Practica 1

Jose Antonio Ruiz Millan

20 de marzo de 2018

Ejercicio sobre la búsqueda iterativa de óptimos

Gradiente Descendente (7 puntos)

1. (1 punto) Implementar el algoritmo de gradiente descendente Voy a mostrar una implemetacion de las que he utilizado en el fichero ya que según los datos que necesitaba, he implementado varias funciones.

Este algoritmo permite minimizar una funcion dados unos valores (pesos) que se van actualizando con cada iteración de la siguiente manera: $w_{i+1} = w_i - \eta \bigtriangledown f(w_i)$ donde η es la tasa de aprendizaje y $\bigtriangledown f(w_i)$ es el valor de $f'(w_i)$

```
gradienteDescendenteej2 <- function(func, derv, wini, niter, paso, umbral){
pt1 <- wini
i <- 1
puntos=matrix(wini,nrow=2)

while(func(pt1[1,1],pt1[2,1]) > umbral && i < niter){
grdnt <- derv(pt1[1,1],pt1[2,1])
pt1[1,1] <- pt1[1,1]-paso*grdnt[1]
pt1[2,1] <- pt1[2,1]-paso*grdnt[2]
i <- i+1
puntos<-cbind(puntos,pt1)
}
puntos
}
```

2. (2 puntos) Considerar la función $E(u, v) = (u^3e^{(v-2)} - 4v^3e^{-u})^2$. Usar gradiente descendente para encontrar un mínimo de esta función, comenzando desde el punto $(u, v) = (1, 1)$ y usando una tasa de aprendizaje $\eta = 0,1$.

Para los siguiente apartado haremos uso de la funcion pintar_frontera que la defino aquí para poder verla.

```
pintar_frontera = function(f,puntos,rangox=c(-50,50),rangoy=c(-50,50)) {
  x=seq(rangox[1],rangox[2],length.out = 500)
  y=seq(rangoy[1],rangoy[2],length.out = 500)
  z = outer(x,y,FUN=f)
  if (dev.cur()==1) # no esta abierto el dispositivo lo abre con plot
    plot(1, type="n", xlim=rango, ylim=rango)
  contour(x,y,z, levels = 1:20, xlim =rangox, ylim=rangoy, xlab = "x", ylab = "y")
  points(t(puntos))
  lines(t(puntos))
  points(t(puntos)[1,1],t(puntos)[1,2],col="blue")
  points(t(puntos)[nrow(t(puntos)),1],t(puntos)[nrow(t(puntos)),2],col="red")
}
```

a) Calcular analíticamente y mostrar la expresión del gradiente de la función $E(u, v)$

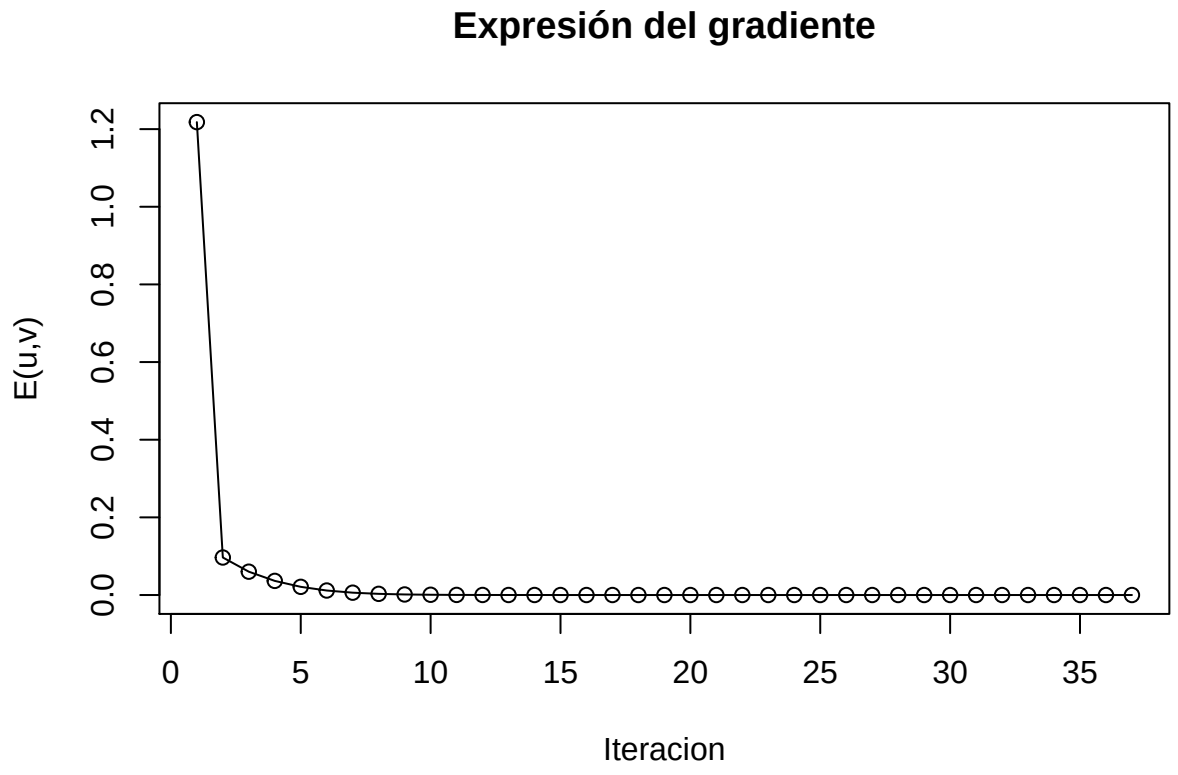
Partiendo de la función que tenemos en el enunciado, el primer paso que tenemos que realizar es calcular las derivadas respecto a u y v de la función. Utilizaré una aplicación matemática que calcula las derivadas,

obteniendo:

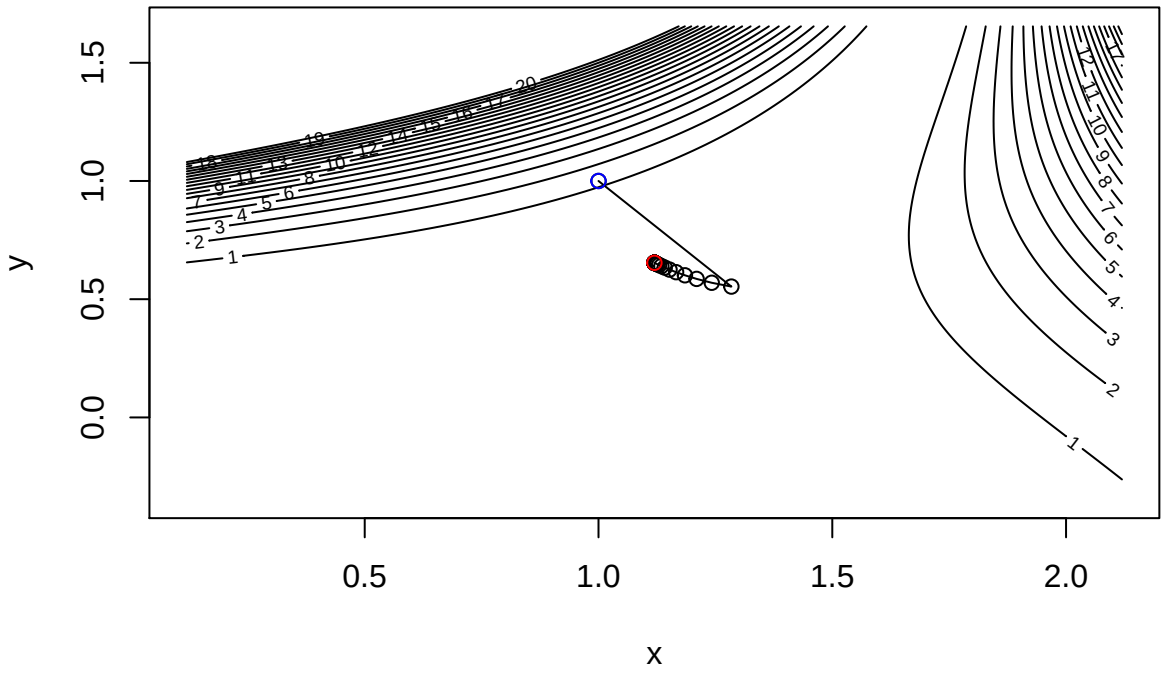
$\frac{\partial}{\partial u} = e^{-2(u+2)}(6u^5e^{2(u+v)} + 8(u-3)u^2v^3e^{u+v+2} - 32e^4v^6)$

$\frac{\partial}{\partial v} = e^{-2(u+2)}(2u^6e^{2(u+v)} - 8u^3v^2(v+3)e^{u+v+2} + 96e^4v^5)$

```
# E(u,v) un poco desarrollada para eliminar los cuadrados
funcion1 <- function(u,v){
  u^6*exp(2*v-4)-8*u^3*v^3*exp(-u+v-2)+16*exp(-2*u)*v^6
}

# E'(u,v), nos deveuelve tanto la derivada respecto a u como respecto a v
derivada1 <- function(u,v){
  c(exp(-2*(u+2))*(6*u^5*exp(2*(u+v))+8*(u-3)*u^2*v^3*exp(u+v+2)-32*exp(4)*v^6),
    exp(-2*(u+2))*(2*u^6*exp(2*(u+v))-8*u^3*v^2*(v+3)*exp(u+v+2)+96*exp(4)*v^5))
}

#Ahora vamos a obtener los valores del gradiente para poder dibujarlo
resultado2 <- gradienteDescendenteej2(funcion1,derivada1,as.matrix(c(1,1),nrow = 2),10e+10,as.double(0.

#Finalmente lo dibujamos, tanto como avanza el gradiente como una imagen del espacio y como a ido itera
plot(funcion1(resultado2[1,],resultado2[2,]),type="o", xlab="Iteracion", ylab="E(u,v)",main="Expresión
```




```
aum <- 1
pintar_frontera(funcion1,resultado2,c(resultado2[1,ncol(resultado2)]-aum,resultado2[1,ncol(resultado2)]
```



b) ¿Cuantas iteraciones tarda el algoritmo en obtener por primera vez un valor de $E(u, v)$ inferior a 10^{-14} . (Usar flotantes de 64 bits)

En un principio, el ejercicio pedía con un valor de $\eta = 0,1$ que aunque fué bastante costoso, finalmente consiguió encontrarlo en la iteracion 734656400

Por otra parte, cuando nos actualizaron el valor $\eta = 0,05$ conseguía llegar en tan solo 37 iteraciones

c) ¿En qué coordenadas (u, v) se alcanza por primera vez un valor igual o menor a 10^{-14} en el apartado anterior.

Para un valor $\eta = 0,1$ se alcanza en el punto [0.01326226,-0.00056566868] un valor $< 10^{-14}$

Por otro lado, con $\eta = 0,05$ se alcanzó en el punto [1.119544,0.653988] un valor $< 10^{-14}$

3. (2 puntos) Considerar ahora la función $f(x, y) = (x - 2)^2 + 2(y + 2)^2 + 2sin(2\pi x)sin(2\pi y)$

Para ello, definiremos la funcion y sus derivadas de la siguiente manera:

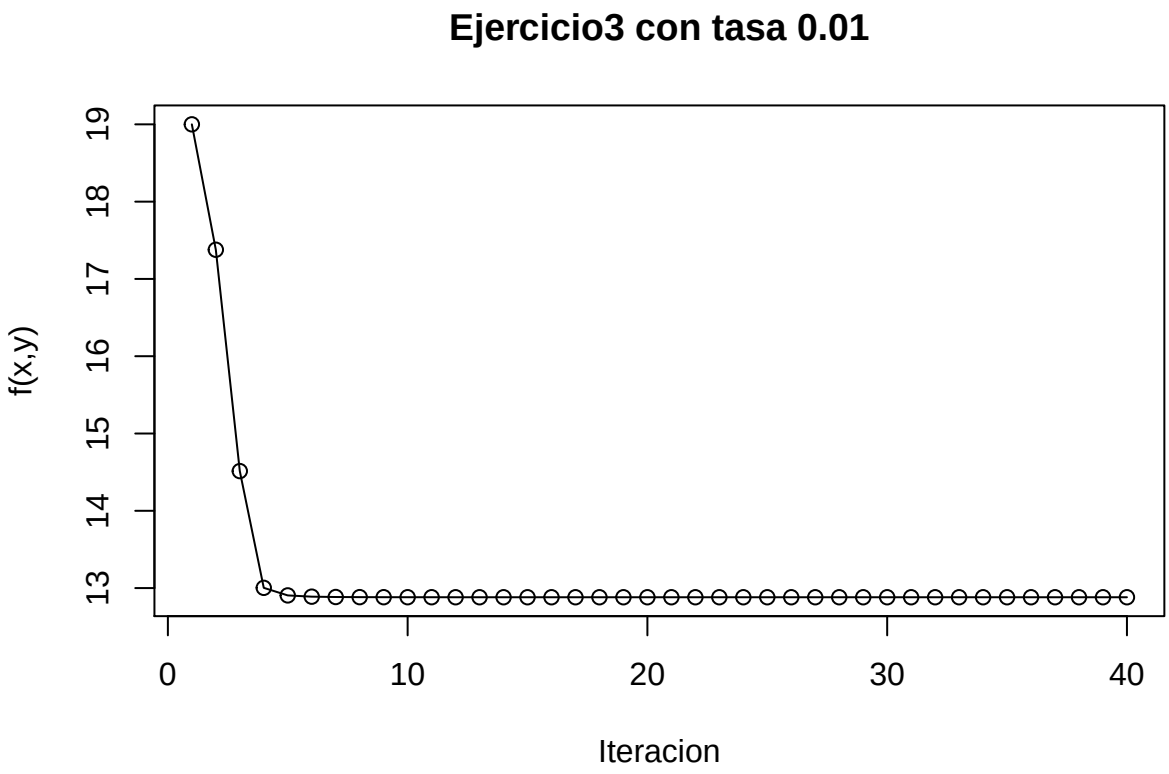
```
#Creamos la funcion algo mas extendida
funcion2 <- function(x,y){
  x^2+2*sin(2*pi*x)*sin(2*pi*y)-4*x+2*y^2+8*y+12
}

#Creamos la derivada que nos devuelve la derivada respecto a x y respecto a y
derivada2 <- function(x,y){
  c(2*(2*pi*cos(2*pi*x)*sin(2*pi*y)+x-2),4*(pi*sin(2*pi*x)*cos(2*pi*y)+y+2))
}
```

a) Usar gradiente descendente para minimizar esta función. Usar como punto inicial $(x_0 = 1, y_0 = 1)$, tasa de aprendizaje $\eta = 0,01$ y un máximo de 50 iteraciones. Generar un grafico de como desciende el valor

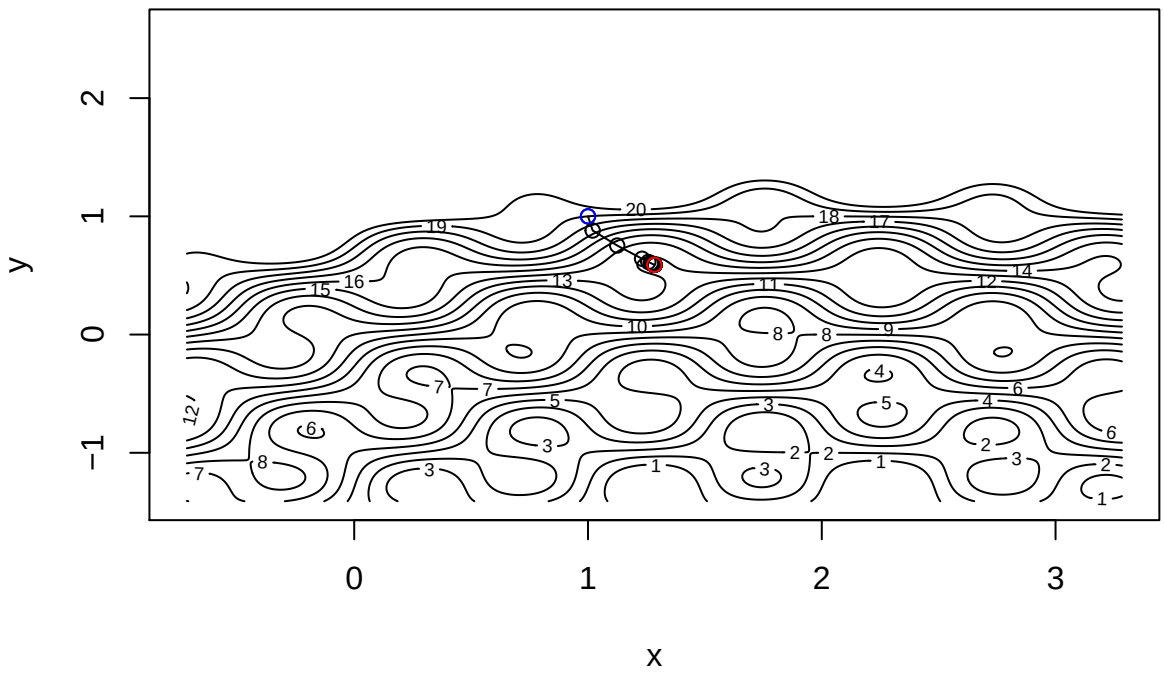
de la función con las iteraciones. Repetir el experimento pero usando $\eta = 0,1$, comentar las diferencias y su dependencia de η.

Para comenzar, como en este ejercicio necesitamos saber datos distintos al primero y la condicion de parada cambia, asique tambien tenemos que definir una nueva funcion *(Gradiente Descendente)* que la podemos ver a continuación.

