PRÁCTICA 2B: REGRESIÓN LINEAL MÚLTIPLE CON GRADIENTE DESCENDENTE

ANÁLISIS ESTADÍSTICO MULTIVARIANTE

GRADO EN CIENCIA E INGENIERÍA DE DATOS

Sumario: En esta práctica veremos cómo estimar los coeficientes del modelo de Regresión Lineal Múltiple (RLM) usando el algoritmo del Gradiente Descendente (GD) para minimizar la función costo, es decir, para minimizar el error cuadrático medio (MSE).
La práctica contempla un ejemplo de introducción al método GD, así como su aplicación en Regresión Lineal Simple y Múltiple. Por último, se ilustra cómo afecta al método GD la existencia de multicolinealidad en RLM.

1. Algoritmo GD para funciones reales de una variable

En diversas técnicas estadísticas o de análisis de datos, se requiere la estimación de parámetros desconocidos. Y en muchas ocasiones, dichos parámetros a estimar surgen de encontrar el mínimo de una función de costo. Es en este contexto donde el método del Gradiente Descendente puede ser de gran utilidad, puesto que proporciona de manera rápida y eficiente el óptimo o al menos una buena aproximación a éste. El libro de Análisis Estadístico Multivariante del profesor Jorge Navarro ha servido como base para el desarrollo de esta práctica.

Para ilustrar el funcionamiento del algoritmo GD, vamos a minimizar la siguiente función real de una variable:

$$J(x) = x^2 + x + 1$$

Recordemos los **pasos del algoritmo GD**:

Paso 0: Sean x_1 valor real y $\alpha > 0$.

Paso 1: Hacer $x_{i+1} = x_i - \alpha J'(x_i)$.

Paso 2: Repetir el paso 1 para $i = 1, \ldots, m$.

Si la derivada es positiva (negativa), es decir si J es creciente (decreciente) en ese punto, el algoritmo mueve el punto hacia la izquierda (derecha) para que J disminuya. El salto es proporcional a α y a la derivada. De esta forma, aunque α sea constante, el salto irá disminuyendo cuando nos acerquemos a la solución (que tiene derivada cero).

El valor inicial x_1 se puede fijar por el usuario o tomar de forma aleatoria (cuanto más cerca esté de la solución, más rápido irá el algoritmo). El paso o salto $\alpha > 0$ (*learning rate*) debe tener un valor adecuado. Si es muy pequeño, el algoritmo irá muy lento, pero si es muy grande, el algoritmo puede diverger.

Si hay un único mínimo local de $J(x)$, este algoritmo nos conducirá a él si tomamos un α adecuado. En particular, si la función costo $J(x)$ es convexa, existirá un único mínimo local. Si hay varios mínimos locales, el algoritmo puede llevarnos a un mínimo local que no sea el mínimo global, y el mínimo local al que nos acerquemos dependerá del valor inicial x_1 que elijamos para el algoritmo.

Vamos a implementar el algoritmo GD para la función costo $J(x) = x^2 + x + 1$. Al ser una función muy sencilla, podemos derivarla manualmente y comprobar que el único mínimo local se alcanza en $x = -0.5$, con valor óptimo $J(-0.5) = 0.75$.

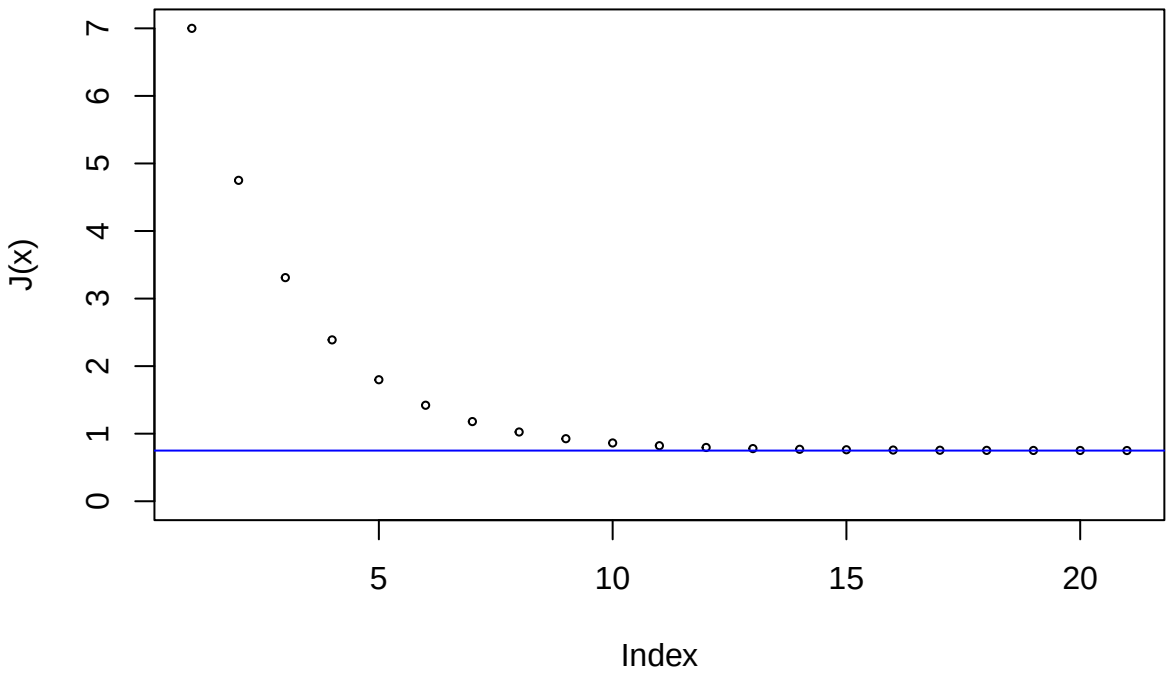
Veamos qué resultados nos proporciona el algoritmo GD para este ejemplo.

```r
#Definimos función costo
J <- function(x) {
  x^2 + x + 1
  }
#Definimos su derivada, necesaria para el Paso 1
Jp <- function(x) {
  2*x + 1
  }
#Fijamos el número de iteraciones a realizar "m" y determinamos el tamaño del vector "x"
m <- 20
x <- 1:(m+1)

#Fijamos el valor inicial del algoritmo.
#Podemos probar a cambiar el valor inicial, acercándolo y alejándolo del óptimo.
x[1] <- 2

#Fijamos el valor del learning rate alpha.
 alfa <- 0.1

#Calculamos todos los valores "x_i" como indica el Paso 1

for (i in 1:m) {
  x[i + 1] <- x[i] - alfa * Jp(x[i])
}

#Mostramos el vector "x" resultado del algoritmo
x
```

