Practica 2

Rafael Nogales

17 de abril de 2016

```
pintar_grafica = function(f) {
    x=y=seq(-50,50,by=0.1)
    z = outer(x,y,FUN=f)
    contour(x,y,z, level=0, drawlabels = FALSE,add = TRUE)  # añade el gráfico
    contour(x,y,z, level=0.5, drawlabels = FALSE,add = TRUE)  # añade el gráfico
    contour(x,y,z, level=1, drawlabels = FALSE,add = TRUE)  # añade el gráfico
    contour(x,y,z, level=2, drawlabels = FALSE,add = TRUE)  # añade el gráfico
    contour(x,y,z, level=3, drawlabels = FALSE,add = TRUE)  # añade el gráfico

}

pinta_puntos = function(m, rangox = NULL, rangoy = NULL ,etiqueta=NULL){
    nptos=nrow(m)
    long = ncol=m

    if(is.null(rangox) && is.null(rangoy)){
        rangox = range(m[,1])
        rangoy = range(m[,1])
    }
    else if(is.null(rangoy))
        rangoy = rangox

    if(is.null(etiqueta))
        etiqueta = 1
    else etiqueta = etiqueta+2

    plot(m,xlab=paste("Pinta ",nptos," Puntos"), ylab="",
         xlim=rangox, ylim=rangoy,col=etiqueta,pch=19)
}
```

Modelos lineales

Gradiente descendente

Implementar el algoritmo de gradiente descendente. a) Considerar la función no lineal de error $E(u,v) = (ue^v - 2ve^{-u})^2$. Usar gradiente descendente y minimizar esta función de error, comenzando desde el punto $(u,v) = (1,1)$ y usando una tasa de aprendizaje $\eta = 0,1$.

1) Calcular analíticamente y mostrar la expresión del gradiente de la función $E(u,v)$
   Solucion basta considerar E como un campo escalar ($E : \mathbb{R}^2 \to \mathbb{R}$) y calcular su gradiente ∇E
   En este caso

$$\nabla E = (\frac{\partial E}{\partial u}, \frac{\partial E}{\partial v}) = (2(ue^v - 2ve^{-u})(e^v + 2ve^{-u}), 2(ue^v - 2ve^{-u})(ue^v - 2e^{-u}))$$

2) ¿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) .

R usa flotantes de 64 bits para ello podemos comprobarlo viendo que la operación $2 - \sqrt{2}^2$ da error a partir de la posición 16:

```
2 - sqrt(2)^2
```

```
## [1] -4.440892e-16
```

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

gradE <- function(u, v){
    Eu <- 2*(u*exp(v) - 2*v*exp(-u))*(exp(v) + 2*v*exp(-u))
    Ev <- 2*(u*exp(v) - 2*v*exp(-u))*(u*exp(v) - 2*exp(-u))
    return(c(Eu, Ev))
}
```

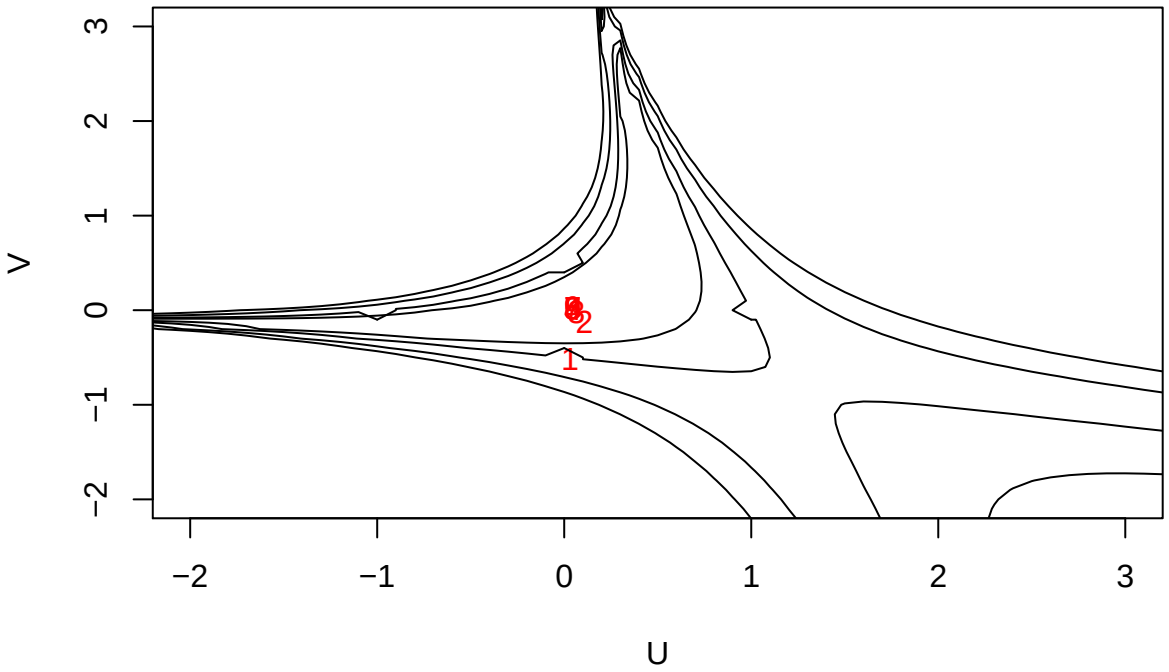
```
GradientDescent <- function(F, gradF, tasa=0.1, wIni=c(0,0), umbral = 10^(-14), max_iter = 100){
    t <- 0
    w <- wIni
    salir <- FALSE
    pointList <- c()

    norma <- function(v){
        sqrt(sum(v^2))
    }

    while(salir == FALSE){
        gt <- gradF(w[1], w[2])
        vt <- -gt
        w <- w + tasa*vt

        pointList[2*t-1] <- w[1]
        pointList[2*t] <- w[2]

        if(F(w[1], w[2]) < umbral){
            print(c("Valor de la funcion menor que umbral:", F(w[1], w[2]) ))
            print(c("Convergencia en ", t, "iteraciones"))

            salir <- TRUE
        }
        t <- t+1
        if(t > max_iter){
            print(c("Max iters alcanzado"))
            salir <- TRUE
        }

    }
    print(c("Stop en ", t-1, "iteraciones"))
    print(c("Valor de F al parar:", F(w[1], w[2]) ))
    print(c("Norma del gradiente al parar:", norma(gt) ))
    m <- matrix(pointList, byrow = TRUE, ncol=2)
    plot(m[,1], m[,2], xlim = c(-2, 3), ylim = c(-2, 3), pch=as.character(1:length(m[,1])), col=2, xlab=
```

```
    pintar_grafica(F)
}
```

Veamos como queda con la funcion E:

```
GradientDescent(E, gradE, wIni = c(1,1), tasa=0.1)
```

```
## [1] "Valor de la funcion menor que umbral:"
## [2] "1.20868339442207e-15"
## [1] "Convergencia en " "9"                "iteraciones"
## [1] "Stop en "    "9"           "iteraciones"
## [1] "Valor de F al parar:" "1.20868339442207e-15"
## [1] "Norma del gradiente al parar:" "2.01918461423345e-06"
```



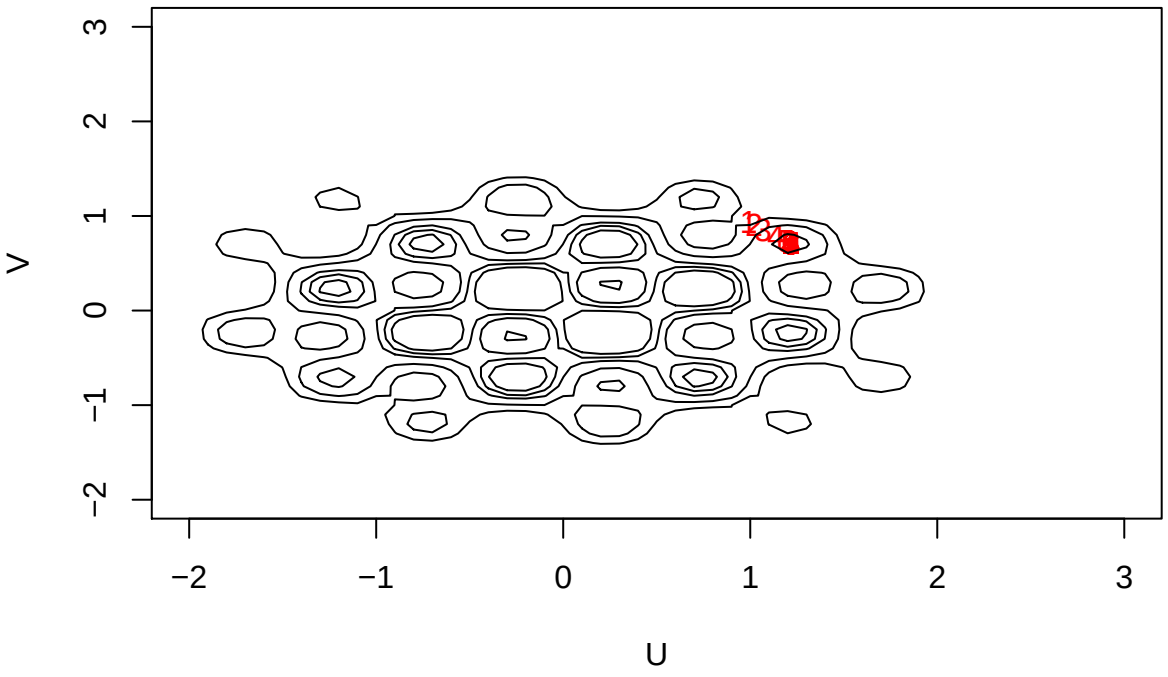
Ahora lo aplicamos a la función $F(x, y) = x^2 + 2y^2 + 2sin(2\pi x)sin(2\pi y)$ pero con una tasa de crecimiento de 0.01 y 50 iteraciones como máximo:

```
f <- function(x, y){
    #Pi con mas decimales
    PI <-  3.14159265358979323846264338 3

    x^2 + 2*y^2 + 2*sin(2*PI*x)*sin(2*PI*y)
}

gradF <- function(x, y){
    #Pi con mas decimales
    PI <-  3.14159265358979323846264338 3

    fx <- 2*x + 4*PI*cos(2*PI*x)*sin(2*PI*y)
    fy <- 4*y + 4*PI*sin(2*PI*x)*cos(2*PI*y)
```

```
    return(c(fx,fy))
}

hessF <- function(x, y){
     #Pi con mas decimales
    PI <-  3.14159265358979323846264338 3
    PI2 <- PI*PI

    fxx <- 2 - 8*PI2*sin(2*PI*x)*sin(2*PI*y)
    fxy <- 8*PI2 * cos(2*PI*x)*cos(2*PI*y)
    fyx <- fxy  # Lema de Schwarz
    fyy <- 4 - 8*PI2*sin(2*PI*x)*sin(2*PI*y)

    H <- matrix(c(fxx, fxy, fyx, fyy), ncol = 2)
}
```

```
GradientDescent(f, gradF, wIni = c(1,1), tasa=0.01, max_iter = 50)
```

```
## [1] "Max iters alcanzado"
## [1] "Stop en "    "50"          "iteraciones"
## [1] "Valor de F al parar:" "0.593269374325836"
## [1] "Norma del gradiente al parar:" "1.03578512786223e-14"
```

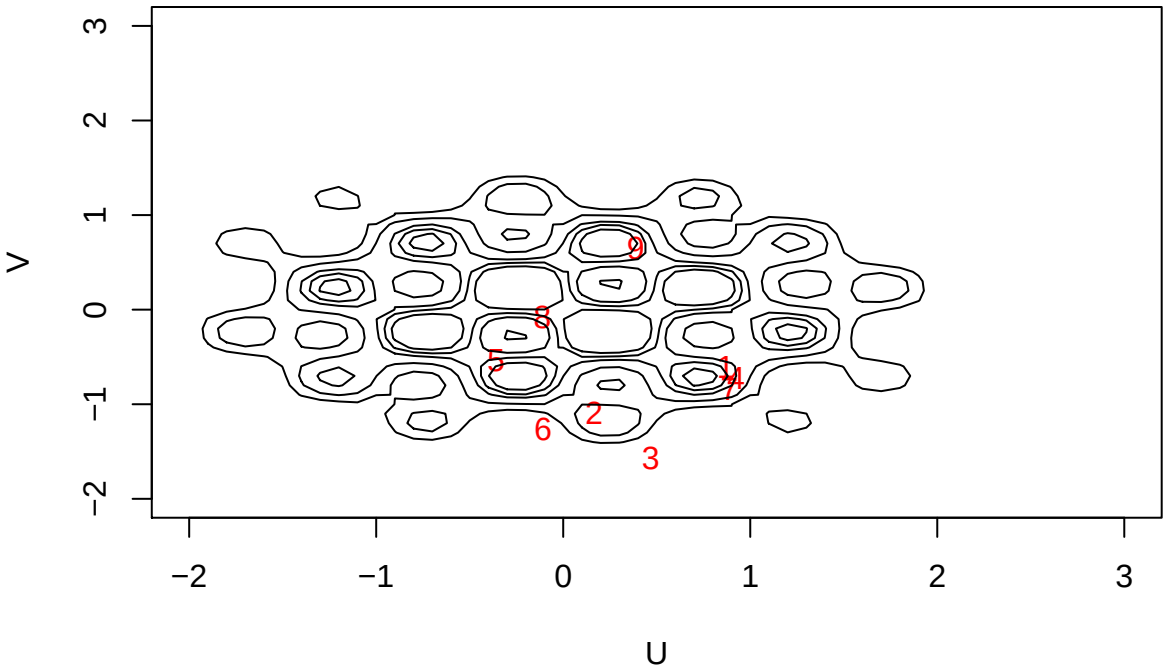


Veamos ahora que ocurre con una tasa de 0.1

```
GradientDescent(f, gradF, wIni = c(1,1), tasa=0.1, max_iter = 50)
```

```
## [1] "Valor de la funcion menor que umbral:"
## [2] "-0.0538800461756588"
## [1] "Convergencia en " "9"                "iteraciones"
```

```
## [1] "Stop en "    "9"           "iteraciones"
## [1] "Valor de F al parar:" "-0.0538800461756588"
## [1] "Norma del gradiente al parar:" "8.87664887332069"
```



Como vemos en el segundo caso no converge, el problema es que la tasa η es demasiado grande para este caso, el motivo es que nosotros multiplicamos $\eta\nabla f(x,y)$ y cuando $||\nabla f(x,y)||$ es grande dar un salto de $0.1*||\nabla f(x,y)||$ puede sacarnos del entorno del minimo (de hecho en esta función cada entorno concavo/convexo tiene un radio de 0.25 respecto del maximo/minimo local) y $||\nabla f(x,y)||$ suele ser mucho mayor.
La gráfica de F es un paraboloide eliptico con "piel de naranja" y nosotros estamos buscando en las proximidades del minimo global del paraboloide (la zona más llana) y aun asi nos salimos.
La solución debe ser cambiar el algoritmo para que no consideremos movernos respecto a $\nabla f(x,y)$ sino que nos movamos respecto $\frac{\nabla f(x,y)}{||\nabla f(x,y)||}$ (vector unitario en la direccion de $\nabla f(x,y)$) y tener una **tasa de movimiento variable**.

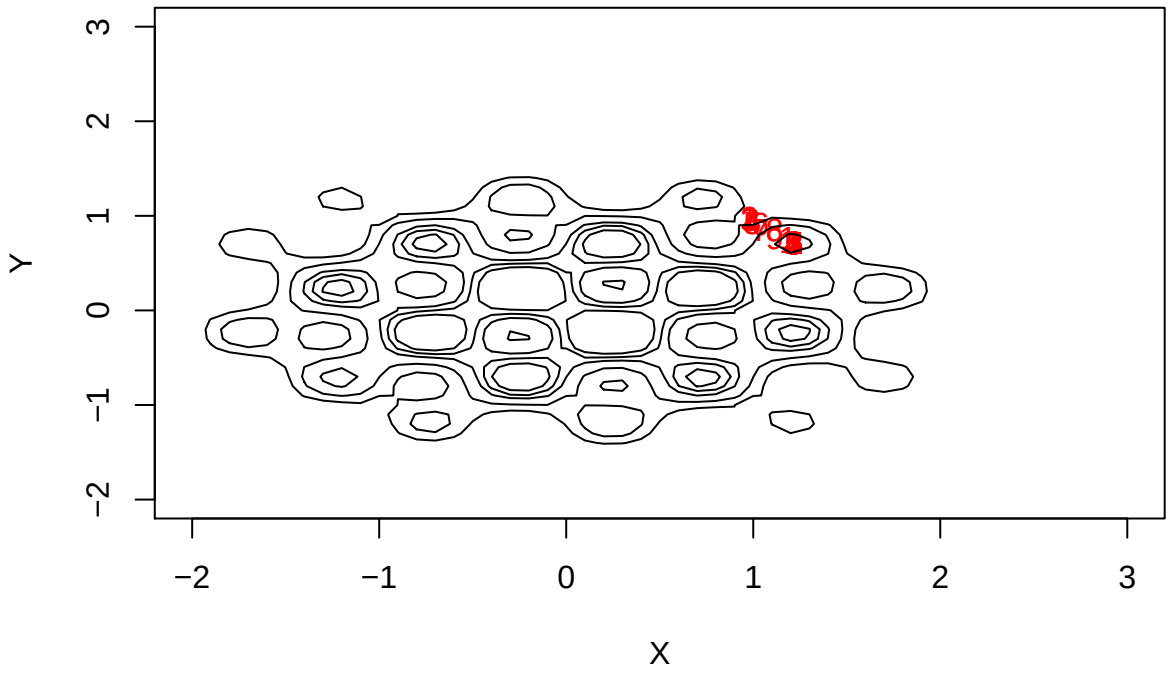
Coord Descendente

```
CoordDescent <- function(F, gradF, tasa=0.1, wIni=c(0,0), umbral = 10^(-14), max_iter = 100, plot=T, xla
    t <- 0
    w <- wIni
    salir <- FALSE
    pointList <- c()
    valueList <- c()

    norma <- function(v){
        sqrt(sum(v^2))
    }
    ronda_impar <- TRUE
    while(salir == FALSE){
        gt <- gradF(w[1], w[2])
        vt <- -gt
```

