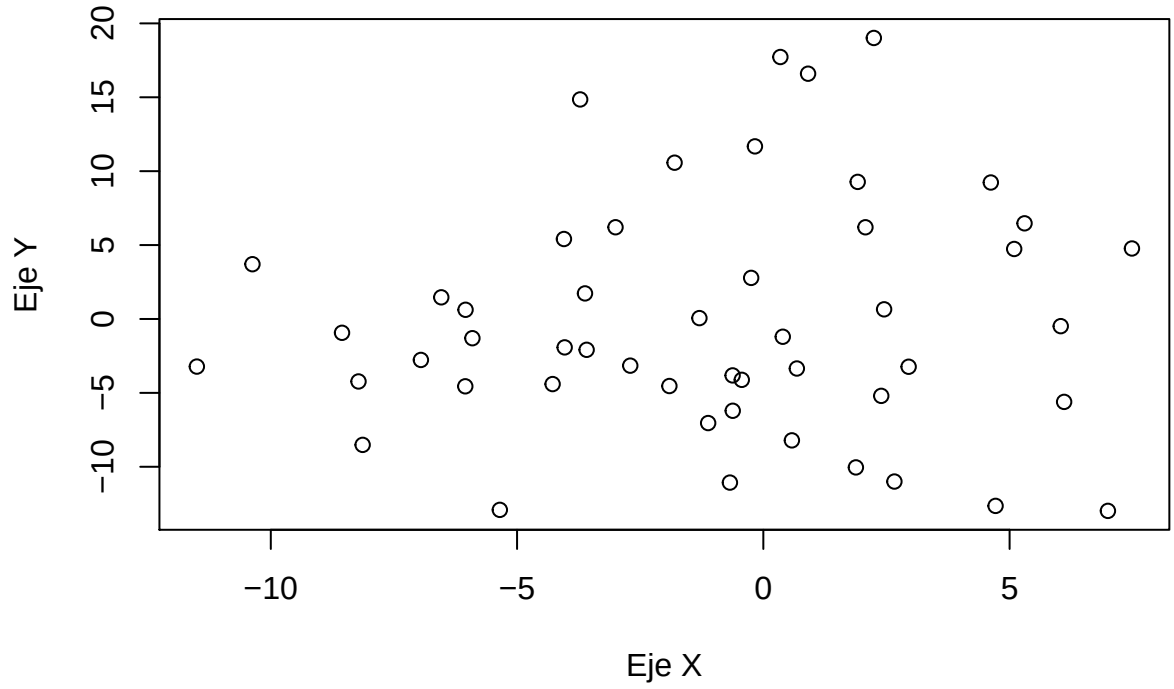
3. Suponer `N = 50, dim = 2, rango = [-50, 50]` en cada dimensión. Dibujar una gráfica de la salida de la función correspondiente.

```r
lista <- simula_unif(50, 2, c(-50,50))
# Obtenemos las coordenadas x e y, que son la primera y segunda columna
# de la lista (primera componente y segunda componente de cada vector
# en la lista)
x <- rapply(lista, function(x) x[1])
y <- rapply(lista, function(x) x[2])
plot(x, y, type = "p", xlab = "Eje X", ylab = "Eje Y")
```



4. Suponer `N = 50, dim = 2, sigma = [5,7]`. Dibujar una gráfica de la salida de la función correspondiente.

```
lista2 <- simula_gauss(50, 2, c(5,7))
# Obtenemos las coordenadas x e y, que son la primera y segunda columna
# de la lista (primera componente y segunda componente de cada vector
# en la lista)
x <- rapply(lista2, function(x) x[1])
y <- rapply(lista2, function(x) x[2])
plot(x, y, type = "p", xlab = "Eje X", ylab = "Eje Y")
```



5. Construir la función `v = simula_recta(intervalo)` que calcula los parámetros `v = (a,b)` de una recta aleatoria `y = ax + b` que corte al cuadrado [-50,50] times$ [-50,50] (Ayuda: Para calcular la recta simular las coordenadas de dos puntos del cuadrado y calcular la recta que pasa por ellos).

```
simula_recta <- function(intervalo) {
  l <- simula_unif(2, 2, intervalo)
  a <- (l[[2]][2] - l[[1]][2]) / (l[[2]][1] - l[[1]][1])
  b <- l[[1]][2] - a * l[[1]][1]
  c(a,b)
}
```

```
simula_recta(c(-50,50))
```

```
## [1] -1.427238 -3.351577
```

6. Generar una muestra 2D de puntos usando `simula_unif()` y etiquetar la muestra usando el signo de la función $f(x, y) = y - ax - b$ de cada punto a una recta simulada con `simula_recta()`. Mostrar una gráfica con el resultado de la muestra etiquetada junto con la recta usada para ello.

Vamos primero a construir una función que nos devuelva la etiqueta +1 o -1 según el signo de la f dada (la evaluación ya ha sido hecha):

```
etiquetar <- function(f) {
  if (f > 0) {
    return(+1)
  }
  else {
    return(-1)
  }
}
```

Ahora generamos la recta con la función indicada y utilizamos la función anterior para etiquetar los puntos haciendo uso también de la función `lapply()`, de la siguiente forma:

```
r <- simula_recta(c(-50,50))
etiquetas <- lapply(1:length(lista), function(i) {
  #Obtenemos los puntos uno a uno y los etiquetamos
  p <- lista[[i]]
  f <- p[2] - r[1]*p[1] - r[2]
  etiquetar(f)
})
```

Esto nos devuelve en `etiquetas` una lista de 50 vectores, todos con una componente. Lo vamos a pasar a un vector con las 50 componentes con la función `unlist()`

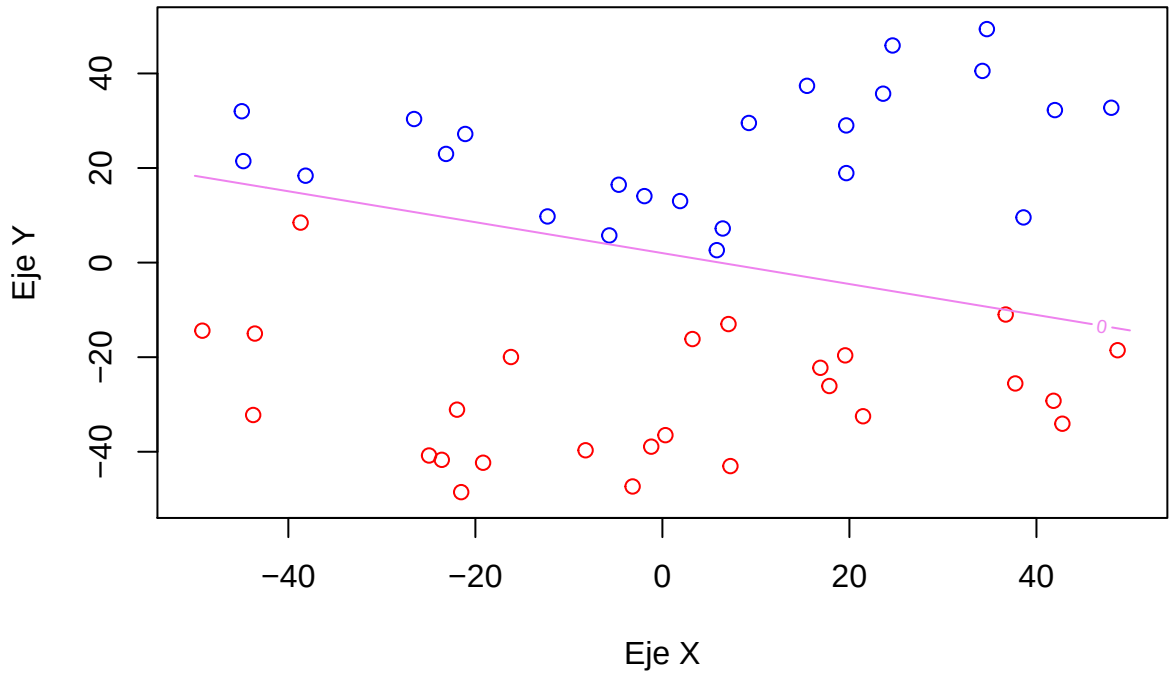
```
etiquetas <- unlist(etiquetas)
#Mostramos las etiquetas
etiquetas
```

```
##  [1]  1 -1 -1 -1 -1  1 -1 -1 -1 -1  1  1 -1  1  1  1  1 -1  1  1  1  1  1
## [24]  1 -1  1 -1 -1 -1  1  1 -1 -1 -1  1 -1 -1 -1 -1  1 -1  1  1 -1  1 -1
## [47] -1  1 -1  1
```

Vamos a dibujar el resultado. Pintamos en una misma gráfica los puntos etiquetados con +1 (en rojo) y los puntos etiquetados con -1 (en azul). Pintamos también la recta que hemos utilizado para etiquetar los puntos:

```
pinta_particion <- function(lista_puntos, etiquetas=NULL, visible=FALSE, f=NULL) {
  x <- rapply(lista_puntos, function(x) x[1])
  y <- rapply(lista_puntos, function(x) x[2])
  if(is.null(etiquetas))
```

```
    etiquetas=1
  else etiquetas = etiquetas+3
  plot(x, y, type = "p", col = etiquetas, xlab = "Eje X", ylab = "Eje Y",
       xlim = c(-50,50), ylim = c(-50,50))

  if(visible) {
    sec <- seq(-50, 50, length.out = 1000)
    z <- outer(sec, sec, f)
    contour(sec, sec, z, col = "violet", levels = 0, add = TRUE)
  }
}

pinta_particion(lista, etiquetas, TRUE, function(x,y) r[1]*x-y+r[2])
```

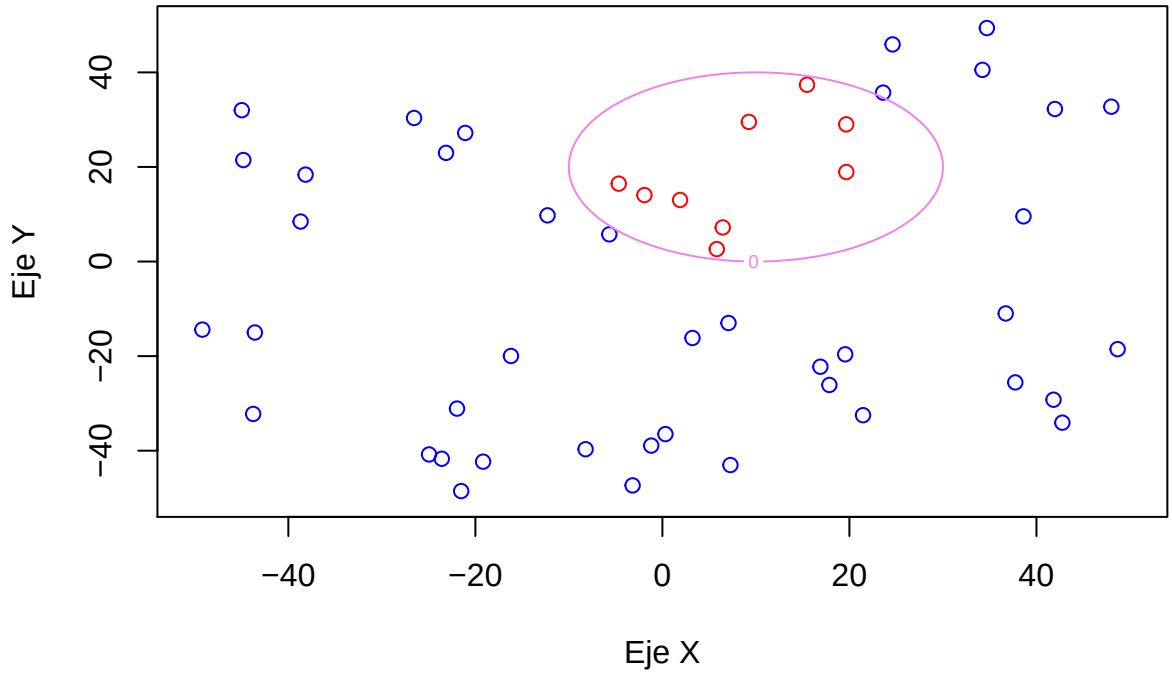


7. Usar la muestra generada en el apartado anterior y etiquetarla con +1,-1 usando el signo de cada una de las siguiente funciones y visualizar el resultado del etiquetado de cada función junto con su gráfica y comparar el resultado con el caso lineal. ¿Qué consecuencias extrae sobre la forma de las regiones positiva y negativa?:

Vamos a utilizar la función para etiquetar que hemos hecho en el apartado anterior y la función para dibujar que ya teníamos hecha también.