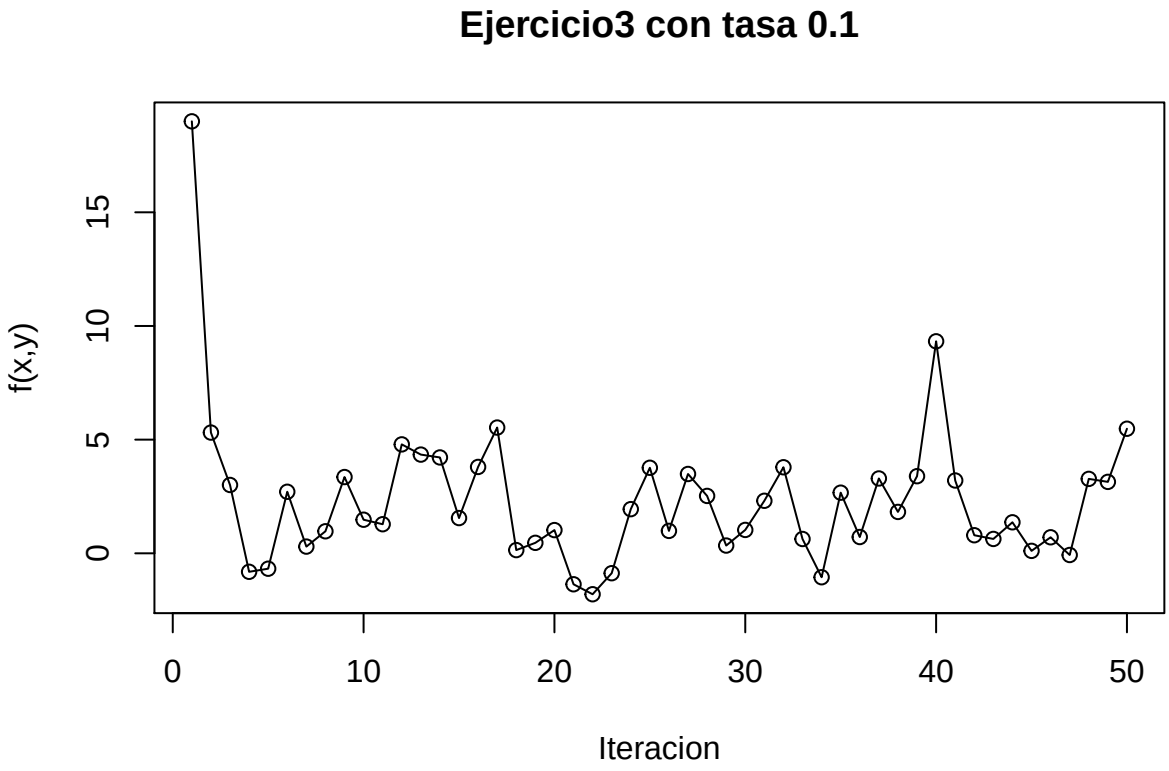
```
#Definimos una nueva funcion en la que lo único que cambia respecto a la anterior es la condicion de pa
gradienteDescendenteej3 <- function(func, derv, wini, niter, paso, umbral){

  #Guardamos el punto inicial
  pt1 <- wini

  #Creamos un nuevo punto que vamos a utilizar para almacenar el valor del punto anterior visitado
  pant <- as.matrix(c(pt1[1,1]+1,pt1[2,1]+1), nrow=2)
  i <- 1

  #Cremamos la matriz de puntos, donde vamos a guardar los respectivos puntos que visitamos
  puntos=matrix(wini,nrow=2)

  # Mientras la diferencia entre el valor del punto anterior y el actual sea mayor que el umbral
  # o las iteraciones no las hayamos alcanzado...
  while(abs(func(pt1[1,1],pt1[2,1]) - func(pant[1,1],pant[2,1])) > umbral && i < niter){
    #Le damos valor al punto anterior
    pant <- pt1

    #Calculamos el gradiente
    grdnt <- derv(pt1[1,1],pt1[2,1])

    #Actualizamos los pesos
    pt1[1,1] <- pt1[1,1]-paso*grdnt[1]
    pt1[2,1] <- pt1[2,1]-paso*grdnt[2]
    i <- i+1

    #Guardamos el punto nuevo con los puntos anteriores
    puntos<-cbind(puntos,pt1)
  }
  puntos
}

#Recogemos el resultado del Gradiente Descendente
resultado3a1 <- gradienteDescendenteej3(funcion2,derivada2,as.matrix(c(1,1),nrow=2),50,as.double(0.01),a

#Dibujamos el resultado
plot(funcion2(resultado3a1[1,],resultado3a1[2,]), main="Ejercicio3 con tasa 0.01", type="o", xlab="Itera
```



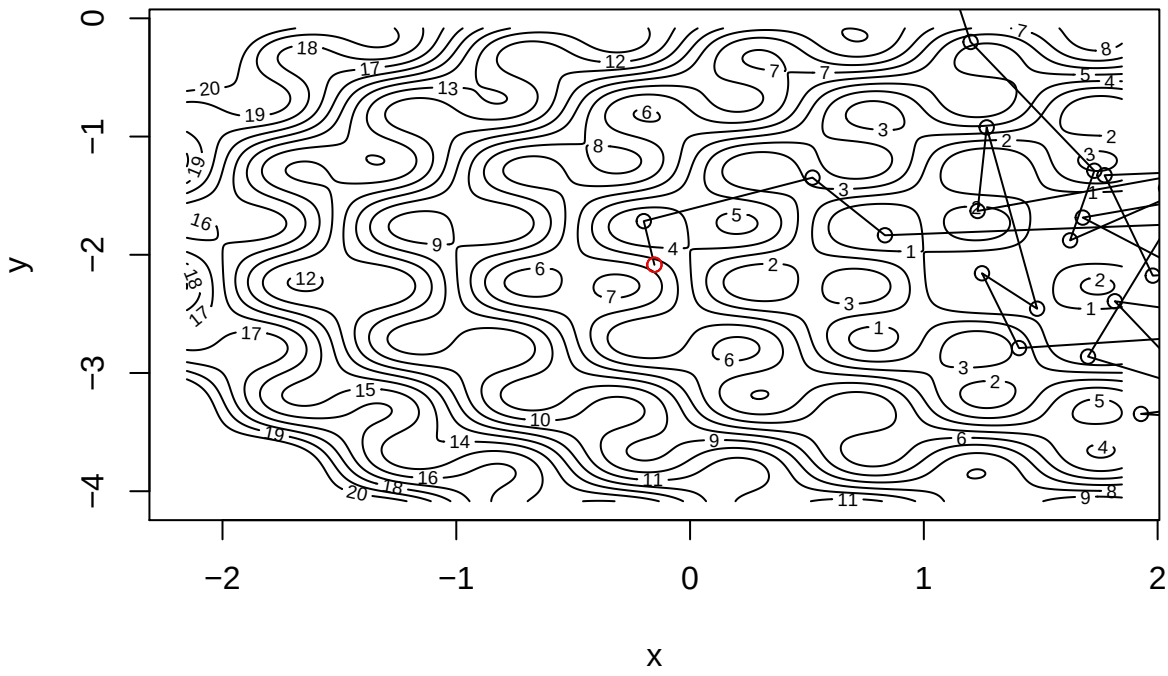

```
aum <- 2
pintar_frontera(funcion2,resultado3a1,c(resultado3a1[1,ncol(resultado3a1)]-aum,resultado3a1[1,ncol(resul
```




```
#Cambiamos ahora la tasa de apredizaje a 0.1
resultado3a2 <- gradienteDescendenteej3(funcion2,derivada2,as.matrix(c(1,1),nrow=2),50,as.double(0.1),as

#Volvemos a dibujarla al igual que antes.
plot(funcion2(resultado3a2[1,],resultado3a2[2,]), main="Ejercicio3 con tasa 0.1", type="o", xlab="Itera
```




```
aum <- 2
pintar_frontera(funcion2,resultado3a2,c(resultado3a2[1,ncol(resultado3a2)]-aum,resultado3a2[1,ncol(resul
```



Podemos ver por las distintas graficas, que con $\eta = 0,01$ la función converge a un mínimo local, sin embargo, cuando cambiamos la $\eta = 0,1$ vemos como la función diverge y va cambiando de un lugar del espacio a otro. Esto quiere decir que un valor grande de tasa de aprendizaje puede llevarnos a diverger y que al contrario, un valor demasiado pequeño de la tasa puede llevarnos a convergencia demasiado directa y no explorar correctamente

Como conclusión, podemos comprobar que calcular un valor óptimo para la tasa de aprendizaje no es tarea fácil ya que nuestra función puede converger o diverger dependiendo de este valor y encontrarnos con resultados que no son los esperados.

b) Obtener el valor mínimo y los valores de las variables (x, y) en donde se alcanzan cuando el punto de inicio se fija: $(2, 1, -2, 1)$, $(3, -3)$,$(1, 5, 1, 5)$,$(1, -1)$. Generar una tabla con los valores obtenidos.

Para este apartado, como nos piden sacar el punto minimo y la iteración donde se encuentra, defino una nueva funcion *Gradiente Descendente* que tambien se puede ver en el siguiente código.

Utilizaremos $\eta = 0,01$ y umbral $= 10^{-14}$.

```
#Funcion gradiente
gradienteDescendenteej3b <- function(func, derv, wini, niter, paso, umbral){
  pt1 <- wini
  pant <- as.matrix(c(pt1[1,1]+1,pt1[2,1]+1), nrow=2)
  i <- 1
  imin <- i
  pmin <- wini

  while(abs(func(pt1[1,1],pt1[2,1]) - func(pant[1,1],pant[2,1])) > umbral && i < niter){
    pant <- pt1
    grdnt <- derv(pt1[1,1],pt1[2,1])
```

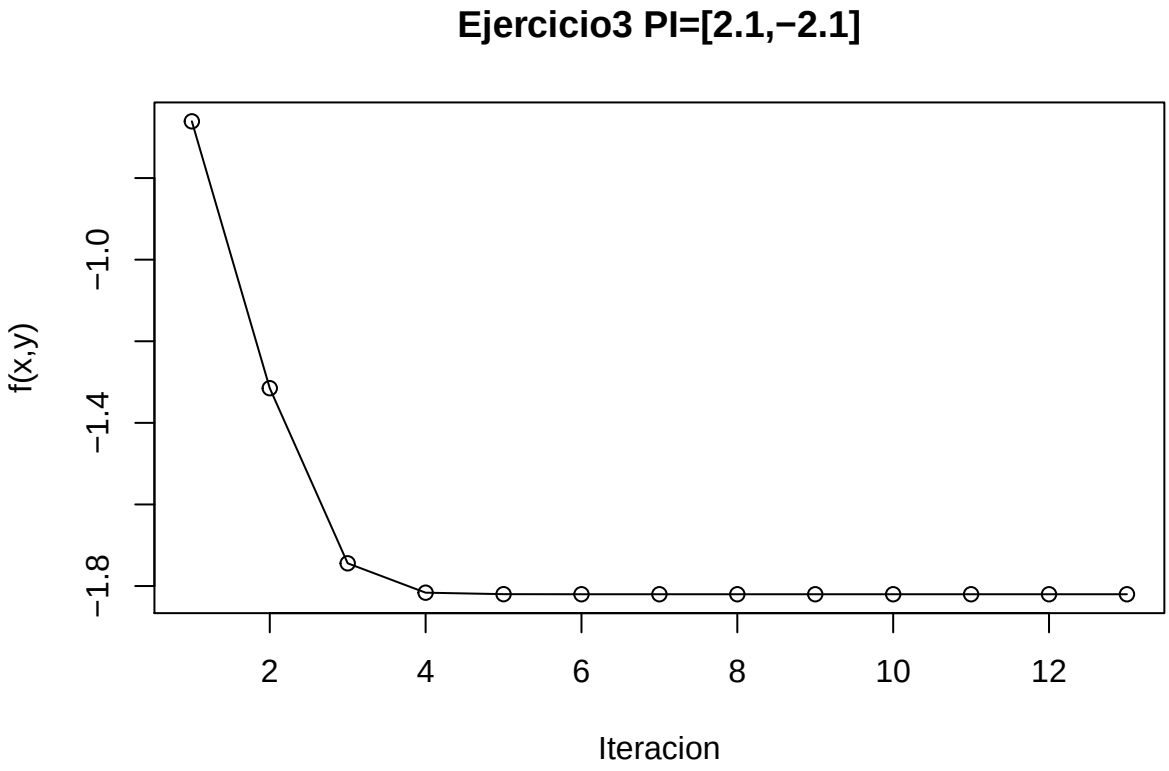
```
    pt1[1,1] <- pt1[1,1]-paso*grdnt[1]
    pt1[2,1] <- pt1[2,1]-paso*grdnt[2]
    i <- i+1
    #Si el valor obtenido es menor que el que tenemos como mínimo, nos quedamos con él
    if(func(pt1[1,1],pt1[2,1]) < func(pmin[1,1],pmin[2,1])){
      pmin <- pt1
      imin <- i
    }
  }
  c(pmin,imin)
}

#Pamos ahora a calcular lo que nos pide el ejercicio
#1 -> Punto de inicio [2.1,-2.1]
#Calculamos el punto y la iteración del mínimo encontrado
resultado3 <- gradienteDescendenteej3b(funcion2,derivada2,as.matrix(c(2.1,-2.1),nrow=2),1e+10,as.double
#Guardo el resultado para crear la tabla
resultados <- matrix(c(resultado3,funcion2(resultado3[1],resultado3[2])),nrow = 1)
#Mostramos los resultados
resultado3
```

```
## [1]  2.243805 -2.237926 13.000000
```

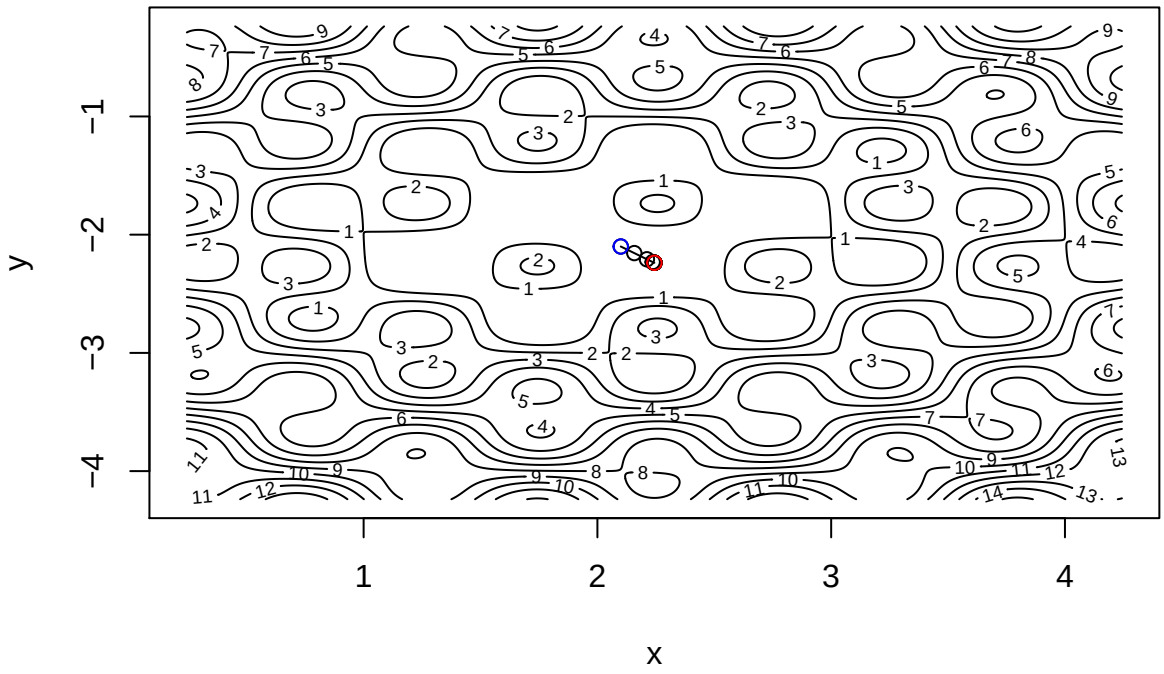
```
funcion2(resultado3[1],resultado3[2])
```

```
## [1] -1.820079
```