```
        w <- if (ronda_impar){
            w + tasa*c(vt[1],0) #Avance en coord x
        }
        else{
            w + tasa*c(0, vt[2]) #Avance en coord y
        }
        ronda_impar = !ronda_impar

        pointList[2*t-1] <- w[1]
        pointList[2*t] <- w[2]

        if( F(w[1], w[2]) < umbral){
            print(c("Valor de la funcion menor que umbral:", F(w[1], w[2]) ))
            print(c("Convergencia en ", t/2, "iteraciones"))

            salir <- TRUE
        }
        t <- t+1

        #Cada iteracion tiene dos partes...
        if(t > 2*max_iter){
            print(c("Max iters alcanzado"))
            salir <- TRUE
        }

    }
    m <- matrix(pointList, byrow = TRUE, ncol=2)
    if(plot){
        print(c("Stop en ", (t-1)/2, "iteraciones"))
        print(c("Valor de F al parar:", F(w[1], w[2]) ))
        print(c("Norma del gradiente al parar:", norma(gt) ))
        plot(m[,1], m[,2], xlim = c(-2, 3), ylim = c(-2, 3), pch=as.character(1:length(m[,1])), col=2, i
        pintar_grafica(F)
    }
    valueList <- F(m[,1], m[,2])
    print(valueList)
    return(w)
}
```

```
CoordDescent(f, gradF, wIni = c(1,1), tasa=0.01, max_iter = 30)
```

```
## [1] "Max iters alcanzado"
## [1] "Stop en "    "30"          "iteraciones"
## [1] "Valor de F al parar:" "0.593269374325836"
## [1] "Norma del gradiente al parar:" "1.03578512786223e-14"
```




```
##  [1] 2.9026215 2.9025757 2.8504756 2.8221615 2.7003107 2.5258783 2.1619174
##  [8] 1.6633265 1.1234922 0.7673465 0.6402452 0.6012702 0.5955646 0.5935710
## [15] 0.5933752 0.5932808 0.5932743 0.5932698 0.5932696 0.5932694 0.5932694
## [22] 0.5932694 0.5932694 0.5932694 0.5932694 0.5932694 0.5932694 0.5932694
## [29] 0.5932694 0.5932694 0.5932694 0.5932694 0.5932694 0.5932694 0.5932694
## [36] 0.5932694 0.5932694 0.5932694 0.5932694 0.5932694 0.5932694 0.5932694
## [43] 0.5932694 0.5932694 0.5932694 0.5932694 0.5932694 0.5932694 0.5932694
## [50] 0.5932694 0.5932694 0.5932694 0.5932694 0.5932694 0.5932694 0.5932694
## [57] 0.5932694 0.5932694 0.5932694 0.5932694
```

```
## [1] 1.218070 0.712812
```

Metodo de Newton

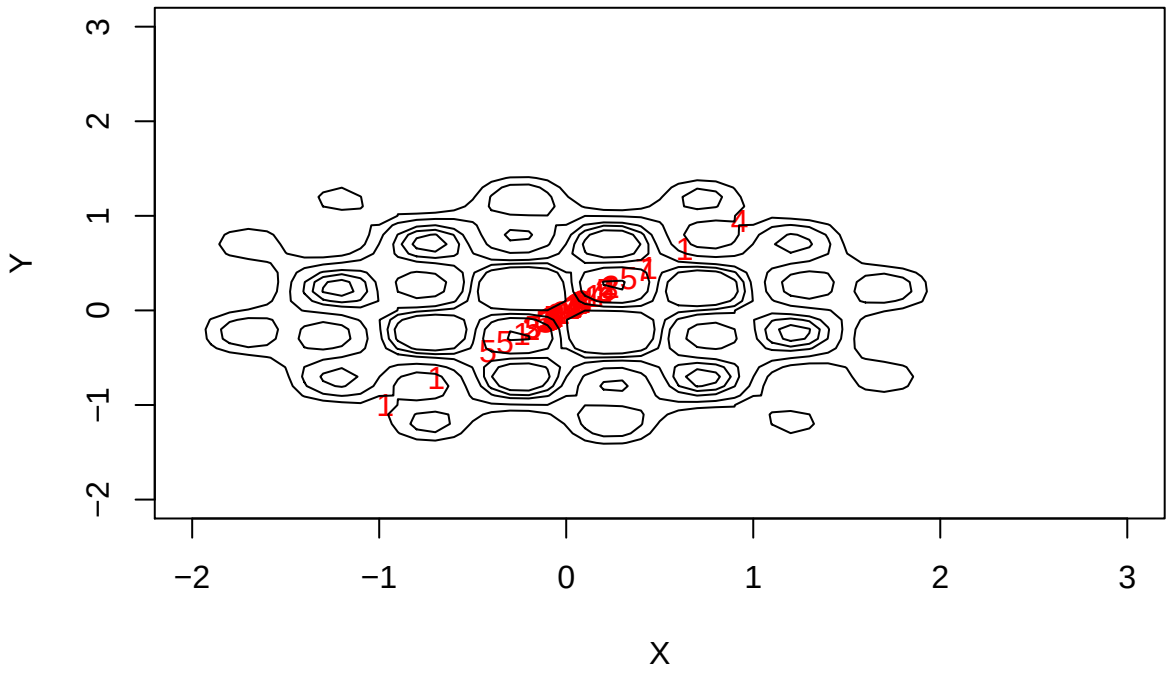
Implementar el algoritmo de minimización de Newton y aplicarlo a la función $f(x, y)$ dada en el ejercicio.1b.
Desarrolle los mismos experimentos usando los mismos puntos de inicio.
Generar un gráfico de como desciende el valor de la función con las iteraciones.
Extraer conclusiones sobre las conductas de los algoritmos comparando la curva de decrecimiento de la función calculada en el apartado anterior y la correspondiente obtenida con gradiente descendente.

```
NewtonMethod <- function(F, gradF,  hessF, wIni=c(0,0), umbral = 10^(-14), max_iter = 100, plot=T, xlab=
    t <- 0
    w <- wIni
    salir <- FALSE
    pointList <- c()
    valueList <- c()

    norma <- function(v){
        sqrt(sum(v^2))
    }
    ronda_impar <- TRUE
```

```
    while(salir == FALSE){
        gt <- gradF(w[1], w[2])
        vt <- -gt
        w <- - solve(hessF(vt[1], vt[2]))%*%c(vt[1],vt[1])

        pointList[2*t-1] <- w[1]
        pointList[2*t] <- w[2]

        if( F(w[1], w[2]) < umbral){
            print(c("Valor de la funcion menor que umbral:", F(w[1], w[2]) ))
            print(c("Convergencia en ", t/2, "iteraciones"))

            salir <- TRUE
        }
        t <- t+1

        #Cada iteracion tiene dos partes...
        if(t > 2*max_iter){
            print(c("Max iters alcanzado"))
            salir <- TRUE
        }

    }
    m <- matrix(pointList, byrow = TRUE, ncol=2)
    if(plot){
        print(c("Stop en ", (t-1)/2, "iteraciones"))
        print(c("Valor de F al parar:", F(w[1], w[2]) ))
        print(c("Norma del gradiente al parar:", norma(gt) ))
        plot(m[,1], m[,2], xlim = c(-2, 3), ylim = c(-2, 3), pch=(as.character(1:length(m[,1]))), col=2
        pintar_grafica(F)
    }
    valueList <- F(m[,1], m[,2])
    print(valueList)
    return(w)
}
```

```
NewtonMethod(f, gradF, hessF, wIni = c(1,1), max_iter = 30)
```

```
## [1] "Max iters alcanzado"
## [1] "Stop en "    "30"          "iteraciones"
## [1] "Valor de F al parar:" "0.445697396783733"
## [1] "Norma del gradiente al parar:" "7.37409694751173"
```




```
##  [1] 0.05402687 0.15579067 0.24329629 0.33105649 0.42077581 0.39440463
##  [7] 0.98293627 0.22142088 0.23390194 2.46113389 1.35436729 0.57025610
## [13] 2.92347376 0.13218086 0.27371184 0.57466683 3.31548685 0.80431819
## [19] 2.16338296 0.06387173 0.07158870 2.17296670 0.05547456 0.62514682
## [25] 0.49617886 1.80785821 0.35364297 0.75112491 1.96425906 0.23844833
## [31] 0.81046233 0.72171989 0.57993625 2.15025237 0.81101293 0.71133374
## [37] 0.51212167 1.82672886 0.56403083 2.10743898 0.16607766 0.20451559
## [43] 1.60445219 2.93152494 0.06286775 0.07237565 0.82505064 0.54215777
## [49] 0.46655818 0.37661475 0.42308076 0.43208487 0.95119678 0.97038485
## [55] 2.01384535 1.78797248 1.85593357 1.71595492 0.40925648 0.44569740
```

```
##           [,1]
## [1,] 0.07553828
## [2,] 0.07755380
```

Regresion logistica

En este ejercicio crearemos nuestra propia función objetivo f (probabilidad en este caso) y nuestro conjunto de datos D para ver cómo funciona regresión logística. Supondremos por simplicidad que f es una probabilidad con valores 0/1 y por tanto que y es una función determinista de x.
Consideremos $d = 2$ para que los datos sean visualizables, y sea $X = [-1, 1] \times [-1, 1]$ con probabilidad uniforme de elegir cada $x \in X$. Elegir una línea en el plano como la frontera entre $f(x) = 1$ (donde y toma valores $+1$) y $f(x) = 0$ (donde y toma valores -1), para ello seleccionar dos puntos aleatorios del plano y calcular la línea que pasa por ambos.

Seleccionar $N = 100$ puntos aleatorios $\{x_n\}$ de X y evaluar las respuestas de todos ellos $\{y_n\}$ respecto de la frontera elegida.

```
RegresionLogistica <- function(muestras, etiquetas, umbral = 0.01, max_iters = 100000){
    w <- c(0,0,0)
    N <- length(etiquetas)
```

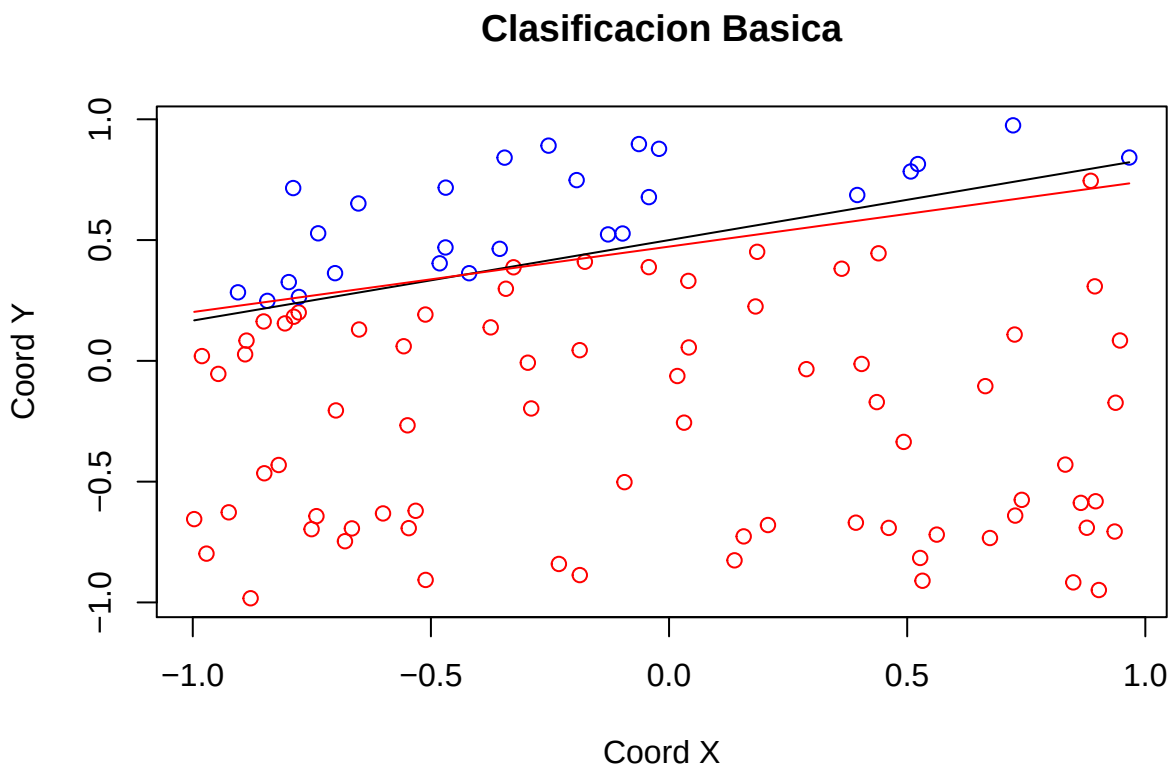
```
    iters <- 0
    tasa <- 0.01

    gradienteError <- function(w, n, muestras, etiquetas){
        return( -etiquetas[n,]*muestras[n,]/(1 + exp(etiquetas[n,]* w%*%muestras[n,]) ) )
    }

    norma <- function(v){
        sqrt(sum(v^2))
    }
    salir <- FALSE
    while(!salir){
        w_prev <- w
        #El hecho de meter esta permutacion es lo que hace que sea "Estocastico"
        #es importante permutar los puntos para eliminar cualquier patron que pueda aparecer al coger
        #las muestras en el mismo orden
        permutacion <- sample(1:N, size = N, replace = FALSE)
        for(i in permutacion){
            w <- w - tasa*gradienteError(w, i, muestras, etiquetas)
            iters <- iters + 1
        }
        if(norma(w - w_prev) < umbral ){
            salir <- TRUE
        }
        if(iters > max_iters){
            salir <- TRUE
            cat("Salimos por maximo iters")
        }
    }
    return(w)
}


puntosClasificadosToMatrix <- function(puntos){
    m <- data.frame(puntos[[1]],puntos[[2]],puntos[[3]]);
    return(data.matrix(m))
}

preparaDatos <- function(puntosClasificados){
    m <- puntosClasificadosToMatrix(puntosClasificados)
    muestras <- m[,1:2]
    numDatos <- length(m[,1])
    muestras <- cbind(rep(1, numDatos), muestras)
    etiquetas <- m[,3]
    etiquetas <- t(t(etiquetas))
    return(list(muestras, etiquetas))
}

simula_unif  <- function(N, dim, minimo, maximo){
    puntos <- list();
    for(i in 1:dim){
        xi <- c();
        xi <- runif(N, minimo, maximo);
```

```
        puntos[[i]] <- xi;
    }
    return(puntos)
}

clasificaPuntosRectav2 <- function(puntos, a, b, dibujar = FALSE){
    # tipo[i] = 1 o -1 ya que TRUE es 1 y FALSE es 0
    # La funcion f(x) = 2x-1 lleva el 0 al -1 y el 1 al 1
    # Por tanto tipo es un vector de -1 y 1
    tipo <- 2*(puntos[[2]] - a*puntos[[1]] - b > 0) - 1;
    if(dibujar){
        #Limites de los ejes para el plot:
        miny = min(puntos[[2]]);
        maxy = max(puntos[[2]]);
        minx = min(puntos[[1]]);
        maxx = max(puntos[[1]]);

        plot(puntos[[1]], puntos[[2]], col=tipo+3, pch=1, xlab="Coord X", ylab="Coord Y", main = "Clasi
        curve(a*x + b, add=TRUE);
    }
    return(list(puntos[[1]], puntos[[2]], tipo));
}
```

```
misPuntos <- simula_unif(100, 2, -1, 1)
#Clasificamos puntos por la recta y = x/3 + 1/2
misPuntosClasificados <- clasificaPuntosRectav2(misPuntos, a = 1/3, b = 0.5, dibujar = T)

datos <- preparaDatos(misPuntosClasificados)

#Muestras es una matriz de dos columnas y etiquetas es un vector columna
muestras <- datos[[1]]
etiquetas <- datos[[2]]

pesos <- RegresionLogistica(muestras, etiquetas)
print(c(-pesos[2]/pesos[3], -pesos[1]/pesos[3]))
```

```
## puntos..1..
##   0.2708853   0.4729226
```

```
curve(-pesos[2]/pesos[3]*x - pesos[1]/pesos[3], col="red", add = T)
```

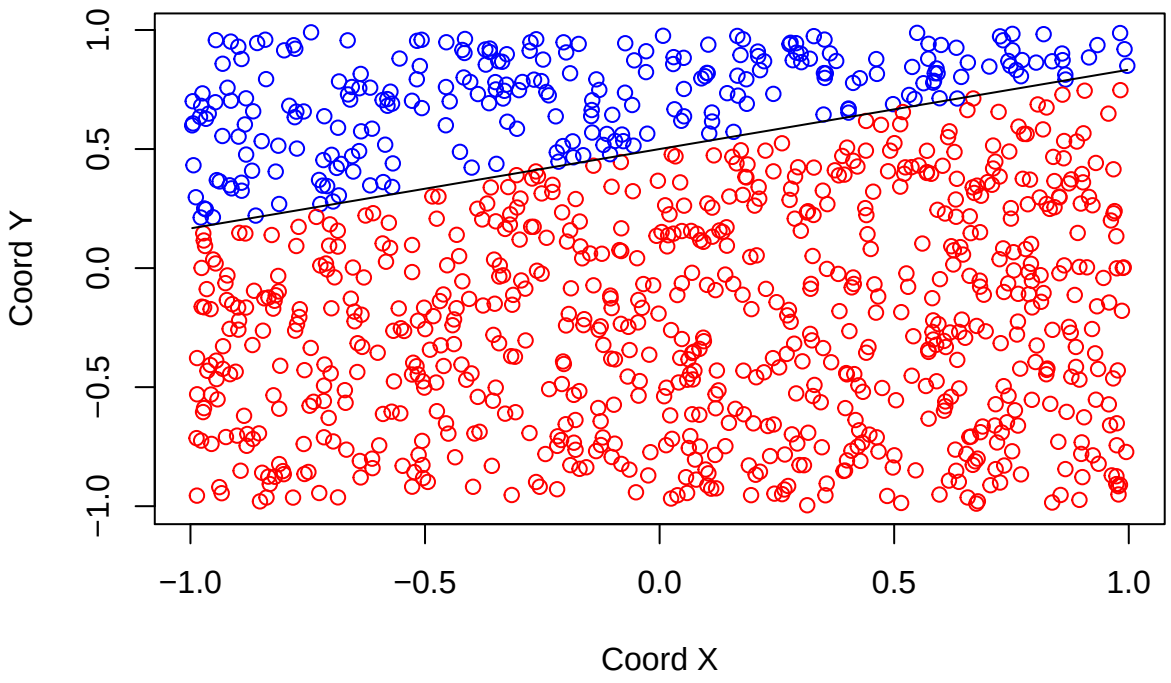