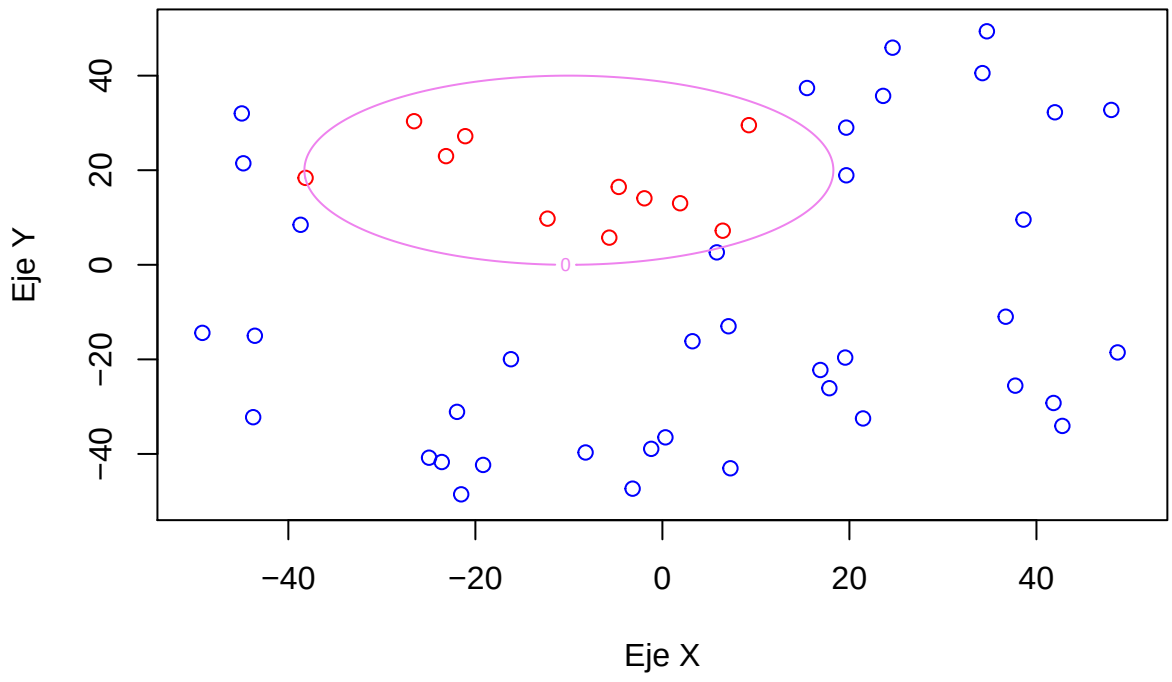
a) $f(x, y) = (x - 10)^2 + (y - 20)^2 - 400$

```
etiquetasFA <- lapply(1:length(lista), function(i) {
  #Obtenemos los puntos uno a uno y los etiquetamos
  p <- lista[[i]]
  f1 <- (p[1]-10)^2 + (p[2] - 20)^2 - 400
  etiquetar(f1)
})
etiquetasFA <- unlist(etiquetasFA)

pinta_particion(lista, etiquetasFA, TRUE, function(x,y) (x-10)^2 + (y-20)^2 - 400)
```



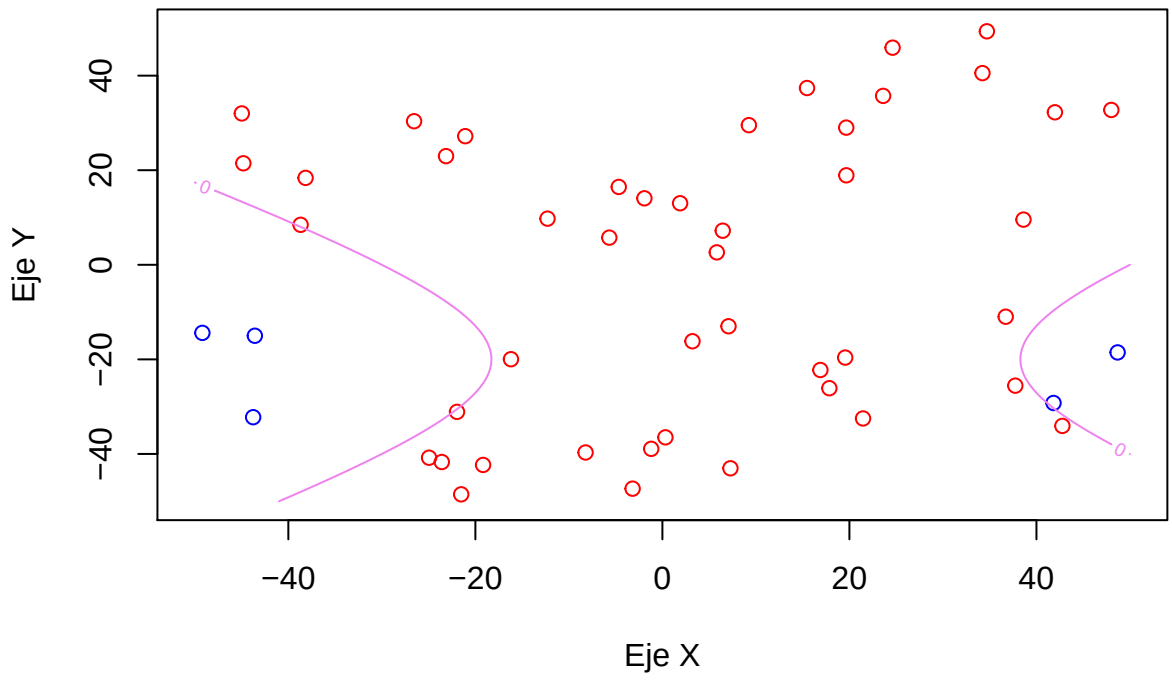
b) $f(x, y) = 0.5 * (x + 10)^2 + (y - 20)^2 - 400$

```
etiquetasFB <- lapply(1:length(lista), function(i) {
  #Obtenemos los puntos uno a uno y los etiquetamos
  p <- lista[[i]]
  f2 <- 0.5*(p[1]+10)^2 + (p[2] - 20)^2 - 400
  etiquetar(f2)
})
etiquetasFB <- unlist(etiquetasFB)

pinta_particion(lista, etiquetasFB, TRUE, function(x,y) 0.5*(x+10)^2 + (y-20)^2 - 400)
```



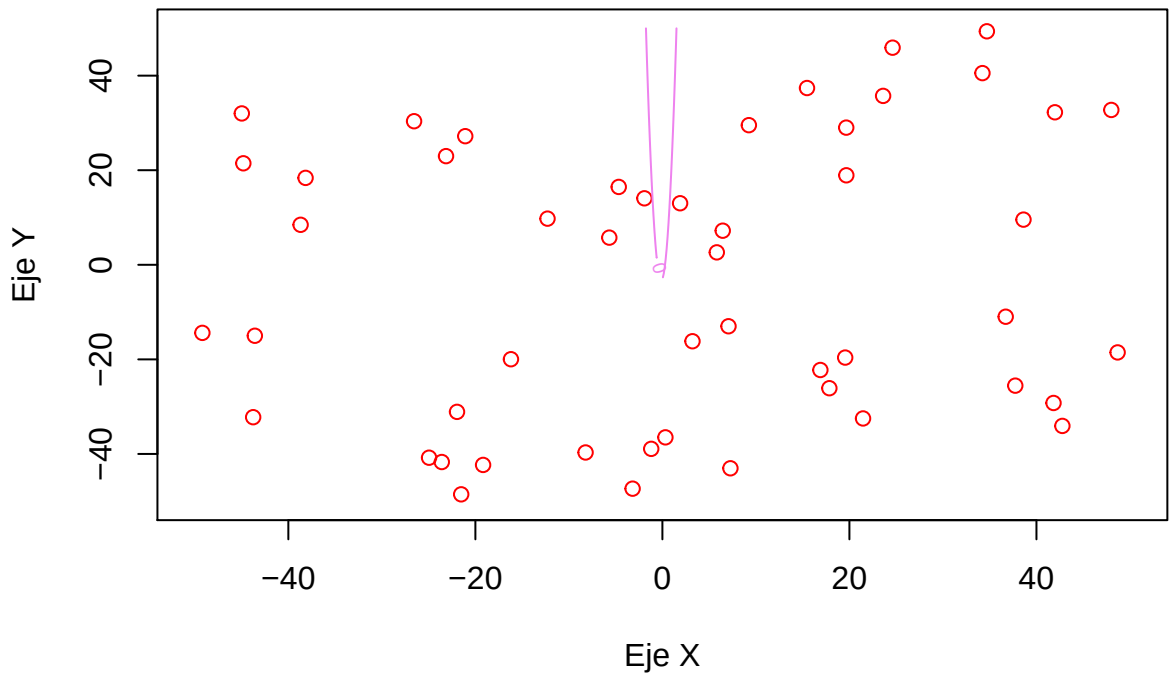
c) $f(x, y) = 0.5 * (x - 10)^2 - (y + 20)^2 - 400$

```
etiquetasFC <- lapply(1:length(lista), function(i) {
  #Obtenemos los puntos uno a uno y los etiquetamos
  p <- lista[[i]]
  f3 <- 0.5*(p[1]-10)^2 - (p[2] + 20)^2 - 400
  etiquetar(f3)
})
etiquetasFC <- unlist(etiquetasFC)

pinta_particion(lista, etiquetasFC, TRUE, function(x,y) 0.5*(x-10)^2 - (y+20)^2 - 400)
```



d) $f(x, y) = y - 20x^2 - 5x + 3$

```
etiquetasFD <- lapply(1:length(lista), function(i) {
  #Obtenemos los puntos uno a uno y los etiquetamos
  p <- lista[[i]]
  f4 <- p[2] - 20*(p[1]^2) - 5*p[1] + 3
  etiquetar(f4)
})
etiquetasFD <- unlist(etiquetasFD)
pinta_particion(lista, etiquetasFD, TRUE, function(x,y) y - 20*x^2 - 5*x + 3)
```



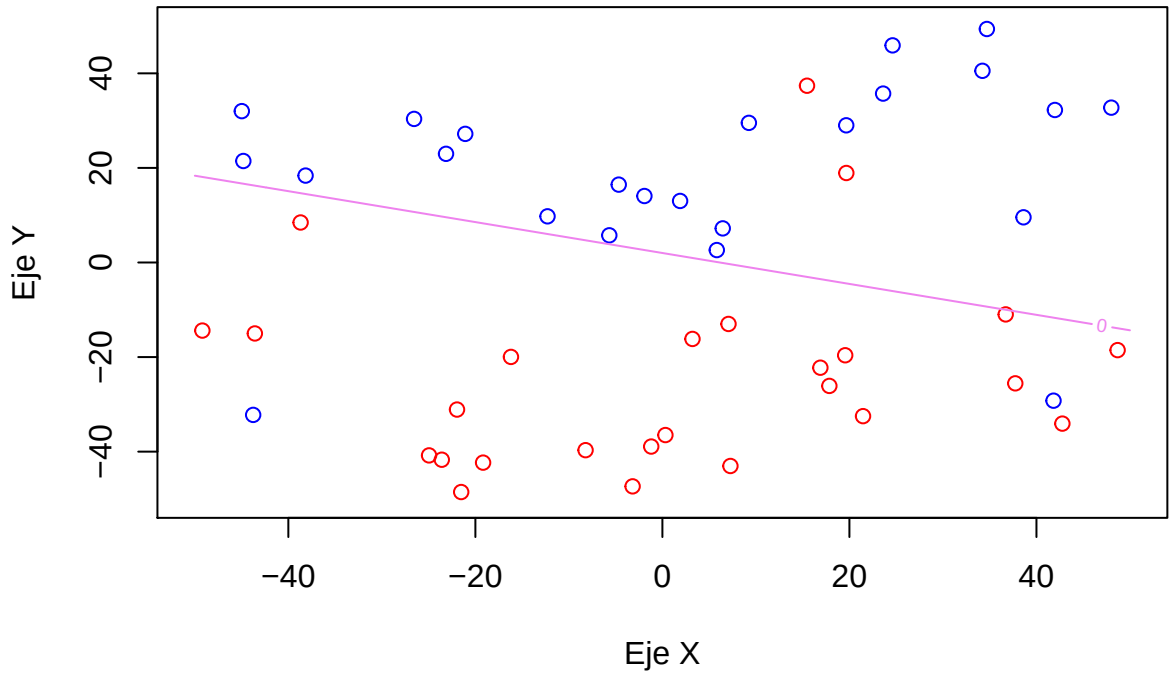
Como vemos, al ser ahora las funciones cuadráticas y no lineales, los datos no son linealmente separables (los que caen dentro de las gráficas se quedan en el centro o a los lados) y por tanto el perceptron no será capaz de parar ante estas clasificaciones.

8. Considerar de nuevo la muestra etiquetada en el apartado 6. Modifique las etiquetas de un 10% aleatorio de muestras positivas y otro 10% de muestras negativas.

Visualice los puntos con las nuevas etiquetas y la recta del apartado 6.

