Trabajo 1 | Programación

Antonio Miguel Morillo Chica

22/03/2018

1. Búsqueda iteractiva de óptimos

1.1 Implementar el algoritmo de gradiante descendente.

Lo primero que hacemos es definir la función dada y creamos otra función para el calculo de la derivadas parciales respecto x e y que se usarán para el *graciante descendente*.

La última función implementa el GD. Además almacena los puntos de la función pasada para poder crear graficas que nos pedirán en futuros partados.

```
E <- function(u,v){
  (u^(3) * exp(v-2) - 4*v^(3)*exp(-u))^2
}

dE <- function(u,v){
  duE <- 2 *  ((u^3)*exp(v-2) - 4* exp(-u)*v^3) * (3*u^(2)*exp(v-2) + 4*exp(-u)*v^3 )
  dvE <- 2 *  ((u^3)*exp(v-2) - 12*exp(-u)*v^2) * (  u^(3)*exp(v-2) - 4*exp(-u)*v^3 )
  c(duE, dvE)
}

GD <- function(f, f_prima, t_aprendizaje = 0.1, umbral = 10^(-14), nIter=5000,
  pInicial=c(1,1)){

  w_anterior <- c(0,0)
  w_nuevo <- pInicial

  iters <- 0
  datos <- rep(nIter)
  datos[iters] <- f(w_nuevo[1], w_nuevo[2])

  cat("\nInicio: ", w_nuevo[1], w_nuevo[2])
  while( (iters < nIter) &  ( f(w_nuevo[1],w_nuevo[2]) > umbral) &
         (abs(f(w_nuevo[1], w_nuevo[2]) - f(w_anterior[1], w_anterior[2])) > umbral) ) {

    w_anterior = w_nuevo

    gradiante <- f_prima(w_nuevo[1], w_nuevo[2])
    n_gradiante <- gradiante * (-1)
    w_nuevo <- w_anterior + t_aprendizaje * n_gradiante
    iters <- iters + 1
    datos[iters] <- f(w_nuevo[1], w_nuevo[2])
  }
  cat("\nSolución: ", w_nuevo[1], w_nuevo[2])
  cat("\nIteracciones:", iters)

  list(valor_funcion = datos[1:iters])
}
```

1.2 Considerar la función E(u,v) = (u³e^(v-2) - 4v³e^(-u))². Usar el gradiante descendente para encontrar un mínimo de esta función, comenzando desde el punto (u, v) = (1, 1) y usando una tasa de aprendizaje mu = 0,1.

- a) Calcular analiticamente y mostrar la expresión del gradiante de la función E(u,v)
- b) ¿Cuántas iteraciones tarda el algoritmo en obtener por primera vez un valor de E(u, v) inferior a 10^-14 . (Usar flotantes de 64 bits)

  Como podemos ver tarda 38 iteracciones con una trasa de aprendizaje de 0.5 y con su punto inicial en el (1,1)

```
GD(E, dE, t_aprendizaje=0.05, umbral=10^(-14), nIter=500, pInicial=c(1,1))
```

```
##
## Inicio:  1 1
## Solución:  1.119544 0.6539881
## Iteracciones: 38
```

```
## $valor_funcion
##  [1] 9.659172e-02 6.007327e-02 3.634715e-02 2.104592e-02 1.156017e-02
##  [6] 6.011457e-03 2.970092e-03 1.404483e-03 6.413349e-04 2.852591e-04
## [11] 1.245051e-04 5.363214e-05 2.289720e-05 9.717265e-06 4.107607e-06
## [16] 1.731833e-06 7.289306e-07 3.064687e-07 1.287579e-07 5.407030e-08
## [21] 2.269928e-08 9.527528e-09 3.998463e-09 1.677915e-09 7.040829e-10
## [26] 2.954354e-10 1.239628e-10 5.201327e-11 2.182392e-11 9.156905e-12
## [31] 3.842049e-12 1.612041e-12 6.763764e-13 2.837921e-13 1.190726e-13
## [36] 4.996011e-14 2.096210e-14 8.795204e-15
```

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

  Para un valor inferior a 10^-14, concretamente $10^{(-14)-10}(-15)$ y una tasa de aprendizaje de 0.05 y como punto inicial, las coordenadas (u,v) donde se alcanzó fue: (1.119544, 0.6539881)

```
GD(E, dE, t_aprendizaje=0.05, umbral=10^(-15), nIter=500, pInicial=c(1,1))
```

```
##
## Inicio:  1 1
## Solución:  1.119544 0.6539881
## Iteracciones: 41
```

```
## $valor_funcion
##  [1] 9.659172e-02 6.007327e-02 3.634715e-02 2.104592e-02 1.156017e-02
##  [6] 6.011457e-03 2.970092e-03 1.404483e-03 6.413349e-04 2.852591e-04
## [11] 1.245051e-04 5.363214e-05 2.289720e-05 9.717265e-06 4.107607e-06
## [16] 1.731833e-06 7.289306e-07 3.064687e-07 1.287579e-07 5.407030e-08
## [21] 2.269928e-08 9.527528e-09 3.998463e-09 1.677915e-09 7.040829e-10
## [26] 2.954354e-10 1.239628e-10 5.201327e-11 2.182392e-11 9.156905e-12
## [31] 3.842049e-12 1.612041e-12 6.763764e-13 2.837921e-13 1.190726e-13
## [36] 4.996011e-14 2.096210e-14 8.795204e-15 3.690261e-15 1.548347e-15
## [41] 6.496498e-16
```

1.3 Considerar ahora la función f(x,y) = (x-2)^2 + 2(y+2)^2 + 2 sin(2pi*x) sin(2pi*y)

Lo primero que hacemos es definir la constante pi ya que no viene por defecto en R. Posteriormente creamos la función dada y creamos otra función para el calculo de la derivadas parciales respecto x e y que se usarán para el *graciante descendente*.

```
F <- function(x,y){
  (x-2)^(2)+2*(y+2)^(2)+2*sin(2*pi*x)*sin(2*pi*y)
}

dF <- function(x,y){
  dxF <- 2*(2*pi*cos(2*pi*x)*sin(2*pi*y)+x-2)
  dyF <- 4*(pi*sin(2*pi*x)*cos(2*pi*y)+y+2)
  c(dxF,dyF)
}
```