```
print(pesos)
```

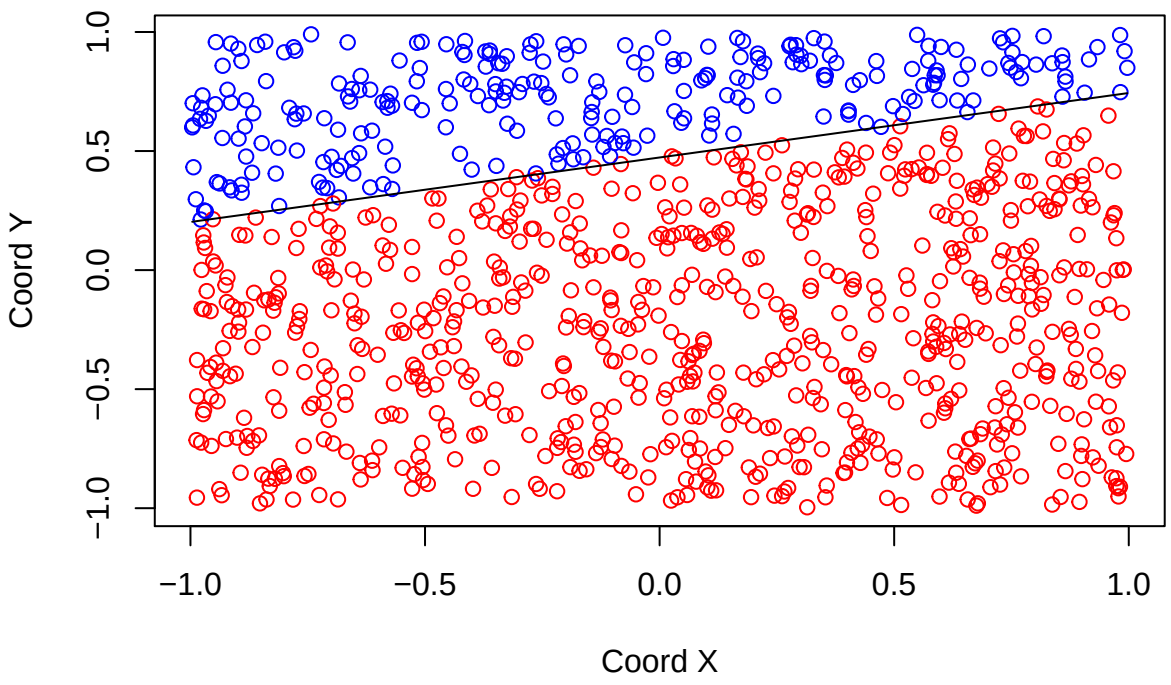
```
##             puntos..1.. puntos..2..
##    -4.042231   -2.315349    8.547342
```

Ahora vamos a estimar E_{out} para ello usamos un numero suficientemente grande de nuevas muestras:

```
nuevasMuestras <- simula_unif(1000, 2, -1,1)
#Clasificamos por la misma recta que antes:
misNuevasMuestrasClasificadas <- clasificaPuntosRectav2(nuevasMuestras, a = 1/3, b = 0.5, dibujar = T)
```




```
nuevosDatos <- preparaDatos(misNuevasMuestrasClasificadas)
etiquetasReales <- nuevosDatos[[2]]

#Coeficientes de la recta de regresion
rectaRegLog <- c(-pesos[2]/pesos[3], -pesos[1]/pesos[3])

misNuevasMuestrasClasificadasRegresion <- clasificaPuntosRectav2(nuevasMuestras, a = rectaRegLog[1], b =
```




```
nuevosDatosEstimados <- preparaDatos(misNuevasMuestrasClasificadasRegresion)
etiquetasEstimadas <- nuevosDatosEstimados[[2]]

cat(c("Recta Real: ", 1/3, 0.5, "\n"))
```

```
## Recta Real:  0.333333333333333 0.5
```

```
cat(c("Recta Regresion: ", rectaRegLog[1], rectaRegLog[2]), "\n")
```

```
## Recta Regresion:  0.27088525023852 0.472922578326791
```

```
#Estimamos el E_out como la proporcion de fallos E_out = numFallos / numMuestras
E_OUT <- mean(etiquetasReales!=etiquetasEstimadas)
#print(etiquetasReales!=etiquetasEstimadas)
print(E_OUT)
```

```
## [1] 0.013
```

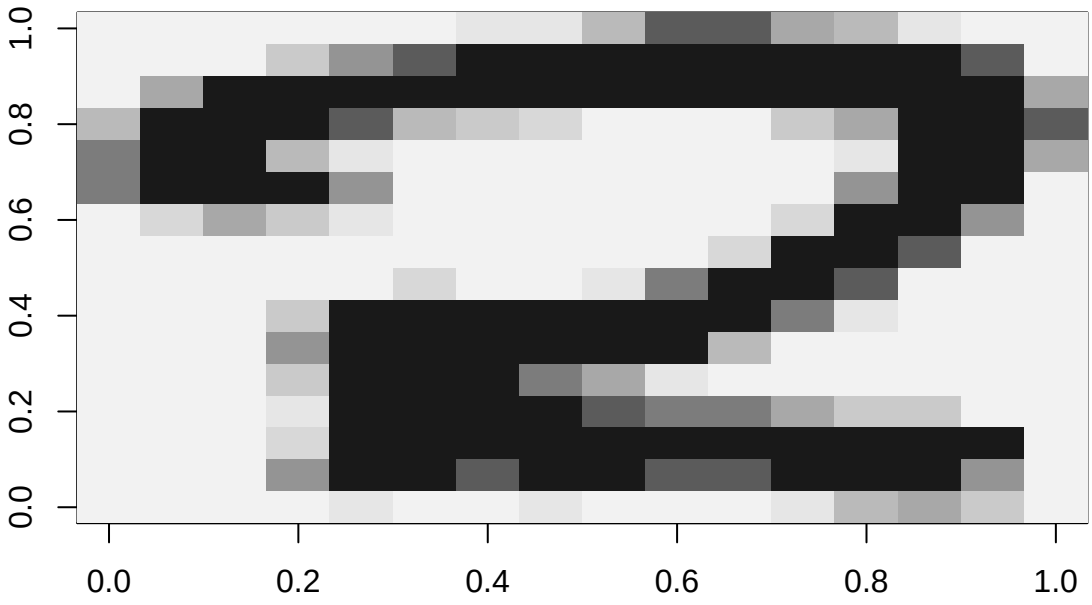
Clasificacion de digitos con Regresion Lineal (de nuevo)

```
zip <- read.table("~/Desktop/UGR/4-CUARTO/Semestre 2/AprendizajeAutomatico/DigitosZip/zip.train", quote=
```

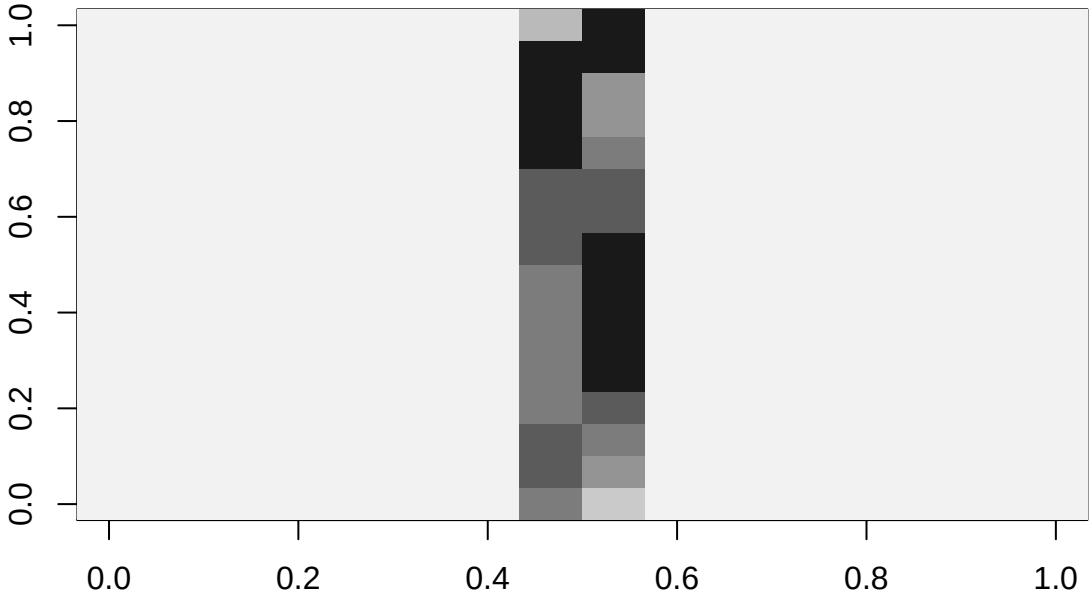
```
## Warning in scan(file, what, nmax, sep, dec, quote, skip, nlines,
## na.strings, : número de items leídos no es múltiplo del número de columnas
```

```r
indicesUnosYCincos <- which(zip$V1 == 1 | zip$V1 == 5)
misNumeros <- zip[indicesUnosYCincos ,]

importarNumeros <- function(data, dibujar = FALSE){
    num_datos <- length(data$V1)
    listaDigitos <- list();
    for(i in 1:num_datos){
        miMatriz_actual <- data[i,]
        listaDigitos[[i]] <- matrix(as.numeric(miMatriz_actual[2:257]),nrow = 16,ncol = 16)
        if(dibujar == TRUE){
            image(z = listaDigitos[[i]], col = rev(grey.colors(start = 0.1, n = 10, end = 0.95)))
        }
    }
    return(listaDigitos)
}

calcularMatrizSimetricaVertical <- function(matriz){
    num_col <- ncol(matriz)
    matrizSimetrica <- matrix(data = NA, nrow = 16, ncol = 16)
    for(i in 1:num_col){
        matrizSimetrica[i,] <- matriz[(num_col+1)-i,]
    }
    return(matrizSimetrica)
}

calcularGradoSimetria <- function(matriz){
    #Calcula el grado de simetria vertical
    matrizSimetrica <- calcularMatrizSimetricaVertical(matriz)
    gradoSimetria <- sum(abs(matriz - matriz[,16:1]))
    return(gradoSimetria)
}

gradoIntensidadMedia <- function(matriz){
    return(mean(matriz))
}

dibujarNumero <- function(matriz){
    image(z = matriz, col = rev(grey.colors(start = 0.1, n = 10, end = 0.95)))
}

miListaNumeros <- importarNumeros(misNumeros)
```

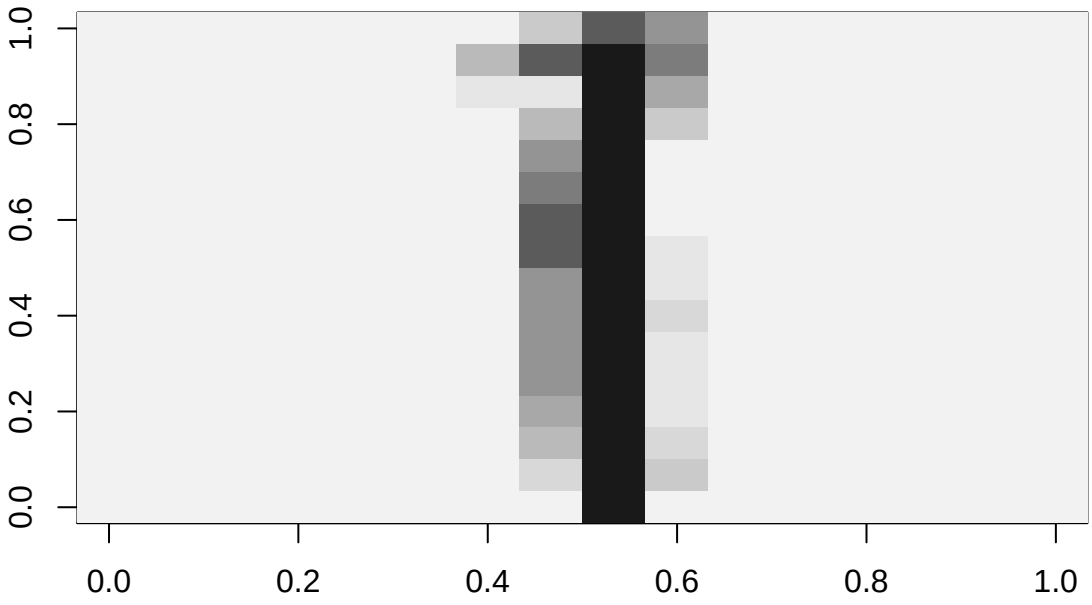
```r
dibujarNumero(miListaNumeros[[1]])
```



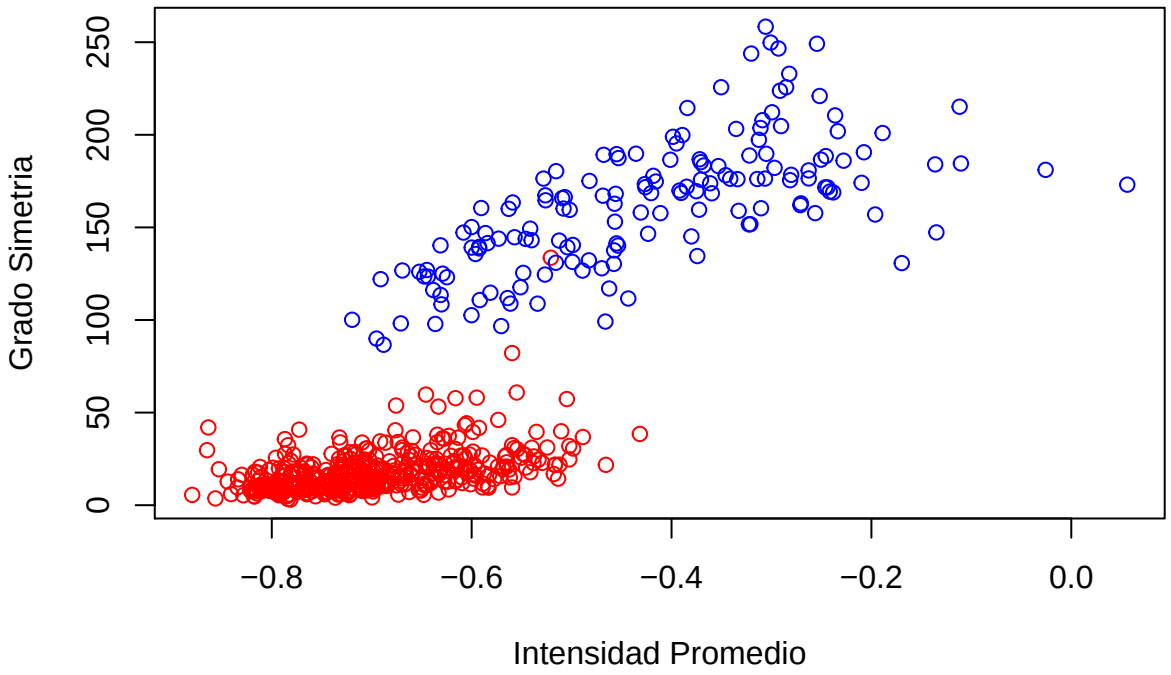

```
dibujarNumero(miListaNumeros[[5]])
```




```
dibujarNumero(miListaNumeros[[3]])
```




```
gradosSimetria <- c()
for(i in 1:length(miListaNumeros)){
    gradosSimetria[i] <- calcularGradoSimetria(miListaNumeros [[i]])
}

gradosIntensidad <- c()
for(i in 1:length(miListaNumeros)){
    gradosIntensidad[i] <- gradoIntensidadMedia(miListaNumeros[[i]])
}

#Podemos ver como quedan el grafico si aniadimos un color a cada punto en funcion de su etiqueta:
miVectorEtiquetas <- misNumeros$V1
miVectorEtiquetas <- (miVectorEtiquetas -3)/2

plot(x=gradosIntensidad , y=gradosSimetria ,col=( miVectorEtiquetas + 3), xlab="Intensidad Promedio", yl
```




```
invertirMatrizSVD <- function(matriz){
    svdDesc <- svd(matriz)
    S_inversa_coef <- 1/svdDesc$d
    S_inversa <- diag(x = S_inversa_coef, nrow = nrow(matriz), ncol = ncol(matriz))
    V <- matrix(svdDesc$v, nrow = nrow(matriz))
    U <- matrix(svdDesc$u, nrow = nrow(matriz))
    matriz_inversa <- V %*% S_inversa %*% t(U)
    return(matriz_inversa)
}
```