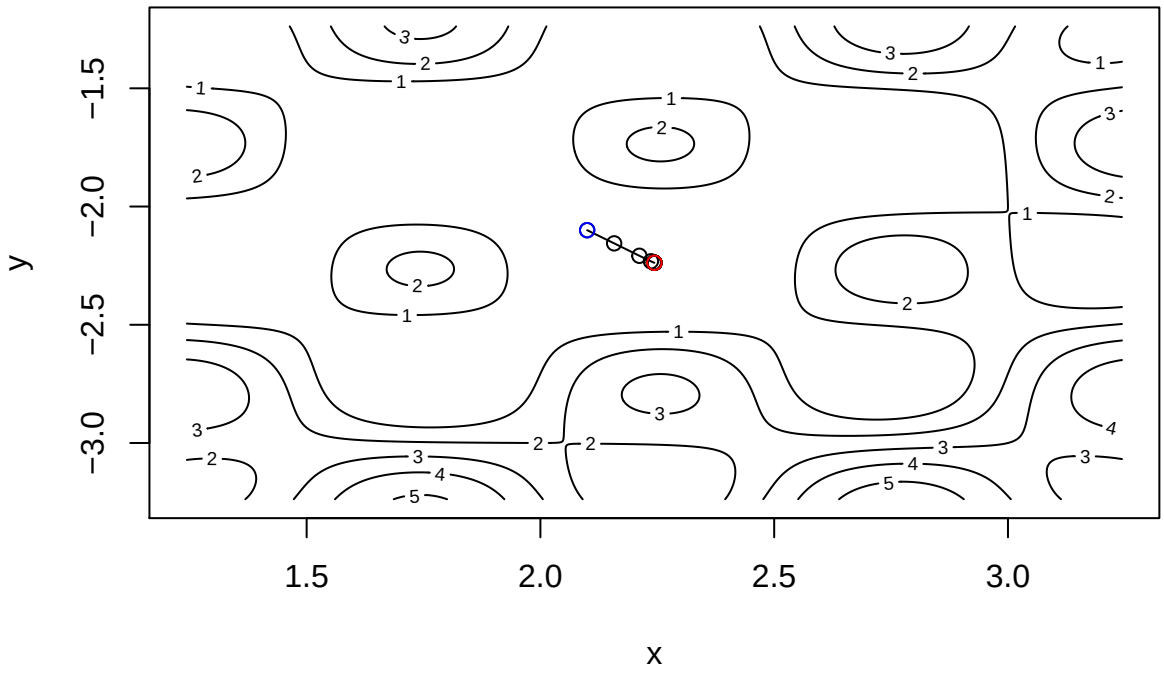
```
#Calculamos de nuevo pero ahora recojemos los dintintos puntos por los que ha pasado utilizando el grad
resultado3 <- gradienteDescendenteej3(funcion2,derivada2,as.matrix(c(2.1,-2.1),nrow=2),1e+10,as.double(
#Dibujamos tanto el valor de la funcion en cada punto como cómo la función se ha movido en el espacio
plot(funcion2(resultado3[1,],resultado3[2,]), main="Ejercicio3 PI=[2.1,-2.1]", type="o", xlab="Iteracio
```




```
aum <- 2
pintar_frontera(funcion2,resultado3,c(resultado3[1,ncol(resultado3)]-aum,resultado3[1,ncol(resultado3)]
```




```
aum <- 1
pintar_frontera(funcion2,resultado3,c(resultado3[1,ncol(resultado3)]-aum,resultado3[1,ncol(resultado3)]
```



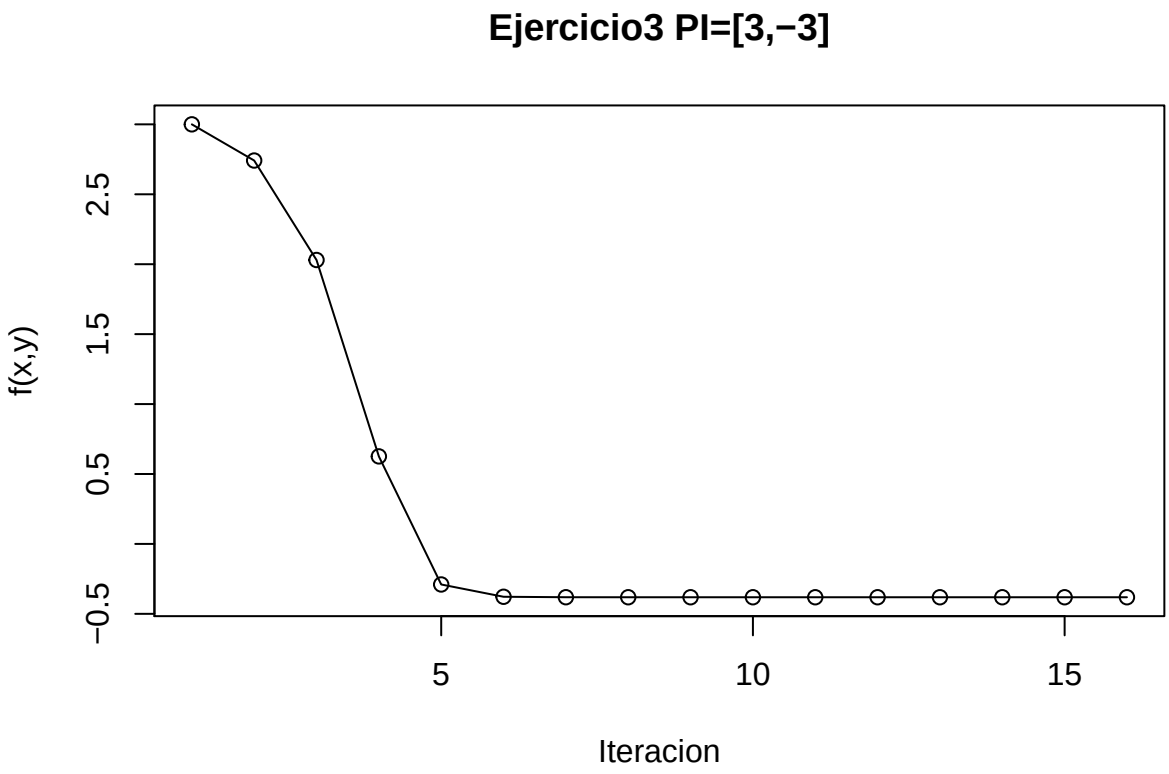

```
# Obtenemos en el punto [2.243805,-2.237926] el valor de -1.820079 con 13 iteraciones

#2 -> Punto de inicio [3,-3]
resultado3 <- gradienteDescendenteej3b(funcion2,derivada2,as.matrix(c(3,-3),nrow=2),1e+10,as.double(0.0
resultados <- rbind(resultados,c(resultado3,funcion2(resultado3[1],resultado3[2])))
resultado3
```

```
## [1]  2.730936 -2.713279 16.000000
```

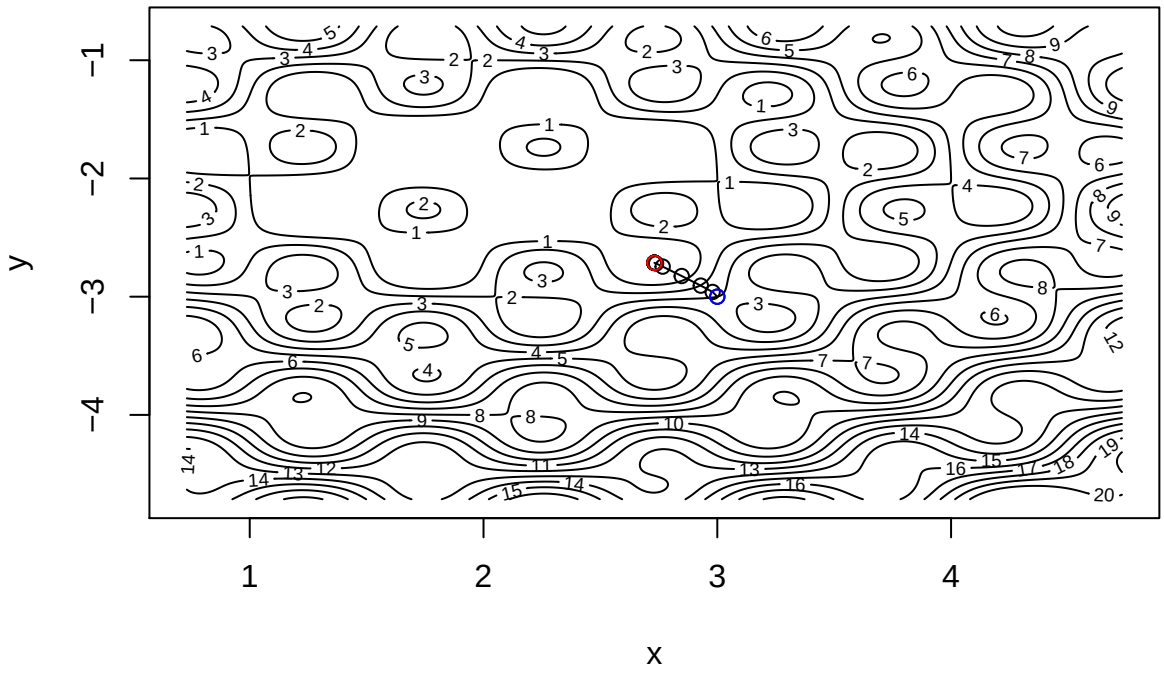
```
funcion2(resultado3[1],resultado3[2])
```

```
## [1] -0.3812495
```

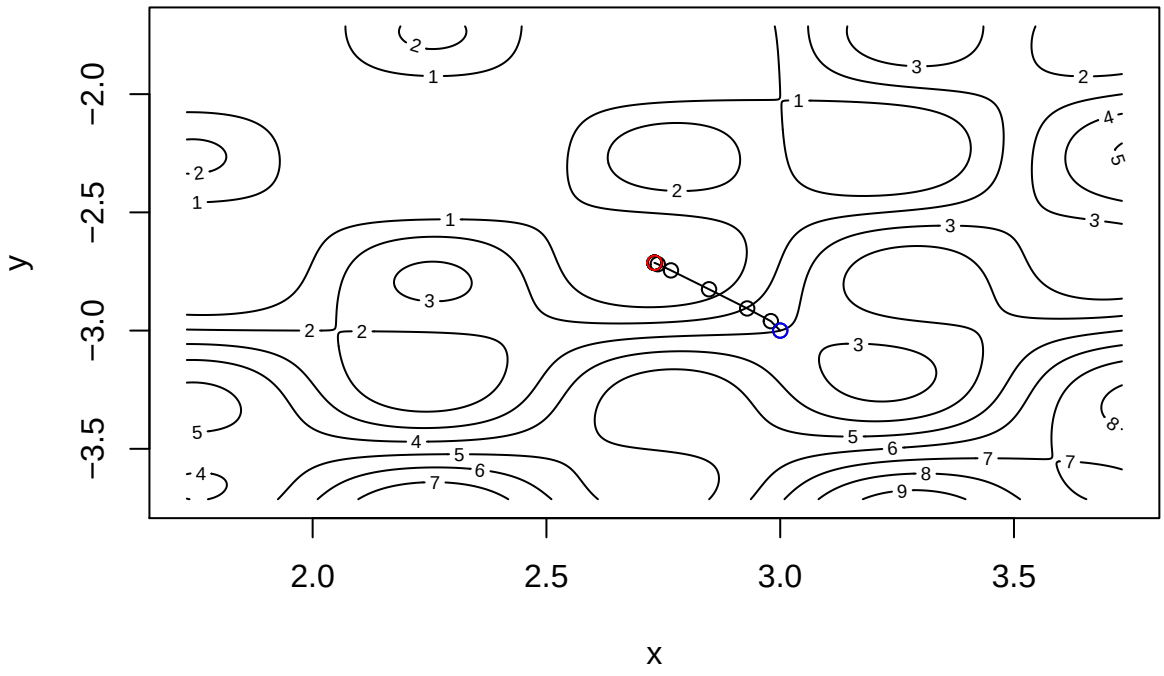
```
resultado3 <- gradienteDescendenteej3(funcion2,derivada2,as.matrix(c(3,-3),nrow=2),1e+10,as.double(0.01
plot(funcion2(resultado3[1,],resultado3[2,]), main="Ejercicio3 PI=[3,-3]", type="o", xlab="Iteracion", y
```




```
aum <- 2
pintar_frontera(funcion2,resultado3,c(resultado3[1,ncol(resultado3)]-aum,resultado3[1,ncol(resultado3)]
```




```
aum <- 1
pintar_frontera(funcion2,resultado3,c(resultado3[1,ncol(resultado3)]-aum,resultado3[1,ncol(resultado3)]
```



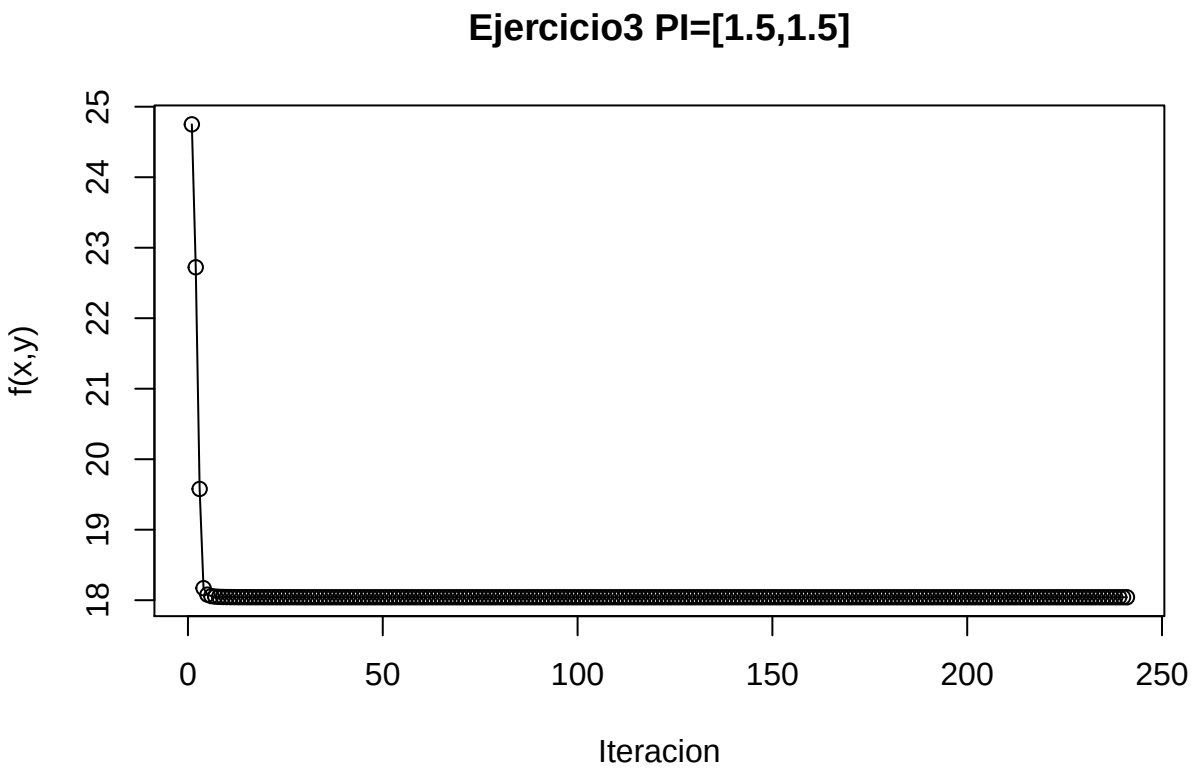

```
# Obtenemos en el punto [2.730936,-2.713279] el valor de -0.3812495 con 16 iteraciones

#3 -> Punto de inicio [1.5,1.5]
resultado3 <- gradienteDescendenteej3b(funcion2,derivada2,as.matrix(c(1.5,1.5),nrow=2),1e+10,as.double(0
resultados <- rbind(resultados,c(resultado3,funcion2(resultado3[1],resultado3[2])))
resultado3
```

```
## [1]   1.779119   1.030946 241.000000
```

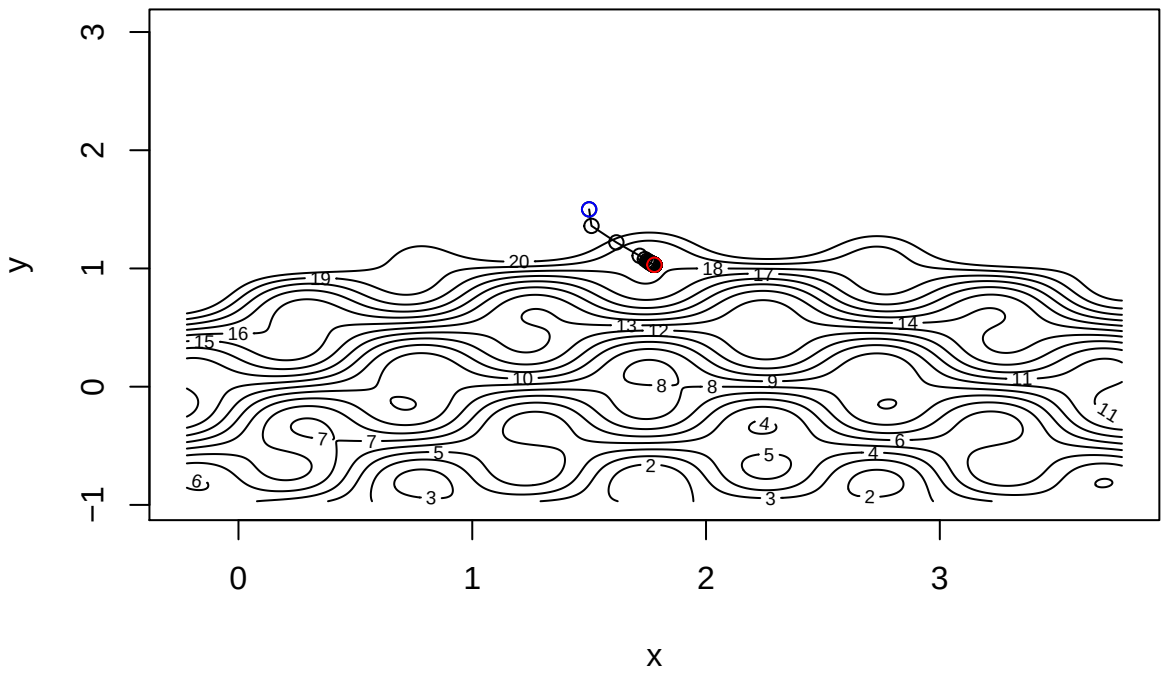
```
funcion2(resultado3[1],resultado3[2])
```

```
## [1] 18.04207
```