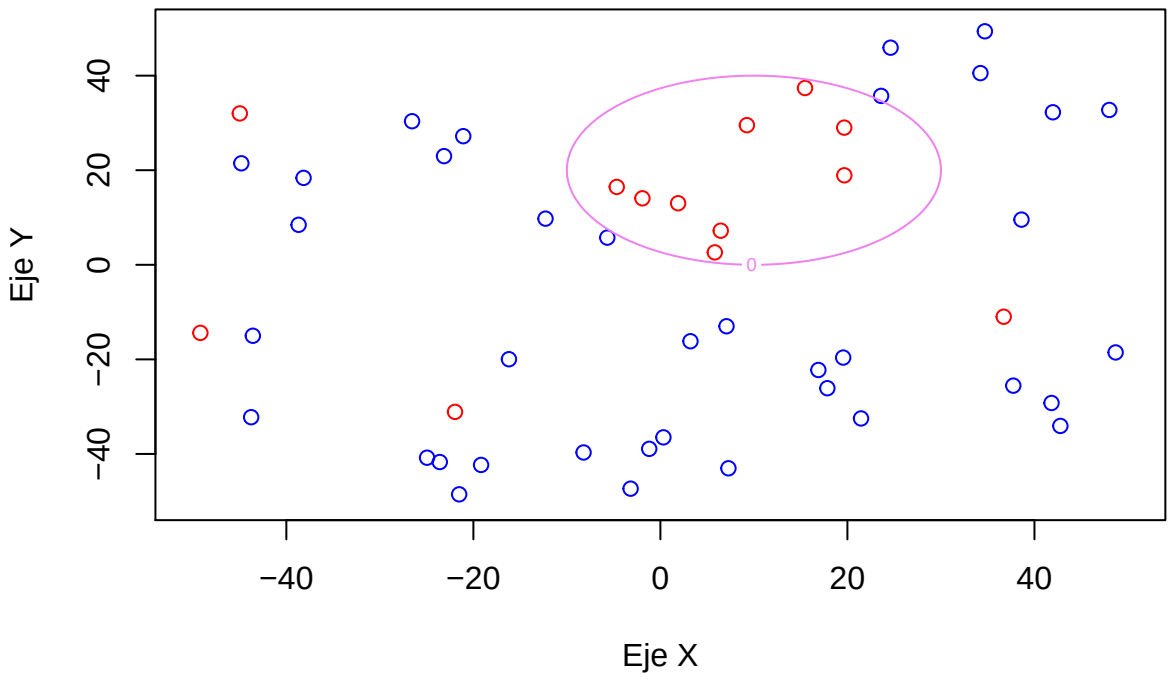
```
cambiar_etiquetas <- function(etiquetas) {
  num <- 1:length(etiquetas)
  etiquetas_cambiadas <- etiquetas
  #Cogemos las posiciones de las etiquetas positivas y negativas
  positivos <- num[etiquetas > 0]
  negativos <- num[etiquetas < 0]
  #Comprobamos que hay algún elemento que cambiar y obtenemos el 10%
  #de posiciones aleatorias
  if(length(positivos)*0.1 > 0) {
    cambiar1 <- sample(positivos, length(positivos)*0.1)
    #Cambiamos las etiquetas que hemos obtenido antes
    etiquetas_cambiadas[cambiar1] <- -1
  }
  if(length(negativos)*0.1 > 0) {
```

```
    cambiar2 <- sample(negativos, length(negativos)*0.1)
    #Cambiamos las etiquetas que hemos obtenido antes
    etiquetas_cambiadas[cambiar2] <- +1
  }

  etiquetas_cambiadas
}

etiquetas2 <- cambiar_etiquetas(etiquetas)
pinta_particion(lista, etiquetas2, TRUE, function(x,y) r[1]*x-y+r[2])
```

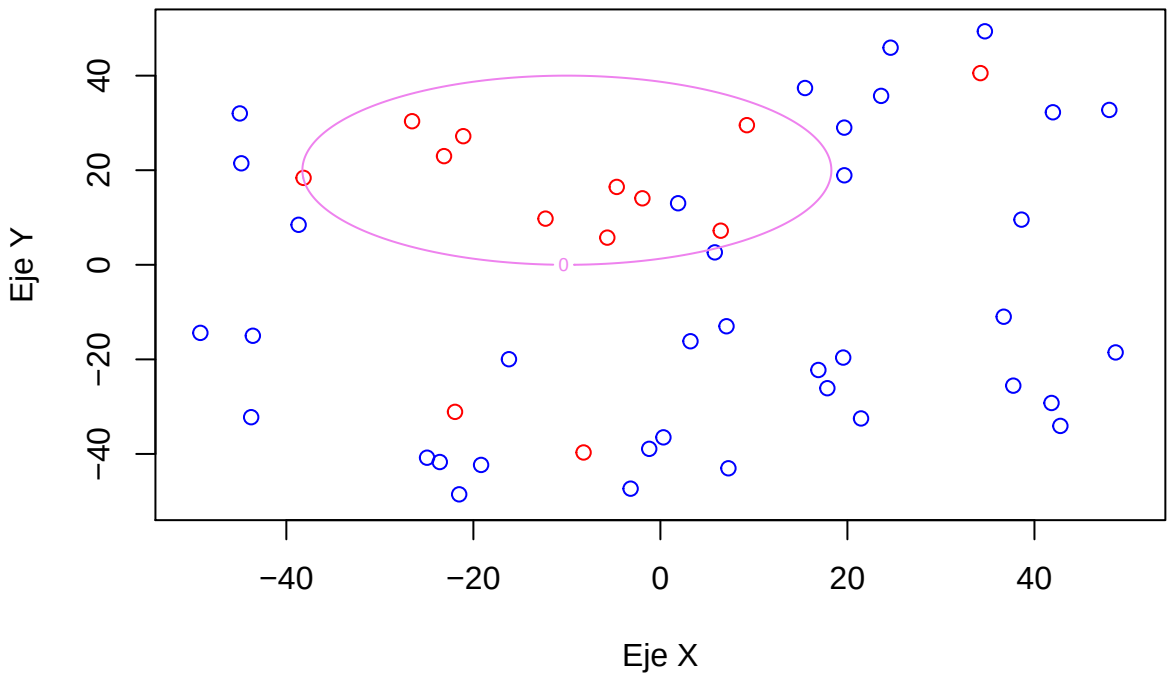


En una gráfica aparte visualice de nuevo los mismos puntos pero junto con las funciones del apartado 7.

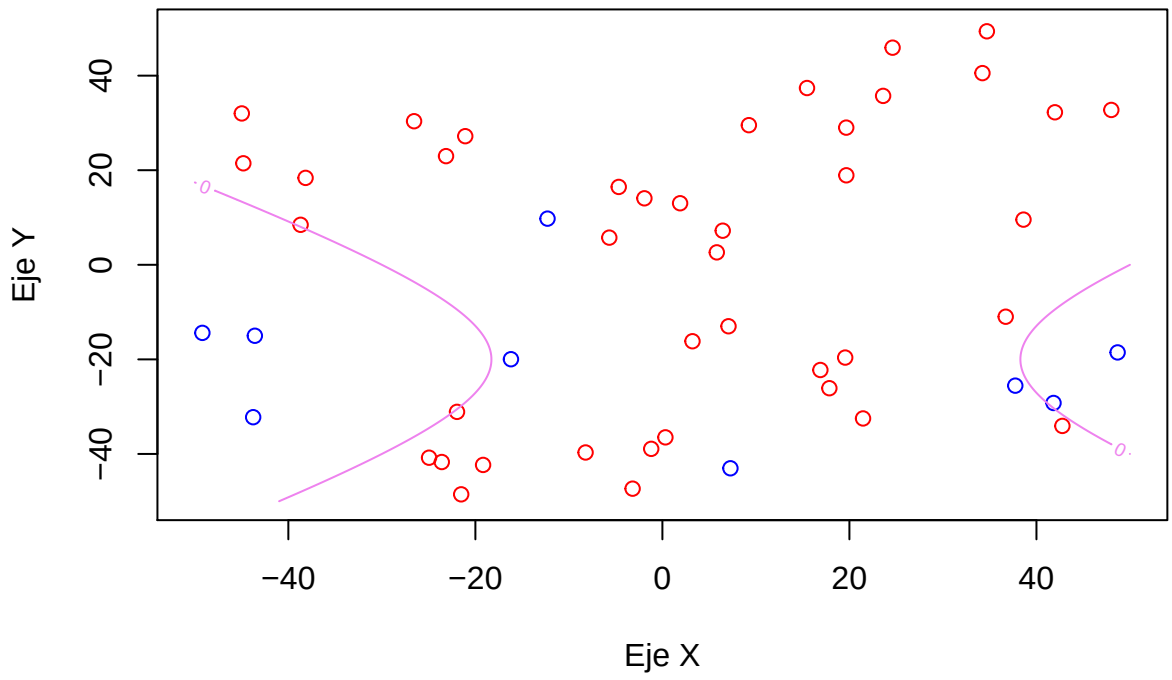
```
pinta_particion(lista, cambiar_etiquetas(etiquetasFA), TRUE, function(x,y) (x-10)^2 + (y-20)^2 - 400)
```



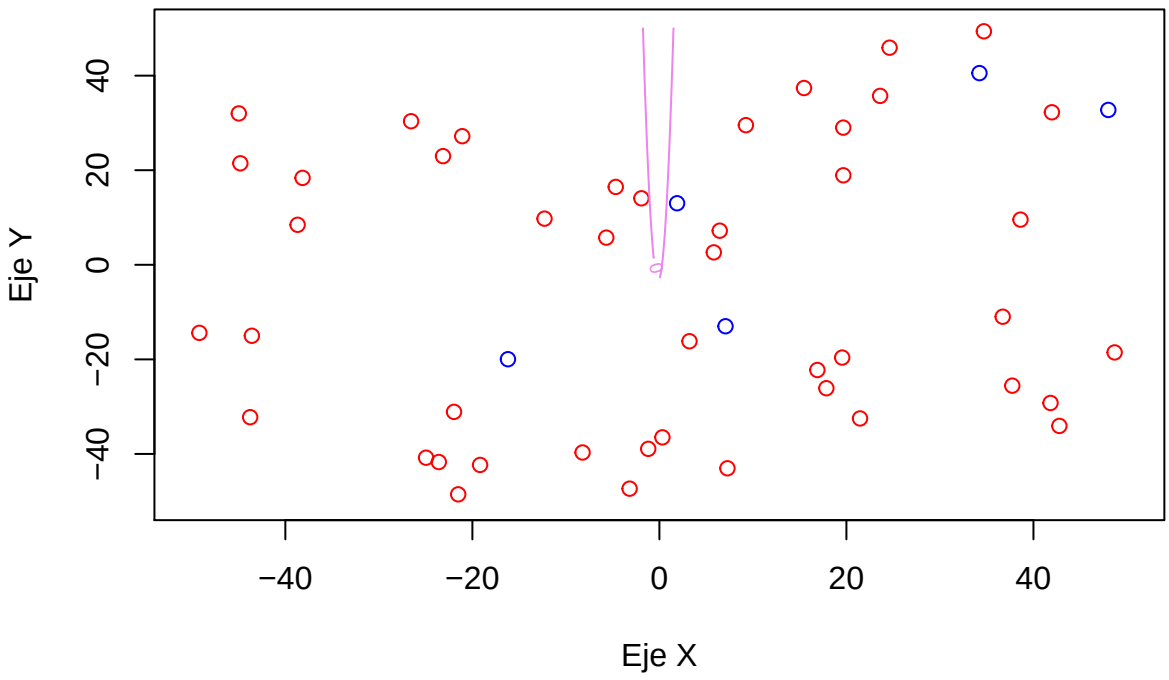

```
pinta_particion(lista, cambiar_etiquetas(etiquetasFB), TRUE, function(x,y) 0.5*(x+10)^2 + (y-20)^2 - 40
```




```
pinta_particion(lista, cambiar_etiquetas(etiquetasFC), TRUE, function(x,y) 0.5*(x-10)^2 - (y+20)^2 - 40
```




```
pinta_particion(lista, cambiar_etiquetas(etiquetasFD), TRUE, function(x,y) y - 20*x^2 - 5*x + 3)
```



Ejercicio de Ajuste del Algoritmo Perceptron

1. Implementar la función `sol = ajusta_PLA(datos, label, max_iter, vini)` que calcula el hiperplano solución a un problema de clasificación binaria usando el algoritmo PLA. La entrada de `datos` es una matriz donde cada item con su etiqueta está representado por una fila de la matriz, `label` el vector de etiquetas (cada etiqueta es un valor +1 o -1), `max_iter` es el número máximo de iteraciones permitidas y `vini` el valor inicial del vector. La salida `sol` devuelve los coeficientes del hiperplano.