```
#Preparacion de datos
miVectorEtiquetas <- misNumeros$V1
miVectorEtiquetas <- (miVectorEtiquetas -3)/2
head(miVectorEtiquetas)
```

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

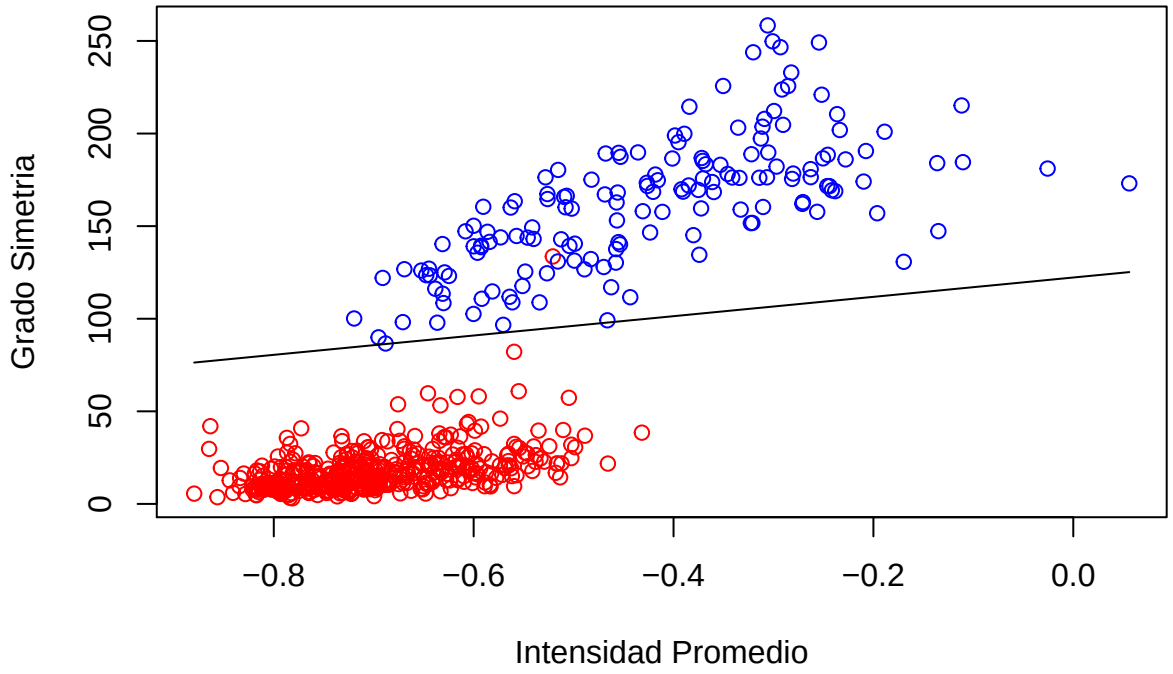
```
datos <- matrix(c(rep(1, length(gradosIntensidad)), gradosIntensidad , gradosSimetria), ncol=3 )
head(datos)
```

```
##      [,1]       [,2]    [,3]
## [1,]    1 -0.1117383 215.162
## [2,]    1 -0.7539141  15.240
## [3,]    1 -0.7722813  18.060
## [4,]    1 -0.7692578   9.472
## [5,]    1 -0.7954375  11.212
## [6,]    1 -0.7159141  27.492
```

```
regress_lin <- function(datos, label){
    X <- datos; #datos es una matriz [N x 3] ---->>> (1, x0, x1)
```

```
    H <- (invertirMatrizSVD(t(X) %*% X) %*% t(X))
    w <- H %*% t(matrix(label, nrow = 1))
    # hiperplano: w1 + w2*x + w3*y = 0 ==>
    coefs <- c(-w[2]/w[3], -w[1]/w[3])
    return(coefs)
}
```

```
recta_reg <- regress_lin(datos, miVectorEtiquetas)
plot(x=gradosIntensidad , y=gradosSimetria ,col=miVectorEtiquetas +3, xlab="Intensidad Promedio", ylab=
curve(recta_reg[1]*x + recta_reg[2], add = T)
```



Calculo de E_{in}:
Para ello comparamos la etiqueta real con la etiqueta que deberia tener si lo clasificasemos por la recta de regresion:

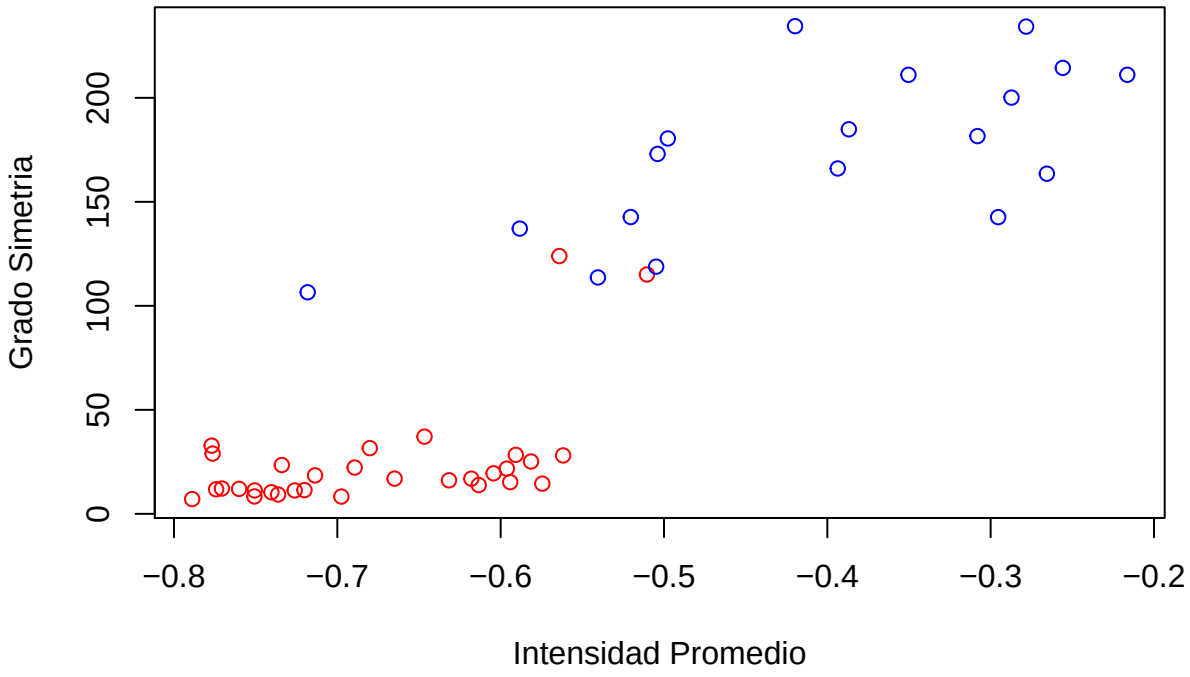
```
misNumeros <- list(datos[,2], datos[,3])
misNumerosClas <- clasificaPuntosRectav2(misNumeros,recta_reg[1], recta_reg[2],dibujar = F)
etiquetasRegresion <- misNumerosClas[[3]]
vectorFallos <- miVectorEtiquetas != etiquetasRegresion
E_in <- mean(vectorFallos)
print(E_in)
```

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

Calculo de E_{test}:
Para ello comparamos la etiqueta real con la etiqueta que deberia tener si lo clasificasemos por la recta de regresion:

```
zipTest <- read.table("~/Desktop/UGR/4-CUARTO/Semestre 2/AprendizajeAutomatico/DigitosZip/zip.test", qu
```

```r
indicesUnosYCincosTest <- which(zipTest$V1 == 1 | zipTest$V1 == 5)
misNumerosTest <- zipTest[indicesUnosYCincosTest ,]

miListaNumerosTest <- importarNumeros(misNumerosTest)

gradosSimetria <- c()
for(i in 1:length(miListaNumerosTest)){
    gradosSimetria[i] <- calcularGradoSimetria(miListaNumerosTest [[i]])
}

gradosIntensidad <- c()
for(i in 1:length(miListaNumerosTest)){
    gradosIntensidad[i] <- gradoIntensidadMedia(miListaNumerosTest[[i]])
}

#Podemos ver como quedan el grafico si aniadimos un color a cada punto en funcion de su etiqueta:
miVectorEtiquetasTest <- misNumerosTest$V1
miVectorEtiquetasTest <- (miVectorEtiquetasTest -3)/2

plot(x=gradosIntensidad , y=gradosSimetria ,col=( miVectorEtiquetasTest + 3), xlab="Intensidad Promedio
```




```r
datosTest <- matrix(c(rep(1, length(gradosIntensidad)), gradosIntensidad , gradosSimetria), ncol=3 )

misNumerosTest <- list(datosTest[,2], datosTest[,3])
misNumerosTestClas <- clasificaPuntosRectav2(misNumerosTest,recta_reg[1], recta_reg[2],dibujar = F)
etiquetasRegresionTest <- misNumerosTestClas[[3]]
vectorFallosTest <- miVectorEtiquetasTest != etiquetasRegresionTest

E_test <- mean(vectorFallosTest)
print(E_test)
```

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

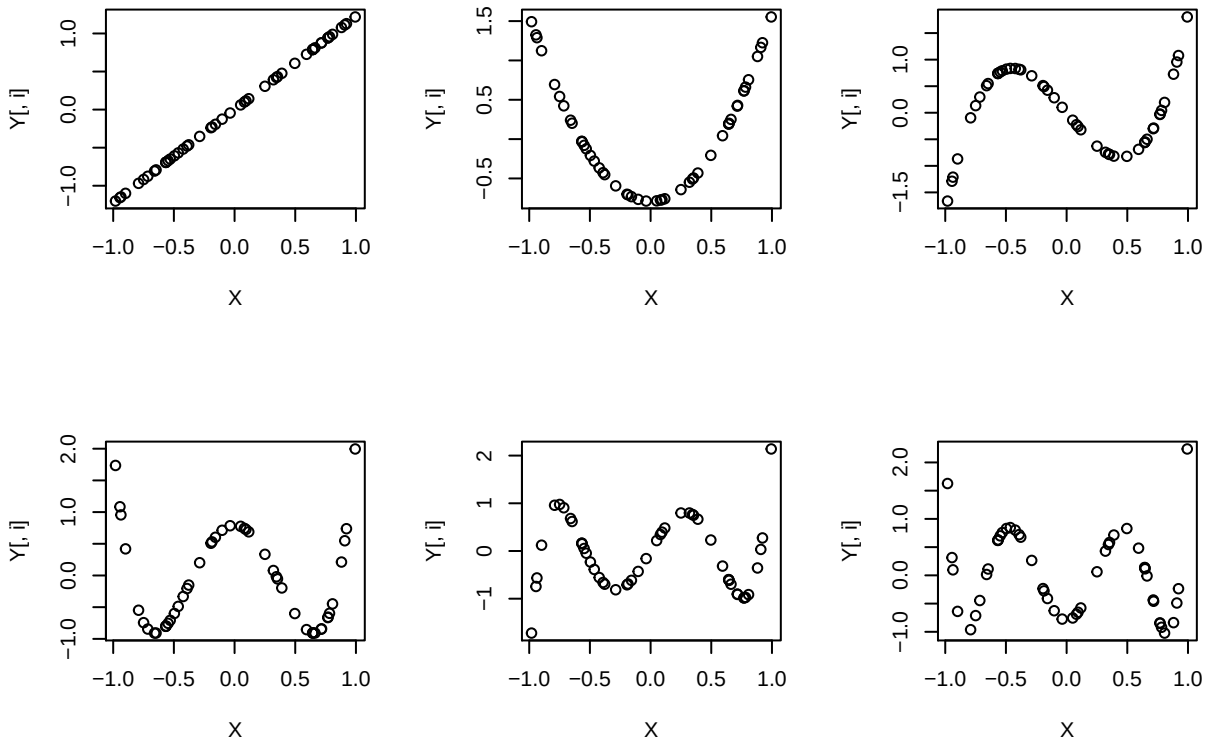
Falta ver una cota del Eout ...

Sobreajuste

Vamos a construir un entorno que nos permita experimentar con los problemas de sobreajuste. Consideremos el espacio de entrada $\mathcal{X} = [-1, 1]$ con una densidad de probabilidad uniforme, $P(x) = \frac{1}{2}$. Consideramos dos modelos H_2 y H_{10} representando el conjunto de todos los polinomios de grado 2 y grado 10 respectivamente. La función objetivo es un polinomio de grado Q_f que escribimos como $f(x) = \sum_{q=0}^{Q_f} a_q L_q(x)$, donde $L_q(x)$ son los polinomios de Legendre. El conjunto de datoses $\mathcal{D} = \{(x_1, y_1), ..., (x_N, y_N)\}$ donde $y_n = f(x_n) + \sigma\epsilon n$ y las $\{\epsilon n\}$ son variables aleatorias **i.i.d.** $\mathcal{N}(0, 1)$ y σ^2 la varianza del ruido. Comenzamos realizando un experimento donde suponemos que los valores de Q_f,N,σ, están especificados, para ello:
Generamos los coeficientes aq a partir de muestras de una distribución $\mathcal{N}(0, 1)$ y escalamos dichos coeficientes de manera que $\mathbb{E}_{a,x}[\{^2] = 1$ ("Ayuda": Dividir los coeficientes por $\sqrt{\sum_{q=0}^{Q_f} \frac{1}{2q+1}}$
Generamos un conjunto de datos,$x_1, ..., x_N$ muestreando de forma independiente $P(x)$ y los valores $y_n = f(x_n) + \sigma\epsilon n$.
Sean g_2 y g_{10} los mejores ajustes a los datos usando H_2 y H_{10} respectivamente, y sean $E_{out}(g_2)$ y $E_{out}(g_{10})$ sus respectivos errores fuera de la muestra.
a) Calcular g_2 y g_{10}

```
#Cargamos las dependencias para usar polinomios de Legendre
#(aunque podrian usarse los de Chebyshev tambien)
library("polynom", lib.loc="/Library/Frameworks/R.framework/Versions/3.2/Resources/library")
library("orthopolynom", lib.loc="/Library/Frameworks/R.framework/Versions/3.2/Resources/library")
```

```
X <- runif(50, min = -1, max = 1)

normalized.p.list <- legendre.polynomials(n= 6, normalized=TRUE)
k=1:6
lY =  sapply (X,FUN = function(x) polynomial.values( normalized.p.list,x))
Y = matrix(unlist(lY), nrow= 50, byrow = T)

par(mfrow=c(2,3))
for(i in 2:7){
 plot(X, Y[,i])
}
```

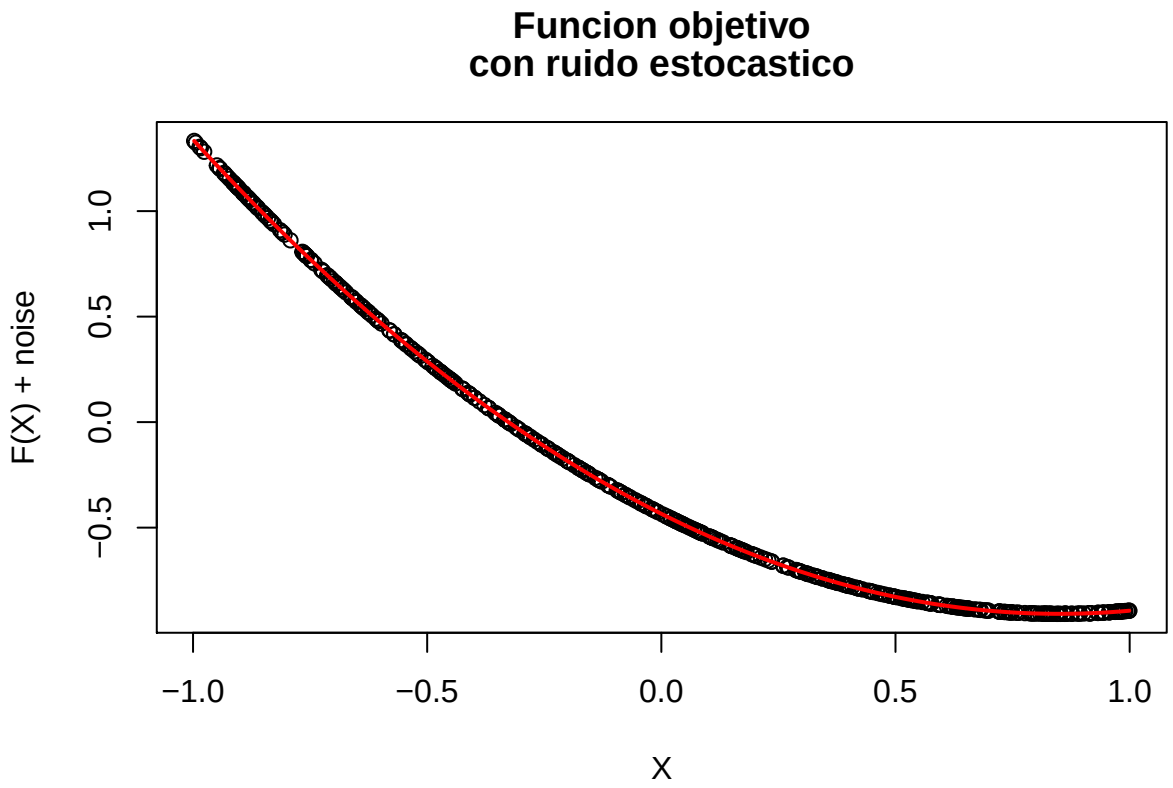



```
set.seed(6)
#Parametros
Qf <- 2         #Grado del polinomio objetivo (ruido deterministico si H no llegase a entenderla)
sigma2 <- 0   #Intensidad del ruido estocastico
N <- 500         #Numero de muestras para el "training"

Qf <- Qf +1          #Qf = 6 indicaba grado 5, despues de esto indica grado 6
X <- runif(N, min = -1, max = 1)
aq <- rnorm(n = Qf, mean = 0, sd = 1)

#La pista que dan en el enunciado esta mal
aq_norm <- sqrt(sum(aq^2))
aq <- aq/aq_norm

normalized.p.list <- legendre.polynomials(n= Qf, normalized=TRUE)
rm(Target)
```

```
## Warning in rm(Target): objeto 'Target' no encontrado
```

```
Target <- 0
for(i in 1:Qf){
    Target <- Target + normalized.p.list[[i]]*aq[i]
}

lY <-  sapply(X,FUN = function(x) polynomial.values( list(Target),x))
Y  <- matrix(unlist(lY), nrow= N, byrow = T)
Y[,1] <- Y[,1] + rnorm(length(Y[,1]),0,sigma2)
par(mfrow=c(1,1))
plot(X, Y[,1], main = c("Funcion objetivo", "con ruido estocastico"), ylab = "F(X) + noise")
```

```
f <- as.function(Target)
curve(expr = f, add = T, col="red", lwd=2)
```




```
Target
```

```
## -0.434431 - 1.116085*x + 0.6568002*x^2
```

```
integral(Target^2, limits = c(-1,1))
```

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

```
regress_lin <- function(datos, label){
    X <- datos; #datos es una matriz [N x 3] ---->>> (1, x0, x1)
    H <- (invertirMatrizSVD(t(X) %*% X) %*% t(X))
    w <- H %*% t(matrix(label, nrow = 1))
    # hiperplano: w1 + w2*x + w3*y = 0 ==>
    return(w)
}
```

Calculamos ahora g2