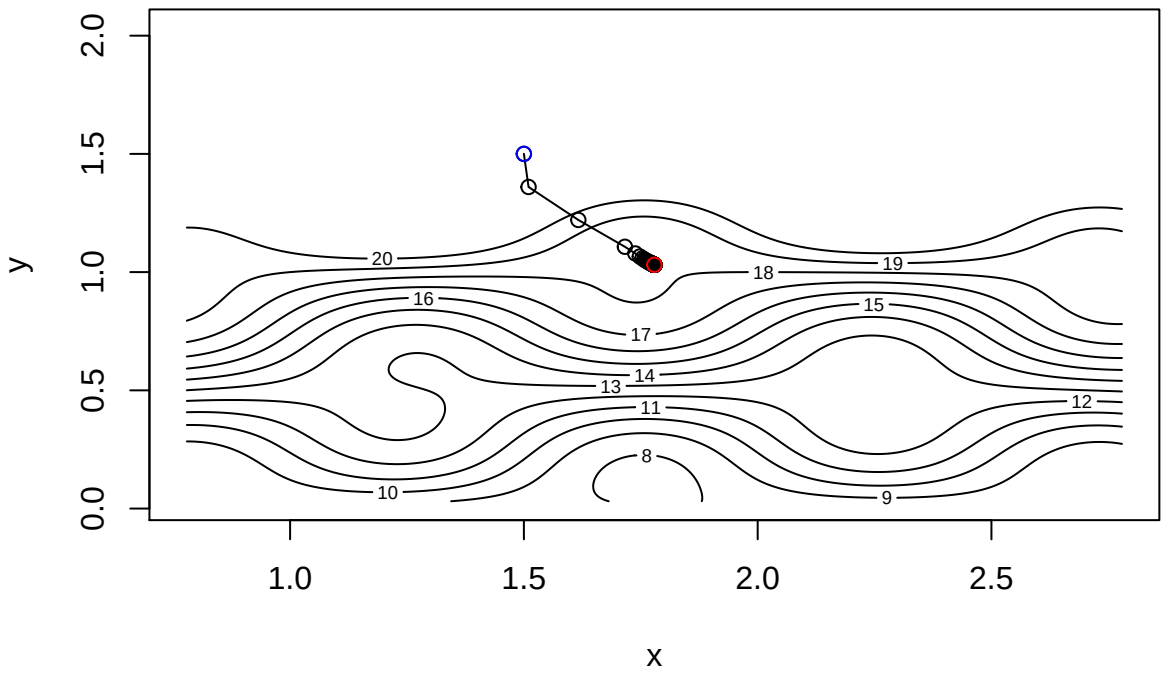
```
resultado3 <- gradienteDescendenteej3(funcion2,derivada2,as.matrix(c(1.5,1.5),nrow=2),1e+10,as.double(0
plot(funcion2(resultado3[1,],resultado3[2,]), main="Ejercicio3 PI=[1.5,1.5]", type="o", xlab="Iteracion
```




```
aum <- 2
pintar_frontera(funcion2,resultado3,c(resultado3[1,ncol(resultado3)]-aum,resultado3[1,ncol(resultado3)]
```




```
aum <- 1
pintar_frontera(funcion2,resultado3,c(resultado3[1,ncol(resultado3)]-aum,resultado3[1,ncol(resultado3)]
```



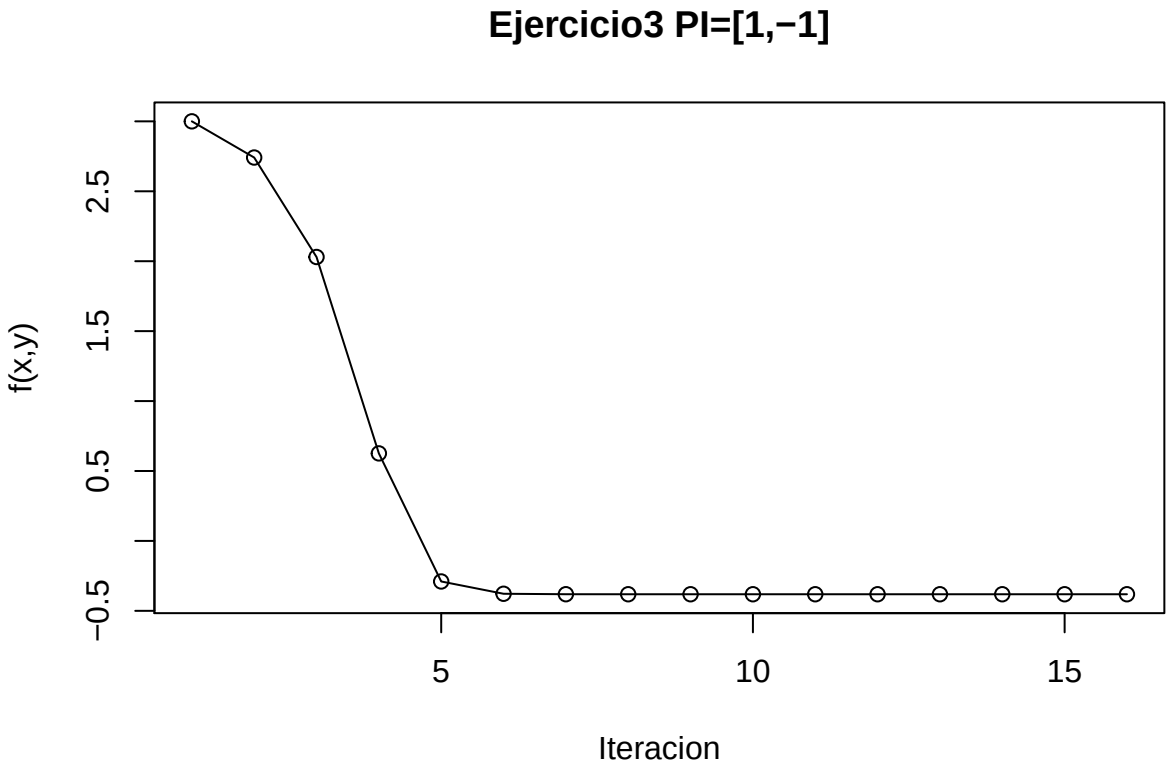

```
# Obtenemos en el punto [1.779119,1.030946] el valor de 18.04207 con 241 iteraciones

#4 -> Punto de inicio [1,-1]
resultado3 <- gradienteDescendenteej3b(funcion2,derivada2,as.matrix(c(1,-1),nrow=2),1e+10,as.double(0.0
resultados <- rbind(resultados,c(resultado3,funcion2(resultado3[1],resultado3[2])))
resultado3
```

```
## [1]  1.269064 -1.286721 16.000000
```

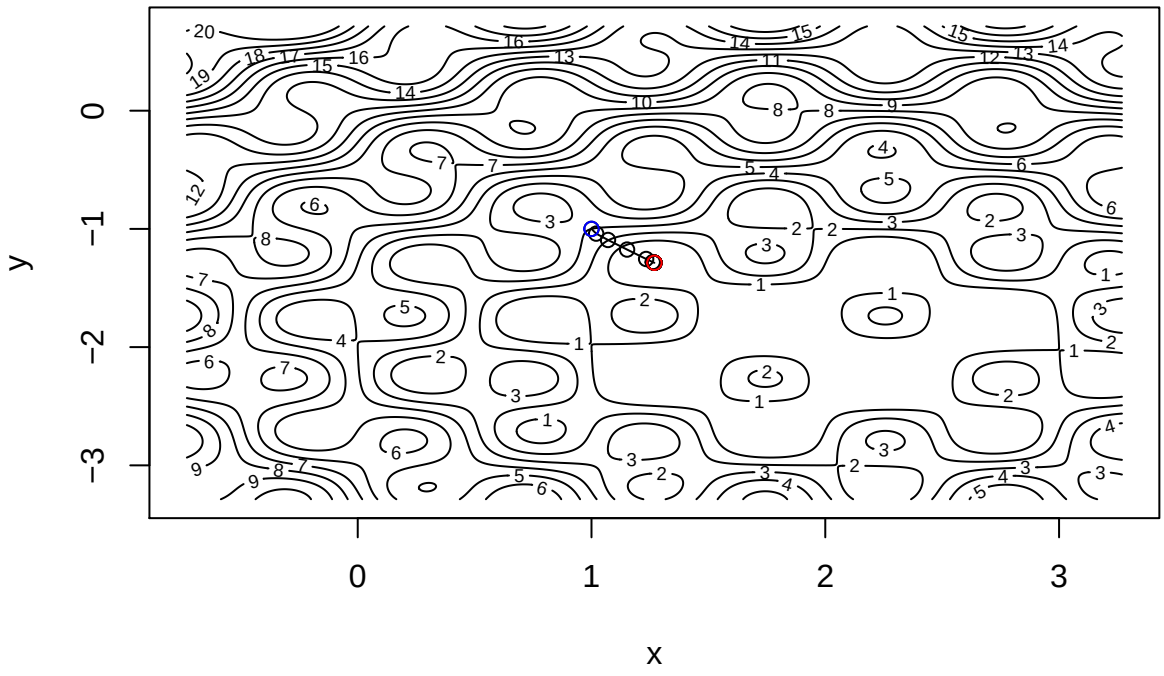
```
funcion2(resultado3[1],resultado3[2])
```

```
## [1] -0.3812495
```

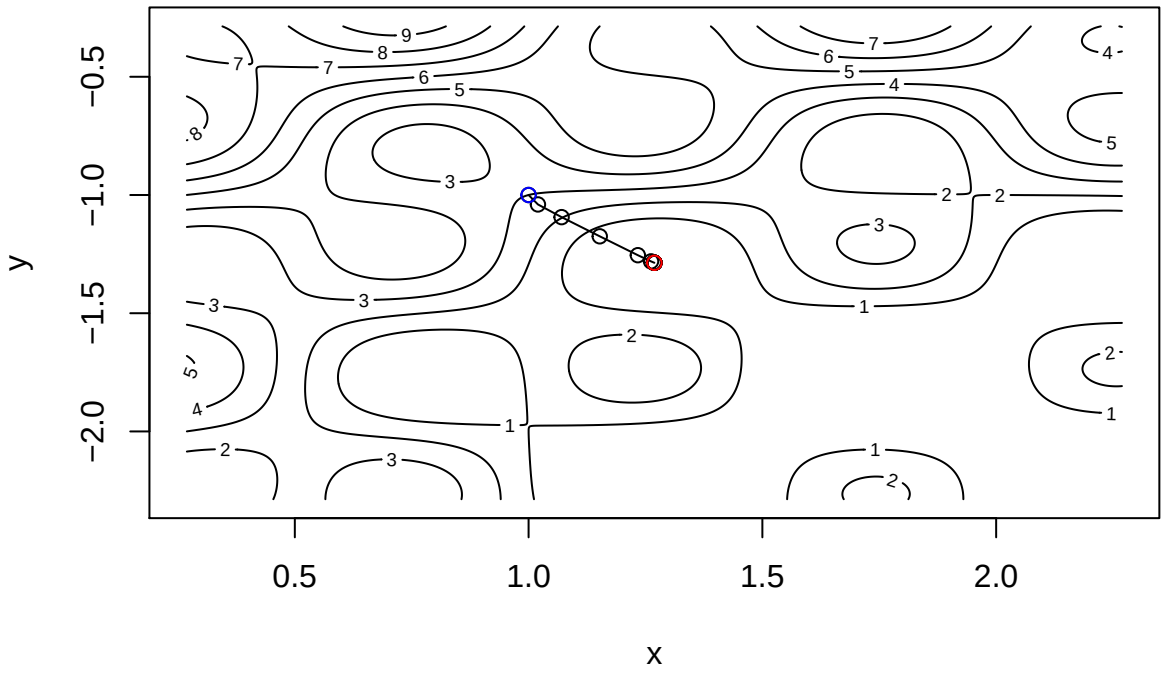
```
resultado3 <- gradienteDescendenteej3(funcion2,derivada2,as.matrix(c(1,-1),nrow=2),1e+10,as.double(0.01
plot(funcion2(resultado3[1,],resultado3[2,]), main="Ejercicio3 PI=[1,-1]", type="o", xlab="Iteracion", y
```




```
aum <- 2
pintar_frontera(funcion2,resultado3,c(resultado3[1,ncol(resultado3)]-aum,resultado3[1,ncol(resultado3)]
```




```
aum <- 1
pintar_frontera(funcion2,resultado3,c(resultado3[1,ncol(resultado3)]-aum,resultado3[1,ncol(resultado3)]
```




```
# Obtenemos en el punto [1.269064,-1.286721] el valor de -0.3812495 con 16 iteraciones
```

Por último vamos a realizar la tabla comparativa entre las distintas ejecuciones que hemos obtenido

	x	y	Iteración	Valor mínimo
[2.1,-2.1]	2.243805	-2.237926	13	-1.8200785
[3,-3]	2.730936	-2.713279	16	-0.3812495
[1.5,1.5]	1.779119	1.030946	241	18.0420723
[1,-1]	1.269064	-1.286721	16	-0.3812495

4. (2 punto) ¿Cuál sería su conclusión sobre la verdadera dificultad de encontrar el mínimo global de una función arbitraria?

Podemos concluir como resumen que calcular el mínimo de una funcion utilizando gradiente descendente tiene un costo más alto de lo que parece, ya que para obtener resultados buenos debemos de ajustar bastante bien tanto la tasa de aprendizaje como el punto de inicio para el comienzo que no es fácil hacer una estimación correcta del valor de esto parametros.

Ejercicio sobre regresión lineal

1. (2.5 puntos) Estimar un modelo de regresión lineal a partir de los datos proporcionados de dichos números (Intensidad promedio, Simetria) usando tanto el algoritmo de la pseudoinversa como Gradiente descendente estocástico (SGD). Las etiquetas serán {-1, 1}, una para cada vector de cada uno de los números. Pintar las soluciones obtenidas junto con los datos usados en el ajuste. Valorar la bondad del resultado usando Ein y Eout (para Eout calcular las

predicciones usando los datos del fichero de test). (usar Regress_Lin(datos, label) como llamada para la función (opcional)).

El primer paso de todos será cargar los datos tanto del train como del test. Declararemos la función fsimetría que nos permitirá calcular la simetría de la imagen

```
fsimetria <- function(A){
  A = abs(A-A[,ncol(A):1])
  -sum(A)
}

setwd("./datos")

#TRAIN
digit.train <- read.table("zip.train",quote="\"", comment.char="", stringsAsFactors=FALSE)

digitos15.train = digit.train[digit.train$V1==1 | digit.train$V1==5,]
digitos.train = digitos15.train[,1]    # vector de etiquetas del train
ndigitos.train = nrow(digitos15.train)  # numero de muestras del train

# se retira la clase y se monta una matriz 3D: 599*16*16
grises.train = array(unlist(subset(digitos15.train,select=-V1)),c(ndigitos.train,16,16))
rm(digit.train)
rm(digitos15.train)

intensidad.train <- apply(grises.train, 1, mean)

simetria.train <- apply(grises.train, 1, fsimetria)

rm(grises.train)
digitos.train <- replace(digitos.train, digitos.train == 5, -1)
datosTr = as.matrix(cbind(intensidad.train,simetria.train))

#########################################################################################################

#TEST

digit.test <- read.table("zip.test",quote="\"", comment.char="", stringsAsFactors=FALSE)

digitos15.test = digit.test[digit.test$V1==1 | digit.test$V1==5,]
digitos.test = digitos15.test[,1]    # vector de etiquetas del test
ndigitos.test = nrow(digitos15.test)  # numero de muestras del test

# se retira la clase y se monta una matriz 3D: 599*16*16
grises.test = array(unlist(subset(digitos15.test,select=-V1)),c(ndigitos.test,16,16))
grises.test <- as.numeric(grises.test)
grises.test <- array(grises.test,(c(ndigitos.test,16,16)))
rm(digit.test)
rm(digitos15.test)

intensidad.test <- apply(grises.test, 1, mean)

simetria.test <- apply(grises.test, 1, fsimetria)
rm(grises.test)
digitos.test <- replace(digitos.test, digitos.test == 5, -1)
```

```
datosTst = as.matrix(cbind(intensidad.test,simetria.test))
```

Una vez cargados todos los datos en datosTr y datosTst pasamos a crear la función Regress_Lin:

Primero, vamos a realizar los pasos con el algoritmo de la pseudoinversa y despues iremos con el GDE, como vamos a utilizar en este caso el algoritmo de la pseudoinversa, en las transparencias de teoría podemos ver que:

$X\dagger = (X^T X)^{-1} X^T$

```
#Regresion lineal :: pseudoinversa
Regress_Lin <- function(datos, label){
 # Añadimos al final la columna de 1 ya que nuestra variable independiente es w3
  datos <- cbind(datos,1)
  # multiplicamos la matriz traspuesta de datos por la matriz de datos
  x <- t(datos)%*%datos
  # Calculamos la inversa de esta matriz
  x.inv <- solve(x)
  # Multiplicamos la inversa calculada por la traspuesta de los datos = pseudoinversa = (X^T*X)^-1*X^T
  x.pseudo.inv <- x.inv %*% t(datos)
  # Calculamos los pesos
  w <- x.pseudo.inv %*% label
  w
}
```

Por otra parte, tenemos que calcular E_{in} sobre este ajuste, como la codificación que usamos es binaria ({-1,1}) utilizaremos la siguiente función para calcular el error.