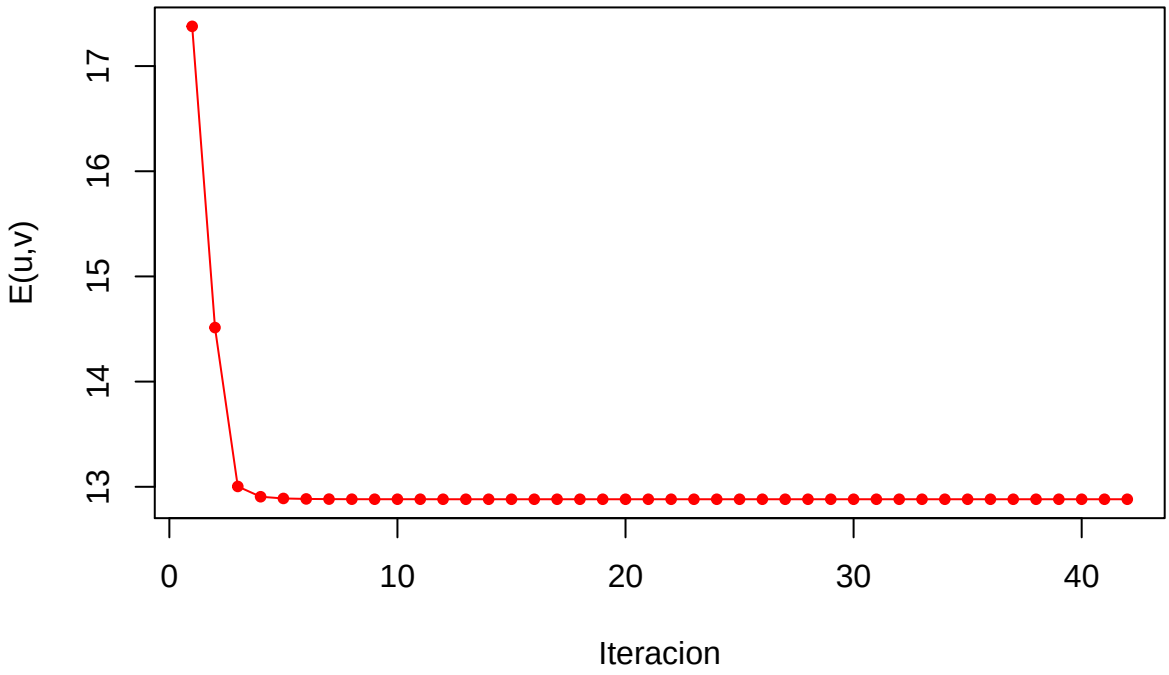
a) **Usar gradiente descendente para minimizar esta función. Usar como punto inicial (x_0=1,y_0=1), tasa de aprendizaje mu=0,01 y un máximo de 50 iteraciones. Generar un gráfico de cómo desciende el valor de la función con las iteraciones. Repetir el experimento pero usando mu=0,1, comentar las diferencias y su dependencia de mu.**

Como podemos observar en las diguientes gráficas existen diferencias muy significativas, en la primera gracias que la tasa de aprendizaje es muy pequeña tarda algunas iteracciones en ajustarse al máximo, en cambio en la segunda como es más rapida solo le bantan 3 iteracciones. La primera es muy buena pero gastamos muchas iterracciones por lo que si ajustasemos la tasa de aprendizaje ligeramente más alta tendríamos una buena aproximación sin hacer tantas iteracciones.

```
# mu = 0.01
datos1 = GD(F,dF, t_aprendizaje=0.01, umbral=10^(-14), nIter=50, pInicial=c(1,1))
```

```
##
## Inicio:  1 1
## Solución:  1.284043 0.5901838
## Iteracciones: 42
```

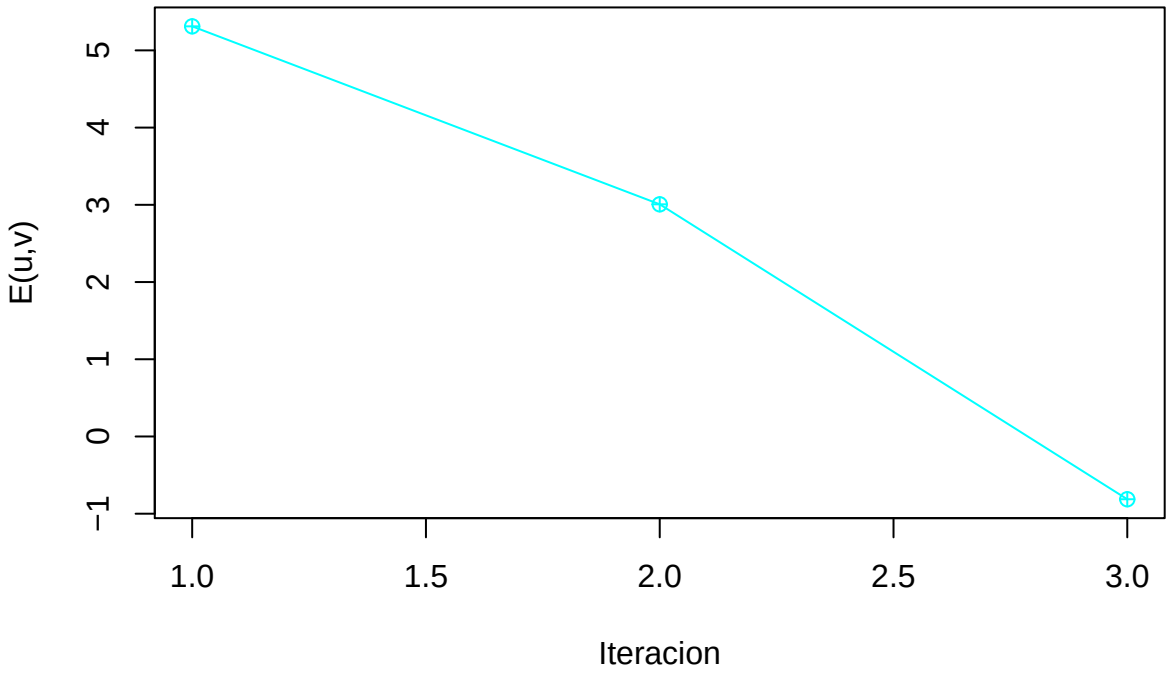
```
plot(datos1$valor_funcion, type = "o", pch = 20, col = 50, xlab = "Iteracion",
     ylab = "E(u,v)", main = "Gradiente Descendiente E(u,v)")
```




```
Sys.sleep(3)

# mu = 0.1
datos2 = GD(F,dF, t_aprendizaje=0.1,  umbral=10^(-14), nIter=50, pInicial=c(1,1))
```

```
##
## Inicio:  1 1
## Solución:  1.625545 -1.87828
## Iteracciones: 3
```

```
plot(datos2$valor_funcion, type = "o", pch = 10, col = 5, xlab = "Iteracion",
     ylab ="E(u,v)", main = "Gradiente Descendiente E(u,v)")
```