```
H2 <- legendre.polynomials(n= 2, normalized=TRUE)
pol1 <- as.function(H2[[2]])
pol2 <- as.function(H2[[3]])

Z <- matrix(c(rep(1, N), pol1(X), pol2(X)), ncol = 3, byrow = F)
#Z <- matrix(c(rep(1, N), X, X^2), ncol = 3, byrow = F)
Y <- matrix(Y[,1], ncol = 1)
```

```r
w <- regress_lin(datos = Z, label = Y)
print(w)
```

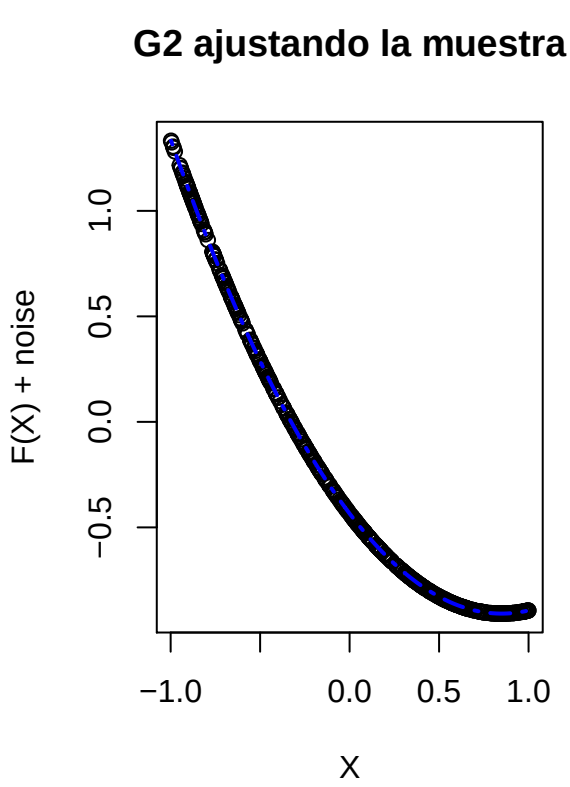
```
##             [,1]
## [1,] -0.2154976
## [2,] -0.9112797
## [3,]  0.2769313
```

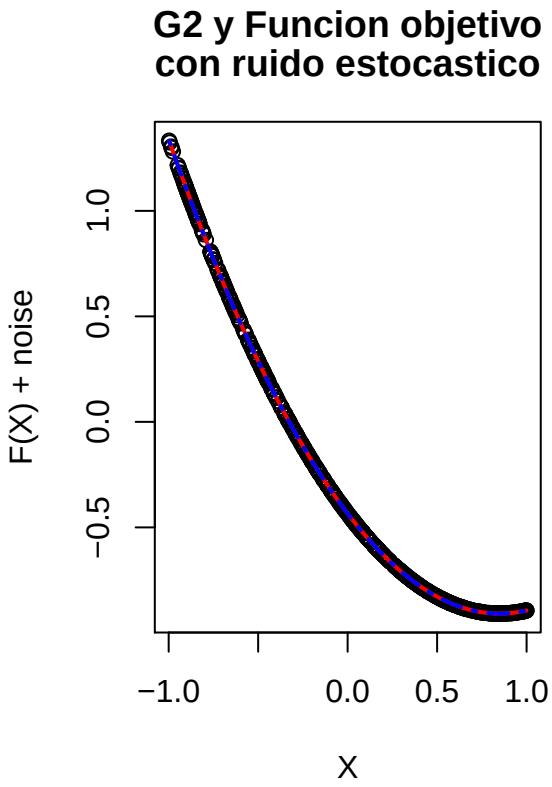
```r
g2 <- as.function(w[1] + w[2]*H2[[2]] + w[3]*H2[[3]])
#g2 <- function(x) {w[1] + w[2]*x + w[3]*x^2}
print(g2)
```

```
## function (x)
## {
##     w <- 0
##     w <- 0.656800185982945 + x * w
##     w <- -1.11608510027022 + x * w
##     w <- -0.434431000217304 + x * w
##     w
## }
## <environment: 0x7fba74128150>
```

Mostramos g2 y la funcion objetivo real

```r
par(mfrow=c(1,2))
plot(X, Y[,1], main = c("G2 ajustando la muestra"), ylab = "F(X) + noise")
curve(g2, add = T, col="blue", lw=2, lty=6 )
plot(X, Y[,1], main = c("G2 y Funcion objetivo", "con ruido estocastico"), ylab = "F(X) + noise")
curve(expr = f, add = T, col="red", lwd=2)
curve(g2, add = T, col="blue", lw=2, lty=6 )
```





Calculamos ahora g10

```
H10 <- legendre.polynomials(n= 10, normalized=TRUE)
pol <- list()
for(i in 1:10){
    pol[[i]] <- as.function(H10[[i+1]])
}
Z <- matrix(rep(1, N), ncol = 1)

for(i in 1:10){
    Z <- cbind(Z, pol[[i]](X))
}

Y <- matrix(Y[,1], ncol = 1)

#Nota:
#Se podria haber hecho Z como (1, x, x^2 ... x^10)
#Z <- matrix(c(rep(1, N), X, X^2, X^3, X^4, X^5, X^6, X^7, X^8, X^9, X^10), ncol = 11, byrow = F)
```

```
w <- regress_lin(datos = Z, label = Y)
print(w)
```

```
##                 [,1]
##  [1,] -2.154976e-01
##  [2,] -9.112797e-01
##  [3,]  2.769313e-01
##  [4,] -9.974660e-16
##  [5,]  1.682682e-16
```

```
##  [6,]  1.942890e-16
##  [7,] -5.010640e-16
##  [8,]  1.664901e-15
##  [9,] -2.703133e-15
## [10,]  1.088756e-15
## [11,] -4.688090e-16
```

```
print(H10)
```

```
## [[1]]
## 0.7071068
##
## [[2]]
## 1.224745*x
##
## [[3]]
## -0.7905694 + 2.371708*x^2
##
## [[4]]
## -2.806243*x + 4.677072*x^3
##
## [[5]]
## 0.7954951 - 7.954951*x^2 + 9.280777*x^4
##
## [[6]]
## 4.397265*x - 20.52057*x^3 + 18.46851*x^5
##
## [[7]]
## -0.7967218 + 16.73116*x^2 - 50.19347*x^4 + 36.80855*x^6
##
## [[8]]
## -5.990715*x + 53.91644*x^3 - 118.6162*x^5 + 73.42906*x^7
##
## [[9]]
## 0.7972005 - 28.69922*x^2 + 157.8457*x^4 - 273.5992*x^6 + 146.571*x^8
##
## [[10]]
## 7.585119*x - 111.2484*x^3 + 433.8688*x^5 - 619.8126*x^7 + 292.6893*x^9
##
## [[11]]
## -0.7974349 + 43.85892*x^2 - 380.1106*x^4 + 1140.332*x^6 - 1384.689*x^8 +
## 584.6464*x^10
```

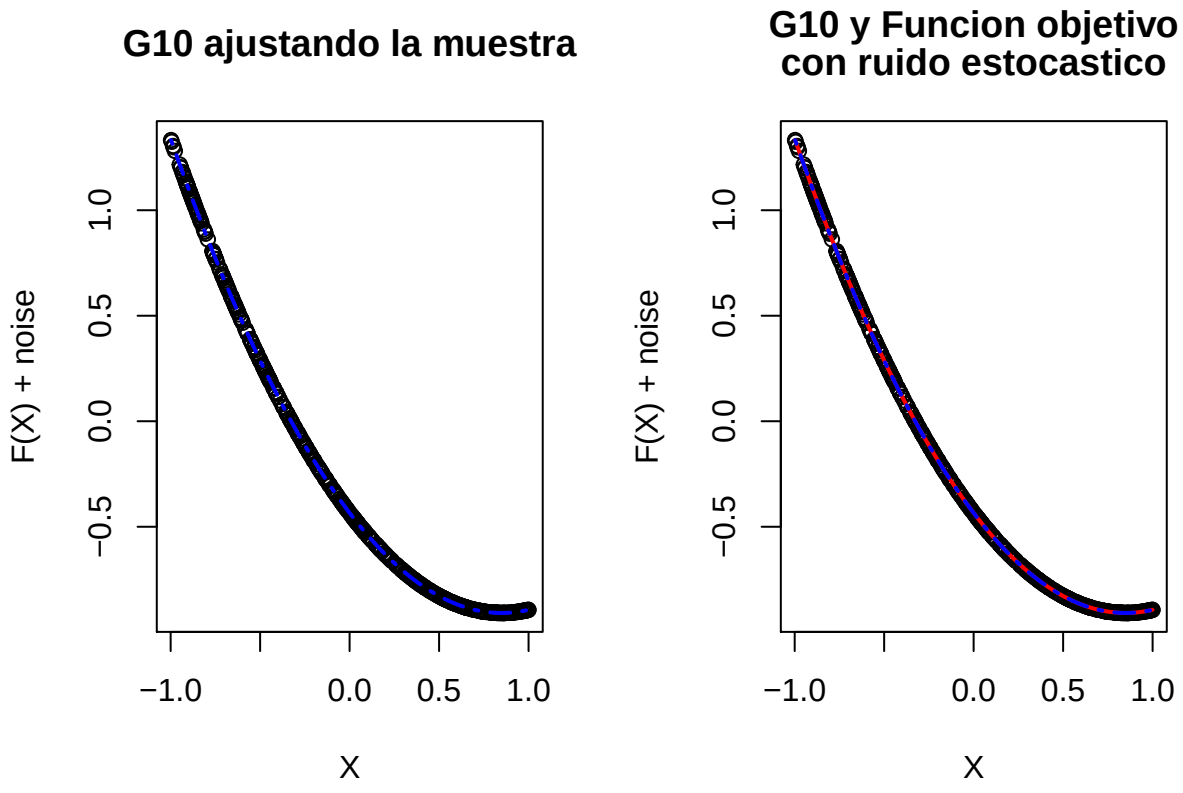
```
#Nos fijamos en que H10[1] no es 1  ... (por eso el bucle empieza en 2)
g10 <- w[1]
for(i in 2:11){
    g10 <- g10 + H10[[i]]*w[i]
}
g10 <- as.function(g10)
print(g10)
```

```
## function (x)
## {
```

```
##     w <- 0
##     w <- -2.7408748287045e-13 + x * w
##     w <- 3.1866714274844e-13 + x * w
##     w <- 2.52953684669724e-13 + x * w
##     w <- -5.52572440446531e-13 + x * w
##     w <- 1.86533660134611e-13 + x * w
##     w <- 2.78481248673395e-13 + x * w
##     w <- -2.21766774949491e-13 + x * w
##     w <- -4.00089684785804e-14 + x * w
##     w <- 0.656800185982982 + x * w
##     w <- -1.11608510027023 + x * w
##     w <- -0.434431000217301 + x * w
##     w
## }
## <environment: 0x7fba73127350>
```

Mostramos g10 y la funcion objetivo real

```
par(mfrow=c(1,2))
plot(X, Y[,1], main = c("G10 ajustando la muestra"), ylab = "F(X) + noise")
curve(g10, add = T, col="blue", lw=2, lty=6 )
plot(X, Y[,1], main = c("G10 y Funcion objetivo", "con ruido estocastico"), ylab = "F(X) + noise")
curve(expr = f, add = T, col="red", lwd=2)
curve(g10, add = T, col="blue", lw=2, lty=6 )
```

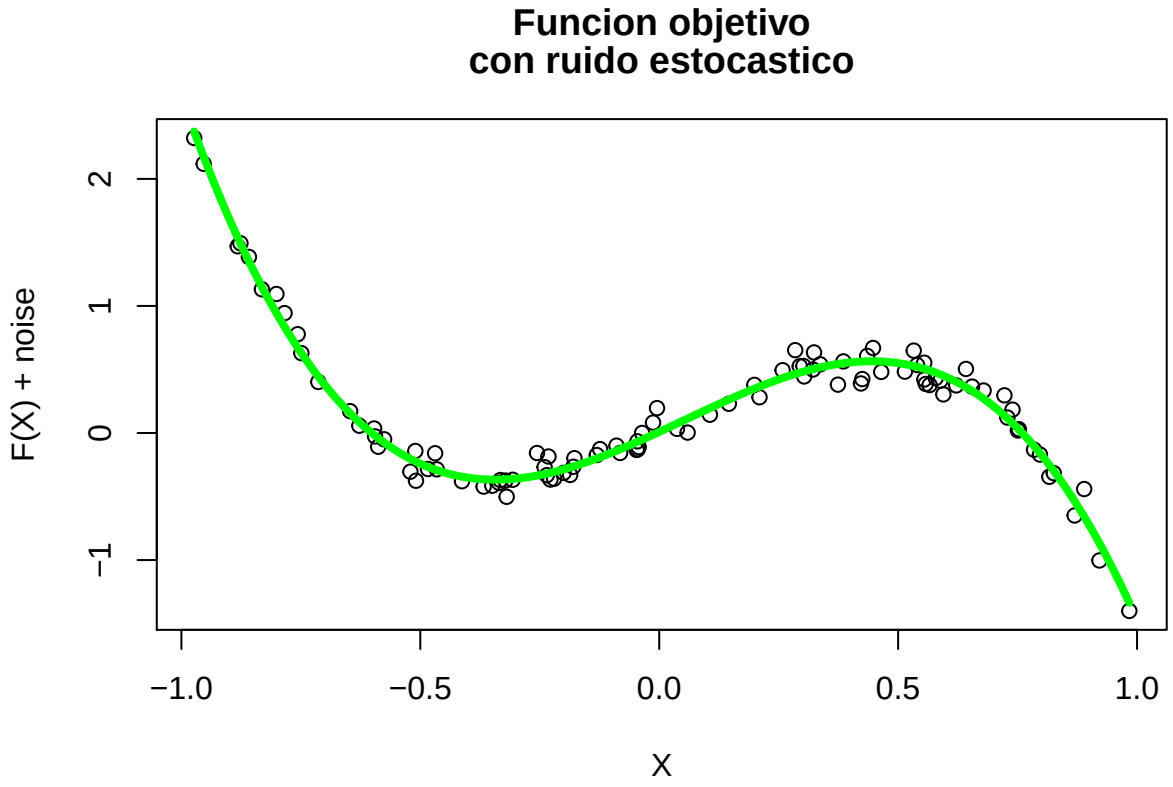



```
errorCuadraticoMedio <- function(X, Y, f){
    N <- length(X)
    return(sum((Y - f(X))^2)/N)
}
```

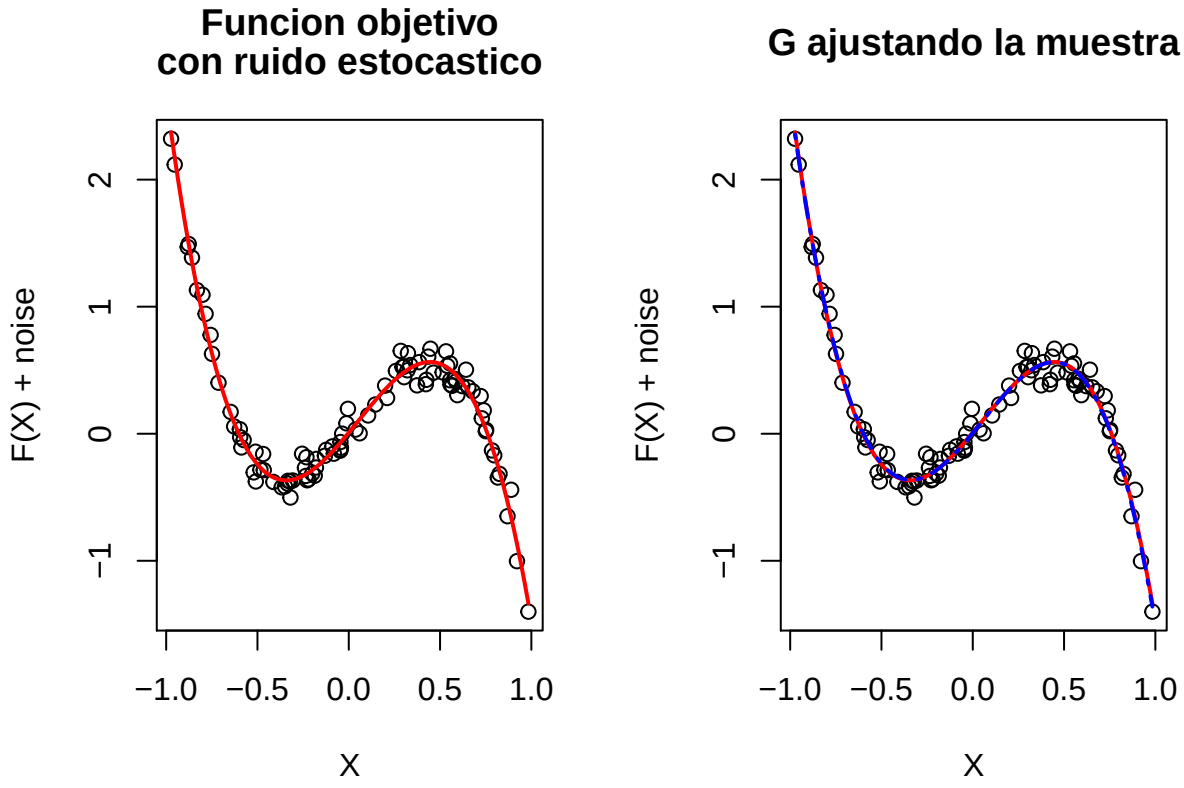
```r
experimento_Overfiting <- function(Qf=20, N=50, sigma=1, gradoH=2, seed=1, dibujar=TRUE, verbose=FALSE){
    set.seed(seed)
    Qf <- Qf+1

    #Puntos
    X <- runif(N, min = -1, max = 1)

    #Coeficientes de Fourier para aproximar 'f' mediante polinomios de Legendre
    aq <- rnorm(n = Qf, mean = 0, sd = 1)
    aq_norm <- sqrt(sum(aq^2))
    aq <- aq/aq_norm

    legendre.pol.list <- legendre.polynomials(n= Qf, normalized=TRUE)

    f <- 0
    for(i in 1:Qf){
        f <- f + legendre.pol.list[[i]]*aq[i]
    }

    f_ <- as.function(f)
    Y <- f_(X)                              # Lista de etiquetas de las muestras
    Y <- Y + rnorm(length(Y),0,sigma^2)     # Añadimos el ruido