$\frac{1}{N}\sum_{i=1}^{N} sign(w^T x_n) \neq y_n$

```
#Calcula el error en representacion binaria
Err_bin <- function(x, y, w){
  # Añadimos al final la columna de 1 ya que nuestra variable independiente es w3
  x <- cbind(x,1)
  suma <- 0
  for(n in 1:nrow(x)){
    if( sign(x[n,] %*% w) != y[n]){
      suma <- suma +1
    }
  }
  suma <- suma/nrow(x)
  suma
}
```

Ahora que tenemos las funciones que necesitamos, calculamos los pesos utilizando la pseudoinversa, dibujamos los resultados y calculamos el error.

Antes de nada, para poder calcular la recta respecto a los pesos, debemos de calcular la pendiente y el punto de intercepción, esto lo haremos sabiendo que tenemos una ecuación que cumple que $w_1 X_1 + w_2 X_2 + w_3 = 0$, despejamos X_2 para obtener la recta del tipo $y = mx + n$.

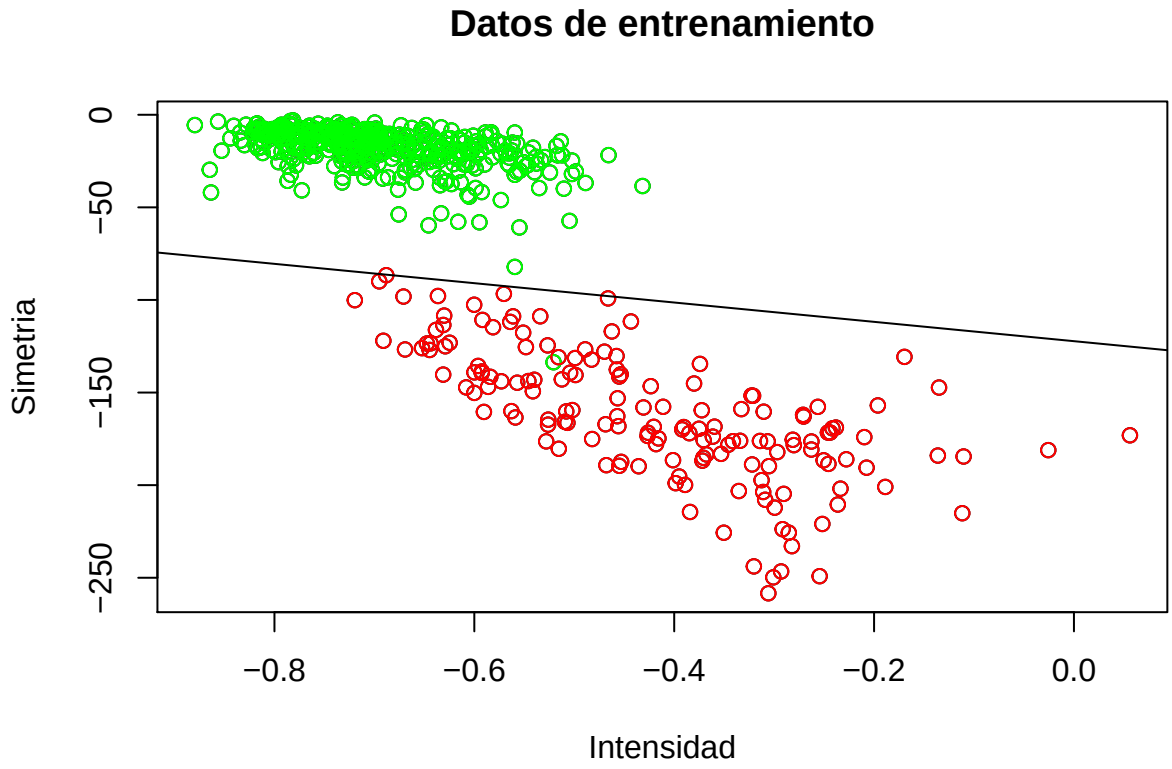
Una vez despejado, obtenemos $X_2 = -\frac{w_1}{w_2} X_1 - \frac{w_3}{w_2}$. Por lo que tenemos que la pendiente de la recta es $-\frac{w_1}{w_2}$ y el punto de intercepción es $-\frac{w_3}{w_2}$

```
#Calculamos los pesos
pesos <- Regress_Lin(datosTr,matrix(digitos.train,ncol=1))
pesos
```

```
##                       [,1]
```

```
## intensidad.train 0.73182914
## simetria.train   0.01402067
##                  1.71412717
```

```
#Dibujamos los datos del train
plot(datosTr[,1],datosTr[,2],xlab = "Intensidad", ylab = "Simetria", main="Datos de entrenamiento")
points(datosTr[,1][digitos.train==1],datosTr[,2][digitos.train==1],col="green")
points(datosTr[,1][digitos.train==-1],datosTr[,2][digitos.train==-1],col="red")
#Dibujamos la recta calculada a traves de los pesos
abline(-pesos[3]/pesos[2],-pesos[1]/pesos[2],col="black")
```



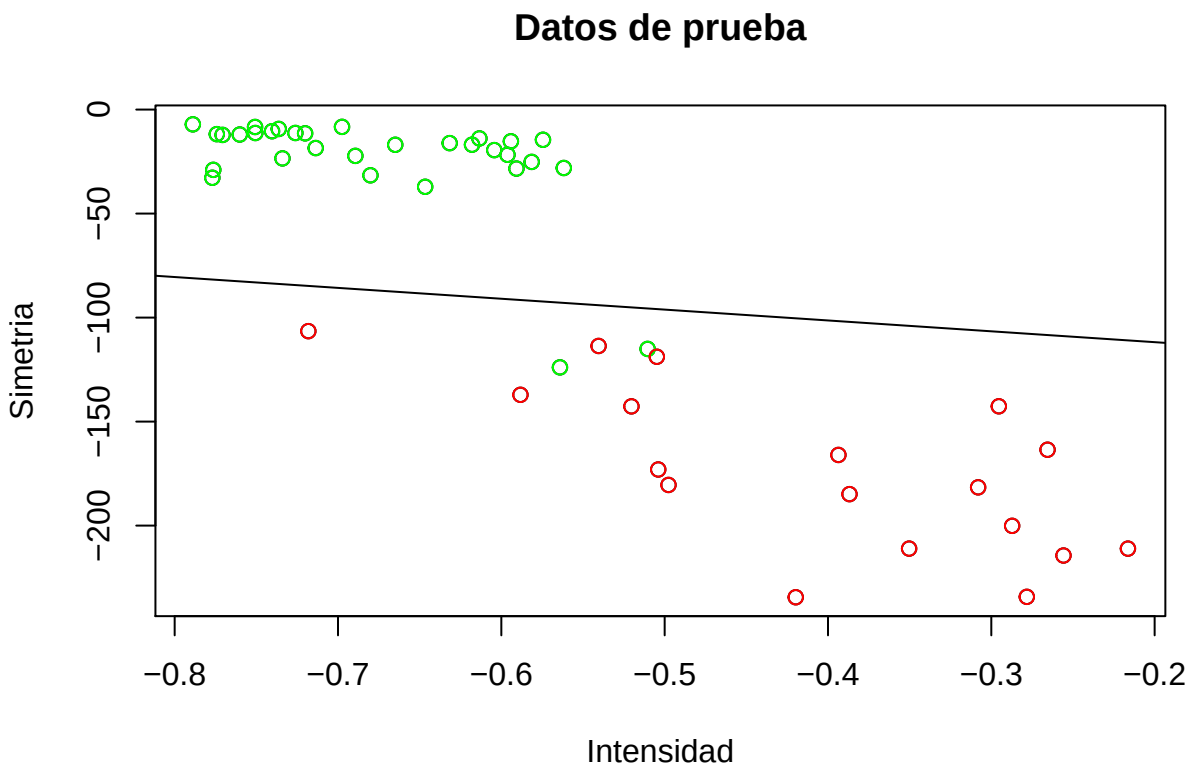
Podemos ver que hace un ajuste bastante correcto, fallando muy pocos elementos. El siguiente paso será calcular el error.

```
#Calculamos el error_in
Err_in <- Err_bin(datosTr,matrix(digitos.train,ncol=1),pesos)
Err_in
```

```
## [1] 0.001669449
```

Ahora vamos a realizar los mismos pasos pero para los datos de test.

```
#Dibujamos igual que antes pero con los datos del test
plot(datosTst[,1],datosTst[,2],xlab = "Intensidad", ylab = "Simetria", main= "Datos de prueba")
points(datosTst[,1][digitos.test==1],datosTst[,2][digitos.test==1],col="green")
points(datosTst[,1][digitos.test==-1],datosTst[,2][digitos.test==-1],col="red")

abline(-pesos[3]/pesos[2],-pesos[1]/pesos[2],col="black")
```



Vemos en la imagen que este ajuste tambien ha sido bastante bueno, ahora pasemos a calcular su error.

```
#Calculamos el error en los datos de test
Err_out <- Err_bin(datosTst,matrix(digitos.test,ncol=1),pesos)
Err_out
```

```
## [1] 0.04081633
```

Podemos ver que el error es un poco más alto que en los datos de prueba pero no obstante, hemos obtenido unos resultados bastante buenos de error.

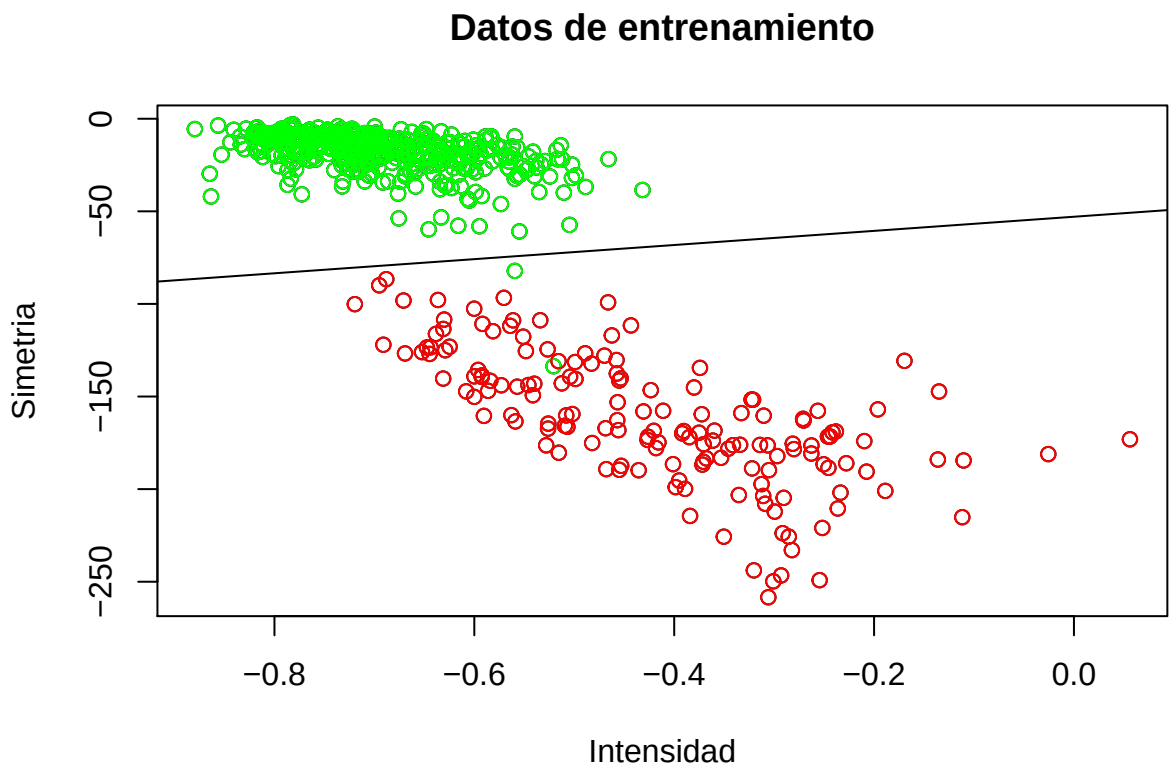
Ahora vamos a hacer uso del GDE para calcular los pesos y minimizar el E_in y seguiremos los mismos pasos que hemos seguido para el algoritmo de la pseudoinversa. Tenemos que definir tanto la funcion $\nabla E_{in} = \frac{-y_n x_n}{1+e^{y_n w^T x_n}}$ como el propio GDE

```
set.seed(3)

G_Err_bin <- function(x,y,w,n){
  sum <- matrix(0.0,nrow=3)
  x <- cbind(x,1)
  for(i in 1:length(n)){
    sum <- sum + ((-y[n[i],]*x[n[i],])/(1+exp((y[n[i],]*(x[n[i],]%*%w)))))
  }
  sum <- sum/length(n)
  sum
}

GDE <- function(funcion, derv, datos, label, niter, paso, umbral,tam){
```

```
  w <- matrix(0.0,nrow=3)
  i <- 1
  valor <- funcion(datos,label,w)
  valorant <- valor+1

  while(valor > umbral & i < niter){
    n <- sample(x = nrow(datos),size = tam,replace = FALSE)
    grad <- derv(datos,label,w,n)
    w <- w - paso*grad
    valorant <- valor
    valor <- funcion(datos,label,w)
    i <- i+1
  }
w
}

# Calculamos de nuevo los pesos pero utilizando GDE
pesos <- GDE(Err_bin,G_Err_bin,datosTr,matrix(digitos.train,ncol=1),1000,as.double(0.01),as.double(10e-
pesos
```

```
##             [,1]
## [1,] -1.47859346
## [2,]  0.03872376
## [3,]  2.04910257
```

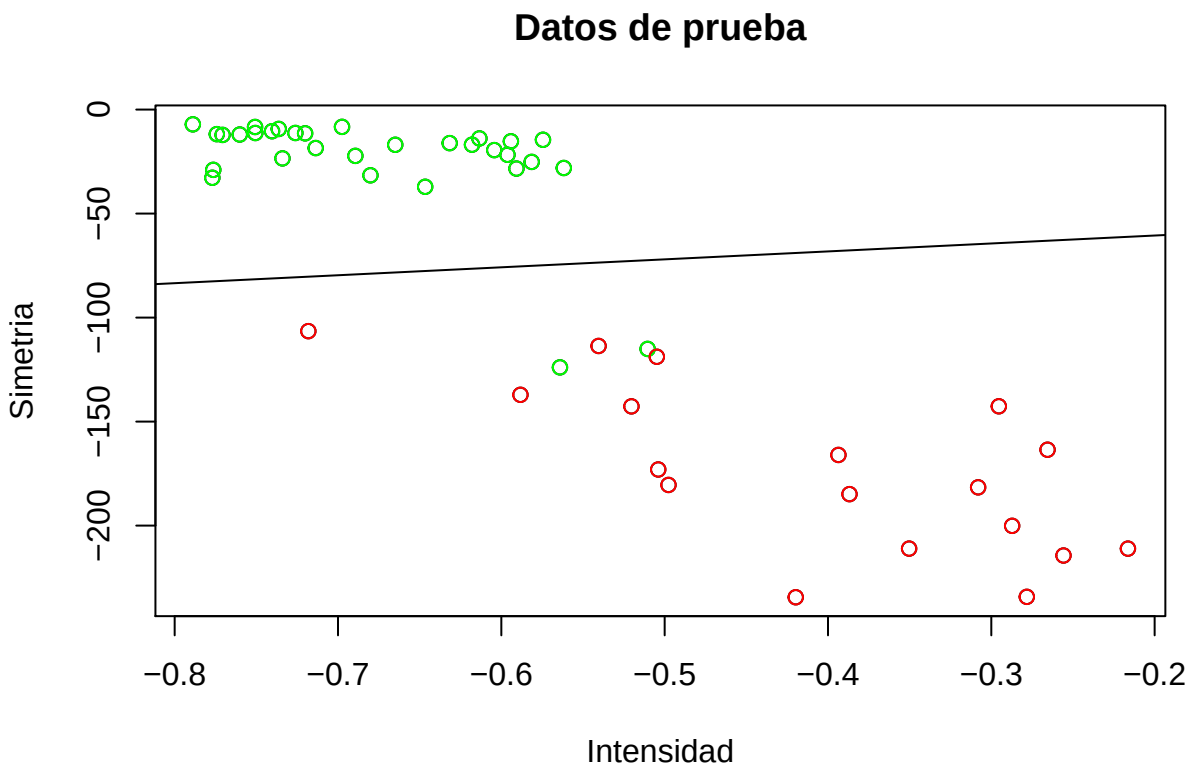
```
#Dibujamos los datos del train
plot(datosTr[,1],datosTr[,2],xlab = "Intensidad", ylab = "Simetria", main="Datos de entrenamiento")
points(datosTr[,1][digitos.train==1],datosTr[,2][digitos.train==1],col="green")
points(datosTr[,1][digitos.train==-1],datosTr[,2][digitos.train==-1],col="red")
#Dibujamos la recta calculada a traves de los pesos
abline(-pesos[3]/pesos[2],-pesos[1]/pesos[2],col="black")
```




```
#Calculamos el error_in
Err_in <- Err_bin(datosTr,matrix(digitos.train,ncol=1),pesos)
Err_in
```

```
## [1] 0.003338898
```

```
#Dibujamos igual que antes pero con los datos del test
plot(datosTst[,1],datosTst[,2],xlab = "Intensidad", ylab = "Simetria", main= "Datos de prueba")
points(datosTst[,1][digitos.test==1],datosTst[,2][digitos.test==1],col="green")
points(datosTst[,1][digitos.test==-1],datosTst[,2][digitos.test==-1],col="red")

abline(-pesos[3]/pesos[2],-pesos[1]/pesos[2],col="black")
```




```
#Calculamos el error en los datos de test
Err_out <- Err_bin(datosTst,matrix(digitos.test,ncol=1),pesos)
Err_out
```