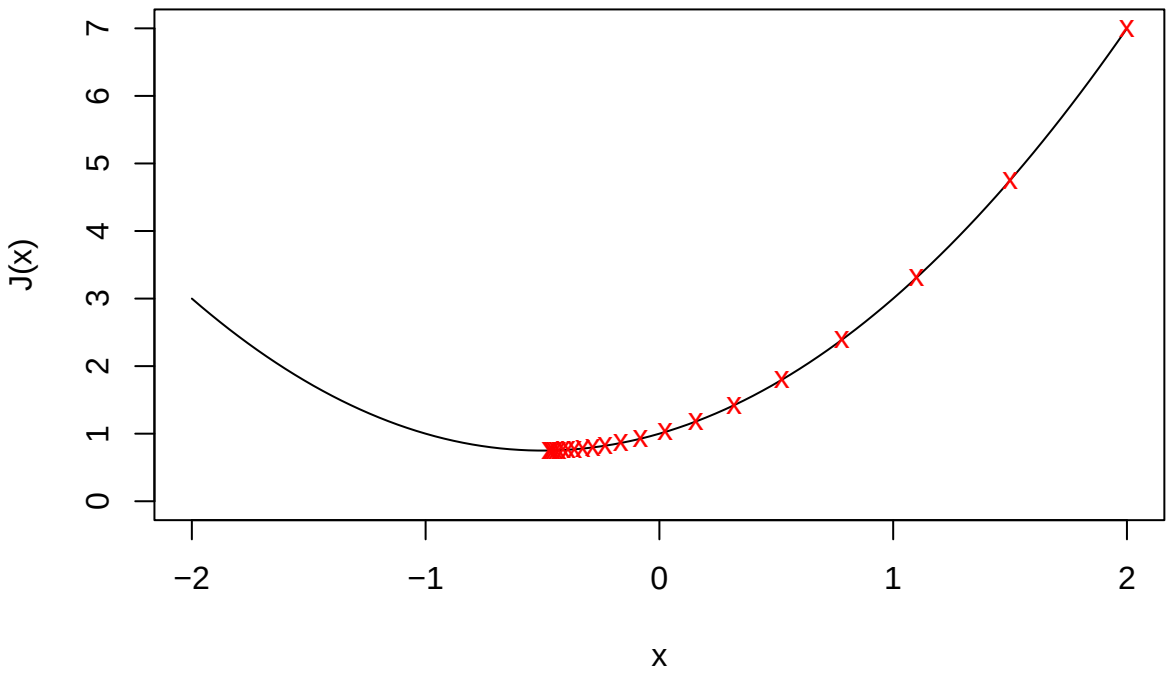
```
##  [1]  2.0000000  1.5000000  1.1000000  0.7800000  0.5240000  0.3192000
##  [7]  0.1553600  0.0242880 -0.0805696 -0.1644557 -0.2315645 -0.2852516
## [13] -0.3282013 -0.3625610 -0.3900488 -0.4120391 -0.4296313 -0.4437050
## [19] -0.4549640 -0.4639712 -0.4711770
```

Podemos representar la función costo $J(x)$ en cada iteración para ver su convergencia al óptimo.

```r
#Representamos la función costo para cada iteración
plot(J(x), ylim = c(0, 7), cex = 0.5)
abline(h = 0.75, col = 'blue')
```




```
#Representamos la función costo respecto de cada valor x_i calculado por el
#algoritmo y comprobamos cómo se acerca al óptimo
curve(J(x), -2, 2, ylim = c(0, 7))
text(x, J(x), 'x', col = 'red')
```



Se deja al lector modificar los valores x_1, α y m para comprobar el efecto de los mismos en la convergencia del algoritmo. Por ejemplo, probar con $x_1 = 3$, $\alpha = 0.1$ y $m = 20$. Probar con $x_1 = 3$, $\alpha = 0.1$ y $m = 100$. Y probar que para $x_1 = 2$ y $\alpha = 1$ el algoritmo diverge.