    #Imprimimos la funcion objetivo con sus muestras ruidosas
    if(dibujar){
        par(mfrow=c(1,1))
        plot(X, Y, main = c("Funcion objetivo", "con ruido estocastico"), ylab = "F(X) + noise")
        curve(expr = f_, add = T, col="green", lwd=4)
    }
    #Comprobamos que f este normalizada (debe salir 1)
    if(verbose){
        print(c("Esta f normalizada correctamente? ", integral(f^2, limits = c(-1,1))))
    }

    #Preparamos conjunto de hipotesis
    H <- legendre.polynomials(n= gradoH, normalized=TRUE)
    pol <- list()
    for(i in 1:gradoH){
        pol[[i]] <- as.function(H[[i+1]])
    }
    #Preparamos Z-space para regresion lineal en el espacio transformado "Z = Phi(X)"
    Z <- matrix(rep(1, N), ncol = 1)
    for(i in 1:gradoH){
       Z <- cbind(Z, pol[[i]](X))
    }
    Y_byCol <- matrix(Y, ncol = 1)
    #Hacemos regresion lineal
    w <- regress_lin(datos = Z, label = Y_byCol)

    #Obtenemos nuestra funcion de ajuste (g)
    g <- w[1]
    for(i in 2:(gradoH+1)){
        g <- g + H[[i]]*w[i]
```

```
    }
    g <- as.function(g)

    #Comprobamos nuestra funcion de ajuste con la funcion objetivo
    if(dibujar){
        par(mfrow=c(1,2))
        plot(X, Y, main = c("Funcion objetivo", "con ruido estocastico"), ylab = "F(X) + noise")
        curve(expr = f_, add = T, col="red", lwd=2)
        plot(X, Y, main = c("G ajustando la muestra"), ylab = "F(X) + noise")
        curve(expr = f_, add = T, col="red", lwd=2)
        curve(g, add = T, col="blue", lw=2, lty=6 )
    }

    #Calculo de E_in
    ECM <- errorCuadraticoMedio(X, Y, g)
    if(verbose){
        print(c("El error medio interno es", ECM))
    }

    #E_out
    #Preparando conjunto muestras de Test para Eout
    N.test <- N/10
    X.test <- runif(N.test, min = -1, max = 1)
    Y.test <- f_(X.test)
    Y.test <- Y.test + rnorm(length(Y.test),0,sigma^2)

    #Imprimimos la funcion objetivo con sus muestras TEST ruidosas
    if(dibujar){
        par(mfrow=c(1,2))
        plot(X, Y, main = c("Funcion objetivo", "con muestras TRAIN"), ylab = "F(X) + noise")
        curve(expr = f_, add = T, col="red", lwd=2)
        plot(X.test, Y.test, main = c("G sobre el TEST"), ylab = "F(X) + noise", xlim = c(-1,1))
        curve(expr = f_, add = T, col="red", lwd=2)
        curve(g, add = T, col="blue", lw=2, lty=6 )
    }
    #Calculo de E_out
    E_out <- errorCuadraticoMedio(X.test, Y.test, g)
    if(verbose){
        print(c("El error medio fuera de la muestra es", E_out))
    }
    return(E_out)
}
```

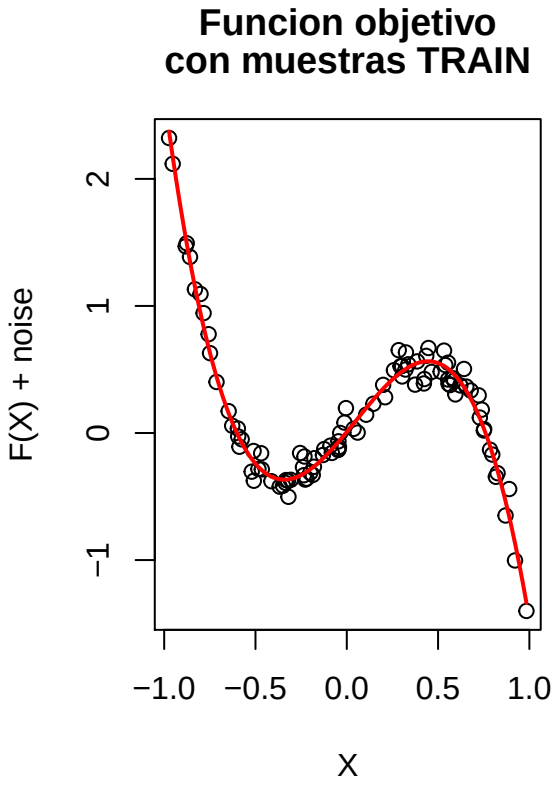
```
experimento_Overfiting(Qf = 3, N = 100, sigma = 0.3, gradoH = 5, dibujar = TRUE, verbose = TRUE)
```

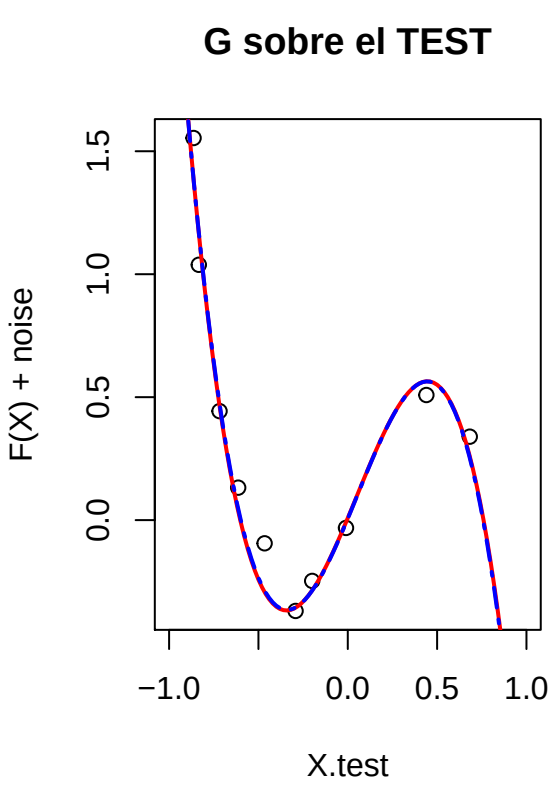



```
## [1] "Esta f normalizada correctamente? "
## [2] "1"
```




```
## [1] "El error medio interno es" "0.00698206571294984"
```

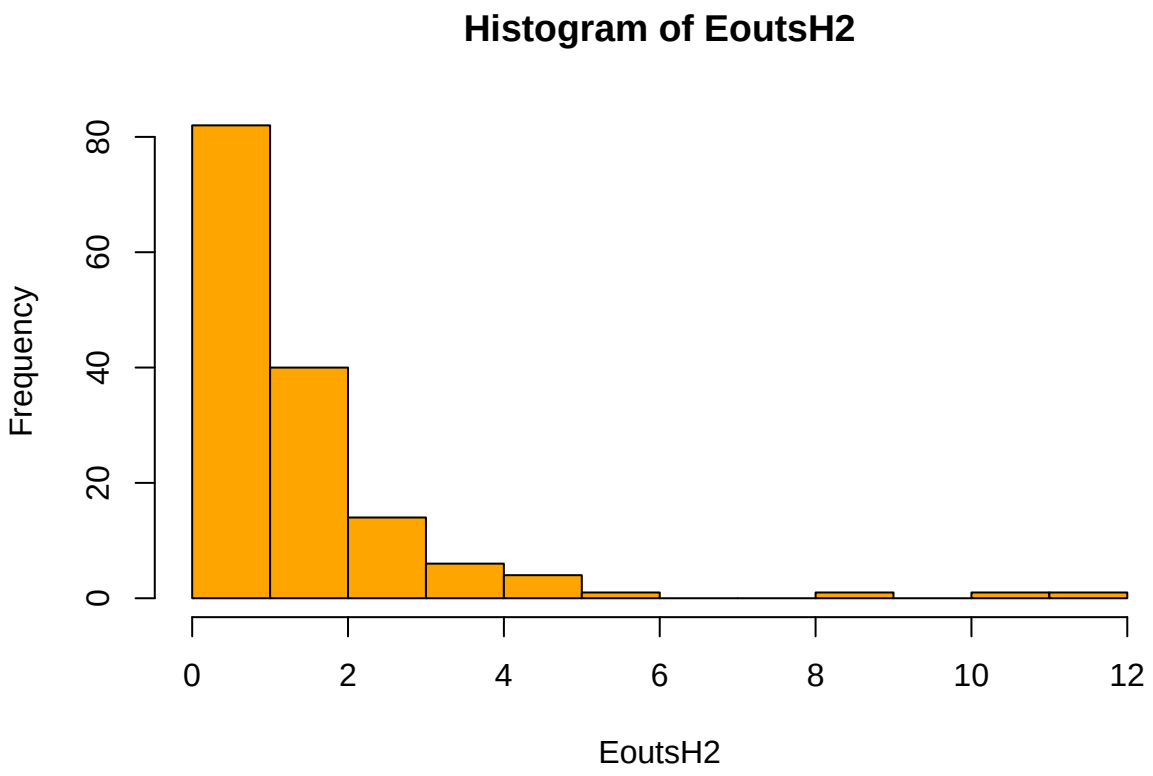




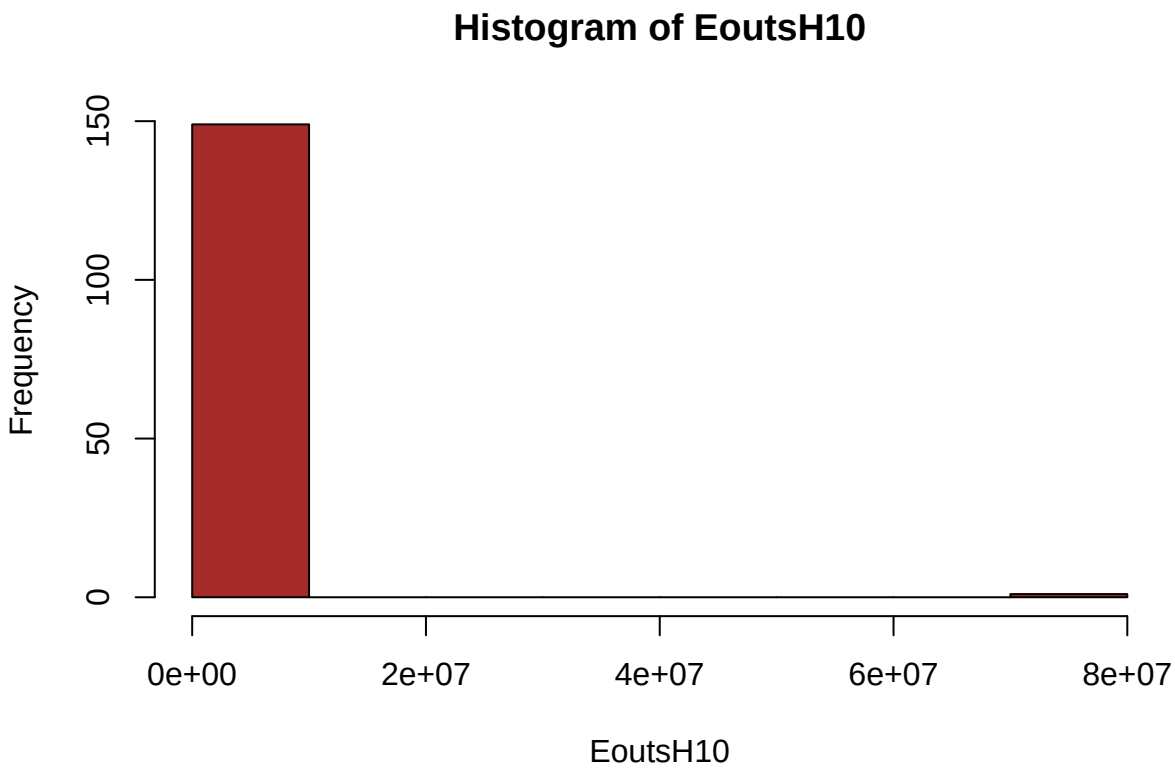

```
## [1] "El error medio fuera de la muestra es"
## [2] "0.0103461448186396"
```

```
## [1] 0.01034614
```

```
#Usando los parametros: Qf = 20, N=50, sigma=1
#Hacemos mas de 100 experimentos con diferentes funciones objetivo y promediamos Eout

experimentoH2 <- function(i){
    experimento_Overfiting(Qf = 3, N = 20, sigma = 1, gradoH = 2,seed = i,dibujar = F)
}

experimentoH10 <- function(i){
    experimento_Overfiting(Qf = 3, N = 20, sigma = 1, gradoH = 10,seed = i,dibujar = F)
}

Num_experimentos <- 150

EoutsH2 <- sapply(1:Num_experimentos, experimentoH2)
EoutsH10 <- sapply(1:Num_experimentos, experimentoH10)

hist(EoutsH2, col="orange")
```




```
hist(EoutsH10, col="brown")
```




```
EoutH2 <- mean(EoutsH2)
EoutH10 <- mean(EoutsH10)
cat(c("Para ",Num_experimentos," experimentos el Eout de H2 medio es:", EoutH2))
```

```
## Para  150  experimentos el Eout de H2 medio es: 1.33734012711921
```

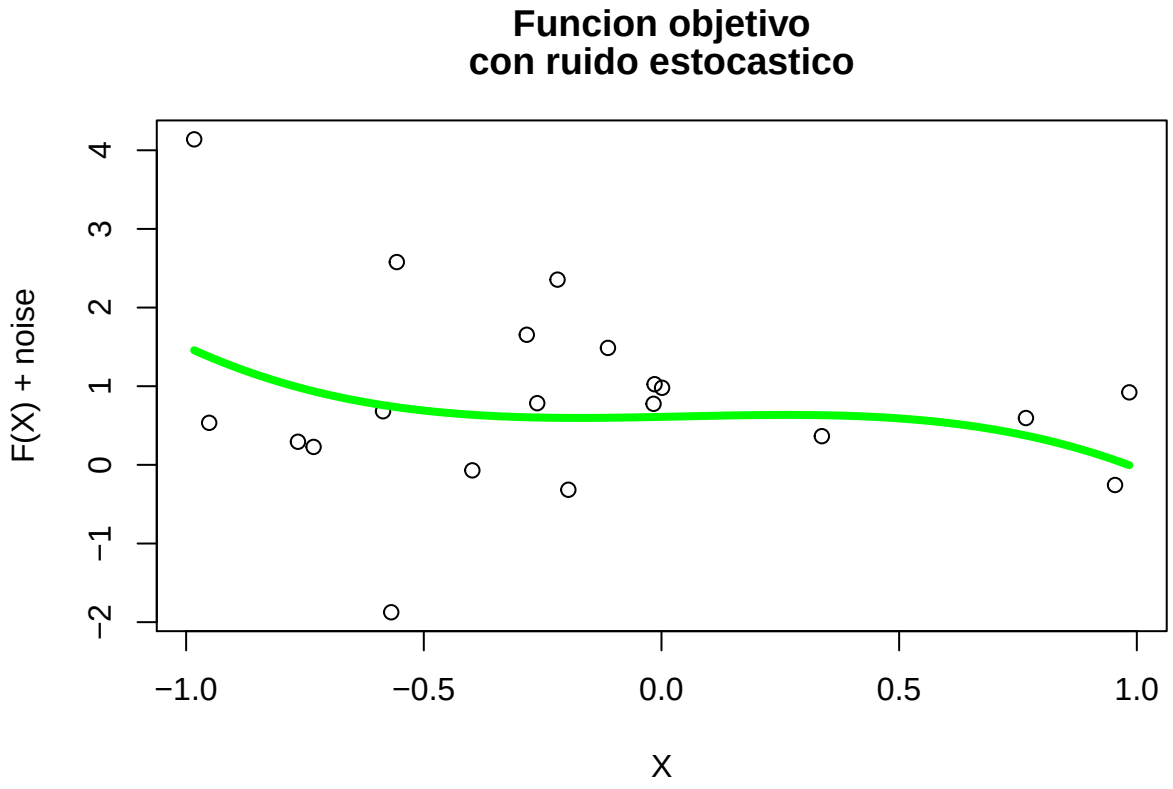
```
cat(c("Para ",Num_experimentos," experimentos el Eout de H10 medio es:", EoutH10))
```

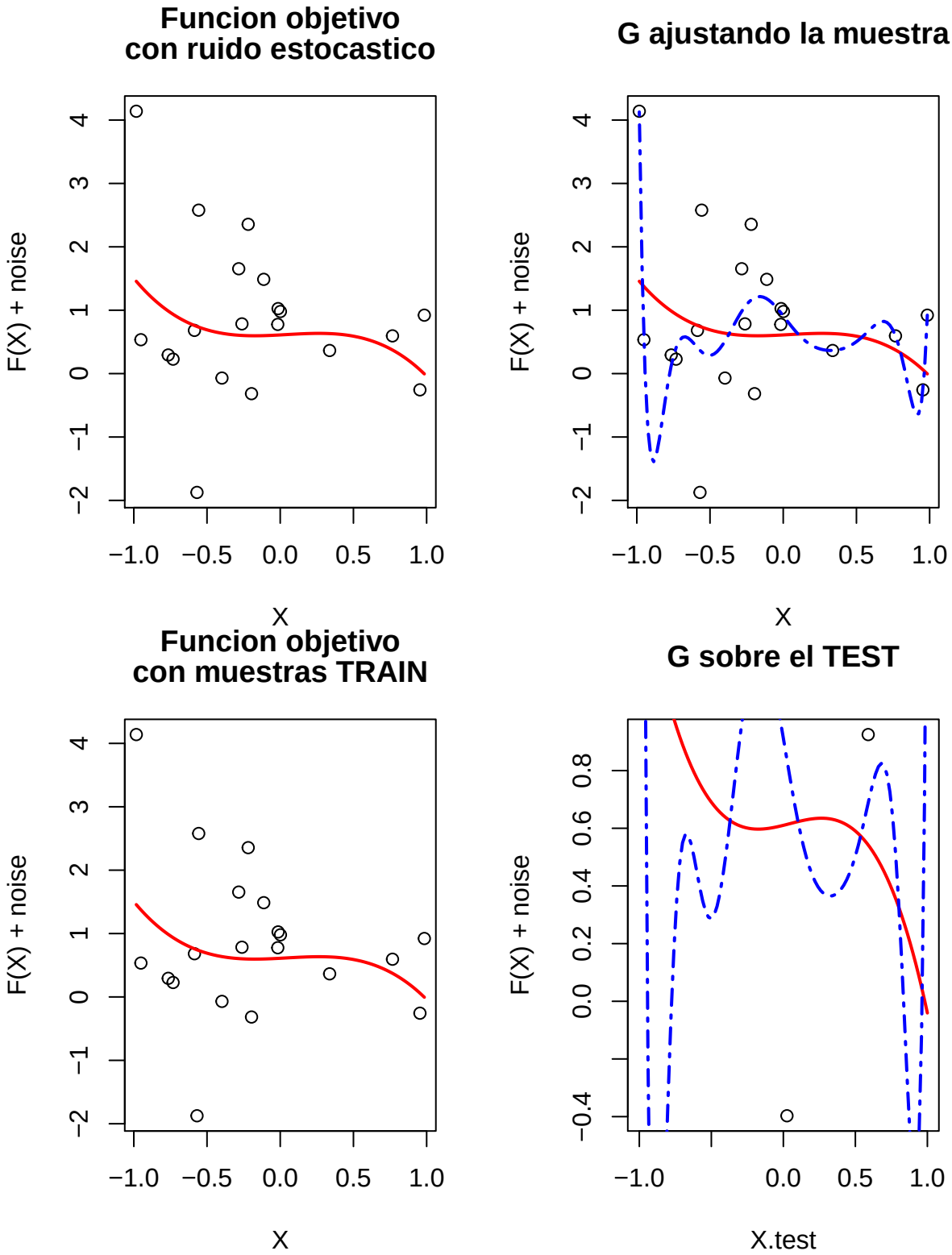
```
## Para  150  experimentos el Eout de H10 medio es: 497612.061168353
```