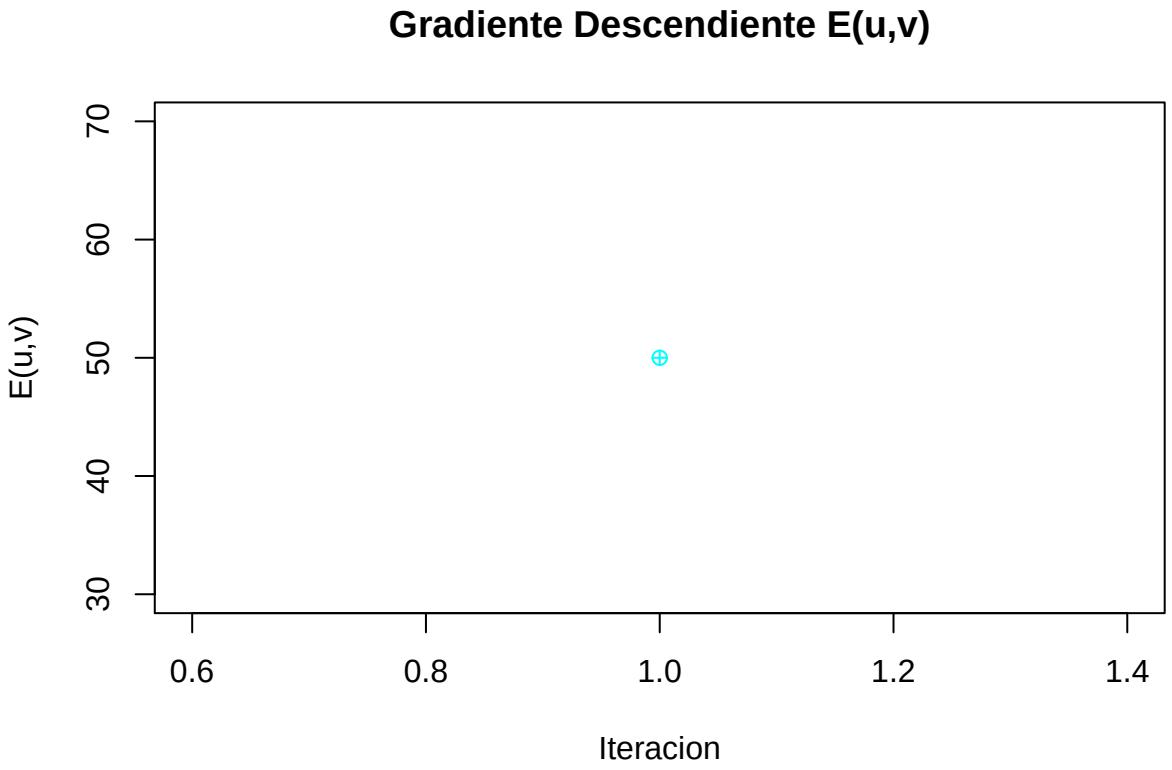

```
Sys.sleep(3)
```

- 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.**

```
datos3 = GD(F,dF, t_aprendizaje=0.1, umbral=10^(-14), nIter=50, pInicial=c(2.1,-2.1))
```

```
##
## Inicio:  2.1 -2.1
## Solución:  2.1 -2.1
## Iteracciones: 0
```

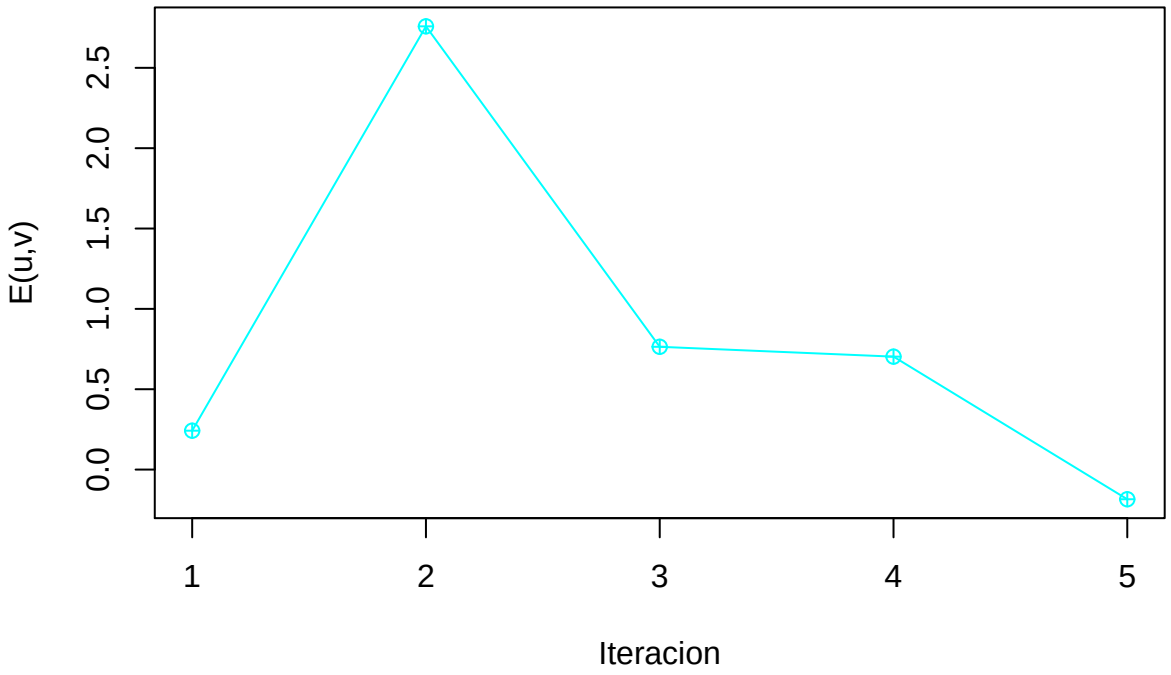
```
plot(datos3$valor_funcion, type = "o", pch = 10, col = 5, xlab = "Iteracion",
     ylab ="E(u,v)", main = "Gradiente Descendiente E(u,v)")
```




```
Sys.sleep(3)

datos4 = GD(F,dF, t_aprendizaje=0.1, umbral=10^(-14), nIter=50, pInicial=c(3,-3))
```

```
##
## Inicio:  3 -3
## Solución:  1.22095 -1.235948
## Iteracciones: 5
```