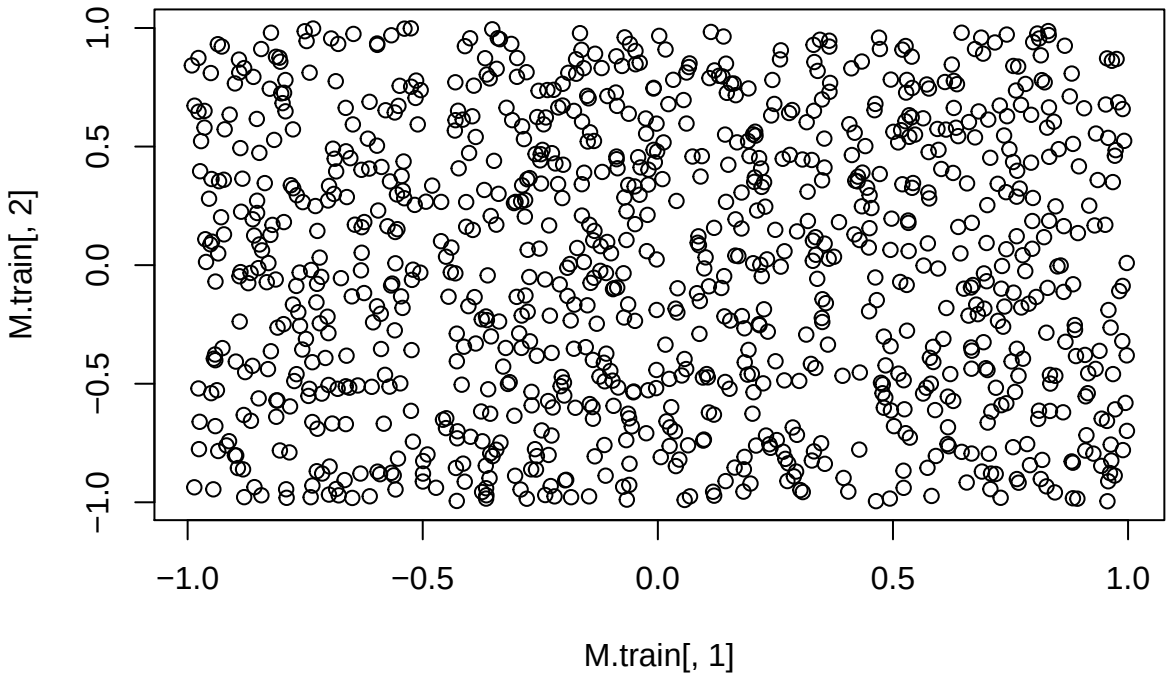
```
## [1] 0.04081633
```

2. En este apartado exploramos como se transforman los errores E_in y E_out cuando aumentamos la complejidad del modelo lineal usado. Ahora hacemos uso de la función simula_unif(N, 2, size) que nos devuelve N coordenadas 2D de puntos uniformemente muestreados dentro del cuadrado definido por $[-size, size] \times [-size, size]$

```
simula_unif = function (N=2,dims=2, rango = c(0,1)){
  m = matrix(runif(N*dims, min=rango[1], max=rango[2]),
             nrow = N, ncol=dims, byrow=T)
  m
}
```

a) Generar una muestra de entrenamiento de N = 1000 puntos en el cuadrado X = $[-1, 1]$Ö$[-1, 1]$. Pintar el mapa de puntos 2D. (ver función de ayuda)

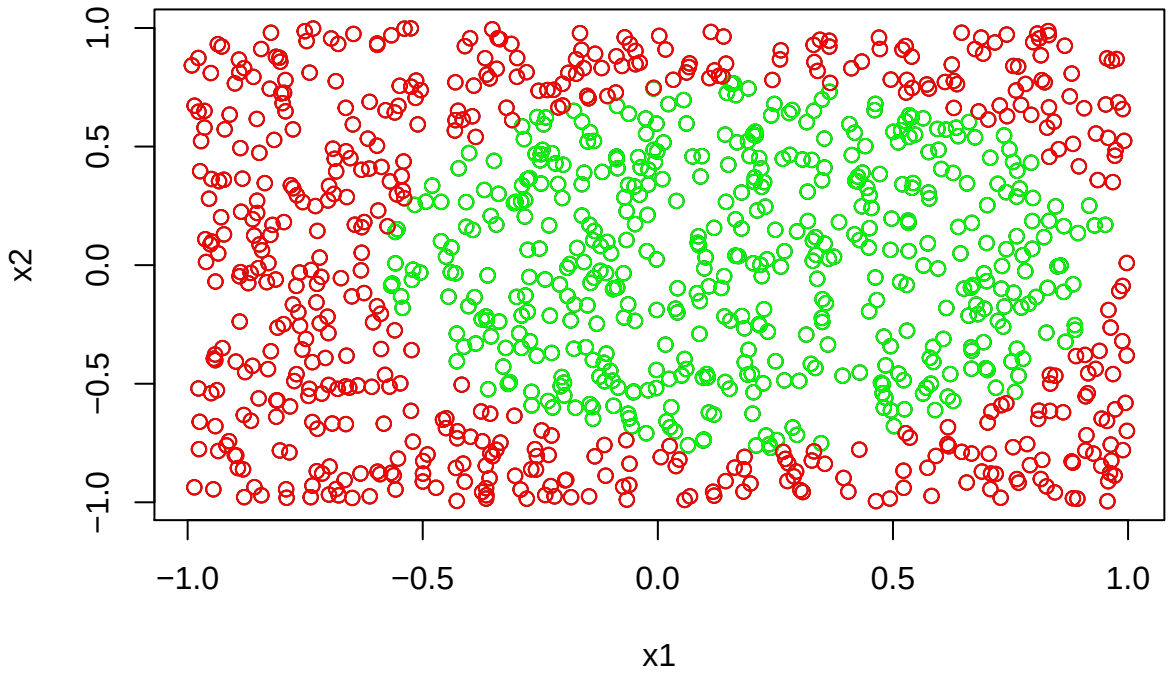
```
# Generamos la muestra
M.train <- simula_unif(1000,2,c(-1,1))
#Mostramos los resultados
plot(M.train[,1],M.train[,2], type ="p")
```



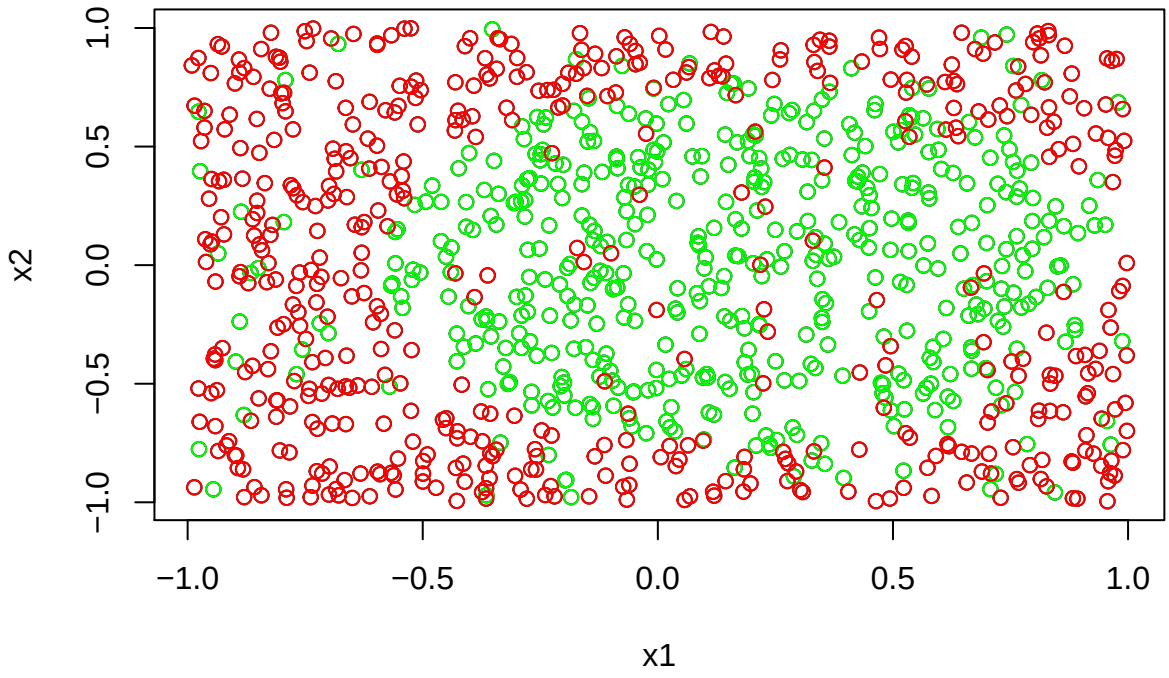
b) Consideremos la función $f(x_1, x_2) = sign((x_1 - 0,2)^2 + x_2^2 - 0,6)$ que usaremos para asignar una etiqueta a cada punto de la muestra anterior. Introducimos ruido sobre las etiquetas cambiando aleatoriamente el signo de un 10 % de las mismas. Pintar el mapa de etiquetas obtenido.

El primer paso será definir la funcion, y despues continuamos con lo que pide el ejercicio. También haré uso de una funcion que voy a cerar que cambia un porcentaje de elementos de un vector que le pasemos.

```
#Definimos la funcion
funcion3 <- function(x,y){
  sign((x-0.2)^2 + y^2 -0.6)
}
#funcion que nos cambia el % de las etiquetas
cambiarValores <- function(datos, nvalores){
  cambios <- sample(x = length(datos), size = nvalores, replace = FALSE)

  datos[cambios] <- datos[cambios]*-1
  datos
}

#Creamos la columna de etiquetas
M.etiquetas.train <- c(funcion3(M.train[,1],M.train[,2]))
plot(M.train[,1],M.train[,2],type ="p", xlab = "x1" , ylab = "x2")
points(M.train[,1][M.etiquetas.train==-1],M.train[,2][M.etiquetas.train==-1],col="green")
points(M.train[,1][M.etiquetas.train==1],M.train[,2][M.etiquetas.train==1],col="red")
```




```
#Cambiamos el signo a el 10% de los valores
M.etiquetas.train <- cambiarValores(M.etiquetas.train,0.1*nrow(M.train))
plot(M.train[,1],M.train[,2], type ="p", xlab = "x1" , ylab = "x2")
points(M.train[,1][M.etiquetas.train==-1],M.train[,2][M.etiquetas.train==-1],col="green")
points(M.train[,1][M.etiquetas.train==1],M.train[,2][M.etiquetas.train==1],col="red")
```



c) Usando como vector de características $(1, x_1, x_2)$ ajustar un modelo de regresion lineal al conjunto de datos generado y estimar los pesos w. Estimar el error de ajuste E_{in} usando Gradiente Descendente Estocástico (SGD).

Para calcular los pesos utilizaremos la pseudoinversa, sin embargo, en los casos anteriores teníamos un vector de caracteristicas $(w_1, w_2, 1)$ y ahora tenemos otro diferente, por lo que tenemos que realizar algunos cambios en las funciones. Por lo que definiré algunas funciones nuevas.

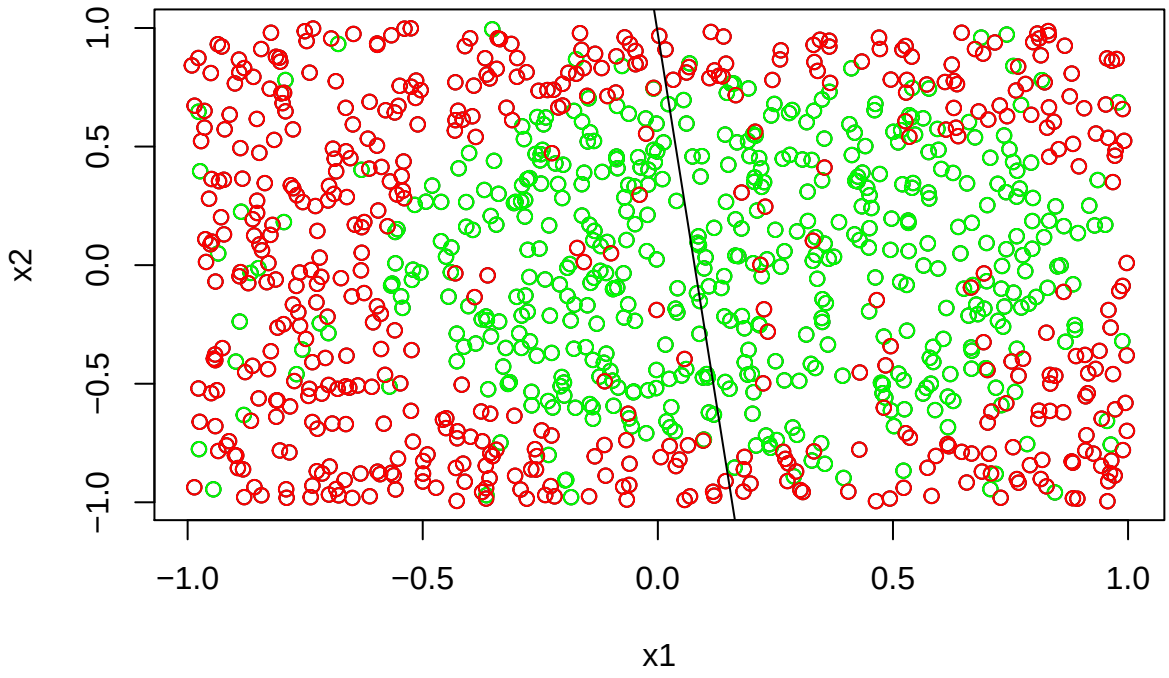
Como también vamos a utilizar el GDE, tendgo que definir otras funciones nuevas ya que la forma de representar los pesos es distinta.

```
Regress_Lin2 <- function(datos, label){
  datos<-datos[,ncol(datos):1]
  # Añadimos al final la columna de 1 ta que nuestra variable independiente es w3
  datos <- cbind(1,datos)
  # multiplicamos la matriz traspuesta de datos por la matriz de datos
  x <- t(datos)%*%datos
  # Calculamos la inversa de esta matriz
  x.inv <- solve(x)
  # Multiplicamos la inversa calculada por la traspuesta de los datos = pseudoinversa = (X^T*X)^-1*X^T
  x.pseudo.inv <- x.inv %*% t(datos)
  # Calculamos los pesos
  w <- x.pseudo.inv %*% label
  w
}

#Calcula el error en representacion binaria
Err_bin2 <- function(x, y, w){
```

```
  x<-x[,ncol(x):1]
  x <- cbind(1,x)
  suma <- 0
  for(n in 1:nrow(x)){
    if( sign(x[n,] %*% w) != y[n]){
      suma <- suma +1
    }
  }
  suma <- suma/nrow(x)
  suma
}

# Gradiente
G_Err_bin2 <- function(x,y,w,n){
  sum <- matrix(0.0,nrow=3)
  x<-x[,ncol(x):1]
  x <- cbind(1,x)
  for(i in 1:length(n)){
    sum <- sum + ((-y[n[i],]*x[n[i],])/(1+exp((y[n[i],]*(x[n[i],]%*%w)))))
  }
  sum <- sum/length(n)
  sum
}

#Calculamos los pesos
pesos <- Regress_Lin2(M.train,matrix(M.etiquetas.train,ncol=1))
pesos
```

```
##             [,1]
## [1,]  0.03274519
## [2,] -0.03358651
## [3,] -0.42014829
```

```
#Dibujamos los puntos y la recta
plot(M.train[,1],M.train[,2],type ="p", xlab = "x1" , ylab = "x2")
points(M.train[,1][M.etiquetas.train==-1],M.train[,2][M.etiquetas.train==-1],col="green")
points(M.train[,1][M.etiquetas.train==1],M.train[,2][M.etiquetas.train==1],col="red")

abline(-(pesos[1]/pesos[2]),-(pesos[3]/pesos[2]),col="black")
```