Aunque pide que la salida sean los coeficientes del hiperplano, en el segundo apartado pide también el número de iteraciones que han sido necesarias para converger, por lo que vamos a devolver una lista cuya primera componente tenga `w` y la segunda componente sea el número de iteraciones necesario para converger.

```
ajusta_PLA <- function(datos, label, max_iter, vini) {
  parada <- F
  fin <- F
  w <- vini
  iter <- 1
  #Mientras no hayamos superado el máximo de iteraciones o
  #no se haya encontrado solución
  while(!parada) {
    #iteramos sobre los datos
```

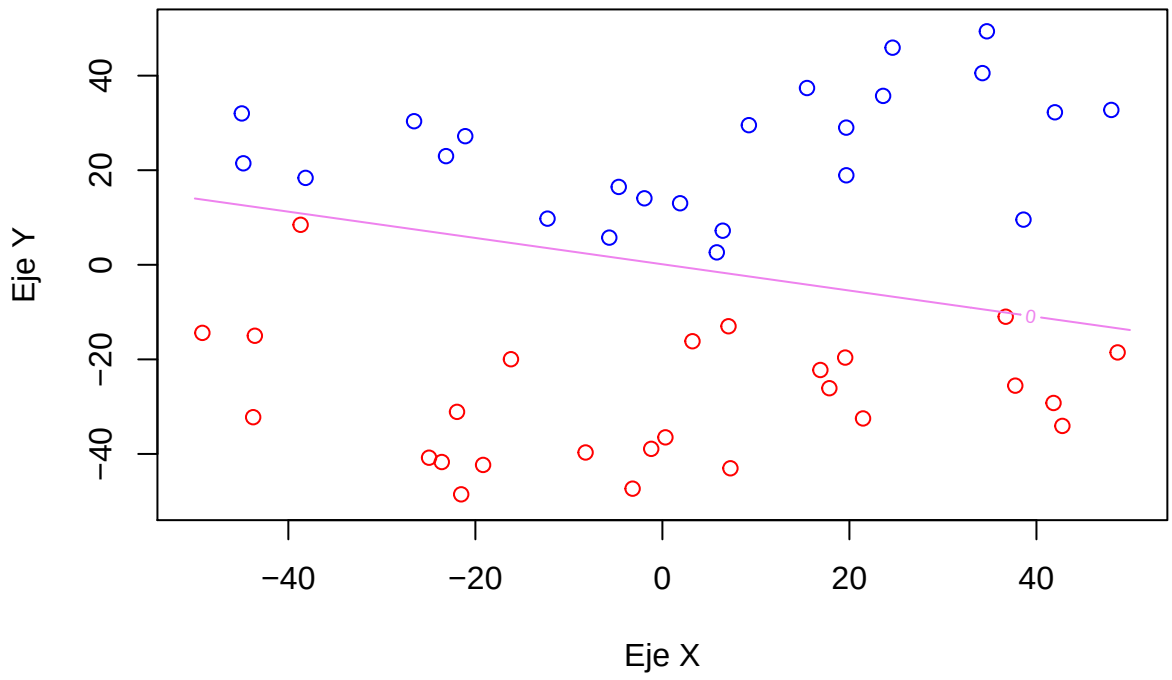
```
    for (j in 1:nrow(datos)) {
      if (sign(crossprod(w, datos[j,])) != label[j]) {
        w <- w + label[j]*datos[j,]
        #La variable fin controla si se ha entrado en el if
        fin <- F
      }
    }
    #Si no se ha entrado en el if, todos los datos estaban bien
    #clasificados y podemos poner a TRUE la variable parada.
    if(fin == T) {
      parada = T
    }
    else {
      fin = T
    }
    iter <- iter + 1
    if (iter >= max_iter) parada = T
  }
  #Devolvemos el hiperplano y el número máximo de iteraciones al que hemos
  #llegado.
  list(w, iter)
}
```

2. Ejecutar el algoritmo PLA con los valores simulados en el apartado 6 del ejercicio 1, inicializando el algoritmo con el vector cero y con vectores de número aleatorios en [0,1] (10 veces). Anotar el número medio de iteraciones necesarias en ambos para converger. Valorar el resultado.

```
#Metemos los datos, que teníamos en una lista llamada "lista" en
#una matriz.
m <- matrix(unlist(lista), 50, 2, byrow=TRUE)
datos <- matrix(1, 50, 3)
datos[1:50, 1:2] <- m
sol <- ajusta_PLA(datos, etiquetas, 20, c(0,0,0))
iter1 <- sol[[2]]
iter1
```

```
## [1] 11
```

```
w <- sol[[1]]
w <- -w / w[2]
pinta_particion(lista, etiquetas, TRUE, function(x,y) y-w[3]-w[1]*x)
```



En este caso el número de iteraciones necesario para que el algoritmo converja es 11. Vamos a hacerlo ahora generando números aleatorios entre 0 y 1.

```
waleatorios <- simula_unif(10, 3, c(0,1))
iteraciones <- lapply(1:10, function(i) {
  wi <- waleatorios[[i]]
  sol <- ajusta_PLA(datos, etiquetas, 200, wi)
  sol[[2]]
})

iteraciones <- unlist(iteraciones)
mean(iteraciones)
```

```
## [1] 10.9
```

Generando números aleatorios entre 0 y 1 para el w inicial, el número medio de iteraciones que son necesarias para que el algoritmo converja es 10.9. Hay que tener en cuenta que las iteraciones que necesita para converjer dependen de los datos (en mi caso, de que he fijado la semilla a 237). Con otros datos podría ser mucho más o incluso menos.

3. Ejecutar el algoritmo PLA con los datos generados en el apartado 8 del ejercicio 1, usando valores de 10, 100 y 1000 para `max_iter`. Etiquetar los datos de la muestra usando la función solución encontrada y contar el número de errores respecto de las etiquetas originales. Valorar el resultado.

Vamos a hacer primero una función que cuente las diferencias entre dos vectores de etiquetas, es decir, la cantidad de posiciones en los que dos vectores de etiquetas tienen valores distintos.

```
cuenta_diferencias <- function(etiquetas1, etiquetas2) {
  vf <- etiquetas1 == etiquetas2
  length(vf[vf == FALSE])
}
```

Vamos a hacer ahora una función que nos devuelva el número de errores respecto de las etiquetas originales:

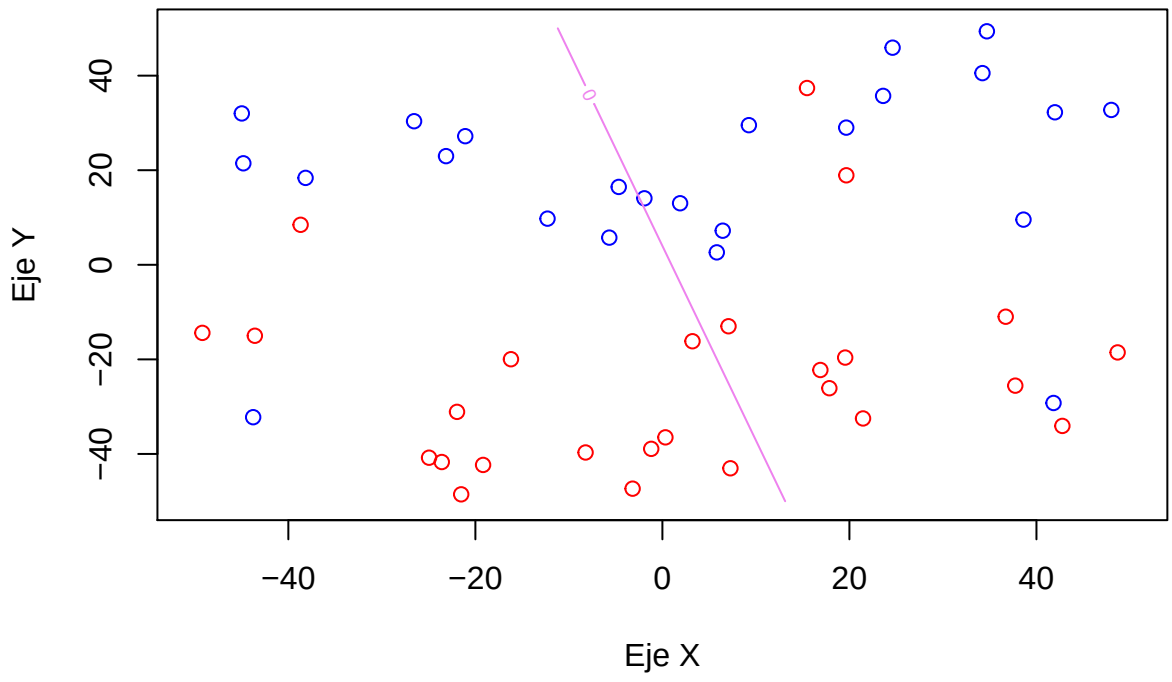
```
cuenta_errores <- function(sol_PLA, etiquetas_originales) {
  w <- sol_PLA[[1]]
  w <- -w / w[2]
  #Recordemos que los datos que hay en la matriz "datos" son los mismos
  #puntos que hay en la matriz "lista"
  etiquetas_cambiadas <- lapply(1:length(lista), function(i) {
    #Obtenemos los puntos uno a uno y los etiquetamos
    p <- lista[[i]]
    f <- -w[1]*p[1] + p[2] - w[3]
    etiquetar(f)
  })
  #Devolvemos el número de errores que da la solución
  cuenta_diferencias(etiquetas_originales, etiquetas_cambiadas)
}
```

Vamos a ejecutarlo ahora con 10, 100 y 1000 iteraciones y vamos a pintar también la solución que da.

```
sol1 <- ajusta_PLA(datos, etiquetas2, 10, c(0,0,0))
cuenta_errores(sol1, etiquetas2)
```

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

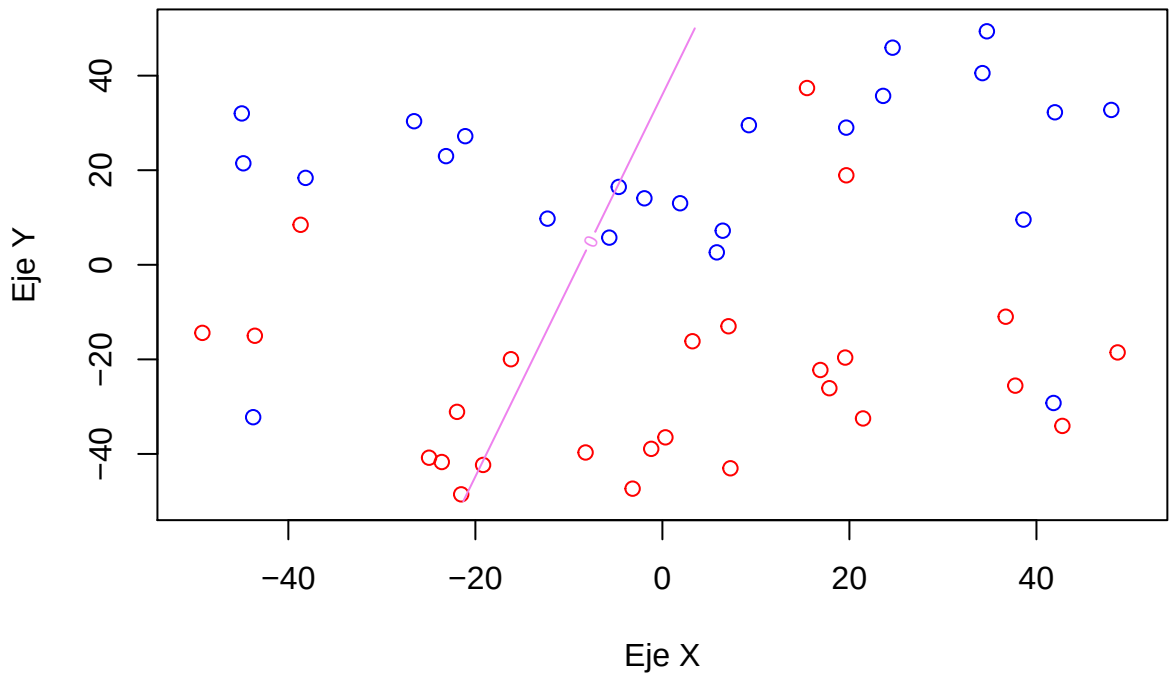
```
w1 <- sol1[[1]]
pinta_particion(lista, etiquetas2, TRUE, function(x,y) w1[1]*x-y+w1[3])
```




```
sol2 <- ajusta_PLA(datos, etiquetas2, 100, c(0,0,0))
cuenta_errores(sol2, etiquetas2)
```

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

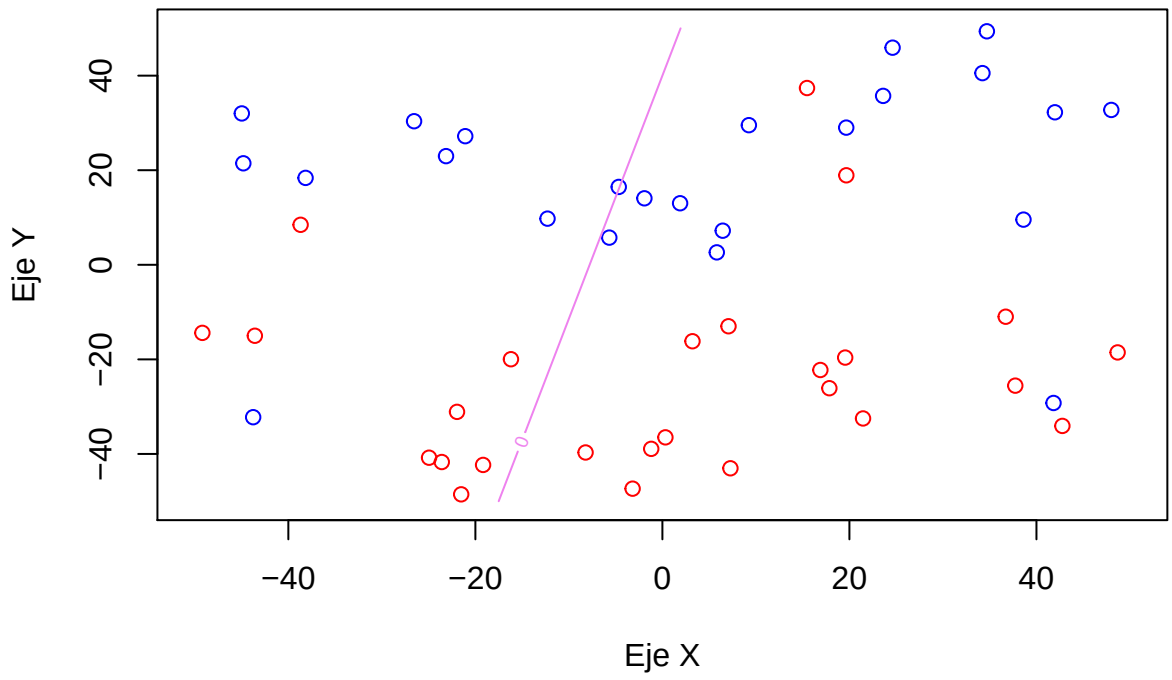
```
w2 <- sol2[[1]]
pinta_particion(lista, etiquetas2, TRUE, function(x,y) w2[1]*x-y+w2[3])
```




```
sol3 <- ajusta_PLA(datos, etiquetas2, 1000, c(0,0,0))
cuenta_errores(sol3, etiquetas2)
```

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

```
w3 <- sol3[[1]]
pinta_particion(lista, etiquetas2, TRUE, function(x,y) w3[1]*x-y+w3[3])
```



Como vemos, en los tres casos el número de puntos mal clasificados es 5. Esto es porque al cambiar etiquetas al azar, los datos han dejado de ser linealmente separables, con lo que el perceptron no puede llegar a una solución en la que todos los puntos estén bien clasificados, da igual el número de iteraciones que le pongamos.