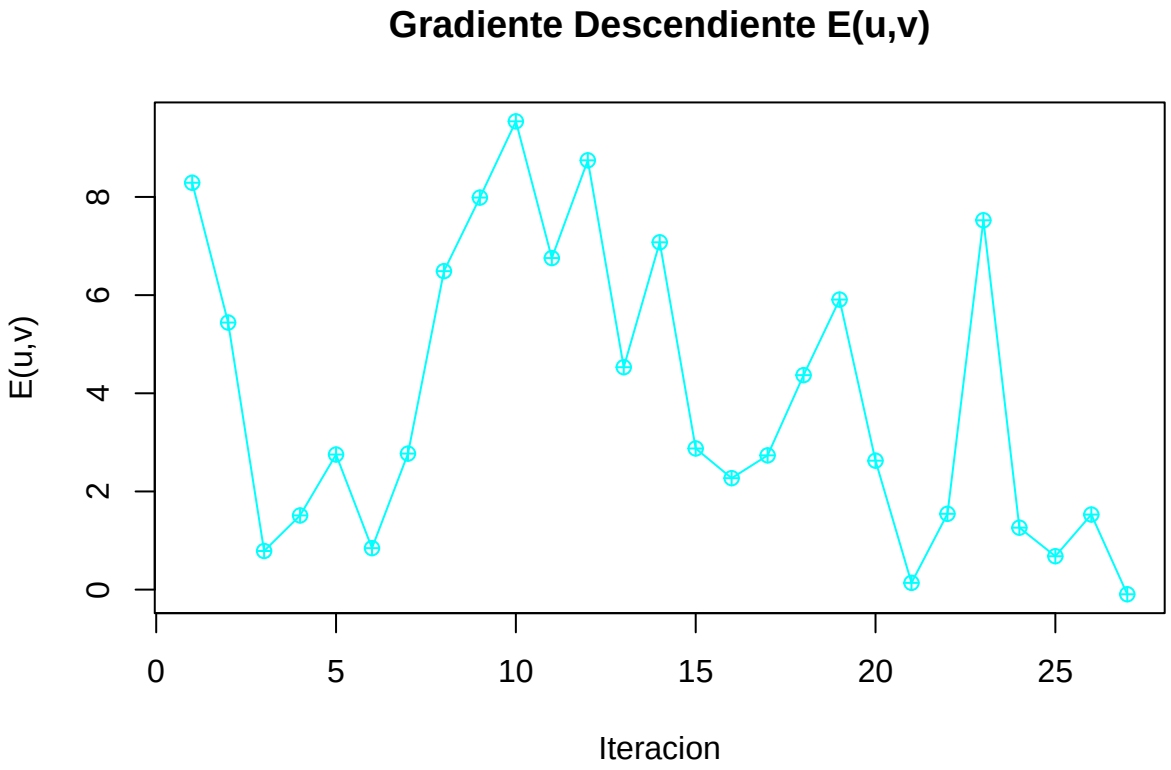
```
plot(datos4$valor_funcion, type = "o", pch = 10, col = 5, xlab = "Iteracion",
     ylab ="E(u,v)", main = "Gradiente Descendiente E(u,v)")
```




```
Sys.sleep(3)

datos5 = GD(F,dF, t_aprendizaje=0.1, umbral=10^(-14), nIter=50, pInicial=c(1.5,1.5))
```

```
##
## Inicio:  1.5 1.5
## Solución:  2.094153 -2.430249
## Iteracciones: 27
```

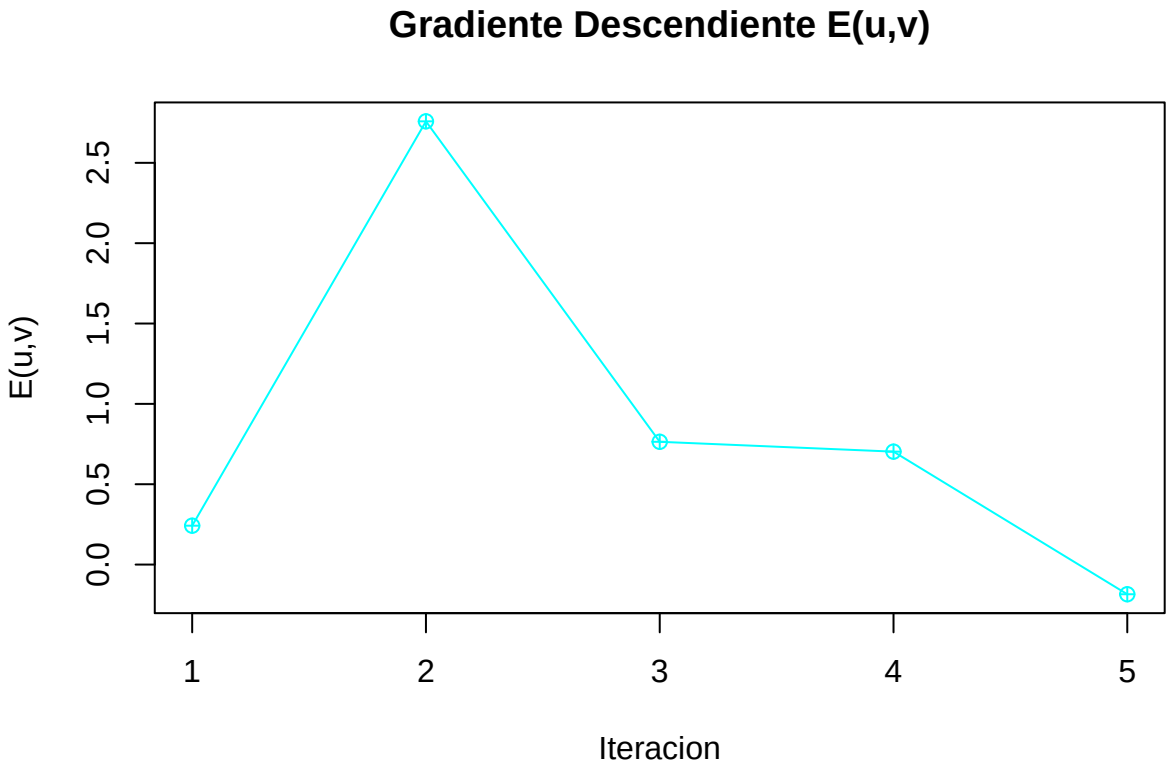
```
plot(datos5$valor_funcion, type = "o", pch = 10, col = 5, xlab = "Iteracion",
     ylab ="E(u,v)", main = "Gradiente Descendiente E(u,v)")
```



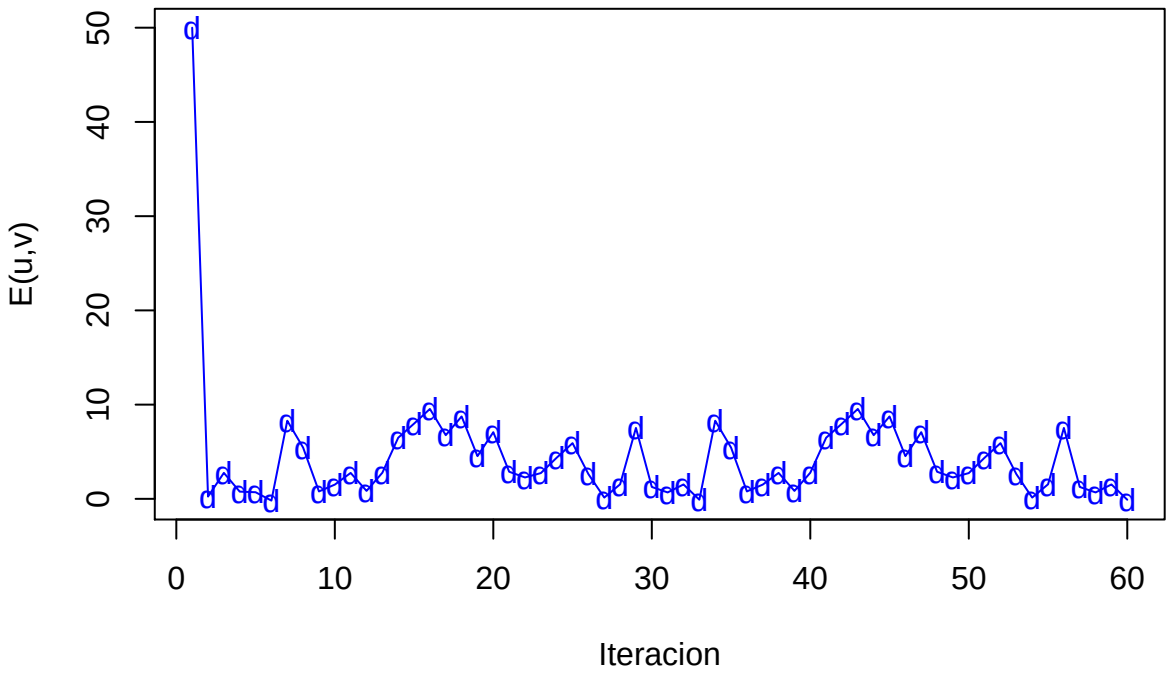

```
Sys.sleep(3)

datos6 = GD(F,dF, t_aprendizaje=0.1, umbral=10^(-14), nIter=50, pInicial=c(1,-1))
```

```
##
## Inicio:  1 -1
## Solución:  2.77905 -2.764052
## Iteracciones: 5
```

```
plot(datos6$valor_funcion, type = "o", pch = 10, col = 5, xlab = "Iteracion",
     ylab ="E(u,v)", main = "Gradiente Descendiente E(u,v)")
```




```
Sys.sleep(3)

todos <- c(datos3$valor_funcion, datos4$valor_funcion, datos5$valor_funcion, datos5$valor_funcion)
plot(todos, type = "o", pch = 100, col = 100, xlab = "Iteracion",
     ylab ="E(u,v)", main = "Gradiente Descendiente E(u,v)")
```



```
Sys.sleep(3)
```