Como es difícil conocer a priori el *learning rate* ($\alpha > 0$) y el número de iteraciones adecuados (m), se puede implementar el algoritmo estableciendo como criterio de parada que la mejora en la función a minimizar sea inferior a una cota fija $\epsilon \geq 0$.

```
#Fijamos la cota epsilon >= 0
#epsilon <- 1/10^9
epsilon <- 0

#Fijamos el máximo número de iteraciones "m" que permitimos en caso
#de que no pare el algoritmo antes atendiendo al epsilon fijado
m_max <- 10000

#Fijamos alpha e iniciamos valores
alpha <- 0.1
x <- array()
x[1] <- 2

mejora <- epsilon + 1
i <- 1

#Repetimos el bucle mientras no se alcance la cota o bien hasta
#el máximo de iteraciones

while(i <= m_max & mejora > epsilon) {
```

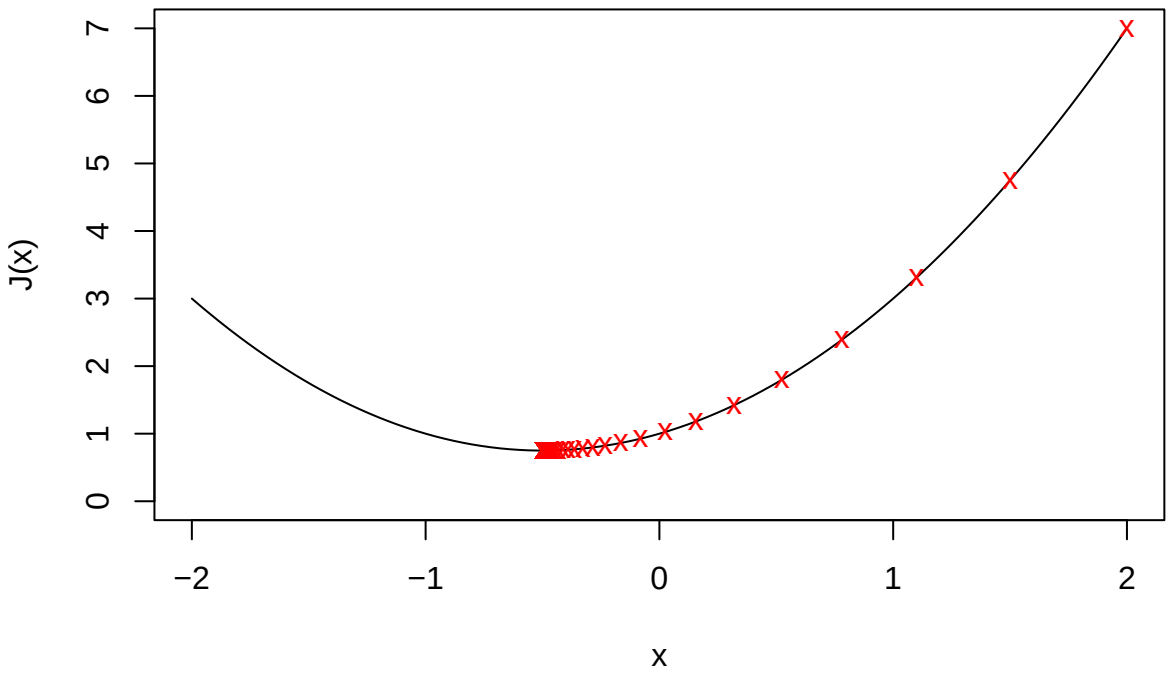
```
  x[i + 1] <- x[i] - alpha * Jp(x[i])
  mejora <- abs(J(x[i]) - J(x[i + 1]))
  i <- i + 1
}
i
```

```
## [1] 88
```

```
x
```

```
##  [1]  2.0000000  1.5000000  1.1000000  0.7800000  0.5240000  0.3192000
##  [7]  0.1553600  0.0242880 -0.0805696 -0.1644557 -0.2315645 -0.2852516
## [13] -0.3282013 -0.3625610 -0.3900488 -0.4120391 -0.4296313 -0.4437050
## [19] -0.4549640 -0.4639712 -0.4711770 -0.4769416 -0.4815533 -0.4852426
## [25] -0.4881941 -0.4905553 -0.4924442 -0.4939554 -0.4951643 -0.4961314
## [31] -0.4969051 -0.4975241 -0.4980193 -0.4984154 -0.4987323 -0.4989859
## [37] -0.4991887 -0.4993510 -0.4994808 -0.4995846 -0.4996677 -0.4997342
## [43] -0.4997873 -0.4998299 -0.4998639 -0.4998911 -0.4999129 -0.4999303
## [49] -0.4999442 -0.4999554 -0.4999643 -0.4999715 -0.4999772 -0.4999817
## [55] -0.4999854 -0.4999883 -0.4999906 -0.4999925 -0.4999940 -0.4999952
## [61] -0.4999962 -0.4999969 -0.4999975 -0.4999980 -0.4999984 -0.4999987
## [67] -0.4999990 -0.4999992 -0.4999994 -0.4999995 -0.4999996 -0.4999997
## [73] -0.4999997 -0.4999998 -0.4999998 -0.4999999 -0.4999999 -0.4999999
## [79] -0.4999999 -0.4999999 -0.5000000 -0.5000000 -0.5000000 -0.5000000
## [85] -0.5000000 -0.5000000 -0.5000000 -0.5000000
```

```
#Representamos la función costo respecto de cada valor x_i calculado
#por el algoritmo y comprobamos cómo se acerca al óptimo
curve(J(x), -2, 2, ylim = c(0, 7))
text(x, J(x), 'x', col = 'red')
```



2. Algortimo GD para Regresión Lineal Simple

Cuando planteamos un problema RLS, la función costo a minimizar es una función real de dos variables: la constante (θ_0) y la pendiente (θ_1).