```
tasaOverfitting <- EoutH10 - EoutH2
cat(c("Para ",Num_experimentos," la tasa de sobreajuste Eout10 - Eout2 =", tasaOverfitting))
```

```
## Para  150  la tasa de sobreajuste Eout10 - Eout2 = 497610.723828226
```

```
    experimento_Overfiting(Qf = 3, N = 20, sigma = 1, gradoH = 10,seed = 9,dibujar = T)
```






```
## [1] 0.7897916
```

b) ¿Por qué normalizamos f? (Ayuda: interpretar el significado de σ) Para que el ruido afecte por igual a todas las funciones.

Para normalizar he dividido por $\sum_{i=0}^{Q_f} a_i^2$ en lugar de dividir por: $\sqrt{\sum_{q=0}^{Q_f} \frac{1}{2q+1}}$ que es lo que dice la ayuda (porque creo que está mal lo que se dice en la ayuda) Mi argumentacion es la siguiente:
$\mathbb{E}_{a,x}[f^2] = 1 \Leftrightarrow \int_{-1}^{1} \sum_{i=0}^{n} (a_i L_i(x))^2 dx = 1$ Donde $L_i(x)$ es el polinomio de Legendre normalizado de grado i
Ahora vamos a desarrollar $\int_{-1}^{1} \sum_{i=0}^{n} (a_i L_i(x))^2 dx$
$\int_{-1}^{1} \sum_{i=0}^{n} (a_i L_i(x))^2 dx = \int_{-1}^{1} \sum_{i=0}^{n} a_i^2 L_i(x)^2 + R(x) dx$

Donde $R(x)$ es una combinacion lineal de polinomios de Legendre de la forma: $\sum_{\mathcal{I}} a_i L_i(x) L_j(x)$ con $i \neq j$.

Luego por la ortogonalidad de los polinomios de Legendre tenemos $\int_{-1}^{1} R(x) dx = 0$
Con lo que $\int_{-1}^{1} \sum_{i=0}^{n} (a_i L_i(x))^2 dx = \int_{-1}^{1} \sum_{i=0}^{n} a_i^2 L_i(x)^2 dx$
Ahora utilizando propiedades usuales de las integrales tenemos:
$\int_{-1}^{1} \sum_{i=0}^{n} a_i^2 L_i(x)^2 dx = \sum_{i=0}^{n} \int_{-1}^{1} a_i^2 L_i(x)^2 dx = \sum_{i=0}^{n} a_i^2 \int_{-1}^{1} L_i(x)^2 dx = \sum_{i=0}^{n} a_i^2$
Donde hemos utilizado en el último paso que los L_i estan normalizados

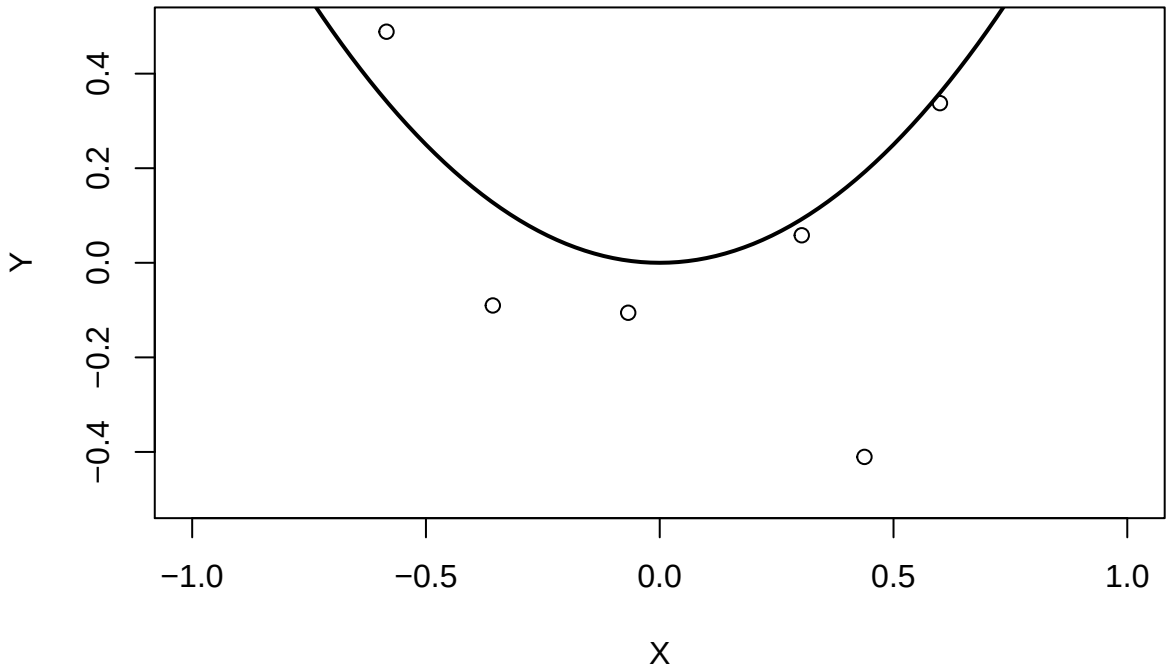
Por tanto para normalizar un polinomio f cualquiera expresado como $f = \sum_{i=0}^{n} (a_i L_i(x))$ (donde $n = Q_f$) lo que debemos hacer es dividir por $\sum_{i=0}^{n} a_i^2$

Regularizacion

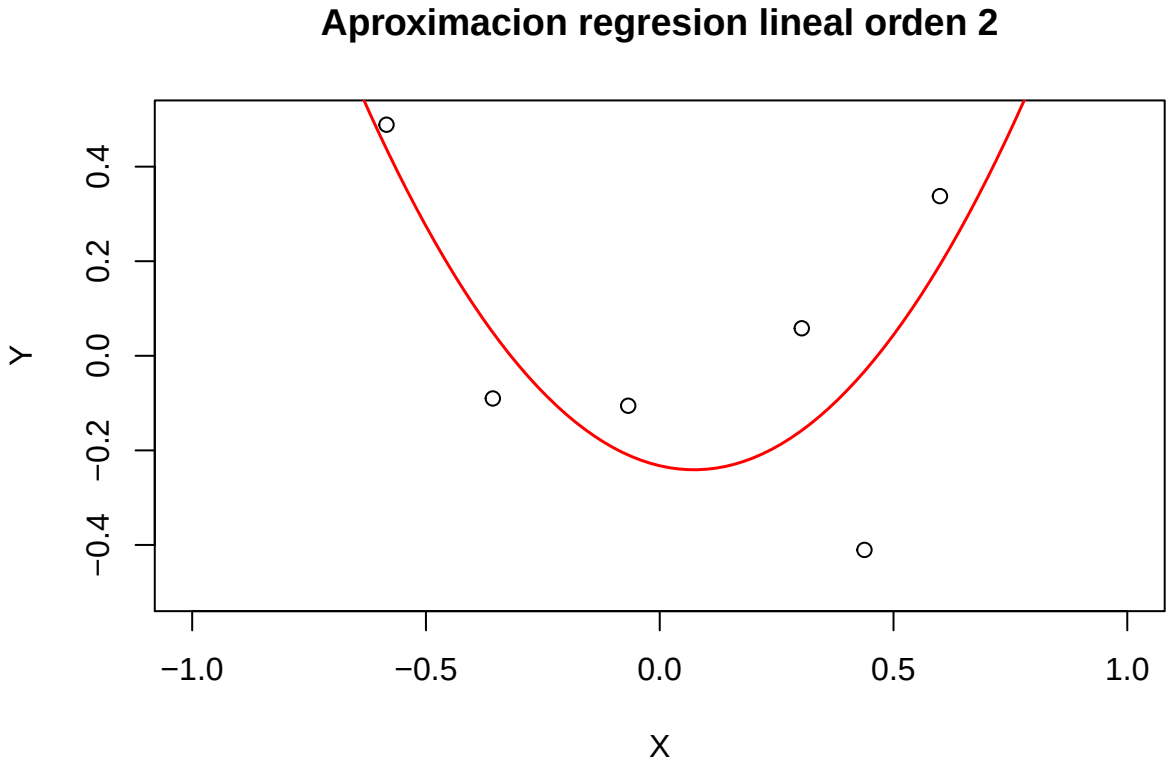
Para $d = 3$ (dimensión) generar un conjunto de N datos aleatorios x_n, y_n de la siguiente forma. Para cada punto x_n generamos sus coordenadas muestreando de forma independiente una $\mathcal{N}(0, 1)$. De forma similar generamos un vector de pesos de $(d+1)$ dimensiones w_f , y el conjunto de valores $y_n = w_f^t x_n + \sigma \epsilon n$, donde ϵn es un ruido que sigue también una $\mathcal{N}(0, 1)$ y σ^2 es la varianza del ruido; fijar $\sigma = 0, 5$. Usar regresión lineal con regularización "weight decay" para estimar w_f con w_{reg}. Fijar el parámetro de regularización a $0, 05/N$.

```
regress_lin_decay <- function(datos, label, lambda){
    X <- datos; #datos es una matriz [N x 3] ---->>> (1, x0, x1)
    XtX <- t(X) %*% X
    lambdaI <- diag(lambda, nrow = nrow(XtX))
    H <- (invertirMatrizSVD(XtX + lambdaI) %*% t(X))
    w <- H %*% t(matrix(label, nrow = 1))
    # hiperplano: w1 + w2*x + w3*y = 0 ==>
    return(w)
}
```

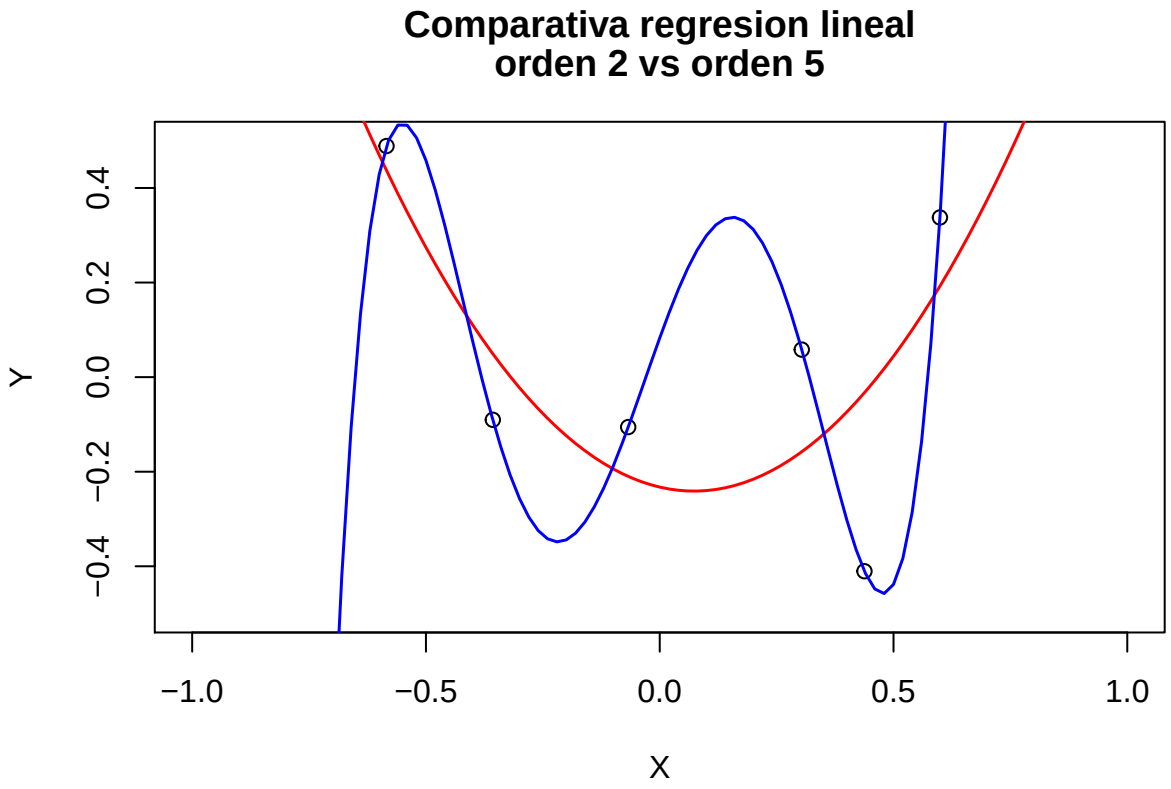
```
set.seed(8) #Para obtener un ejemplo bonito visualmente
N <- 6
sigma2 <- 0.2
par(mfrow=c(1,1)) #Para las graficas

X <- runif(N, -1,1)
Y <- X^2
Ruido <- rnorm(N, 0, sigma2)

Y <- Y + Ruido

plot(X, Y, xlim = c(-1,1), ylim = c(-0.5,0.5), main="Funcion real y datos con ruido")
curve(x^2, add=T, col="black", lw=2)
```



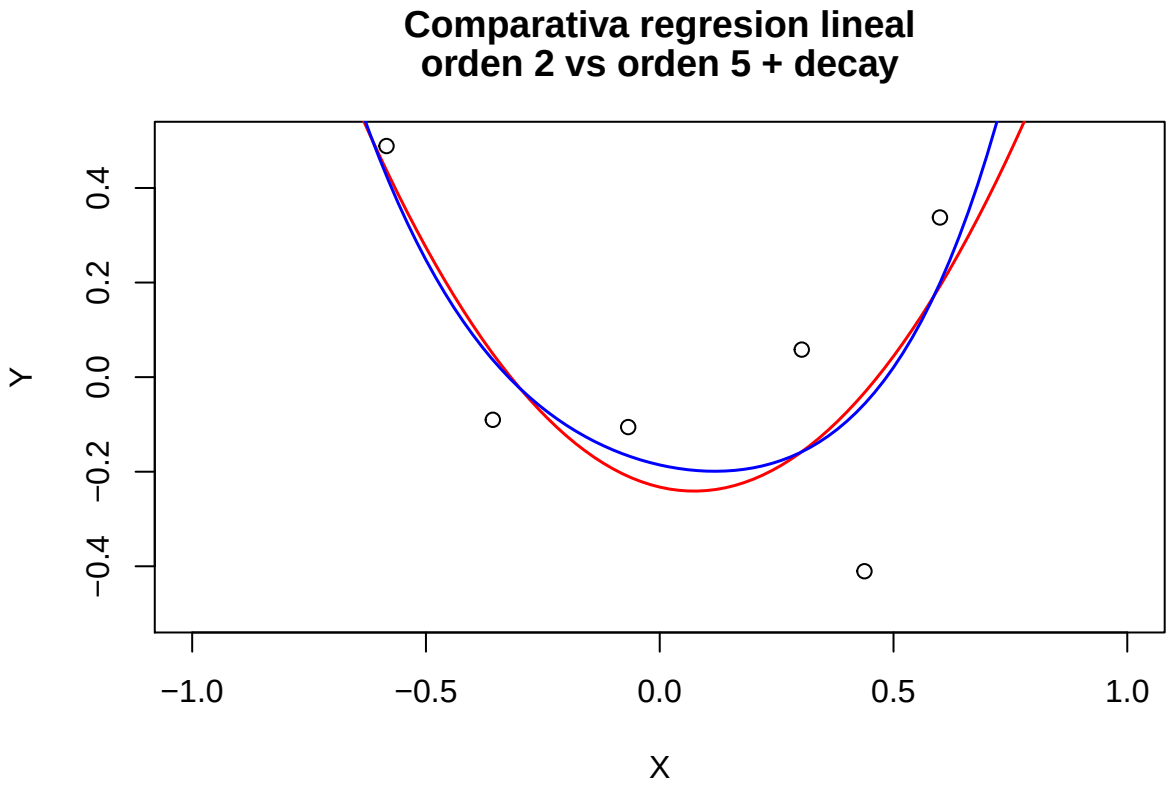

```r
#Aproximacion con polinimio de orden 2 usando regresion lineal:
data <- matrix(c(rep(1, N), X, X^2), ncol = 3, byrow = F)
etiquetas <- t(t(Y))
w2 <- regress_lin(data, etiquetas)
plot(X, Y, xlim = c(-1,1), ylim = c(-0.5,0.5) , main="Aproximacion regresion lineal orden 2")
curve(w2[1] + w2[2]*x + w2[3]*x^2, add=T, col="red", lw=1.5)
```




```r
#Aproximacion con un polinomio de orden 5 (perfect fit)
data <- matrix(c(rep(1, N), X, X^2, X^3, X^4, X^5), ncol = 6, byrow = F)
etiquetas <- t(t(Y))
w5 <- regress_lin(data, etiquetas)
plot(X, Y, xlim = c(-1,1), ylim = c(-0.5,0.5), main=c("Comparativa regresion lineal","orden 2 vs orden 5
curve(w2[1] + w2[2]*x + w2[3]*x^2, add=T, col="red", lw=1.5)
curve(w5[1] + w5[2]*x + w5[3]*x^2 + w5[4]*x^3 + w5[5]*x^4 + w5[6]*x^5 , add=T, col="blue", lw=1.5)
```




```r
#Aproximacion con un polinomio de orden 5 con weight decay (lambda = 0.05/N)
data <- matrix(c(rep(1, N), X, X^2, X^3, X^4, X^5), ncol = 6, byrow = F)
etiquetas <- t(t(Y))
w5decay <- regress_lin_decay(data, etiquetas, lambda = 0.05/N)
plot(X, Y, xlim = c(-1,1), ylim = c(-0.5,0.5), main=c("Comparativa regresion lineal","orden 2 vs orden 5
curve(w2[1] + w2[2]*x + w2[3]*x^2, add=T, col="red", lw=1.5)
curve(w5decay[1] + w5decay[2]*x + w5decay[3]*x^2 + w5decay[4]*x^3 + w5decay[5]*x^4 + w5decay[6]*x^5 , ad
```




```r
#Simula Validacion Cruzada de puntos en R^3
simulaCrossValidation <- function(N, lambda, sigma){
    #Especificacion de parametros:
    xlim <- c(-1,1) #intervalo X
    dim <- 3  #No es un parametro que pueda cambiarse sin modificar el codigo...