1.4 ¿Cuál sería su conclusión sobre la verdadera dificultad de encontrar el mínimo global de una función arbitraria?

La conclusión es que la dificultad de contrar mínimos globales reside en conseguir ajustar una tasa de aprendizaje adecuada para que el ajuste del gradiante sea lo mejor posible.

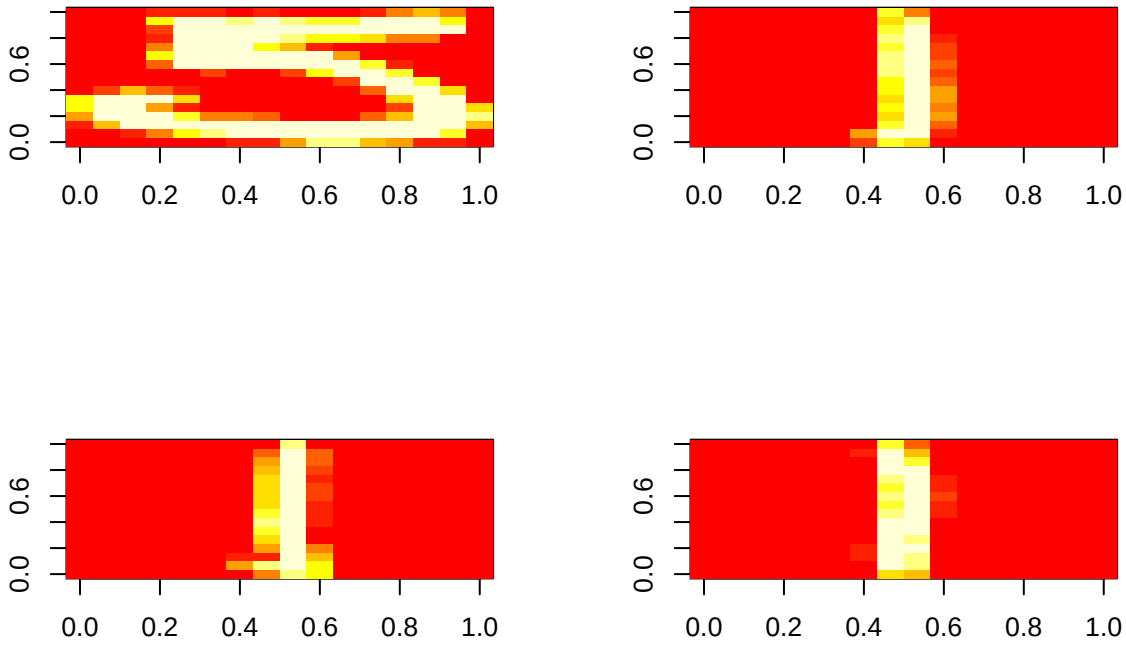
2. Regresión Lineal

Este ejercicio ajusta modelos de regresión a vectores de características extraidos de imágenes de digitos manuscritos. En particular se extraen dos característcas concretas: el valor medio del nivel de gris y simetría del número respecto de su eje vertical. Solo se seleccionarán para este ejercicio las imágenes de los números 1 y 5.

2.1 Estimar un modelo de regresión lineal a partir de los datos proporcionados de dichos números (Intensidad promedio, Simetria) usando tanto el algoritmo de la pseudo-inversa 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 E_in y E_out (para E out calcular las predicciones usando los datos del fichero de test, usar Regress_Lin(datos, label) como llamada para la función (opcional)).

1. Lectura del train

```
digit.train <- read.table("datos/zip.train", quote="\"", comment.char="",
                          stringsAsFactors=FALSE)

# Guardamos los 4 y los 8
digitos15.train = digit.train[digit.train$V1==1 | digit.train$V1==5,]

# Etiquetas
digitos.train = digitos15.train[,1]
ndigitos.train = nrow(digitos15.train)

# Se retira la clase y se monta una matriz de tamanio 16x16
grises = array(unlist(subset(digitos15.train,select=-V1)),c(ndigitos.train,16,16))
grises.train = lapply(seq(dim(grises)[1]), function(m) {matrix(grises[m,,],16)})

# Visualizamos las imágenes
par(mfrow=c(2,2))

for(i in 1:4){
  imagen = grises[i,,16:1] # Se rotan para verlas bien
  image(z=imagen)
}
```




```
Sys.sleep(3)
```