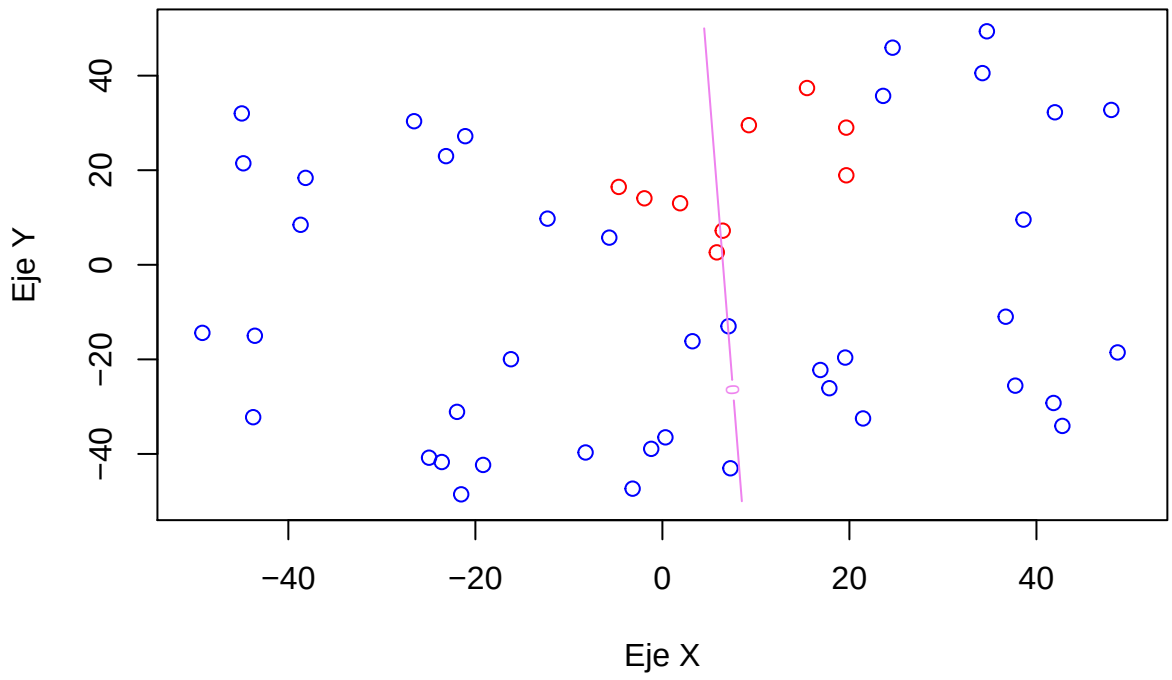
4. Repetir el análisis del punto anterior usando la primera función del apartado 7 del ejercicio 1.

La función es $f(x, y) = (x - 10)^2 + (y - 20)^2 - 400$ y tenemos las etiquetas originales guardadas en un vector llamado `etiquetasFA`.

```
sol1 <- ajusta_PLA(datos, etiquetasFA, 10, c(0,0,0))
cuenta_errores(sol1, etiquetasFA)
```

```
## [1] 30
```

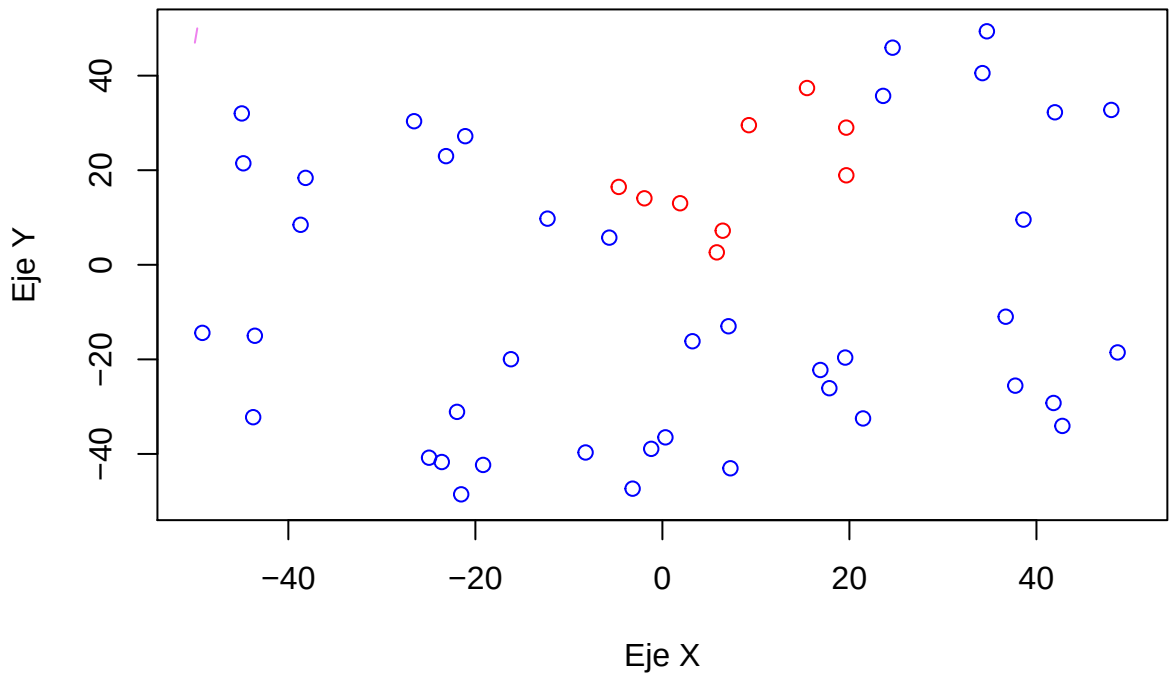
```
w1 <- sol1[[1]]
pinta_particion(lista, etiquetasFA, TRUE, function(x,y) w1[1]*x-y+w1[3])
```




```
sol2 <- ajusta_PLA(datos, etiquetasFA, 100, c(0,0,0))
cuenta_errores(sol2, etiquetasFA)
```

```
## [1] 39
```

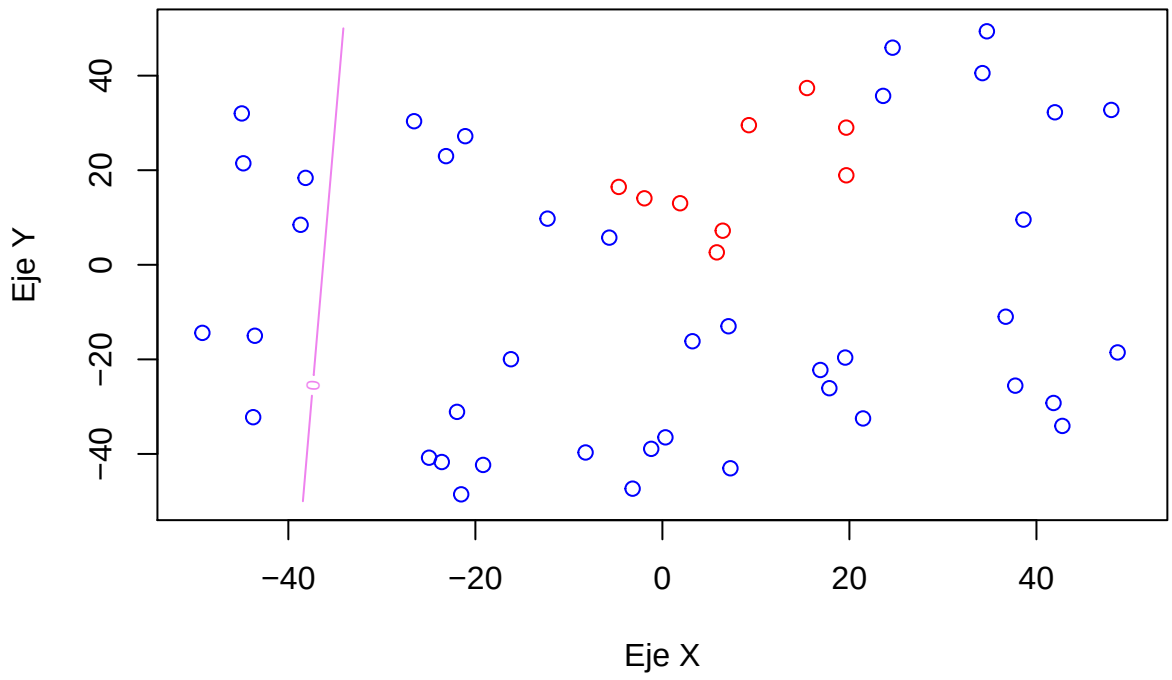
```
w2 <- sol2[[1]]
pinta_particion(lista, etiquetasFA, TRUE, function(x,y) w2[1]*x-y+w2[3])
```




```
sol3 <- ajusta_PLA(datos, etiquetasFA, 1000, c(0,0,0))
cuenta_errores(sol3, etiquetasFA)
```

```
## [1] 17
```

```
w3 <- sol3[[1]]
pinta_particion(lista, etiquetasFA, TRUE, function(x,y) w3[1]*x-y+w3[3])
```



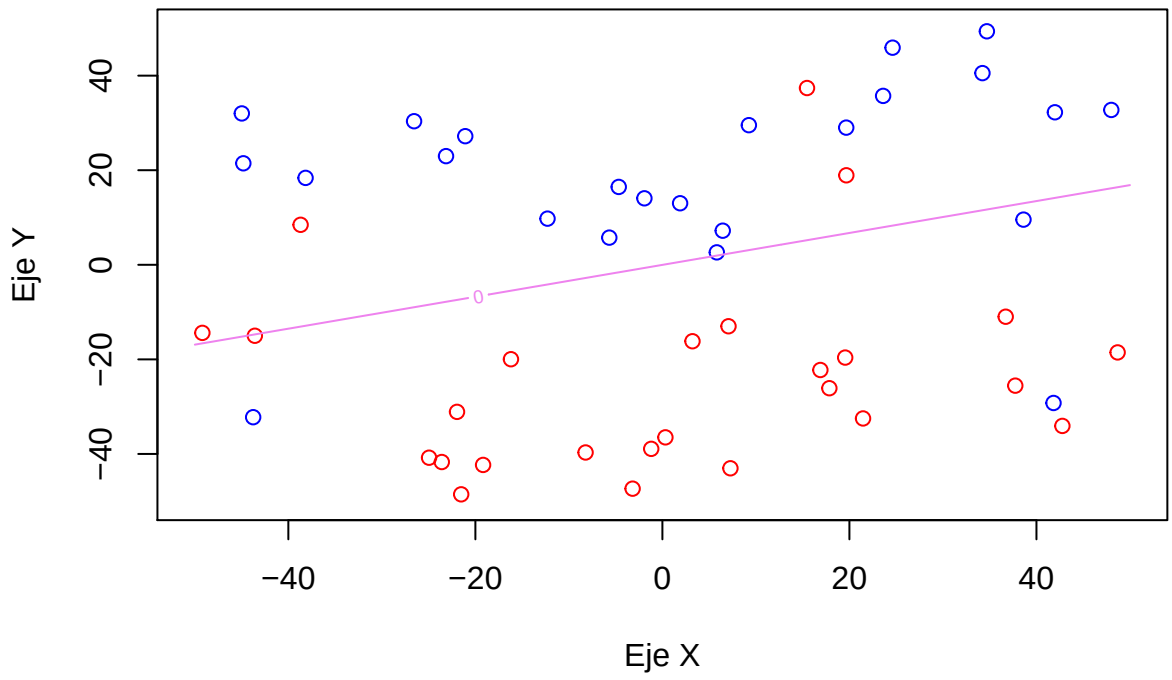
En este caso los datos no eran linealmente separables de entrada, ya que se habían clasificado en base a una función cuadrática, con lo que el perceptron no va a llegar a una solución en la que todos los datos estén bien clasificados.

5. Modifique la función `ajusta_PLA` para que le permita visualizar los datos y soluciones que va encontrando a lo largo de las iteraciones. Ejecute con la nueva versión el apartado 3 del ejericio 2.