Si $J(\theta_0, \theta_1)$ representa la función costo a minimizar, el **algortimo GD para el caso bivariante** es el siguiente:

Paso 0: Sean $\theta_0^{(1)}$ y $\theta_1^{(1)}$ valores reales y $\alpha > 0$.

Paso 1: Hacer simultáneamente

$$\theta_0^{(i+1)} = \theta_0^{(i)} - \alpha \cdot \frac{\partial}{\partial \theta_0} J(\theta_0^{(i)}, \theta_1^{(i)}),$$
$$\theta_1^{(i+1)} = \theta_1^{(i)} - \alpha \cdot \frac{\partial}{\partial \theta_1} J(\theta_0^{(i)}, \theta_1^{(i)}).$$

Paso 2: Repetir el paso 1 para $i = 1, \ldots, m$.

Recordemos que, dado un conjunto de datos bidimensionales $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$, la función de costo empírica para un análisis RLS de dicho conjunto de datos viene dada por:

$$J(\theta_0, \theta_1) = \frac{1}{2n} \sum_{i=i}^{n} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right)^2.$$

Y por tanto las derivadas parciales de la función costo son:

$$\frac{\partial}{\partial\theta_0}J(\theta_0,\theta_1)=\frac{1}{n}\sum_{i=i}^{n}\left(\theta_0+\theta_1x^{(i)}-y^{(i)}\right),$$
$$\frac{\partial}{\partial\theta_1}J(\theta_0,\theta_1)=\frac{1}{n}\sum_{i=i}^{n}\left(\theta_0+\theta_1x^{(i)}-y^{(i)}\right)\cdot x^{(i)}.$$

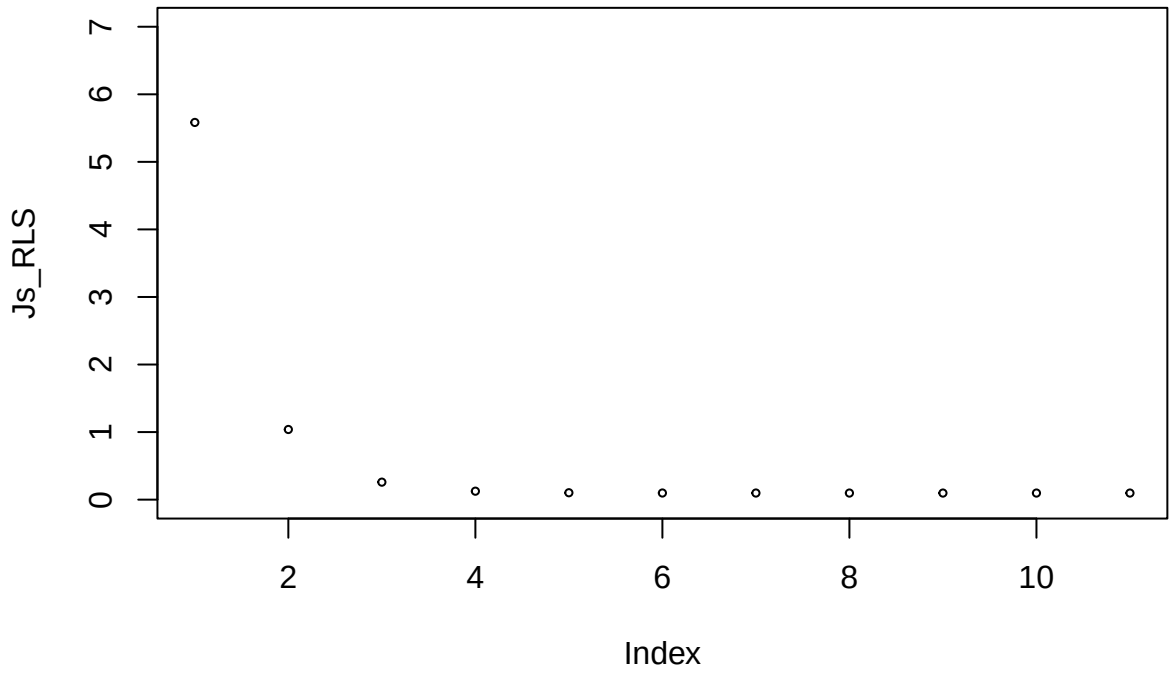
A continuación mostramos cómo programar el algoritmo GD para un ejemplo sencillo de ajuste RLS para los siguiente datos bidimensionales:

$$\{(0,1),(1,2),(1,3),(2,3),(3,4),(4,4)\}.$$

```
#Introducimos los vectores de datos x e y:
x_RLS <- c(0, 1, 1, 2, 3, 4)
y_RLS <- c(1, 2, 3, 3, 4, 4)
n_RLS <- length(x_RLS)

#Definimos la función costo y sus derivadas parciales
h_RLS <- function(theta0, theta1, z) {
  theta0 + theta1 * z
  }
J_RLS <- function(theta0, theta1) {
  sum((h_RLS(theta0, theta1, x_RLS) - y_RLS)^ 2)/(2*n_RLS)
  }
J0_RLS <- function(theta0, theta1) {
  sum((h_RLS(theta0, theta1, x_RLS) - y_RLS))/n_RLS
  }
J1_RLS <- function(theta0, theta1) {
  sum((h_RLS(theta0, theta1, x_RLS) - y_RLS)*x_RLS)/n_RLS
  }
#Fijamos learning rate (alpha), número de iteraciones (m) y
# valores iniciales de los parámetros theta
alfa_RLS <- 0.1
m_RLS <- 10
theta0_RLS <- array()
theta1_RLS <- array()
theta0_RLS[1] <- 2
theta1_RLS[1] <- 2

#Guardaremos el valor de la función costo para cada iteración en el vector Js_RLS
Js_RLS <- array()
Js_RLS[1] <- J_RLS(theta0_RLS[1], theta1_RLS[1])

#Repetimos el Paso 1 "m_RLS" veces
for (i in 1:m_RLS) {
  theta0_RLS[i + 1] <- theta0_RLS[i] - alfa_RLS*J0_RLS(theta0_RLS[i], theta1_RLS[i])
  theta1_RLS[i + 1] <- theta1_RLS[i] - alfa_RLS*J1_RLS(theta0_RLS[i], theta1_RLS[i])
  Js_RLS[i + 1] <- J_RLS(theta0_RLS[i + 1], theta1_RLS[i + 1])
# print(Js_RLS[i + 1])
  }

#Guardamos los valores de los theta y del costo en un dataframe
resultados_RLS <- data.frame(theta0_RLS, theta1_RLS, Js_RLS)
```

Representamos los valores obtenidos de la función costo para comprobar su convergencia:

```
plot(Js_RLS, ylim = c(0, 7), cex = 0.5)
```



Podemos obtener la solución óptima y el valor óptimo exactos usando la función *lm()* de R, que realiza el ajuste lineal por mínimos cuadrados. Comparamos la solución exacta con la proporcionada por el algoritmo GD.

```
modelo_RLS <- lm(y_RLS ~ x_RLS)
minimo_exacto <- J_RLS(modelo_RLS$coefficients[1], modelo_RLS$coefficients[2])

#Comparamos valores óptimos
print("Valor óptimo exacto:")
```

```
## [1] "Valor óptimo exacto:"
```

```
minimo_exacto
```

```
## [1] 0.0974359
```

```
print("Valor óptimo según GD:")
```

```
## [1] "Valor óptimo según GD:"
```

```
Js_RLS[m_RLS + 1] #Valor costo óptimo según método GD
```

```
## [1] 0.09744391
```

```
#Comparamos las soluciones óptimas
print("Solución óptima exacta:")
```

```
## [1] "Solución óptima exacta:"
```

```
modelo_RLS$coefficients #solución exacta
```

```
## (Intercept)       x_RLS
##   1.5076923   0.7230769
```

```
print("Solución óptima según GD:")
```

```
## [1] "Solución óptima según GD:"
```

```
theta0_RLS[m_RLS + 1] #solución GD
```

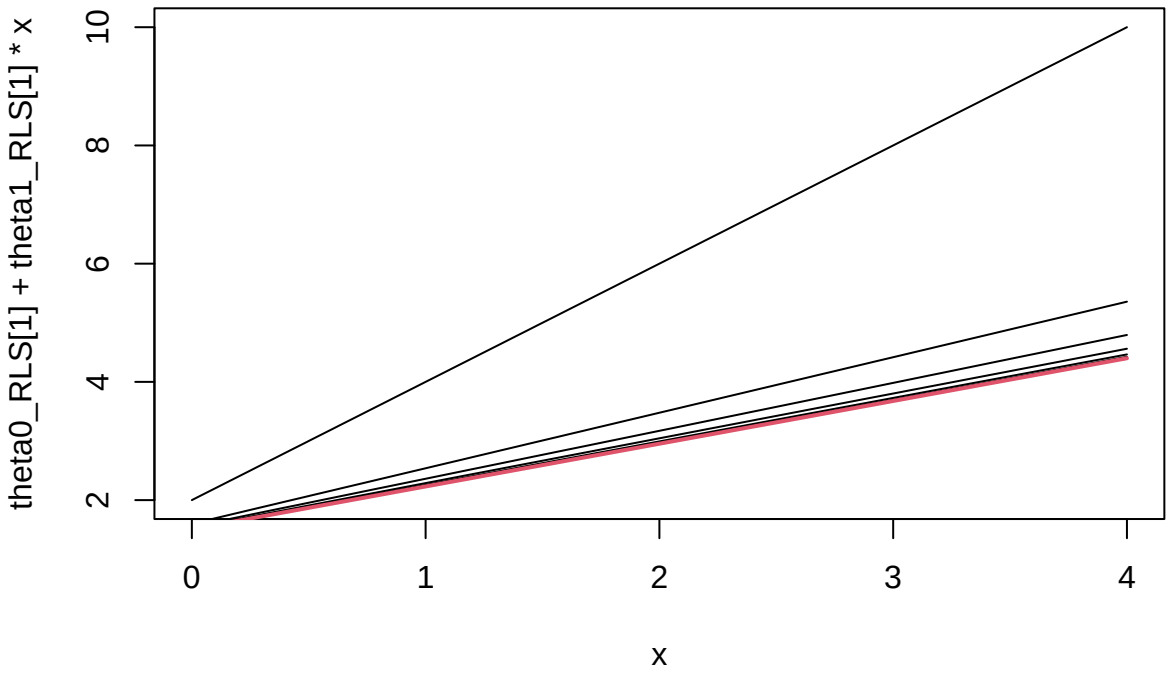
```
## [1] 1.514462
```

```
theta1_RLS[m_RLS + 1] #solución GD
```

```
## [1] 0.7207397
```