    #Generamos las muestras con sus etiquetas y los pesos que queremos estimar
    X <- simula_unif(N, dim,  xlim[1], xlim[2])
    data <- matrix(c(rep(1, N), X[[1]], X[[2]], X[[3]]), ncol=4, byrow = F)
    wf <- rnorm(dim+1, 0, 1)
    wf <- t(t(wf))
    Y <- data %*% wf
    Ruido <- rnorm(N, 0, sigma)
    Y <- Y + Ruido*sigma

    #Estimamos los pesos wf mediante regresion lineal
    wreg <- regress_lin_decay(datos = data, label = Y, lambda = lambda)
    #print(wf)
    #print(wreg)

    error.i <- vector(length = N)
    #Comenzamos validacion cruzada para estimar el error:
    for(i in 1:N){
        data.test <- data[i,]
        Y.test <- Y[i,]
        data.train <- data[-i,]
        Y.train <- Y[-i,]
        wreg.i <- regress_lin_decay(data.train, Y.train, lambda/N)
        error.i[i] <- (Y.test - (data.test%*%wreg.i))^2
```

```
    }
    #print(error.i)
    #E_cv <- mean(error.i)
    #print(E_cv)
    return(error.i)
}
simulaCrossValidation(118, 0.05, 0.5)
```

```
##   [1] 6.951511e-04 6.184219e-03 4.358690e-02 2.506952e-02 1.567335e-01
##   [6] 1.242632e-01 4.651272e-02 1.012847e-01 1.861922e-01 2.315051e-03
##  [11] 1.772184e-02 1.764092e-04 1.186816e-01 8.934001e-02 1.764215e-05
##  [16] 9.314840e-02 2.901539e-03 3.001482e-02 5.605638e-02 4.143434e-02
##  [21] 4.072335e-02 6.992529e-02 2.030026e-02 1.726488e-02 4.120152e-02
##  [26] 2.089157e-03 4.630081e-04 5.216746e-03 8.664280e-02 1.661728e-01
##  [31] 3.307911e-02 8.745695e-02 7.049538e-02 2.447414e-02 1.898811e-03
##  [36] 1.266557e-04 2.282554e-03 4.221888e-03 7.749595e-03 1.853439e-01
##  [41] 1.034804e-01 1.585514e-01 6.275269e-03 2.215163e-03 5.826316e-02
##  [46] 1.416403e-02 8.268816e-04 1.098459e-02 3.086916e-02 2.036628e-01
##  [51] 5.592631e-02 1.380213e-03 6.830127e-02 5.566416e-02 1.852267e-02
##  [56] 6.804876e-02 7.453650e-03 5.766949e-03 2.362987e-03 1.006952e-02
##  [61] 8.128905e-02 7.143782e-03 8.352371e-02 6.306686e-04 2.872531e-04
##  [66] 3.766268e-02 4.400994e-02 2.115305e-02 1.695383e-02 6.090467e-06
##  [71] 7.043669e-03 1.735960e-02 9.469560e-03 2.805907e-01 1.768876e-01
##  [76] 1.553528e-02 2.024623e-02 1.164888e-02 1.118263e-02 9.668761e-02
##  [81] 1.062611e-03 9.871771e-02 7.109343e-02 1.527921e-02 3.323185e-02
##  [86] 2.147129e-02 1.749793e-01 1.914467e-03 1.767128e-02 9.477145e-03
##  [91] 3.992188e-01 3.372762e-02 1.337927e-03 3.588298e-03 5.009277e-02
##  [96] 7.488184e-03 1.299428e-02 2.759836e-03 7.253652e-03 2.603904e-02
## [101] 1.280148e-04 6.404927e-03 6.451533e-02 9.532439e-02 1.284437e-01
## [106] 4.363644e-03 1.339202e-02 5.555742e-03 8.514087e-03 1.086873e-01
## [111] 3.366953e-02 1.106198e-01 2.538268e-02 7.671200e-07 9.058312e-02
## [116] 9.512821e-03 2.302248e-02 6.055539e-03
```

```
listaValores <- seq(18, 118, by = 10)
listaErrores <- list()
for(i in listaValores){
    listaErrores <- simulaCrossValidation(i, 0.05, 0.5)
}
print(listaErrores)
```

```
##   [1] 2.110674e-01 4.104962e-02 2.070255e-03 8.027727e-03 3.053583e-02
##   [6] 1.568738e-01 3.808032e-01 1.469764e-02 5.077410e-02 6.691635e-03
##  [11] 2.174552e-03 2.766122e-04 5.537514e-01 5.993266e-04 2.424681e-01
##  [16] 2.090233e-02 1.212627e-01 1.339690e-02 5.526666e-02 3.249661e-02
##  [21] 3.153050e-03 2.045770e-02 1.149411e-03 6.125122e-03 4.237952e-02
##  [26] 1.786393e-02 1.368482e-02 4.385086e-03 1.346339e-01 2.596682e-02
##  [31] 6.593994e-02 2.062387e-03 3.252141e-04 3.541077e-02 4.278575e-02
##  [36] 8.598278e-02 2.527037e-01 6.249341e-02 7.553529e-03 1.365503e-02
##  [41] 1.225936e-02 1.205550e-01 1.022504e-02 9.535477e-02 4.397991e-01
##  [46] 5.705813e-05 2.986453e-02 5.193934e-02 8.397258e-02 3.719731e-03
##  [51] 3.935539e-02 1.752498e-01 7.928625e-03 2.742608e-02 7.817292e-02
##  [56] 2.848919e-02 1.063495e-01 2.296827e-01 3.036832e-02 3.170849e-02
```

```
##  [61] 1.405887e-02 8.233515e-02 9.429310e-02 2.354476e-03 1.138953e-01
##  [66] 9.336707e-02 1.406360e-02 3.126354e-03 8.321177e-02 7.764153e-02
##  [71] 3.286772e-03 7.962715e-02 1.809324e-01 4.879914e-01 6.433003e-01
##  [76] 7.210269e-02 2.068142e-02 1.399187e-02 2.191579e-01 5.911907e-03
##  [81] 2.564772e-02 1.962654e-01 3.792997e-02 2.794764e-02 8.443313e-03
##  [86] 7.925525e-03 2.874915e-04 4.064723e-02 1.963543e-02 3.882380e-03
##  [91] 1.090633e-01 2.760842e-02 2.256514e-03 3.096753e-02 9.000772e-02
##  [96] 5.319583e-02 1.113626e-02 8.267830e-02 6.830136e-02 1.186493e-01
## [101] 1.843449e-01 4.890605e-02 2.273886e-02 3.755872e-02 4.390037e-02
## [106] 6.300158e-02 1.174568e-01 5.609239e-03 3.093797e-04 3.158833e-02
## [111] 7.813579e-02 5.133470e-02 2.456893e-03 9.414801e-04 9.767043e-03
## [116] 2.463991e-02 2.399670e-02 7.589690e-04
```

```
experimentoCV <- function(N, lambda, sigma){
    errores <- simulaCrossValidation(N, lambda, sigma)
    return(c(errores[1], errores[2], mean(errores)) )
}

M <- 1000
for(N in listaValores){
    e1  <- vector(length = M)
    e2  <- vector(length = M)
    ecv <- vector(length = M)
    for(i in 1:M){
        aux <- experimentoCV(N, 0.05, 0.5)
        e1[i] <- aux[1]
        e2[i] <- aux[2]
        ecv[i] <- aux[3]
    }
    m <- matrix(c(mean(e1), mean(e2), mean(ecv), var(e1), var(e2), var(ecv)),
                ncol = 3, byrow = T)
    cat(c("Valores para N=", N, " \n"))
    print(m)
}
```

```
## Valores para N= 18
##             [,1]       [,2]        [,3]
## [1,] 0.07655991 0.08214743 0.081936747
## [2,] 0.01135691 0.01250354 0.001008905
## Valores para N= 28
##             [,1]       [,2]         [,3]
## [1,] 0.07687060 0.07492822 0.0739623052
## [2,] 0.01011712 0.01230339 0.0004560911
## Valores para N= 38
##              [,1]       [,2]         [,3]
## [1,] 0.070312590 0.07412570 0.0710501039
## [2,] 0.009562615 0.01017326 0.0002809191
## Valores para N= 48
##             [,1]        [,2]         [,3]
## [1,] 0.06858370 0.062432823 0.0679595997
## [2,] 0.01016368 0.007203714 0.0002018994
## Valores para N= 58
##             [,1]        [,2]         [,3]
## [1,] 0.07135479 0.064164307 0.0675676083
```

```
## [2,] 0.01015797 0.007616106 0.0001704661
## Valores para N= 68
##              [,1]        [,2]        [,3]
## [1,] 0.068288525 0.065337499 0.065881299
## [2,] 0.009686404 0.009174426 0.000145272
## Valores para N= 78
##              [,1]        [,2]         [,3]
## [1,] 0.069176364 0.065935180 0.0660082755
## [2,] 0.009645813 0.008905777 0.0001180285
## Valores para N= 88
##             [,1]        [,2]         [,3]
## [1,] 0.06922312 0.063995580 0.0655268210
## [2,] 0.01018316 0.007427425 0.0001052061
## Valores para N= 98
##              [,1]        [,2]         [,3]
## [1,] 0.061962091 0.064032501 6.478778e-02
## [2,] 0.008116001 0.007916901 8.852286e-05
## Valores para N= 108
##              [,1]        [,2]         [,3]
## [1,] 0.066716811 0.068298550 6.485722e-02
## [2,] 0.007947872 0.009071743 7.551112e-05
## Valores para N= 118
##              [,1]        [,2]         [,3]
## [1,] 0.064531588 0.066127813 6.487764e-02
## [2,] 0.008898642 0.009146999 7.400862e-05
```

¿Cuál debería de ser la relación entre el promedio de los valores de e_1 y el de los valores de E_{cv} ? ¿y el de los valores de e_2? Argumentar la respuesta en base a los resultados de los experimentos.
El promedio de e_1 debe ser el mismo que el de cualquier e_i ya que hay muchos datos y las varianzas son pequeñas. Como $e_{cv} = \sum_{i=0}^{M} e_i$ tenemos que el promedio debe ser el mismo.

¿Qué es lo que contribuye a la varianza de los valores de e_1?
Que los valores de cada ejecucion son independientes

Si los errores de validación-cruzada fueran verdaderamente independientes, ¿cual sería la relación entre la varianza de los valores de e_1 y la varianza de los de E_{cv}?

$var(E_{cv}) = \frac{1}{N^2} var(\sum_{i=0}^{N} e_i)) = \frac{1}{N^2} var(\sum_{i=0}^{N} e_i)) = \frac{1}{N^2} \sum_{i=0}^{N} var(e_i)) = \frac{1}{N^2} N var(e_1)) = \frac{1}{N} var(e_1))$

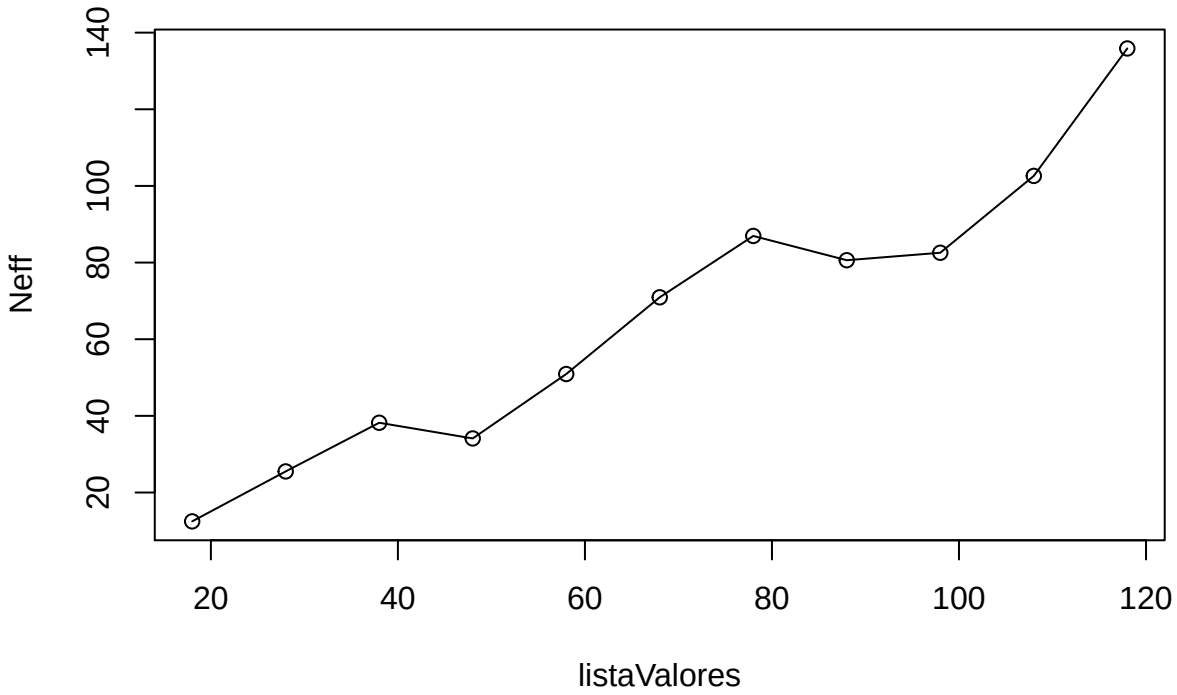
Una medida del número efectivo de muestras nuevas usadas en el cálculo de E_{cv} es el cociente entre la varianza de e_1 y la varianza de E_{cv}. Explicar por qué, y dibujar, respecto de N, el número efectivo de nuevos ejemplos (N_{eff}) como un porcentaje de N. NOTA: Debería de encontrarse que N_{eff} está cercano a N.

```
M <- 1000
Neff <- vector(length = length(listaValores))
index <- 1
for(N in listaValores){
    e1  <- vector(length = M)
    ecv <- vector(length = M)
    for(i in 1:M){
        aux <- experimentoCV(N, 0.05, 0.5)
        e1[i] <- aux[1]
        ecv[i] <- aux[3]
    }
    Neff[index] <-  var(e1)/var(ecv)
    index <- index + 1
```

```
    #print(Neff)
}
print(Neff)
```

```
##  [1]  12.47000  25.50153  38.20379  34.11013  50.91621  70.94388  86.94863
##  [8]  80.61562  82.56420 102.61626 135.87088
```

```
plot(listaValores, Neff, type = "o")
```



Como hemos visto antes $var(E_{cv}) = \frac{1}{N} var(e_i)$ por tanto $\frac{var(E_{cv})}{var(e_i)} = \frac{1}{N} \implies N_{eff} = N$

Si se incrementa la cantidad de regularización, ¿debería N_{eff} subir o bajar?. Argumentar la respuesta. Ejecutar el mismo experimento con $\lambda = 2,5/N$ y comparar los resultados del punto anterior para verificar la conjetura.

La N_{eff} debe bajar ya que estás disminuyendo la varianza de los e_i (tambien disminuyes la varianza de los e_{cv} pero esta disminucion es provocada por la de los e_i y por tanto se sigue manteniedo $var(E_{cv}) = \frac{1}{N} var(e_i)$) Como $var(E_{cv}) = \frac{1}{N} var(e_i)$ y $var(e_i)$ baja tenemos que $var(e_i) \to \alpha var(e_i) \implies N_{eff} \to \alpha N_{eff}$ con $\alpha \in (0, 1)$ luego el N_eff baja.

```
M <- 1000
Neff_reg <- vector(length = length(listaValores))
index <- 1
for(N in listaValores){
    e1  <- vector(length = M)
    ecv <- vector(length = M)
    for(i in 1:M){
        aux <- experimentoCV(N, 2.5, 0.5)
        e1[i] <- aux[1]
        ecv[i] <- aux[3]
    }
```

```
    Neff_reg[index] <-  var(e1)/var(ecv)
    index <- index + 1
    #print(Neff_reg)
}
print(Neff_reg)
```

```
##  [1]  11.37378  28.62859  37.78410  47.05016  44.21657  57.62352  83.85068
##  [8]  90.44454  86.29685  92.08953 123.69307
```

```
plot(listaValores, Neff_reg, type = "o", col="blue")
points(listaValores, Neff, type="o", col="orange")
points(listaValores, listaValores, type = "l", add=T, col="black")
```

```
## Warning in plot.xy(xy.coords(x, y), type = type, ...): "add" is not a
## graphical parameter
```