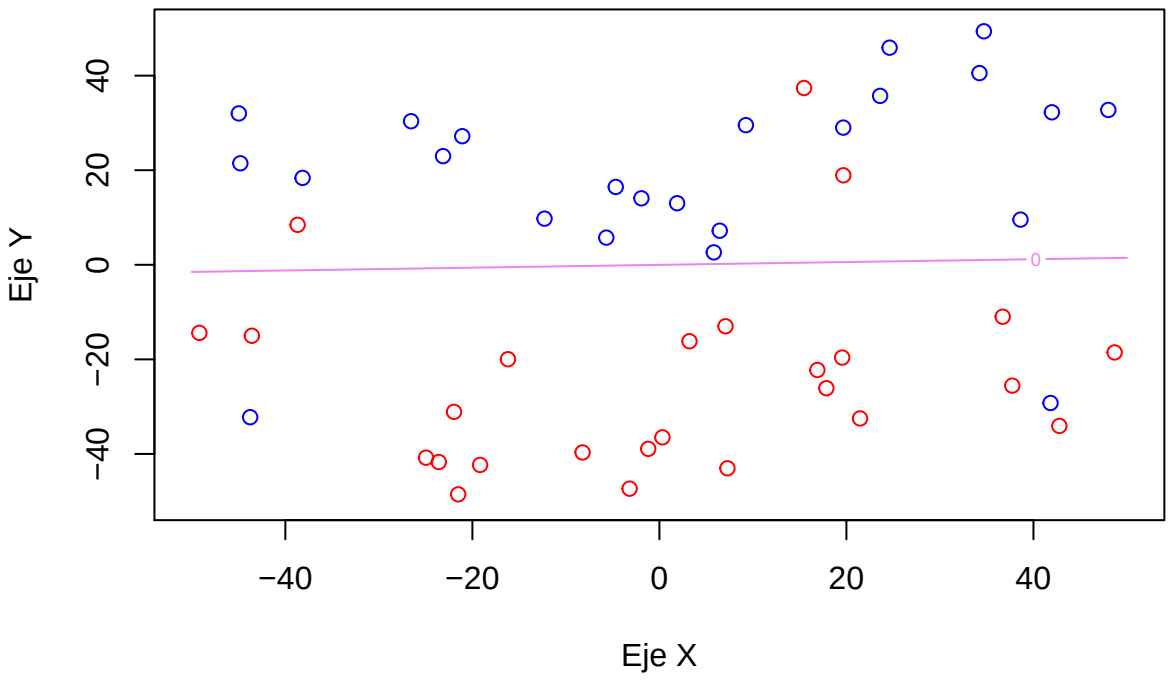
```
ajusta_PLA_sol <- function(datos, label, max_iter, vini) {
  parada <- F
  fin <- F
  w <- vini
  iter <- 1
  #Mientras no hayamos superado el máximo de iteraciones o
  #no se haya encontrado solución
  while(!parada) {
    #iteramos sobre los datos
    for (j in 1:nrow(datos)) {
      if (sign(crossprod(w, datos[j,])) != label[j]) {
        w <- w + label[j]*datos[j,]
        #La variable fin controla si se ha entrado en el if
        fin <- F
      }
    }
```

```
    #Pintamos la gráfica
    w2 <- -w / w[2]
    pinta_particion(lista, label, TRUE, function(x,y) -w2[1]*x + y - w2[3])

    #Paramos la ejecución para que se pueda ver la gráfica
    cat("Presione una tecla para continuar")
    t <- readline()

    #Si no se ha entrado en el if, todos los datos estaban bien
    #clasificados y podemos poner a TRUE la variable parada.
    if(fin == T) {
      parada = T
    }
    else {
      fin = T
    }
    iter <- iter + 1
    if (iter >= max_iter) {parada = T}
  }
  #Devolvemos el hiperplano y el número máximo de iteraciones al que hemos
  #llegado.
  list(w, iter)
}
```

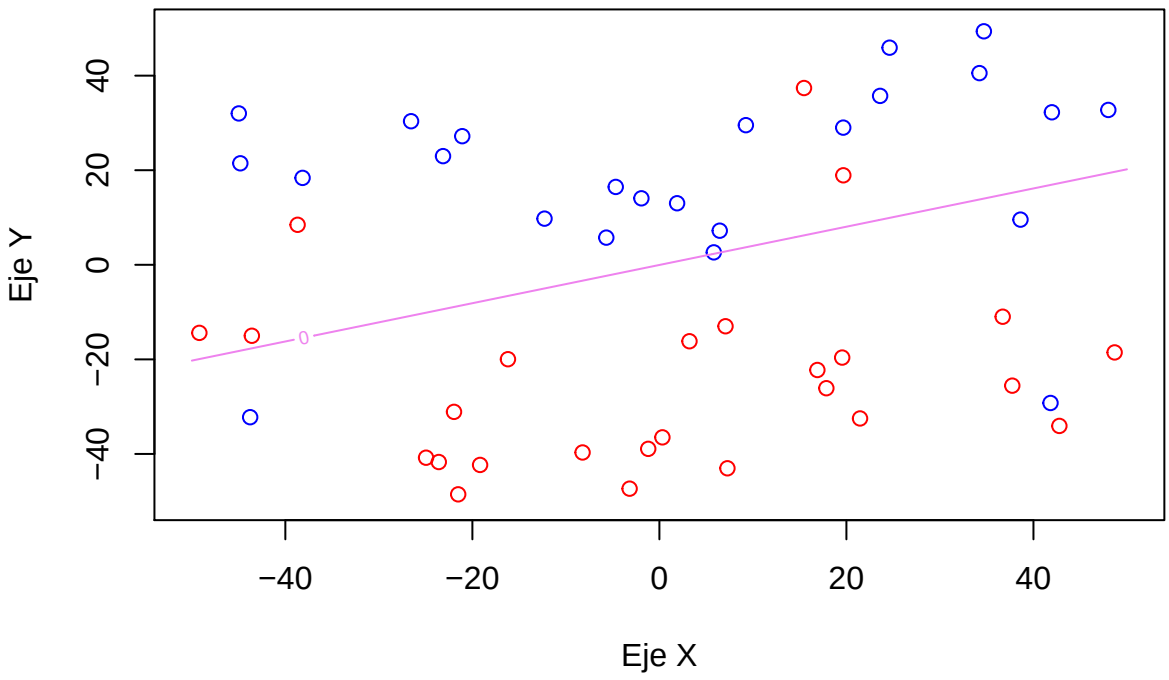
```
#Llamamos a la función
ajusta_PLA_sol(datos, etiquetas2, 10, c(0,0,0))
```



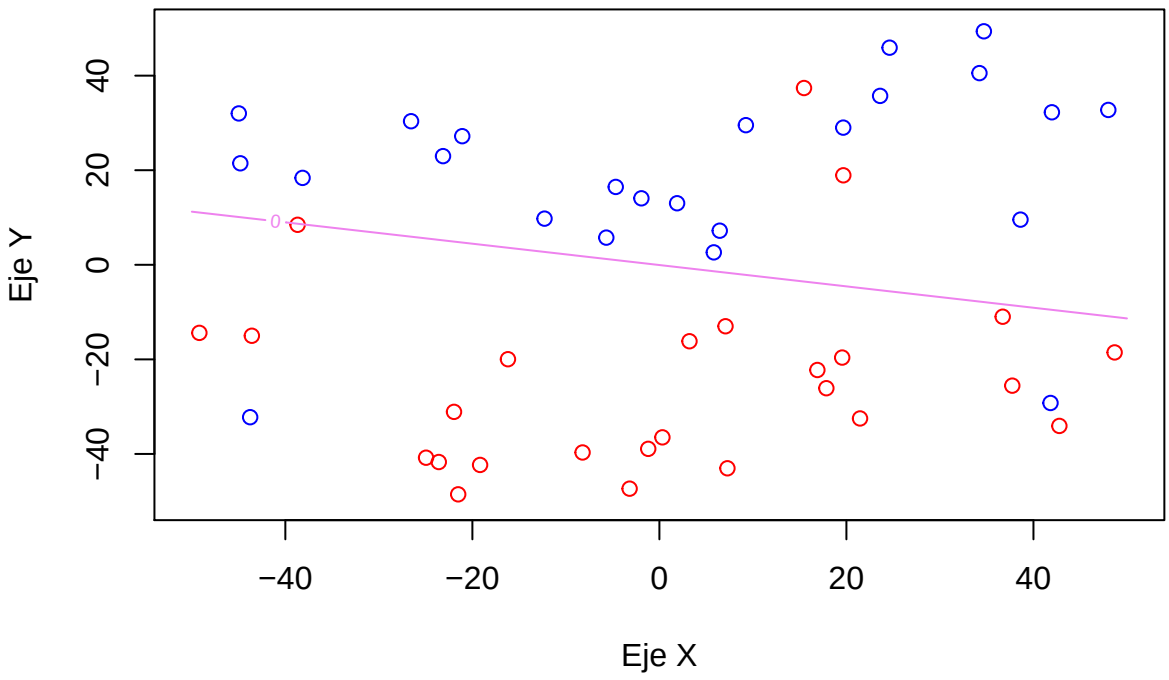

```
## Presione una tecla para continuar
```



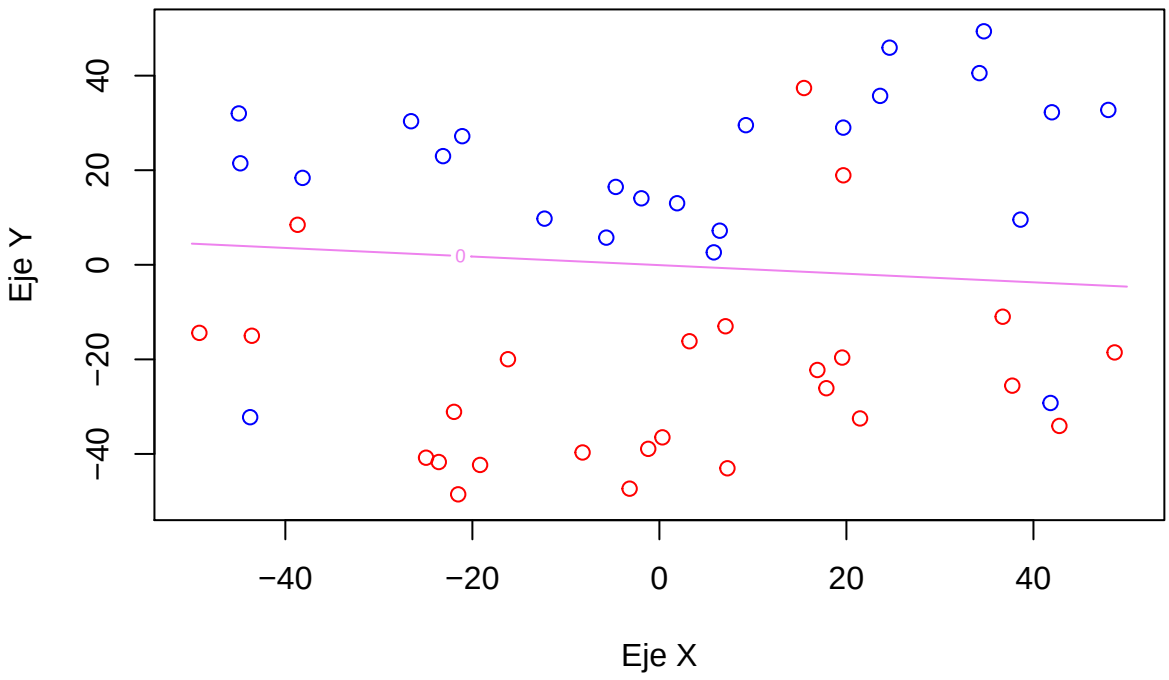

```
## Presione una tecla para continuar
```



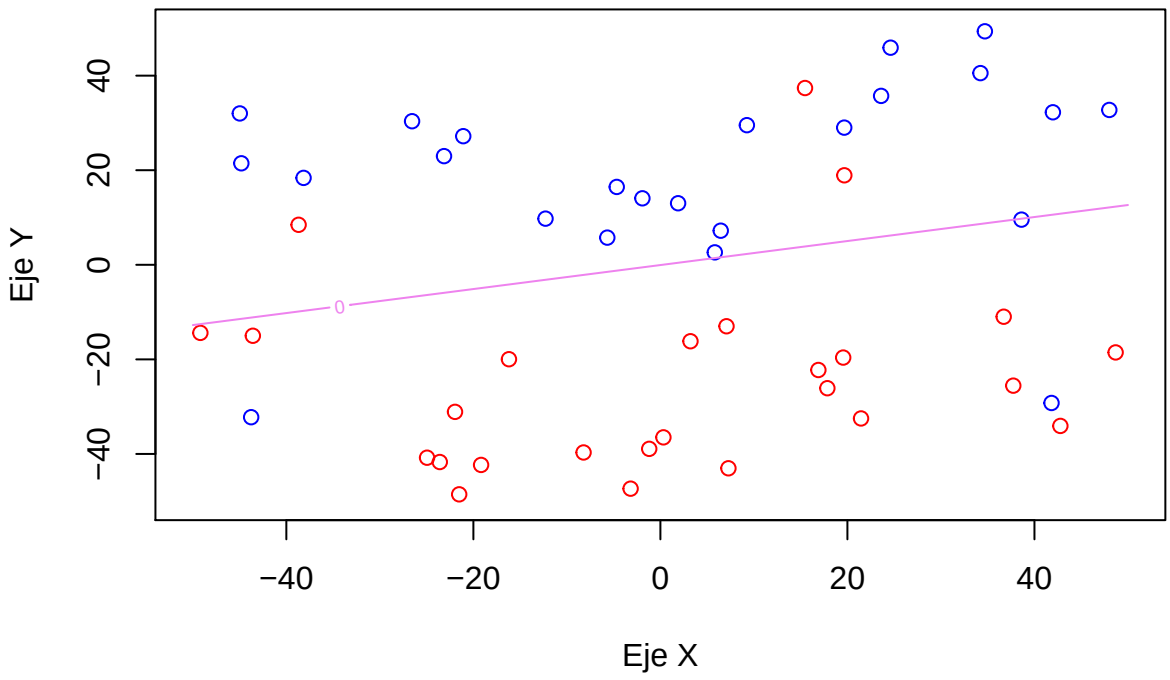

```
## Presione una tecla para continuar
```