Representamos las rectas ajustadas por el método GD en cada iteración y observamos cómo se acercan a la recta ajustada óptima (la que minimiza el error cuadrático medio):

```
curve(theta0_RLS[1] + theta1_RLS[1] * x, from = 0, to = 4)
for (j in 2:m_RLS + 1) {
  curve(theta0_RLS[j] + theta1_RLS[j] * x, from = 0, to = 4, add = TRUE)
}

curve(modelo_RLS$coefficients[1] + modelo_RLS$coefficients[2]*x,
      from = 0, to = 4, add = TRUE, col = 2, lwd = 2)
```



Teniendo en cuenta que la función coste se definió como la mitad del MSE, podemos comprobar que el coste

óptimo coincide con la mitad del MSE para el modelo de regresión:

```
MSE <- mean(modelo_RLS$residuals^2)
MSE
```

```
## [1] 0.1948718
```

```
2*minimo_exacto
```

```
## [1] 0.1948718
```

NOTA: El metodo GD puede funcionar mejor si tipificamos primero los vectores de datos x e y para que estén en la misma escala. Esto implica que habría que deshacer el cambio correspondiente al final del proceso para estimar correctamente los coeficientes de regresión.

3. Algortimo GD para Regresión Lineal Múltiple

En esta sección mostramos cómo adaptar el algoritmo GD para estimar los parámetros de un análisis RLM. Siguiendo los mismos pasos mostrados para el caso de RLS, vamos a aplicar el algoritmo GD a un pequeño conjunto de datos con 2 regresores (X_1 y X_2).

```
#Introducimos los datos muestrales, incluyendo el vector de unos para la cte
x0_RLM <- c(1, 1, 1, 1, 1, 1)
x1_RLM <- c(0, 1, 1, 2, 3, 4)
x2_RLM <- c(2, 2, 4, 4, 2, 3)
y_RLM <- c(1, 2, 3, 3, 4, 4)

n_RLM <-length(x1_RLM)
d <- data.frame(x0_RLM, x1_RLM, x2_RLM, y_RLM)
k <- 2 #número regresores

#Metemos los datos muestrales en una matriz M (matriz de diseño)
M <- matrix(1, nrow = n_RLM, ncol = k + 1)
M[,2] <- x1_RLM
M[,3] <- x2_RLM
# M

#Definimos función costo usando la matriz de diseño M
h_RLM <- function(theta, x) {
  sum(theta*x)
  }
J_RLM <- function(theta,M) {
  0.5*sum((M %*% theta - y_RLM)^2)/n_RLM
  }

#Fijamos iteraciones, learning rate y valores iniciales
m_RLM <- 10 # Número de interaciones
alfa_RLM <- 0.1 # learning rate
theta_RLM <- c(1, 1, 1) #valores iniciales de los theta

J2_RLM <- array() #Vector con actualizaciones de la función costo en cada iteración
hv_RLM <- array() #Vector con actualizaciones de los valores ajustados en cada iteración
Z2_RLM <- matrix(NA, nrow = m_RLM + 1, ncol = k + 1) #Matriz con actualizaciones de
#los theta en cada iteración
J2_RLM[1] <- J_RLM(theta_RLM, M)
Z2_RLM[1,] <-theta_RLM
```

```r
#Repetimos el Paso 1 "m_RLM" veces
for (i in 1:m_RLM) {
  hv_RLM <- M %*% theta_RLM
  theta_RLM <- theta_RLM - (alfa_RLM/n_RLM) * t(M) %*% (hv_RLM - y_RLM)
  J2_RLM[i + 1] <-J_RLM(theta_RLM, M)
  Z2_RLM[i + 1,] <- theta_RLM
}

#Guardamos los valores de los theta y del costo en un dataframe
resultados_RLM <- data.frame(Z2_RLM, J2_RLM)
colnames(resultados_RLM) <- c("theta0", "theta1", "theta2", "costo")
# View(resultados)
```

Podemos obtener la solución óptima y el valor óptimo exactos de dos formas diferentes. Los comparamos con la solución proporcinada por el algoritmo GD.

```r
A <- t(M) %*% M
B <- solve(A)
# A %*%B
thetas_exactos <- B %*% t(M) %*% y_RLM
print("Solución óptima exacta forma 1:")
```

```
## [1] "Solución óptima exacta forma 1:"
```

```r
thetas_exactos  #óptimos exactos
```

```
##           [,1]
## [1,] 0.8129032
## [2,] 0.7032258
## [3,] 0.2580645
```

```r
modelo_RLM <- lm(y_RLM ~ x1_RLM + x2_RLM)
print("Solución óptima exacta forma 2:")
```

```
## [1] "Solución óptima exacta forma 2:"
```

```r
modelo_RLM$coefficients #óptimos exactos
```

```
## (Intercept)      x1_RLM      x2_RLM
##   0.8129032   0.7032258   0.2580645
```

```r
print("Solución óptima según GD:")
```

```
## [1] "Solución óptima según GD:"
```

```r
Z2_RLM[m_RLM + 1, ] #óptimos según GD
```

```
## [1] 0.7800776 0.6709765 0.2917599
```

```r
print("Valor óptimo exacto:")
```

```
## [1] "Valor óptimo exacto:"
```

```r
MSE <- mean(modelo_RLM$residuals^2)
MSE #valor óptimo exacto
```

```
## [1] 0.1419355
```

```r
2*J_RLM(thetas_exactos, M) #valor óptimo exacto
```

```
## [1] 0.1419355
```

```
print("Valor óptimo según GD:")
```