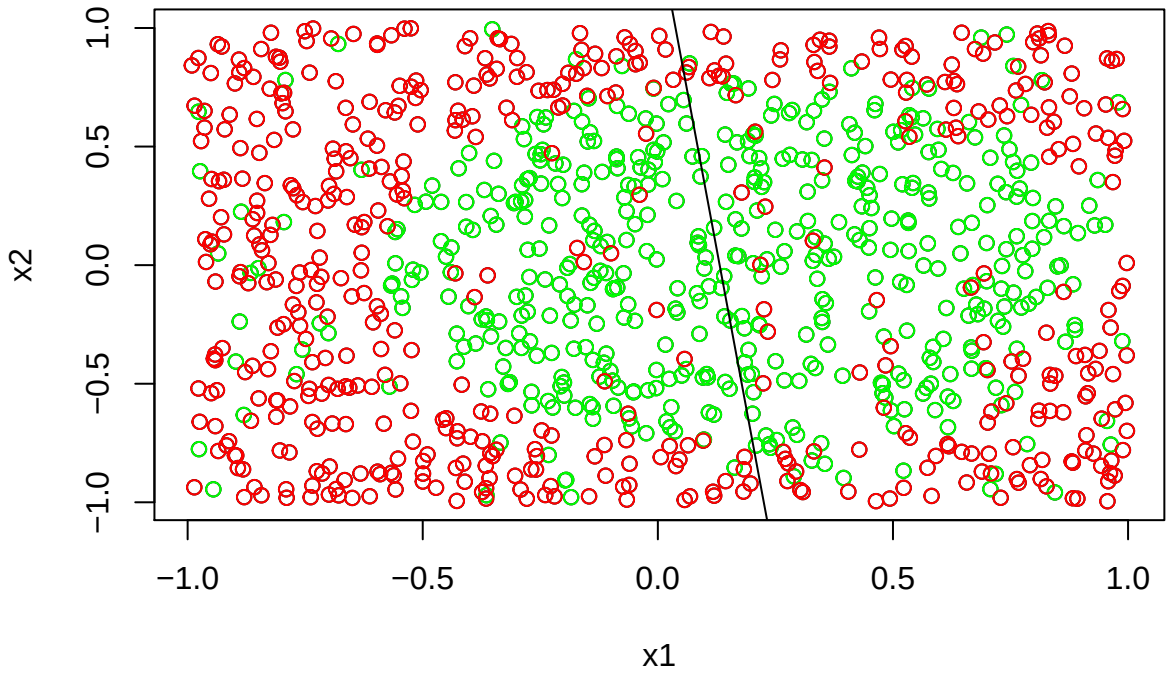

```
#Calculamos el error
Err_puntos <- Err_bin2(M.train,matrix(M.etiquetas.train,ncol=1),pesos)
Err_puntos
```

```
## [1] 0.409
```

```
#Ahora realizamos los mismos pasos pero con el GDE
#Calculamos los pesos
pesos <- GDE(Err_bin2, G_Err_bin2, M.train,matrix(M.etiquetas.train,ncol=1),800,as.double(0.01),as.doubl
pesos
```

```
##              [,1]
## [1,]  0.05289332
## [2,] -0.03776183
## [3,] -0.40240069
```

```
#Dibujamos los puntos y la recta
plot(M.train[,1],M.train[,2],type ="p", xlab = "x1" , ylab = "x2")
points(M.train[,1][M.etiquetas.train==-1],M.train[,2][M.etiquetas.train==-1],col="green")
points(M.train[,1][M.etiquetas.train==1],M.train[,2][M.etiquetas.train==1],col="red")

abline(-(pesos[1]/pesos[2]),-(pesos[3]/pesos[2]),col="black")
```




```
#Calculamos el error
Err_puntos <- Err_bin2(M.train,matrix(M.etiquetas.train,ncol=1),pesos)
Err_puntos
```

```
## [1] 0.417
```

d) Ejecutar todo el experimento definido por (a)-(c) 1000 veces (generamos 1000 muestras diferentes) y
- Calcular el valor medio de los errores Ein de las 1000 muestras
- Generar 1000 puntos nuevos por cada iteración y calcular con ellos el valor de Eout en dicha iteración. Calcular el valor medio de Eout en todas las iteraciones.

Para calcular los pesos podemos utilizar tanto la pseudoinversa como el GDE, en mi caso voy a utilizar la pseudoinversa.

```
Err_in_total <- 0
Err_out_total <- 0
for(i in 1:1000){
  #Creamos los 1000 puntos aleatorios
  M.train <- simula_unif(1000,2,c(-1,1))
  # Creamos sus etiquetas
  M.etiquetas.train <- c(funcion3(M.train[,1],M.train[,2]))
  #Cambiamos aleatoriamente un 10% de ellos
  M.etiquetas.train <- cambiarValores(M.etiquetas.train,0.1*nrow(M.train))
  # Creamos los pesos utilizando el algortirmo de la pseudoinversa
  pesos <- Regress_Lin2(M.train,matrix(M.etiquetas.train,ncol=1))
  # Calculamos E_in
  Err_puntos_in <- Err_bin2(M.train,matrix(M.etiquetas.train,ncol=1),pesos)
  # Sumamos E_in de esta iteracion al total
```

```
  Err_in_total <- Err_in_total + Err_puntos_in
  # Simulamos otros 1000 puntos nuevos para test
  M.test <- simula_unif(1000,2,c(-1,1))
  # Calculamos sus etiquetas
  M.etiquetas.test <- c(funcion3(M.test[,1],M.test[,2]))
  # Calculamos E_out
  Err_puntos_out <- Err_bin2(M.test,matrix(M.etiquetas.test,ncol=1),pesos)
  # Sumamos el E_out al total
  Err_out_total <- Err_out_total + Err_puntos_out
}
Err_in_total <- Err_in_total/1000
Err_out_total <- Err_out_total/1000
#Mostramos los resultados
Err_in_total
```

```
## [1] 0.398051
```

```
Err_out_total
```

```
## [1] 0.37609
```

e) Valore que tan bueno considera que es el ajuste con este modelo lineal a la vista de los valores medios obtenidos de E_{in} y E_{out}

Podemos ver por los resultados, que casi siempre se obtiene un error mayor en los datos de aprendizaje que en los de test. Esto se debe a que está aprendiendo mal ya que hemos cambiado un 10% de los datos, por lo que forzamos a que falle. Con esto puedo concluir que no solo importa que tengamos un bien metodo de aprendizaje, sino que los datos que tenemos deben ser representativos y estar correctamente diseñados para que el ajuste sea lo más correcto posible.

bonus

1. (2 puntos) **Método de Newton** Implementar el algoritmo de minimización de Newton y aplicarlo a la función $f(x, y)$ dada en el ejercicio.3. Desarrolle los mismos experimentos usando los mismos puntos de inicio.

- Generar un gráfico de como desciende el valor de la función con las iteraciones.
- Extraer conclusiones sobre las conductas de los algoritmos comparando la curva de decrecimiento de la función calculada en el apartado anterior y la correspondiente obtenida con gradiente descendente.

Para el algoritmo de Newton he utilizado los apuntes que hay en el libro *"The Elements of Statistical Learning" pag.139pdf 120libro*

Una vez vista la teoría, para realizar este método necesitamos calcular la matriz Hessiana que no es mas que, $H = \left\{ \begin{array}{cc} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{array} \right\}$

Para ello utilizaré varias funciones para repartir los distintos módulos, crearé una funcion que crea la matriz hessiana, crearé el propio metodo de Newton y crearé las distintas derivadas.

Por último utilizare otra funcion que voy a llamar *newton2* para que una me devuelva el punto mínimo e iteracion y otro me devuelva todos los puntos por lo que hemos pasado.

```
#Declaramos la funcion
funcion <- function(x,y){
  x^2+2*sin(2*pi*x)*sin(2*pi*y)-4*x+2*y^2+8*y+12
}
```

```r
#Segunda derivada respecto x^2
derivadaxx <- function(x,y){
  2-8*pi^2*sin(2*pi*x)*sin(2*pi*y)
}

#Segunda derivada respecto xy
derivadaxy <- function(x,y){
  8*pi^2*cos(2*pi*x)*cos(2*pi*y)
}

#Segunda derivada respecto y^2
derivadayy <- function(x,y){
  4-8*pi^2*sin(2*pi*x)*sin(2*pi*y)
}

#Derivada respecto x e y por separado
derivada <- function(x,y){
  c(2*(2*pi*cos(2*pi*x)*sin(2*pi*y)+x-2),4*(pi*sin(2*pi*x)*cos(2*pi*y)+y+2))
}

#Funcion que crea la matriz hessiana
crear_hessiana <- function(punto, dxx, dxy, dyy){
  matrix(
    c(
      dxx(punto[1,1],punto[2,1]),
      dxy(punto[1,1],punto[2,1]),
      dxy(punto[1,1],punto[2,1]),
      dyy(punto[1,1],punto[2,1])
    ),nrow = 2, ncol = 2, byrow=TRUE)
}

#Metodo de Newton - "The Elements of Statistical Learning" pag.139pdf 120libro
newton <- function(func, derv, dxx, dxy, dyy, wini, niter, umbral){
  w <- wini
  i <- 1
  puntos=matrix(wini,nrow=2)
  want <- as.matrix(c(w[1,1]+1,w[2,1]+1), nrow=2)

  while(abs(func(w[1,1],w[2,1]) - func(want[1,1],want[2,1])) > umbral && i < niter){
    want <- w
    hessiana.inv <- solve(crear_hessiana(w,dxx,dxy,dyy))
    grdnt <- derv(w[1,1],w[2,1])
    w <- w - hessiana.inv %*% grdnt
    i <- i+1
    puntos<-cbind(puntos,w)
  }
  puntos
}

#Devuelve el minimo
newton2 <- function(func, derv, dxx, dxy, dyy, wini, niter, umbral){
  w <- wini
  want <- as.matrix(c(w[1,1]+1,w[2,1]+1), nrow=2)
```

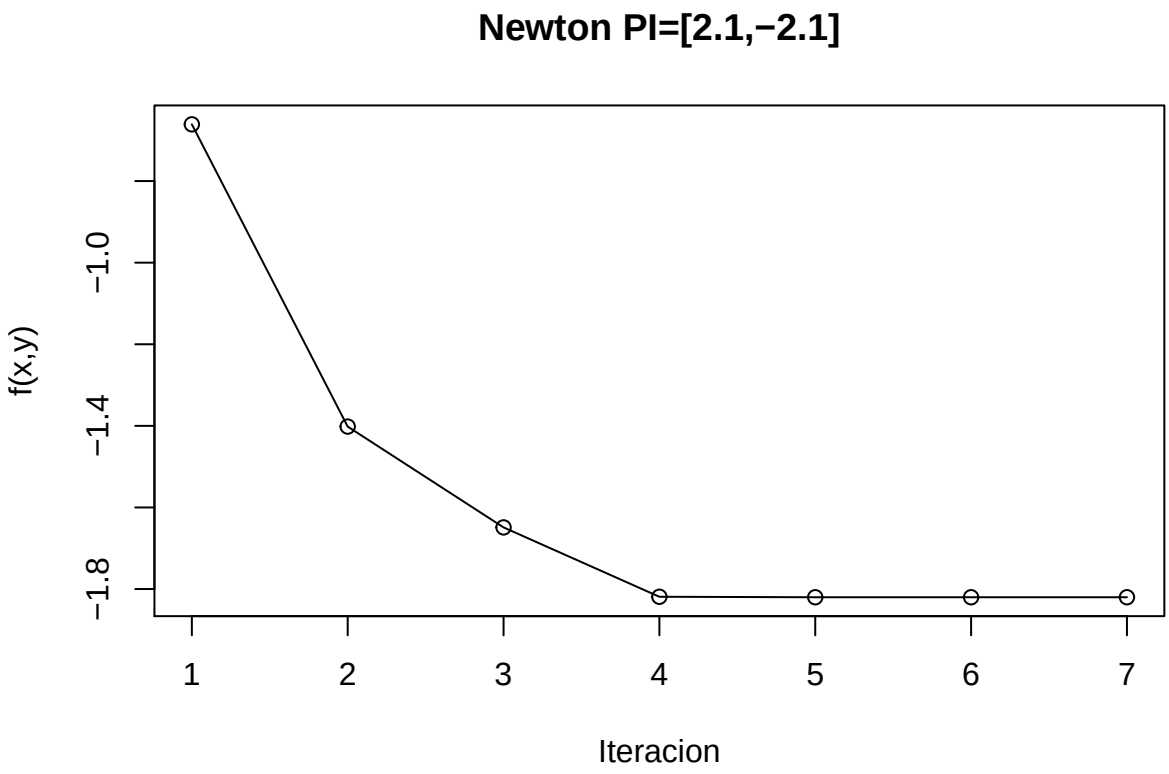
```
  i <- 1
  imin <- i
  wmin <- wini

  while(abs(func(w[1,1],w[2,1]) - func(want[1,1],want[2,1])) > umbral && i < niter){
    want <- w
    hessiana.inv <- solve(crear_hessiana(w,dxx,dxy,dyy))
    grdnt <- derv(w[1,1],w[2,1])
    w <- w - hessiana.inv %*% grdnt
    i <- i+1
    if(func(w[1,1],w[2,1]) < func(wmin[1,1],wmin[2,1])){
      wmin <- w
      imin <- i
    }
  }
  c(wmin,imin)
}

# Pasamos a la ejecución de los distintos tipos.
#1
resultado3 <- newton2(funcion,derivada,derivadaxx,derivadaxy,derivadayy,as.matrix(c(2.1,-2.1),nrow=2),1
resultado3
```

```
## [1]  1.756195 -1.762074  6.000000
```

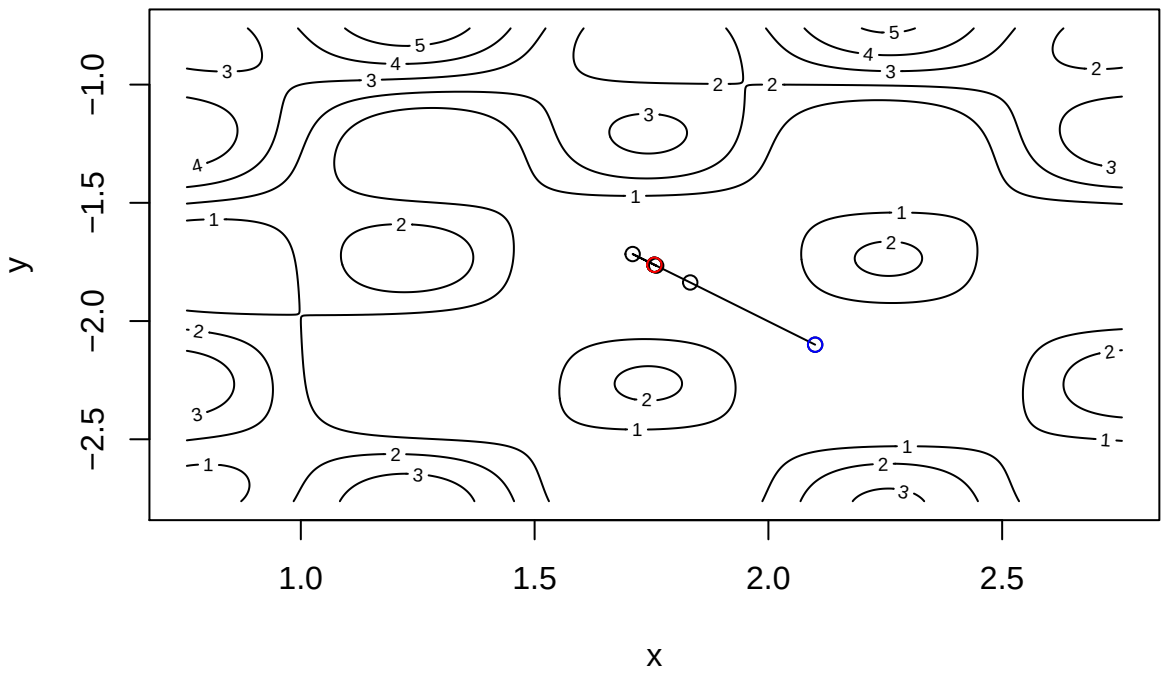
```
funcion(resultado3[1],resultado3[2])
```

```
## [1] -1.820079
```

```
resultado3 <- newton(funcion,derivada,derivadaxx,derivadaxy,derivadayy,as.matrix(c(2.1,-2.1),nrow=2),1e
plot(funcion(resultado3[1,],resultado3[2,]), main="Newton PI=[2.1,-2.1]", type="o", xlab="Iteracion", y
```




```
aum <- 1
pintar_frontera(funcion,resultado3,c(resultado3[1,ncol(resultado3)]-aum,resultado3[1,ncol(resultado3)]+a
```



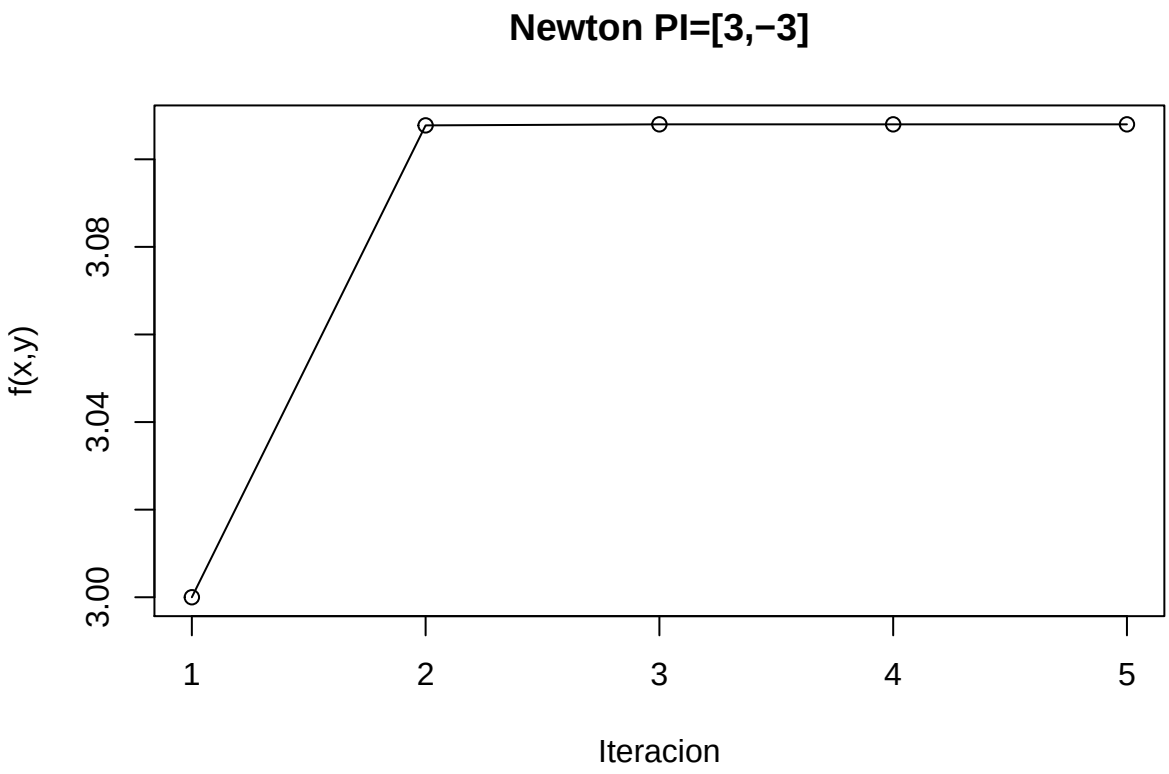

```
# Obtenemos en el punto [1.756195,-1.762075] el valor de -1.820079 con 6 iteraciones

#2
resultado3 <- newton2(funcion,derivada,derivadaxx,derivadaxy,derivadayy,as.matrix(c(3,-3),nrow=2),1e+10
resultado3
```

```
## [1]  3 -3  1
```