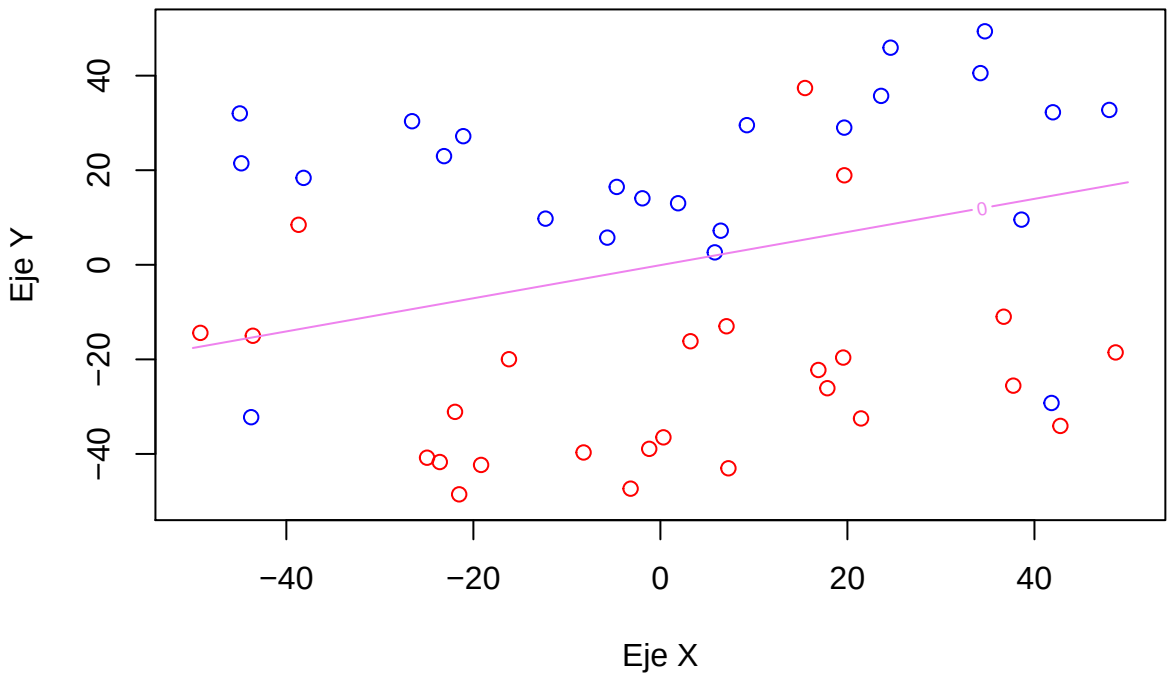

```
## Presione una tecla para continuar
```



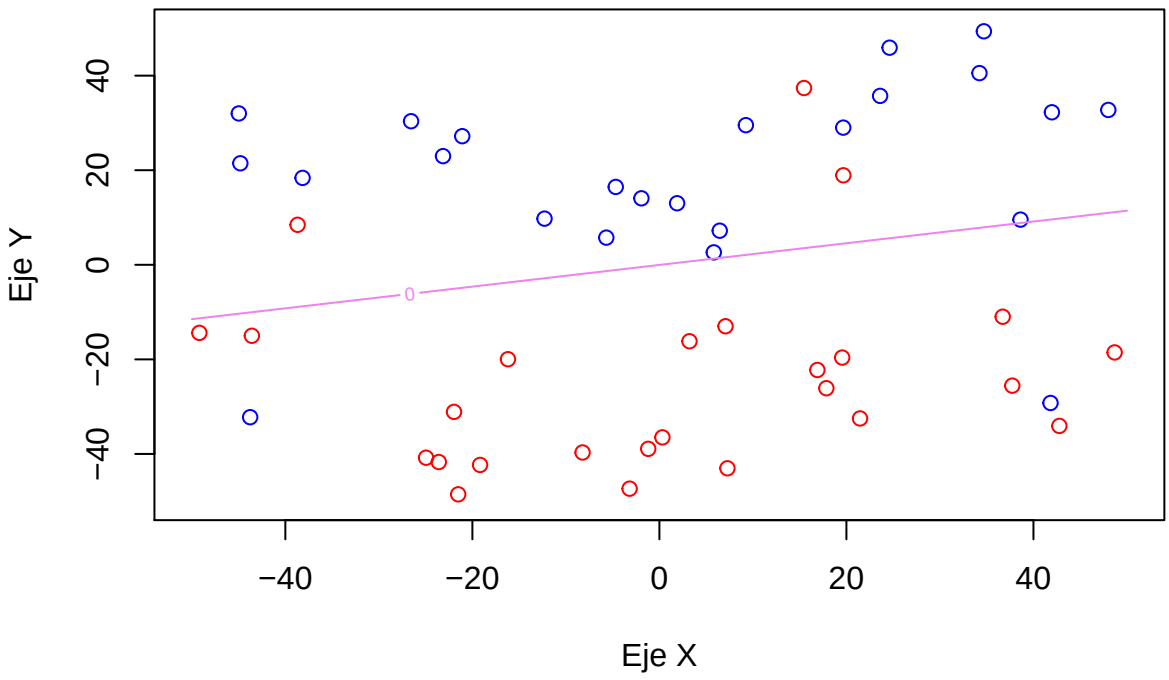

```
## Presione una tecla para continuar
```



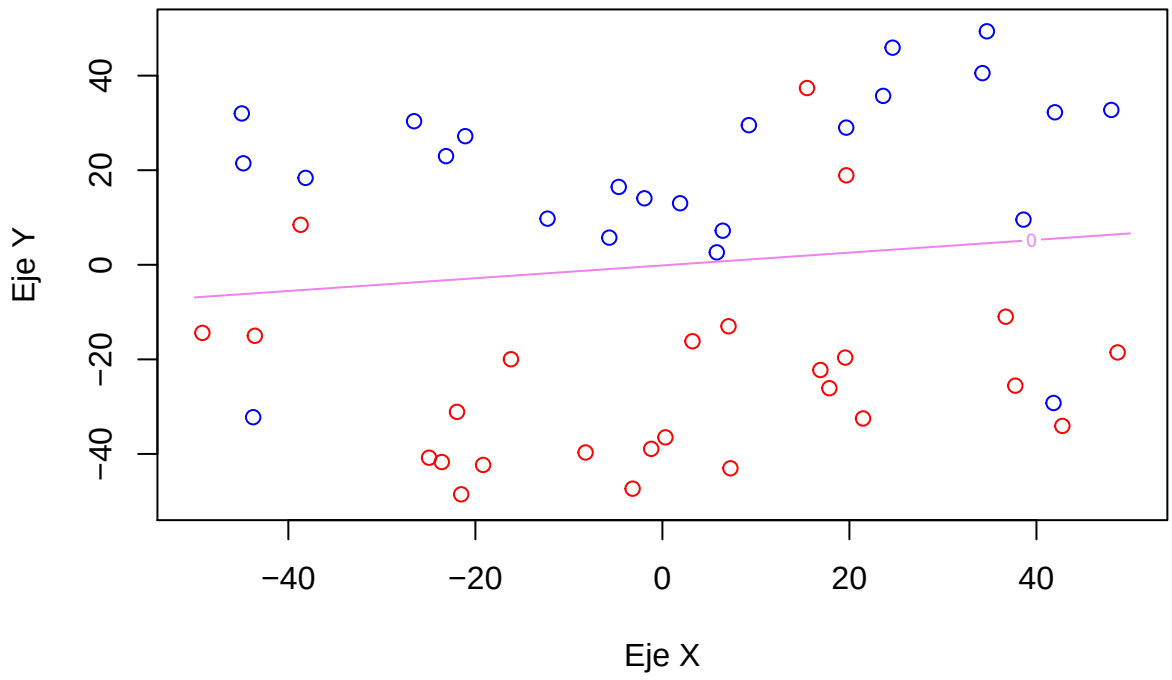

```
## Presione una tecla para continuar
```




```
## Presione una tecla para continuar
```




```
## Presione una tecla para continuar
```




```
## Presione una tecla para continuar
```

```
## [[1]]
## [1] -4.112858 30.409589  4.000000
##
## [[2]]
## [1] 10
```

??se supone que hay que hacer esto con 100 y con 1000????

6. A la vista de la conducta de las soluciones observadas en el apartado anterior, proponga e implemente una modificación de la función original `sol = ajusta_PLA_MOD(...)` que permita obtener soluciones razonables sobre datos no linealmente separables. Mostrar y valorar el resultado encontrado usando los datos del apartado 7 del ejercicio 1.

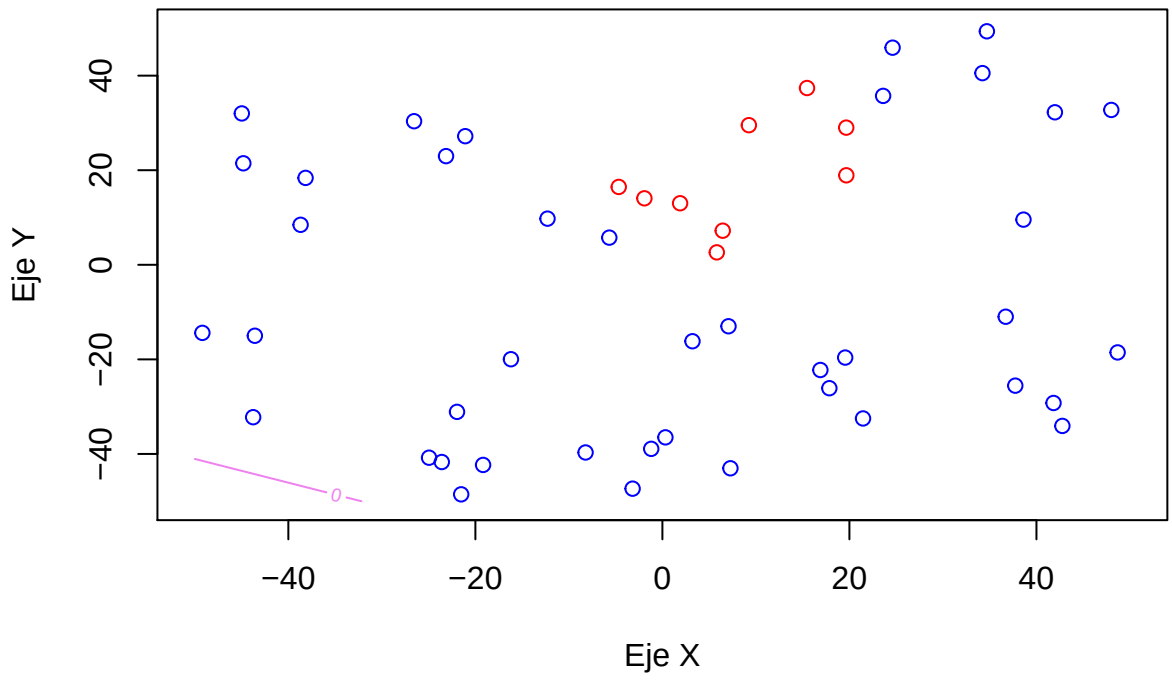
Lo que vamos a hacer es ir contando los errores que hay al etiquetar con la solución que va devolviendo el algoritmo en cada iteración y quedarnos con aquella que tenga menos errores.

```
ajusta_PLA_MOD <- function(datos, label, max_iter, vini) {
  parada <- F
  fin <- F
  w <- vini
```

```
  wmejor <- w
  iter <- 1
  errores_mejor <- nrow(datos)

  #Mientras no hayamos superado el máximo de iteraciones o
  #no se haya encontrado solución
  while(!parada) {
    #iteramos sobre los datos
    for (j in 1:nrow(datos)) {
      if (sign(crossprod(w, datos[j,])) != label[j]) {
        w <- w + label[j]*datos[j,]
        #La variable fin controla si se ha entrado en el if
        fin <- F
      }
    }

    #Contamos el número de errores que hay en la solución actual y si
    #es menor que el número de errores en la mejor solución de las que
    #llevamos, nos quedamos con la actual
    l <- list(w, 1)
    errores_actual <- cuenta_errores(l, label)
    if(errores_actual < errores_mejor) {
      wmejor <- w
      errores_mejor <- errores_actual
    }

    #Si no se ha entrado en el if, todos los datos estaban bien
    #clasificados y podemos poner a TRUE la variable parada.
    if(fin == T) {
      parada = T
    }
    else {
      fin = T
    }
    iter <- iter + 1
    if (iter >= max_iter) parada = T
  }
  #Devolvemos el hiperplano, el número máximo de iteraciones al que hemos
  #llegado y el número de errores de la mejor solución que hemos encontrado
  list(wmejor, iter, errores_mejor)
}
```

Vamos a probarlo ahora con las funciones del ejercicio 7 (es decir, con las etiquetas que tenemos almacenadas en etiquetasFA, etiquetasFB, etiquetasFC y etiquetasFD)