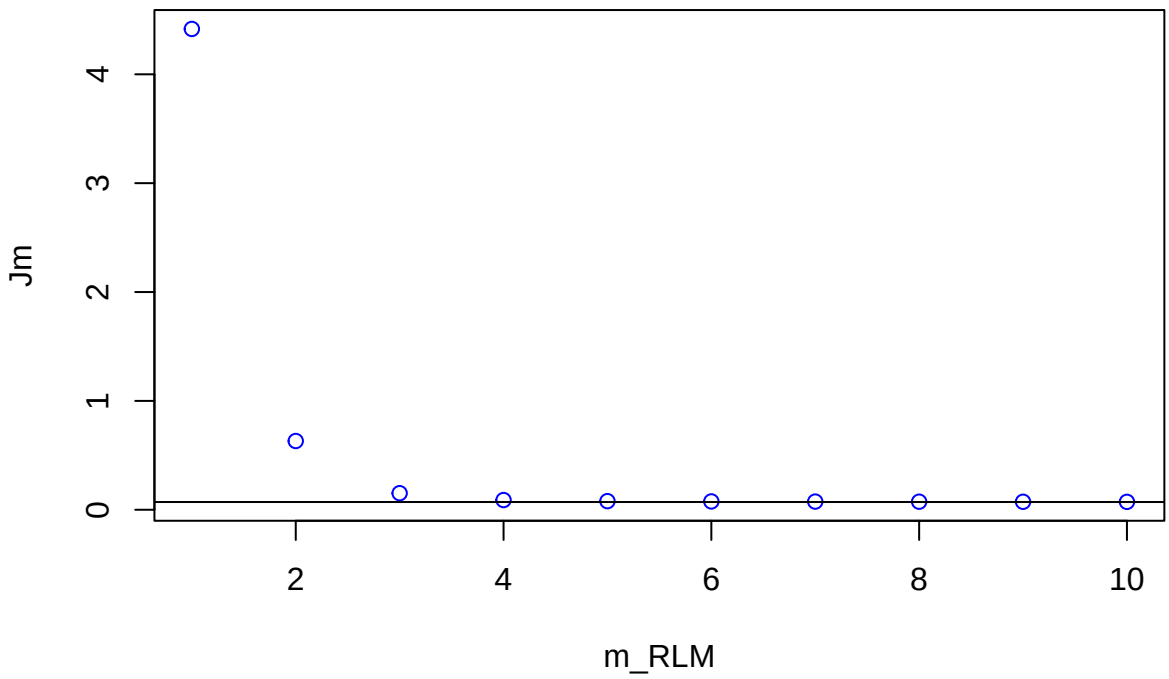
```
## [1] "Valor óptimo según GD:"
```

```
2*J_RLM(Z2_RLM[m_RLM + 1,], M) #valor óptimo según GD
```

```
## [1] 0.1444385
```

Representamos la convergencia de la funcion costo:

```
Jm <-1:m_RLM
for (i in 1:m_RLM) {
  Jm[i] <- J_RLM(Z2_RLM[i, ], M)
  }
min_RLM <-J_RLM(modelo_RLM$coefficients, M)
plot(Jm[1:m_RLM], col = 'blue', xlab = 'm_RLM', ylab = 'Jm')
abline(h = min_RLM)
```



Se deja al lector probar a aumentar el valor de m (número de iteraciones) para comprobar que podemos alcanzar el óptimo con el algoritmo GD. Basta con encontrar una combinación adecuada de valores para m, α y $\theta's$ iniciales.

4. Algortimo GD para RLM con presencia de multicolinealidad

En esta sección veremos cómo se comporta el algortimo GD cuando los datos del análisis RLM presentan multicolinealidad (asociación lineal entre los predictores).

Con el fin de probar el algoritmo GD para diferentes conjuntos de datos (con 2 regresores) y variando los

valores que deben prefijarse (número de iteraciones, *learning rate*, valores iniciales de los $\theta's$), vamos a definir el algoritmo GD en una función con múltiples salidas. Nos apoyaremos en el código de la sección anterior.

```
h_RLM <- function(theta, x) {
  sum(theta*x)
  }
J_RLM <- function(theta, M) {
  0.5*sum((M %*% theta - y_RLM)^2)/n_RLM
  }

algoritmo_GD <- function(x0_RLM, x1_RLM, x2_RLM, y_RLM, m_RLM, alfa_RLM, theta_RLM){
  n_RLM <- length(x1_RLM)
  k <- 2 #número regresores
  M <- matrix(1, nrow = n_RLM, ncol = k + 1)
  M[, 2] <- x1_RLM
  M[, 3] <- x2_RLM
  J2_RLM <- array()
  hv_RLM <- array()
  Z2_RLM <- matrix(NA, nrow = m_RLM + 1, ncol = k + 1)
  J2_RLM[1] <- J_RLM(theta_RLM, M)
  Z2_RLM[1, ] <- theta_RLM
  for (i in 1:m_RLM) {
    hv_RLM <- M %*% theta_RLM
    theta_RLM <- theta_RLM - (alfa_RLM/n_RLM) * t(M) %*% (hv_RLM - y_RLM)
    J2_RLM[i + 1] <- J_RLM(theta_RLM, M)
    Z2_RLM[i + 1, ] <- theta_RLM
  }
  resultados_RLM <- data.frame(Z2_RLM, J2_RLM)
  colnames(resultados_RLM) <- c("theta0", "theta1", "theta2", "costo")
  mylist <- list(resultados_RLM, M)
  return(mylist)
  }
```