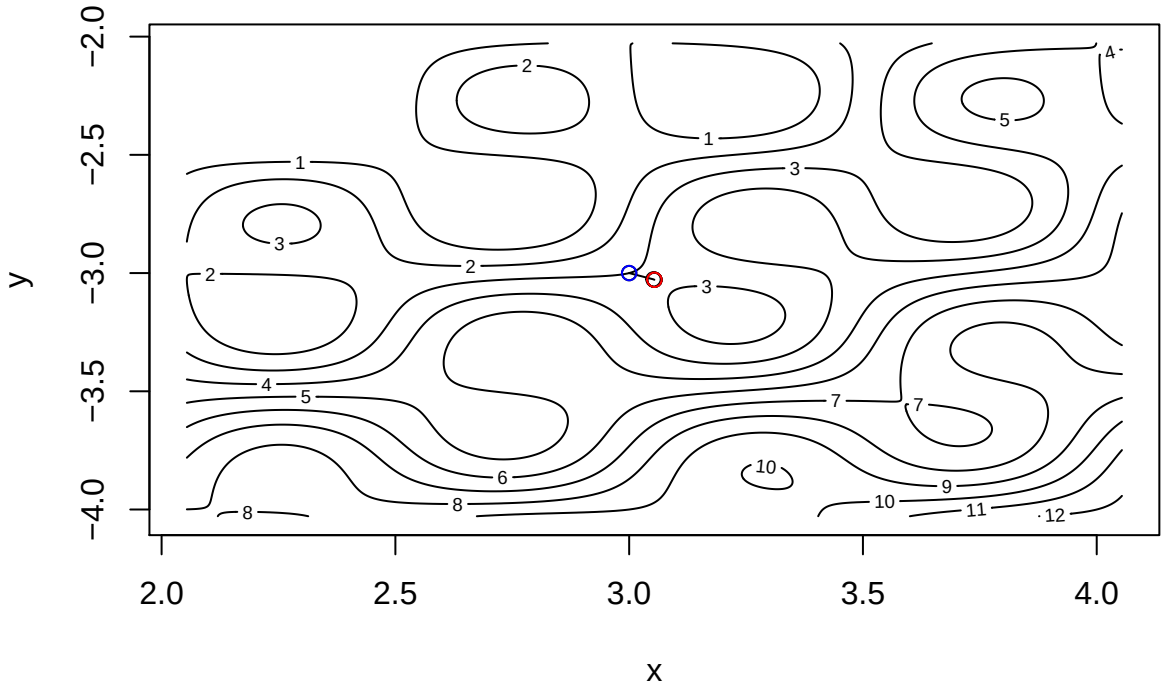
```
funcion(resultado3[1],resultado3[2])
```

```
## [1] 3
```

```
resultado3 <- newton(funcion,derivada,derivadaxx,derivadaxy,derivadayy,as.matrix(c(3,-3),nrow=2),1e+10,a
plot(funcion(resultado3[1,],resultado3[2,]), main="Newton PI=[3,-3]", type="o", xlab="Iteracion", ylab=
```




```
aum <- 1
pintar_frontera(funcion,resultado3,c(resultado3[1,ncol(resultado3)]-aum,resultado3[1,ncol(resultado3)]+a
```



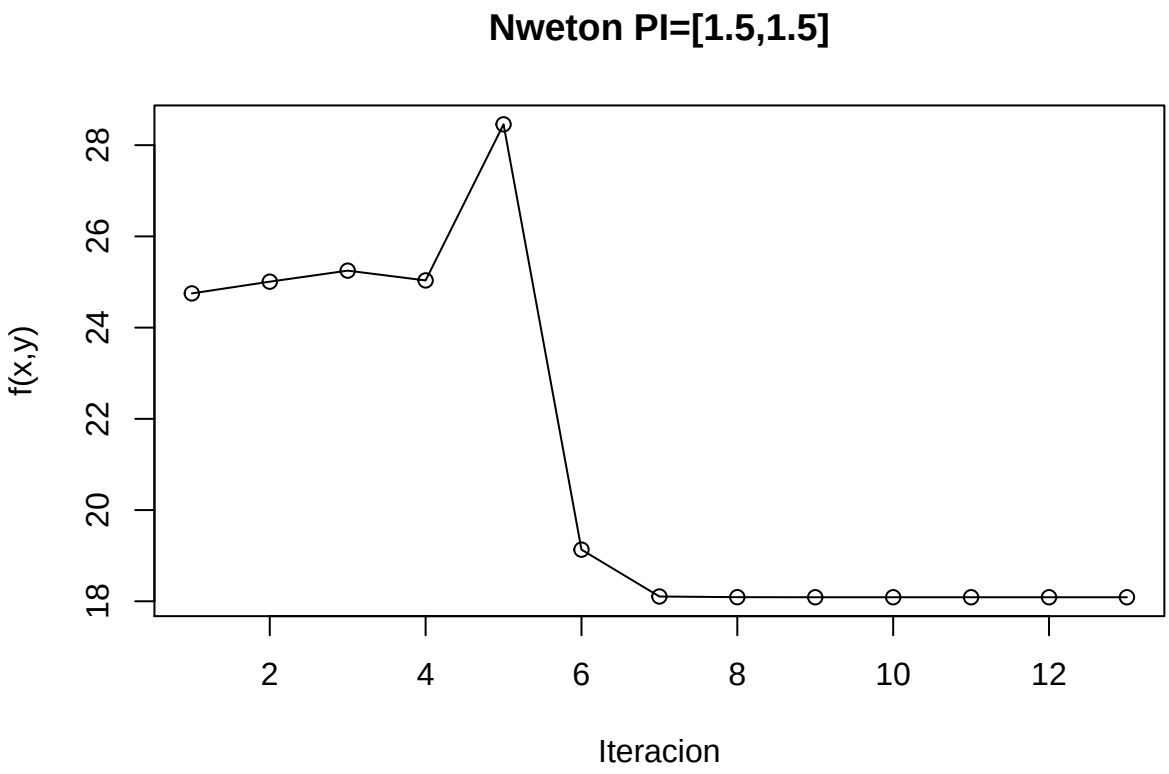

```
# Obtenemos en el punto [3,-3] el valor de 3 con 1 iteraciones

#3
resultado3 <- newton2(funcion,derivada,derivadaxx,derivadaxy,derivadayy,as.matrix(c(1.5,1.5),nrow=2),1e-
resultado3
```

```
## [1]  1.7048310  0.9731716 12.0000000
```

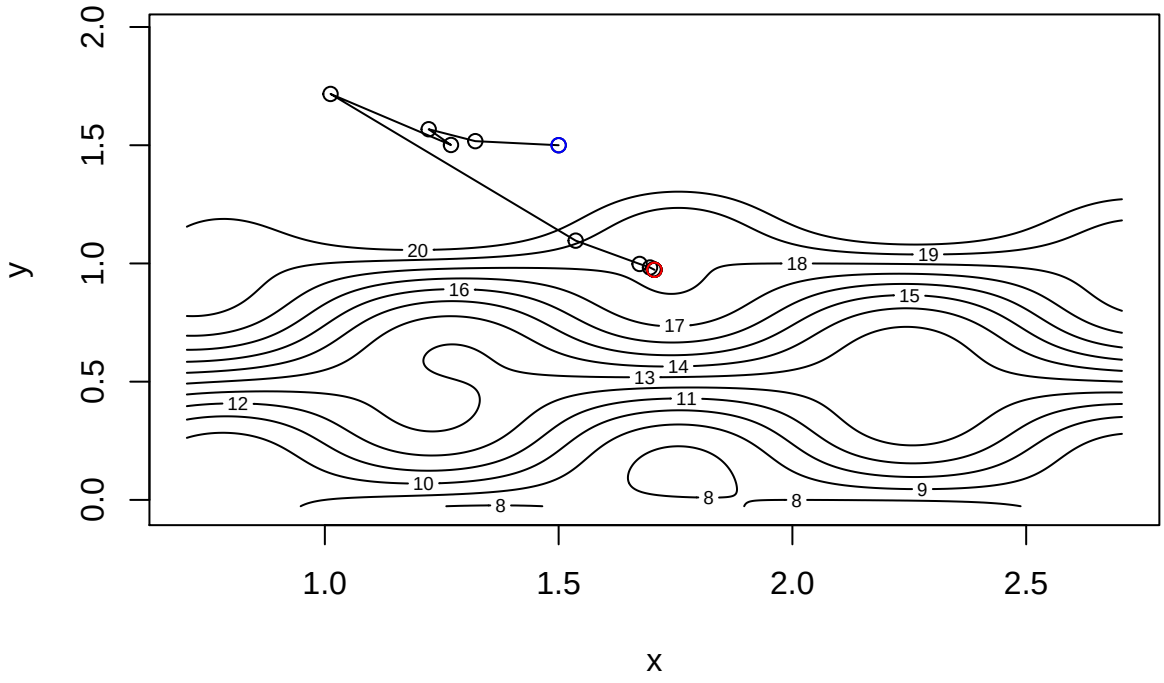
```
funcion(resultado3[1],resultado3[2])
```

```
## [1] 18.08874
```

```
resultado3 <- newton(funcion,derivada,derivadaxx,derivadaxy,derivadayy,as.matrix(c(1.5,1.5),nrow=2),1e+
plot(funcion(resultado3[1,],resultado3[2,]), main="Nweton PI=[1.5,1.5]", type="o", xlab="Iteracion", yla
```




```
aum <- 1
pintar_frontera(funcion,resultado3,c(resultado3[1,ncol(resultado3)]-aum,resultado3[1,ncol(resultado3)]+a
```



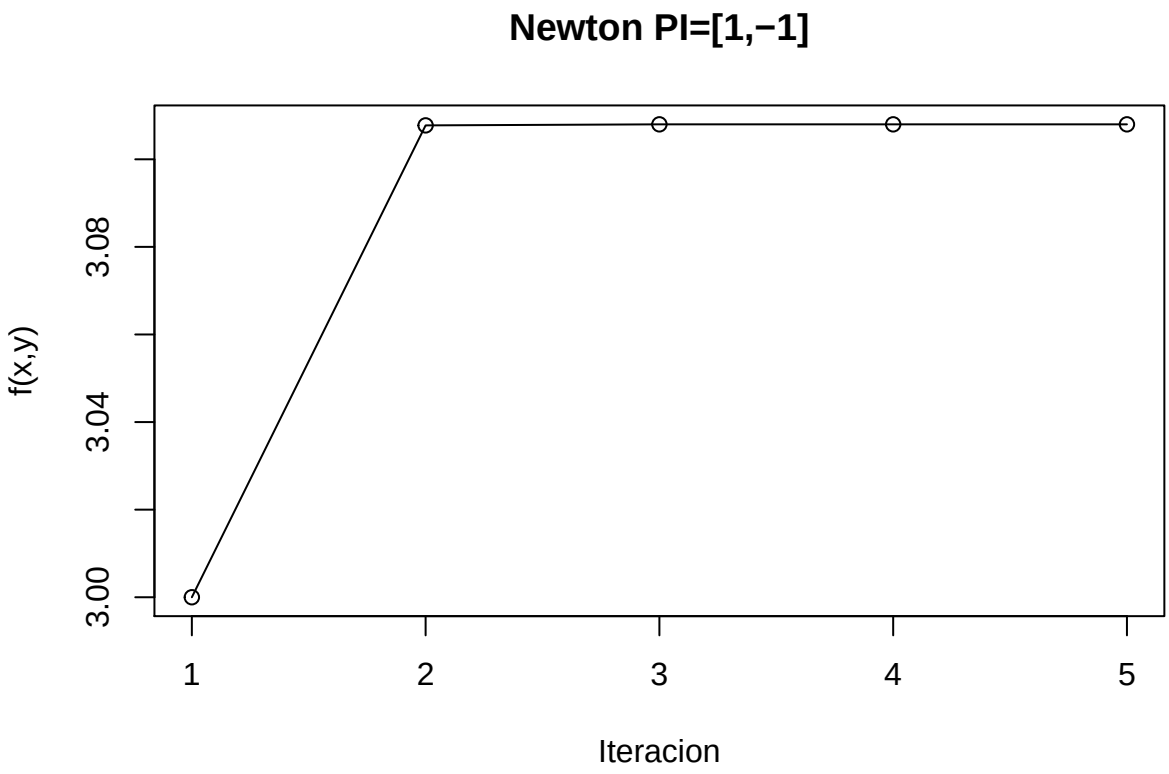

```
# Obtenemos en el punto [1.7048310,0.9731716] el valor de 18.08874 con 12 iteraciones

#4
resultado3 <- newton2(funcion,derivada,derivadaxx,derivadaxy,derivadayy,as.matrix(c(1,-1),nrow=2),1e+10
resultado3
```

```
## [1]  1 -1  1
```

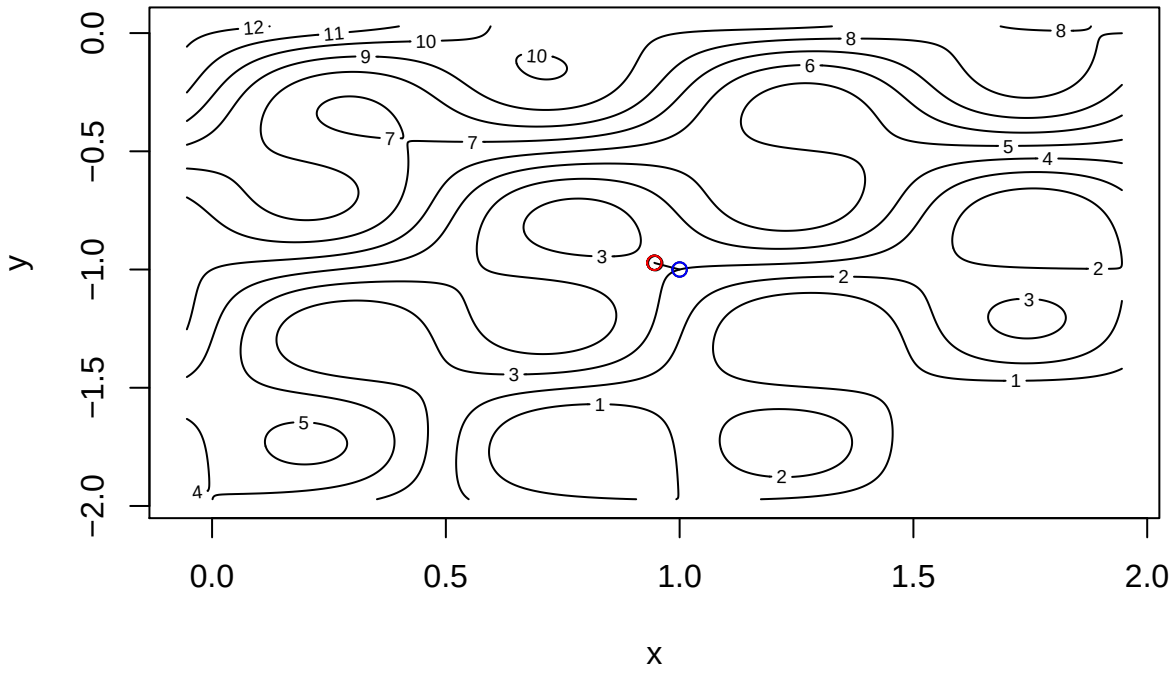
```
funcion(resultado3[1],resultado3[2])
```

```
## [1] 3
```

```
resultado3 <- newton(funcion,derivada,derivadaxx,derivadaxy,derivadayy,as.matrix(c(1,-1),nrow=2),1e+10,a
plot(funcion(resultado3[1,],resultado3[2,]), main="Newton PI=[1,-1]", type="o", xlab="Iteracion", ylab=
```




```
aum <- 1
pintar_frontera(funcion,resultado3,c(resultado3[1,ncol(resultado3)]-aum,resultado3[1,ncol(resultado3)]+a
```




```
# Obtenemos en el punto [1,-1] el valor de 3 con 1 iteraciones
```

Podemos ver comparando las gráficas que obtuvimos en el ejercicio3 (página 6) que el método de Newton por lo general tarda menos iteraciones en cumplir los términos que hemos puesto de parada, sin embargo podemos ver en las gráficas que en el caso 2 y 4 no ha conseguido minimizar ya que no sabe salir de la colina en la que se encuentra hasta llegar al minimo local mas cercano. En los otros dos casos obtenemos unos buenos resultados pero en todo caso podemos diferenciar claramente el avance de un método y otro con los parametros que hemos establecido de umbral $= 10^{-14}$ y un numero de iteraciones 10^{10} (aunque nunca lleguemos).