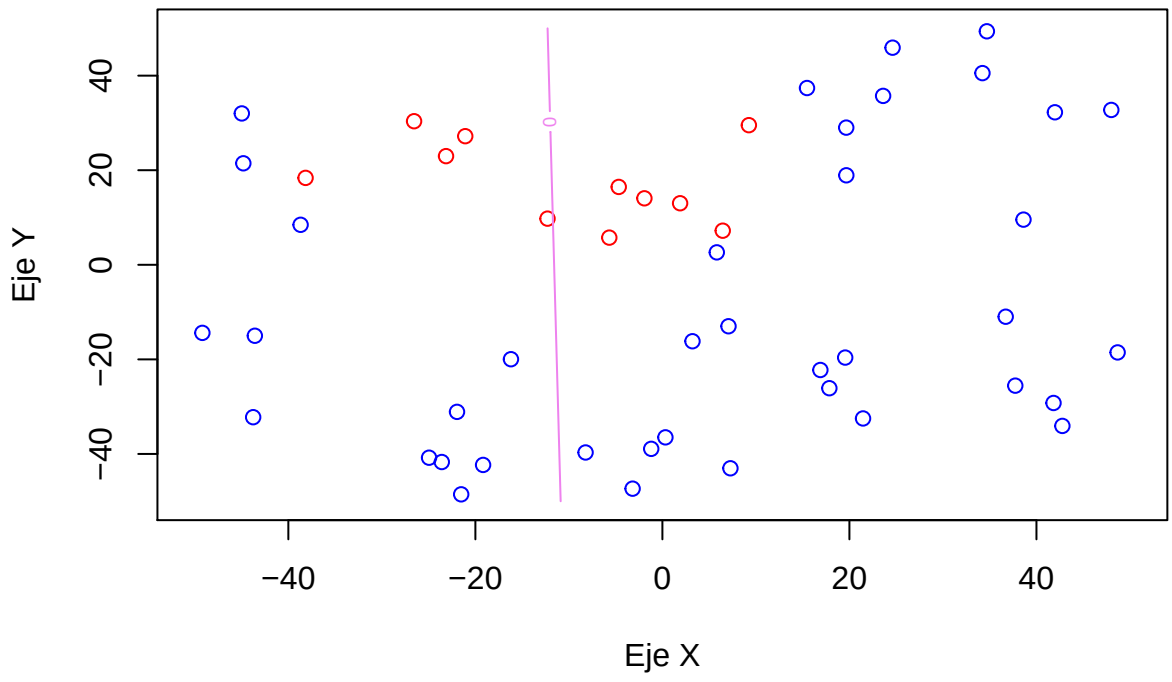
```
sol1_MOD <- ajusta_PLA_MOD(datos, etiquetasFA, 1000, c(0,0,0))
w1 <- sol1_MOD[[1]]
errores <- sol1_MOD[[3]]
w1 <- -w1 / w1[2]
pinta_particion(lista, etiquetasFA, TRUE, function(x,y) -w1[1]*x + y - w1[3])
```




```
errores
```

```
## [1] 9
```

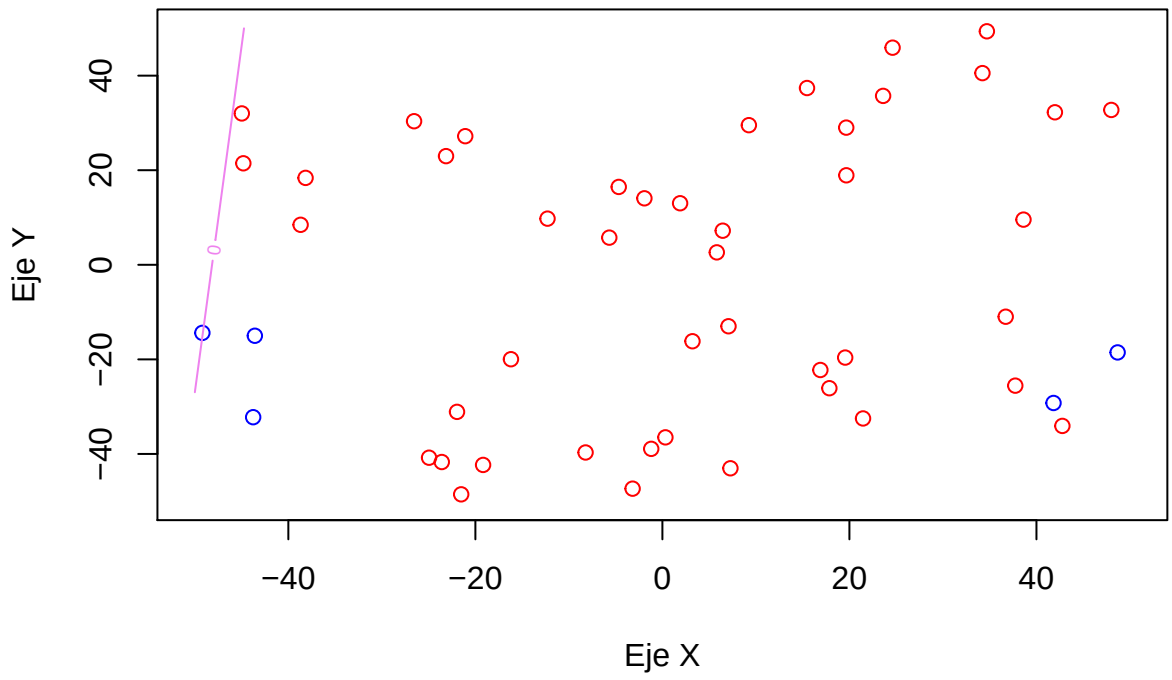
```
sol2_MOD <- ajusta_PLA_MOD(datos, etiquetasFB, 1000, c(0,0,0))
w2 <- sol2_MOD[[1]]
errores <- sol2_MOD[[3]]
w2 <- -w2 / w2[2]
pinta_particion(lista, etiquetasFB, TRUE, function(x,y) -w2[1]*x + y - w2[3])
```




```
errores
```

```
## [1] 18
```

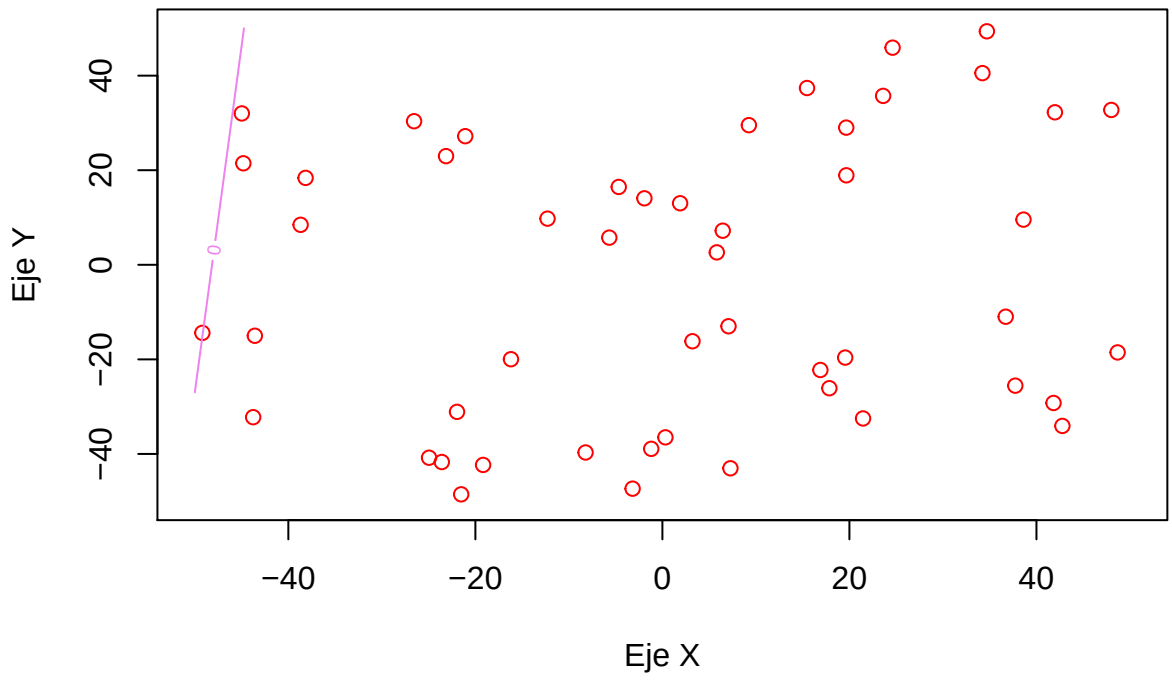
```
sol3_MOD <- ajusta_PLA_MOD(datos, etiquetasFC, 1000, c(0,0,0))
w3 <- sol3_MOD[[1]]
errores <- sol3_MOD[[3]]
w3 <- -w3 / w3[2]
pinta_particion(lista, etiquetasFC, TRUE, function(x,y) -w3[1]*x + y - w3[3])
```




```
errores
```

```
## [1] 4
```

```
sol4_MOD <- ajusta_PLA_MOD(datos, etiquetasFD, 1000, c(0,0,0))
w4 <- sol4_MOD[[1]]
errores <- sol4_MOD[[3]]
w4 <- -w4 / w4[2]
pinta_particion(lista, etiquetasFD, TRUE, function(x,y) -w3[1]*x + y - w3[3])
```




```
errores
```

```
## [1] 0
```

3. Ejercicio sobre Regresión Lineal

1. Abra el fichero ZipDigits.info disponible en la web del curso y lea la descripsión de la representación numérica de la base de datos de números manuscrito que hay en el fichero ZipDigits.train

2. Lea el fichero ZipDigits.train dentro de su código y visualice las imágenes. Seleccione sólo las instancias de los números 1 y 5. Guárdelas como matrices de tamaño 16×16

3. Para cada matriz de números calcularemos su valor medio y su grado de simetría vertical. Para calcular la simetría calculamos la suma del valor absoluto de las diferencias en cada píxel entre la imagen original y la imagen que obtenemos invirtiendo el orden de las columnas. Finalmente le cambiamos el signo.

4. Representar en los ejes {X=Intensidad Promedio, Y=Simetría} las instancias seleccionadas de 1's y 5's.

5. Implementar la función `sol = Regress_Lin(datos, label)` que permita ajustar un modelo de regresión lineal (usar SVD). Los datos de entrada se interpretan igual que en clasificación.

6. Ajustar un modelo de regresión lineal a los datos de (Intensidad Promedio, Simetría) y pintar la solución junto con los datos. Valorar el resultado.

7. En este ejercicio exploramos cómo funciona la regresión lineal en problemas de clasificación. Para ello generamos datos usando el mismo procedimiento que en ejercicios anteriores. Suponemos $X = [-10, 10] \times [-10, 10]$ **y elegimos muestras aleatorias uniformes dentro de** X**. La función** f **en cada caso será una recta aleatoria que corta a** X **y que asigna etiqueta a cada punto con el valor de su signo. En cada ejecución generamos una nueva función** f**.**

a) Fijar el tamaño de muestra $N = 100$. Usar regresión lineal para encontrar g y evaluar E_{in}, (el porcentaje de puntos incorrectamente clasificados). Repetir el experimento 1000 veces y promediar los resultados. ¿Qué valor obtiene para E_{in}?

b) Fijar el tamaño de muestra $N = 100$. Usar regresión lineal para encontrar g y evaluar E_{out}. Para ello generar 1000 puntos nuevos y usarlos para estimar el error fuera de la muestra, E_{out} (porcentaje de puntos mal clasificados). De nuevo, ejecutar el experimento 1000 veces y tomar el promedio. ¿Qué valor obtiene de E_{out}? Valore los resultados.

c) Ahora fijamos $N = 10$, ajustamos regresión lineal y usamos el vector de pesos encontrado como un vector inicial de pesos para PLA. Ejecutar PLA hasta que converja a un vector de pesos final que separe completamente la muestra de entrenamiento. Anote el número de iteraciones y repita el experimento 1000 veces. ¿Cuál es el valor promedio de iteraciones que tarda PLA en converger? (En cada iteración de PLA elija un punto aleatorio del conjunto de mal clasificados). Valore los resultados.

8. En este ejercicio el uso de transformación no lineales. Consideremos la función objetivo $f(x_1, x_2) = sign(x_1^2 + x_2^2 - 0.6)$. Generar una muestra de entrenamiento de $N = 1000$ puntos a partir de $X = [-10, 10] \times [-10, 10]$ muestreando cada punto $x \in X$ uniformemente. Generar las salidas usando el signo de la función en los puntos muestreados. Generar ruido sobre las etiquetas cambiando el signo de las salidas a un 10% de puntos del conjunto aleatorio generado.

a) Ajustar regresión lineal, para estimar los pesos ω. Ejecutar el experimento 1000 veces y calcular el valor promedio del error de entrenamiento E_{in}. Valorar el resultado.

b) Ahora, consideremos $N = 1000$ datos de entrenamiento y el siguiente vector de variables: $(1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$. Ajustar de nuevo regresión lineal y calcular el nuevo vector de pesos $\widehat{\omega}$. Mostrar el resultado.

c) Repetir el experimento anterior 1000 veces calculando en cada ocasión el error fuera de la muestra. Para ello generar en cada ejecución 1000 puntos nuevos y valorar sobre ellos la función ajustada. Promediar los valores obtenidos. ¿Qué valor obtiene? Valorar el resultado.