2. Intensidad de train:

```
intensidad.train = unlist(lapply(grises.train, FUN=mean))
```

3. Simetria del train:

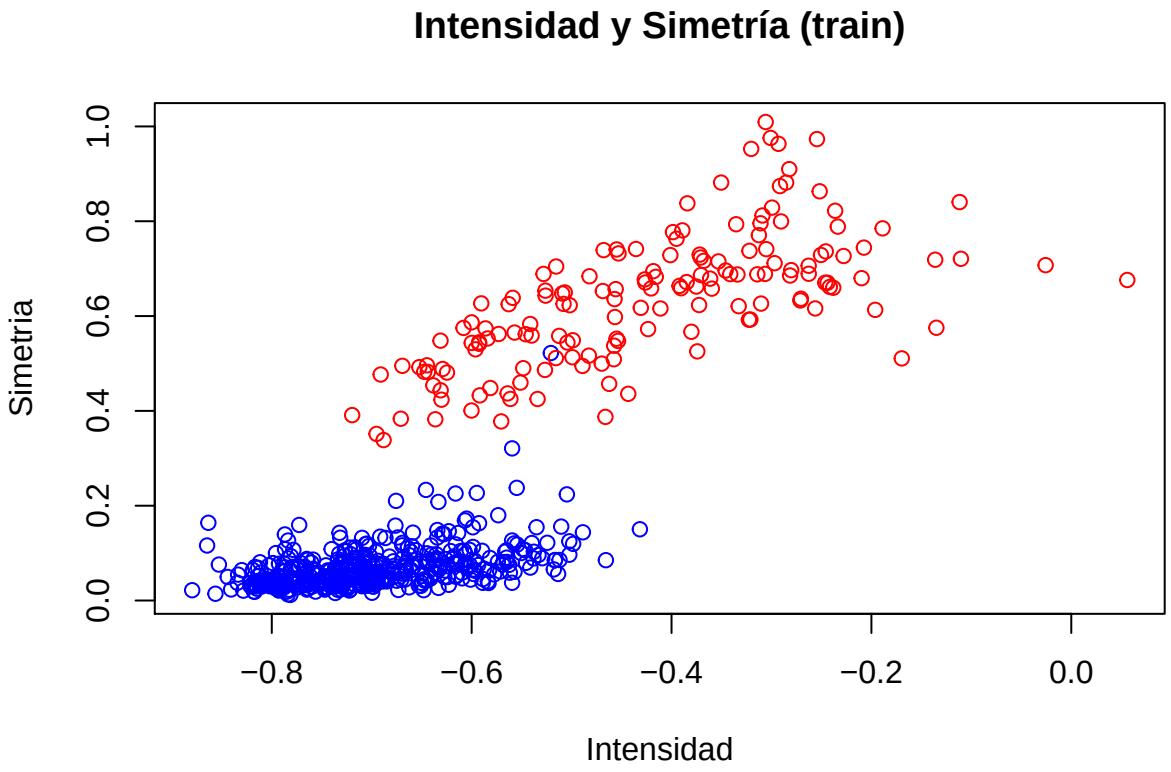
```
simetria <- function(matriz){

  matriz_original = matriz[1:256]
  matriz_invertida = matriz[,ncol(matriz):1]
  diferencia_matriz = matriz_original - matriz_invertida
  simetria = mean(abs(diferencia_matriz))

  simetria
}
```

4. Rerecodificar digitos del train:

```
etiquetas15.train <- digitos15.train[,1]
etiquetas15.train <- (etiquetas15.train-3)/(-2)
```

5. Composición de los datos para mostrarlos:

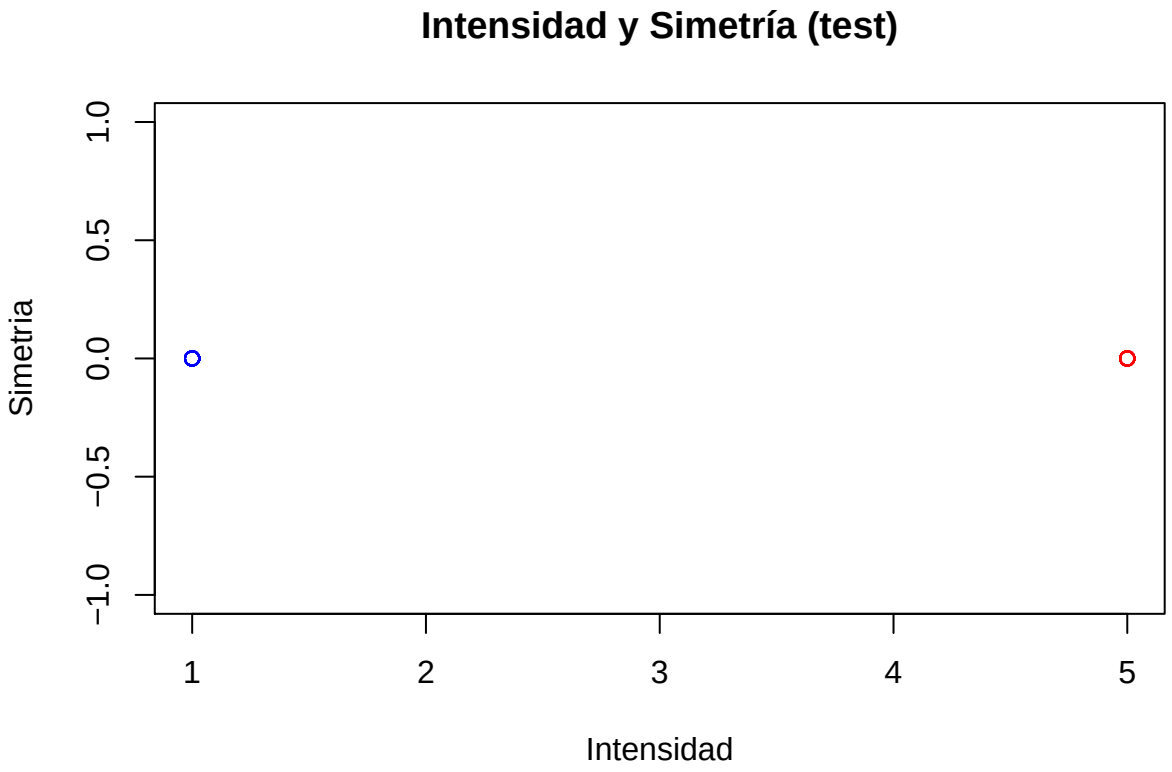
```
# Guardamos
simetria.train = unlist(lapply(grises.train, simetria))

plot(x=intensidad.train, y=simetria.train, col=etiquetas15.train+3, xlab = "Intensidad",
     ylab = "Simetria", main="Intensidad y Simetría (train)")
```




```
Sys.sleep(3)
```