Consideremos ahora un conjunto de datos con multicolinealidad perfecta entre los regresores $X1$ y $X2$. Se deja al lector probar con diferentes valores iniciales de los $\theta's$ ($theta\_new$) y del número de iteraciones (m_new) para comprobar que el algoritmo GD converge a diferentes soluciones. Además, dichas soluciones están relacionadas atendiendo a la expresión que relaciona los regresores X_1 y X_2, concretamente cumplen de forma aproximada que:

$\theta_0 = 1.5077$ y que $0.7231 = \theta_1 + 2 * \theta_2$,

debido a que la recta ajustada al eliminar X_2 es:

$Y = 1.5077 + 0.7231 \cdot X_1$

```
#Observar que hay multicolinealidad perfecta porque X2 = 2*X1
x0_new <- c(1, 1, 1, 1, 1, 1)
x1_new <- c(0, 1, 1, 2, 3, 4)
# x2_new <- c(0,2,2,4,6,8)
x2_new <- c(0, 2, 2, 4, 6, 7.8)
y_new <- c(1, 2, 3, 3, 4, 4)
m_new <- 100 # Número de interaciones
alfa_new <- 0.05 # learning rate
theta_new <- c(2, 2, 2) #valores iniciales de los theta
resultados_2 <- algoritmo_GD(x0_new, x1_new, x2_new, y_new,
                             m_new, alfa_new, theta_new)
thetas_costo_GD_2 <- resultados_2[[1]]
```

```
matriz_M_2 <- resultados_2[[2]]
```

```
#Solución proporcionada por algoritmo GD
thetas_costo_GD_2[m_new + 1, ] #thetas y coste según GD
```

```
##       theta0   theta1      theta2      costo
## 101 1.525488 0.905989 -0.09536862 0.09875609
```

¿Qué sucede si decidimos obtener los óptimos exactos de las dos formas que vimos en la sección anterior?

En el caso de la primera forma, que requiere calcular la inversa de la matriz $(M' * M)$, nos dará mensaje de error indicando que la matriz no es invertible (debido a la multicolinealidad). En el caso de usar la función *lm()* de R, advierte la multicolinealidad perfecta y el programa decide suprimir el regresor X_2.

```
#Primera forma
A <- t(matriz_M_2) %*% matriz_M_2
A
```

```
##      [,1] [,2]   [,3]
## [1,]  6.0 11.0  21.80
## [2,] 11.0 31.0  61.20
## [3,] 21.8 61.2 120.84
```

```
B <- solve(A)
thetas_exactos_forma1 <- B %*% t(matriz_M_2) %*% y_new
thetas_exactos_forma1
```

```
##           [,1]
## [1,]  1.307692
## [2,] -9.076923
## [3,]  5.000000
```

```
#Segunda forma
modelo_RLM_new <- lm(y_new ~ x1_new + x2_new)
modelo_RLM_new$coefficients
```

```
## (Intercept)      x1_new      x2_new
##    1.307692   -9.076923    5.000000
```

```
rms::vif(modelo_RLM_new)
```

```
##   x1_new   x2_new
## 2602.083 2602.083
```

Se deja al lector probar con datos donde exista multicolinealidad, pero no perfecta. Por ejemplo, consideremos los datos anteriores pero modificando la última entrada del regresor X_2, quedando de la siguiente manera: $X_2 = (0, 2, 2, 4, 6, 7.8)$. En este caso, sigue existiendo relación lineal estrecha entre X_1 y X_2, pero no es perfecta.

¿Varía la solución del método GD si cambiamos el valor inicial de los thetas y el número de iteraciones? Sí, al igual que lo hacía en el caso de la multicolinealidad perfecta, es decir, se pueden encontrar infinitas soluciones de los $\theta's$ cumpliendo de forma aproximada que:

$\theta_0 = 1.5077$ y que $0.7231 = \theta_1 + 2 * \theta_2$.

¿Podemos en este caso obtener la inversa de (M'*M)? ¿Cuánto valen los thetas óptimos obtenidos de la forma1? En este caso la matriz sí es invertible y los $\theta's$ óptimos valen:

$\theta_0 = 1.307692, \theta_1 = -9.076923, \theta_2 = 5,$.

¿Cuál es la solución óptima usando la forma2, es decir, usando la función *lm()*? En este caso proporciona los mismos coeficientes que la forma1:

$\theta_0 = 1.307692, \theta_1 = -9.076923, \theta_2 = 5,$

pero podemos comprobar que el error estándar de dichos coeficientes es muy alto y los factores de varianza inflada (VIFs) de X_1 y X_2 también son muy altos.

CONCLUSIÓN: Ante la presencia de multicolinealidad, debemos extraer del modelo RLM los predictores necesarios para que se solvente el problema; en nuestro ejemplo, lo adecuado sería eliminar el predictor X_2 y trabajar solo con X_1 para tener un modelo más fiable y estable.