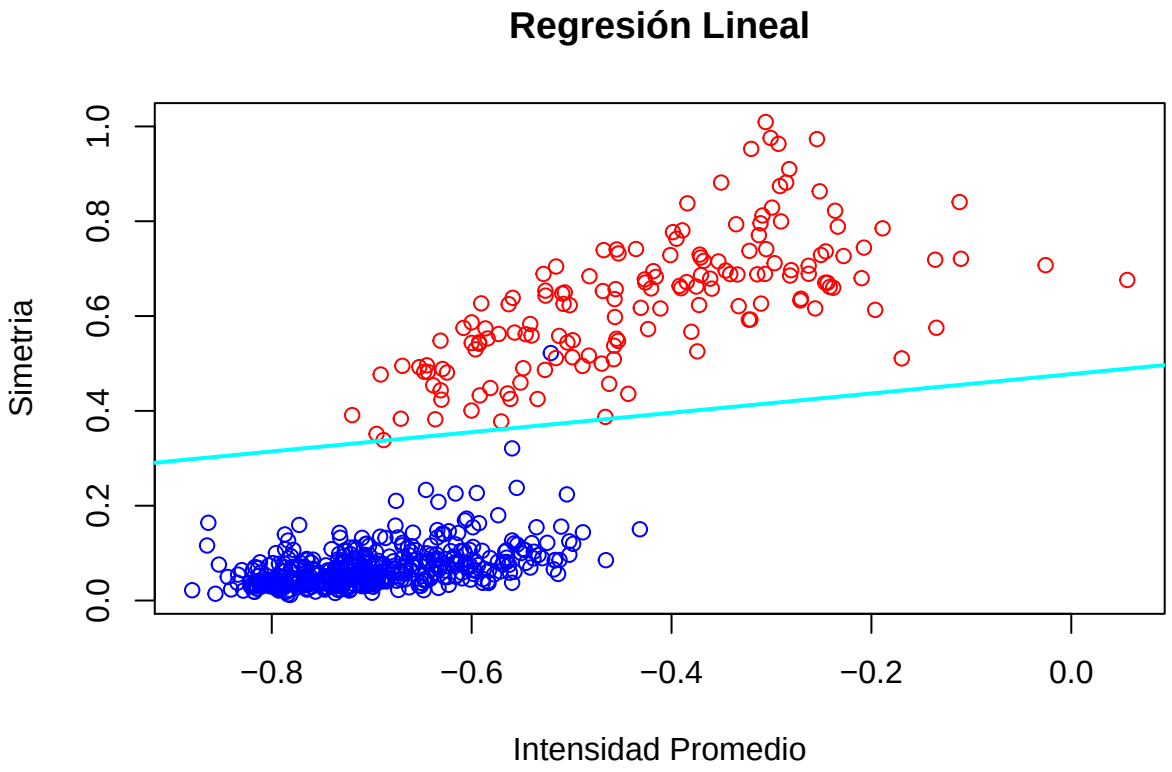
6. Los paso 6,7,8 y 9 son repetir para el test:

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

# Guardamos los 4 y los 8
digitos15.test = digit.test[digit.test[,1]==1 | digit.test[,1]==5,]

# Etiquetas
digitos.test = digitos15.test[,1]
ndigitos.test = nrow(digitos15.test)

# Se retira la clase y se monta una matriz de tamanio 16x16
grises = array(unlist(subset(digitos15.test,select=V1)),c(ndigitos.test,16,16))
grises.test = lapply(seq(dim(grises)[1]), function(m) {matrix(grises[m,,],16)})

intensidad.test = unlist(lapply(grises.test, FUN=mean))

etiquetas15.test <- digitos15.test[,1]
# Convertimos las --etiquetas
etiquetas15.test <- (etiquetas15.test-3)/(-2)

simetria.test = unlist(lapply(grises.test, simetria))

plot(x=intensidad.test, y=simetria.test, col=etiquetas15.test+3, xlab = "Intensidad",
     ylab = "Simetria", main="Intensidad y Simetría (test)")
```



11. Implementación de la regresión lineal para obtención de los pesos, además pinto la grafica de la recta:

```
Regression_Lin <- function(datos, etiq){
  x <- cbind(1, data.matrix(datos))

  x.svd <- svd(x)

  v <- x.svd$v
  d <- x.svd$d

  diagonal <- diag(ifelse(d>0.0001, 1/d, d))
  xx <- v %*% (diagonal^2) %*% t(v)
  pseudoinversa <- xx %*% t(x)
  w <- pseudoinversa %*% as.numeric(etiq)
  c(w)
}

datos.train = cbind(intensidad.train, simetria.train)

plot(datos.train, xlab = "Intensidad Promedio", ylab = "Simetria",
     col = etiquetas15.train+3, main = "Regresión Lineal")

w = Regression_Lin(datos.train, etiquetas15.train)

abline(-w[1]/w[3], -w[2]/w[3], col = 5, lwd = 2)
```



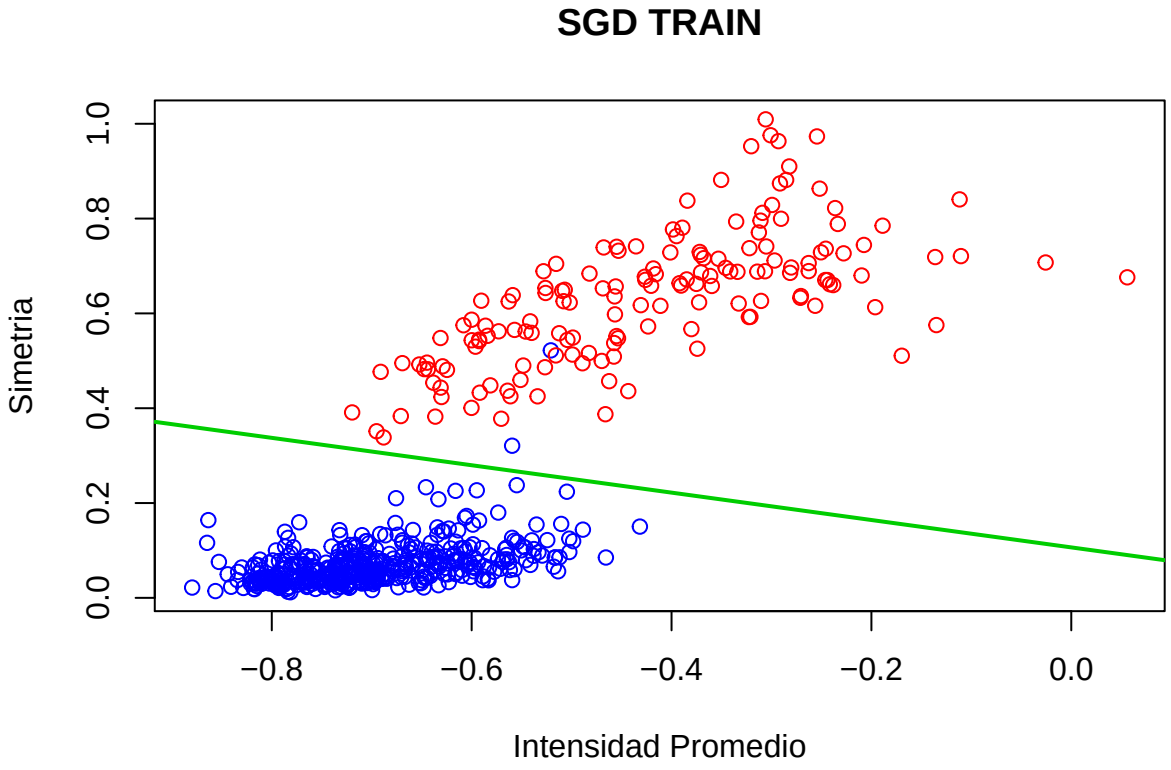
12. Gradiante descendente estocastico con función norma:

```r
norma <- function(w_old, w_new){
  sqrt(sum((w_old - w_new)^2))
}

SGD <- function(datos, etiquetas, vini = c(0,0,0), mu = 0.01, nIter = 500){

  w_nuevo <- vini
  w_anterior <- c(0,0,0)
  iters <- 0

  # Columna de 1's
  datos <- cbind(1, datos)
  seguir <- TRUE

  while (iters < nIter & seguir){
    w_anterior <- w_nuevo
    permutacion <- sample(1:length(etiquetas))

    for(i in permutacion){
      gradiente <- (-etiquetas[i]*datos[i,]) / (1 + exp(etiquetas[i] * w_nuevo %*%datos[i,]))
      w_nuevo <- w_nuevo - mu * gradiente
    }

    iters <- iters + 1

    if(norma(w_anterior, w_nuevo) < mu ) seguir <- FALSE
  }
```

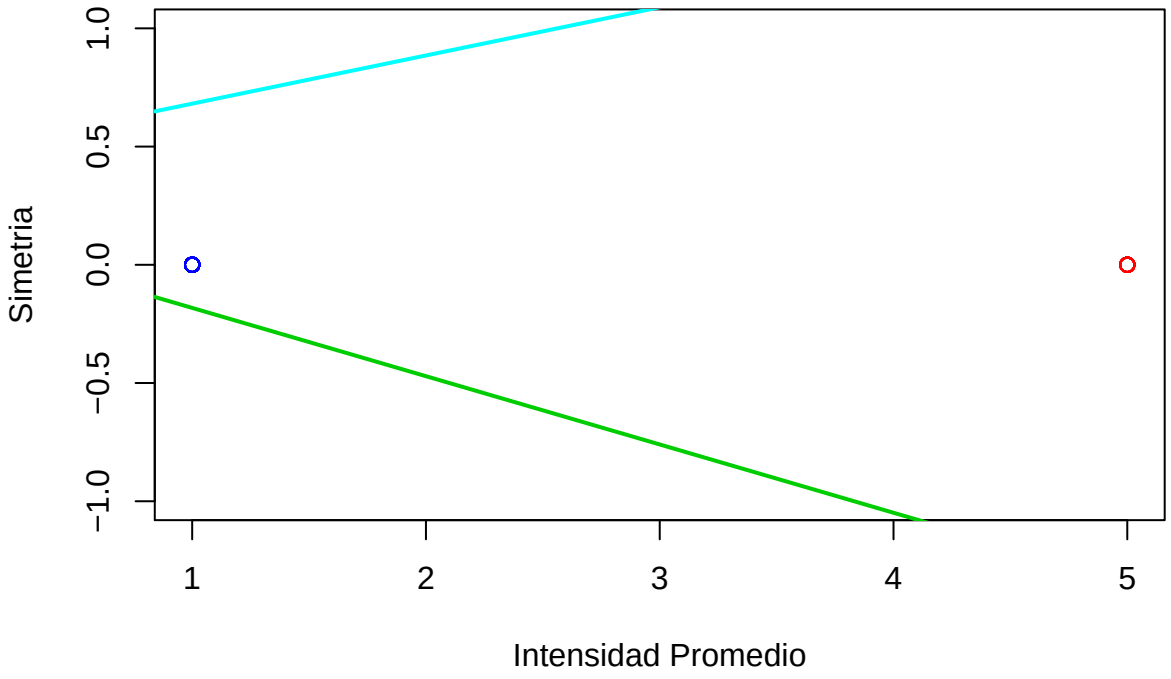
```
  cat("Pesos: ", w_nuevo, "\nValores de la Recta: ", -w_nuevo[1]/w_nuevo[3], -w_nuevo[2]/w_nuevo[3], "\n

  c(w_nuevo,iters)

}
set.seed(3) # se establece la semilla
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
}

simula_recta = function (intervalo = c(-1,1), visible=F){

   ptos = simula_unif(2,2,intervalo)
   a = (ptos[1,2] - ptos[2,2]) / (ptos[1,1]-ptos[2,1])
   b = ptos[1,2]-a*ptos[1,1]

   if (visible) {
       if (dev.cur()==1)
           plot(1, type="n", xlim=intervalo, ylim=intervalo)
       points(ptos,col=3)  #pinta en verde los puntos
       abline(b,a,col=3)   # y la recta
   }
   c(a,b) # devuelve el par pendiente y punto de corte
}
set.seed(3)

datos.train <- cbind(intensidad.train, simetria.train)
plot(datos.train, xlab = "Intensidad Promedio", ylab = "Simetria",
     col = etiquetas15.train+3, main = "SGD TRAIN")
Sys.sleep(3)

pesos01.train <-Regression_Lin(datos.train, etiquetas15.train)
pesos02.train <- SGD(datos.train, etiquetas15.train, pesos01.train, 100)
```

```
## Pesos:  43.52736 -118.0748 -408.853
## Valores de la Recta:  0.1064621 -0.2887952
## Iteraciones:  2
```

```
abline(-pesos02.train[1]/pesos02.train[3], -pesos02.train[2]/pesos02.train[3],
       col = 3, lwd = 2)
```




```
datos.test <- cbind(intensidad.test, simetria.test)

plot(datos.test, xlab = "Intensidad Promedio", ylab = "Simetria",
     col = etiquetas15.test+3, main = "SGD TEST")
Sys.sleep(3)

abline(-pesos01.train[1]/pesos01.train[3], -pesos01.train[2]/pesos01.train[3],
       col = 5, lwd = 2)

abline(-pesos02.train[1]/pesos02.train[3], -pesos02.train[2]/pesos02.train[3],  col = 3, lwd = 2)
```




```
recta_regresion03 <- c(-pesos02.train[1]/pesos02.train[3],
                       - pesos02.train[2]/pesos02.train[3])

# Etiquetamos los puntos a partir de la recta de regresión
etiquetas_regresion01 <- sign(datos.train[,2]
    - recta_regresion03[1]*datos.train[,1]
    - recta_regresion03[2])


Ein <- 0
Ein <- sum(etiquetas15.train != etiquetas_regresion01) / length(etiquetas15.train)

Ein
```

```
## [1] 0.2621035
```

```
recta_regresion03 = c(-pesos02.train[1]/pesos02.train[3],
                       - pesos02.train[2]/pesos02.train[3])

# Etiquetamos los puntos a partir de la recta de regresión
etiquetas_regresion01 <- sign(datos.test[,2] -
  recta_regresion03[1]*datos.test[,1] - recta_regresion03[2])

Etest <- 0
Etest = sum(etiquetas15.test != etiquetas_regresion01) / length(etiquetas15.test)

Etest
```

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

```r
# Obtenemos el tamaño de los datos
N <- nrow(datos.train)

# Calculamos el segundo término de la fórmula
x <- sqrt( (8/N) * log((4*((2*N)^(3) + 1))/0.05) )

# Obtenemos el valor de Eout
cota_Ein_Eout <- Ein + x
cat ("Cota en E_in: ", cota_Ein_Eout)
```

```
## Cota en E_in:  0.8473678
```

```r
# Obtenemos el tamaño de los datos
N <- nrow(datos.test)

# Calculamos el segundo término de la fórmula
x <- sqrt( (1/(2*N)) * log( 2 / 0.05) )

cota_Ein_Eout <- Etest + x
cat ("Cota en E_test: ", cota_Ein_Eout)
```

```
## Cota en E_test:  0.1940145
```

```r
cota_Ein_Eout
```

```
## [1] 0.1940145
```