Trabajo2

Gustavo Rivas Gervilla

1. 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 $E(u,v)$.

El gradiente de esta función es el siguiente:

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

entonces definimos tanto la función de error como el gradiente de la función para poder usarlo en los métodos que iremos implementando a lo largo de la práctica.

```
E <- function(x) {
  u = x[1]
  v = x[2]
  (u*exp(v)-2*v*exp(-u))**2
}

gradE <- function(x) {
  u = x[1]
  v = x[2]
  2*(u*exp(v)-2*v*exp(-u))*c(exp(v)+2*v*exp(-u), u*exp(v)-2*exp(-u))
}
```

2) ¿Cuántas iteraciones tarda el algoritmo en obtener por primera vez un valor de $E(u,v)$ inferior a 10^{-14}.

3) ¿Qué valores de (u,v) obtuvo en el apartado anterior cuando alcanzó el error de 10^{-14}.

Nosotros para este algoritmo así como otro que implementaremos más adelante usaremos distintos criterior de parada:

- Que la función alcance o quede por debajo del mínimo establecido.
- Que la distancia entre dos valores consecutivos calculados de f sea menor que una cierta tolerancia.
- Que la distancia entre los vectores de pesos calculados sea menor que dicha tolerancia.
- Que se alcance el máximo de iteracines permitidas.

Entonces aquí mostramos tanto la implementación del algoritmo de gradiente descendente como la de una función para calcular la distancia entre vectores, en lugar de usar la función *dist* que tiene R que calcula distancias entre filas de una matriz.

```r
d <- function(x,y) {
  sqrt(sum((x-y)**2))
}

gradDesc <- function(f, gradf, eta, w0, tol, max_iter = 1000000, dibujar = FALSE) {
  w = w0 #inicializamos pesos
  n_iters = 1
  valores_f = NULL

  repeat {
    gt = gradf(w) #calculamos el gradiente
    vt = -gt
    w_ant = w
    f_ant = f(w)
    valores_f = c(valores_f, f_ant)
    w = w + eta*vt #actualizamos el vector de pesos
    n_iters = n_iters+1

    #cond. de parada
    if (f(w) < tol || n_iters == max_iter || d(w_ant, w) <= tol || abs(f(w) - f_ant) <= tol){
      valores_f = c(valores_f, f(w))
      break
    }
  }

  if (dibujar)
    plot(seq(n_iters), valores_f, type = "l", ylab = "Valor de f", xlab = "Iteraciones")

  cat("Num de iters empleado", n_iters, "\n")
  cat("Valor de f alcanzado", f(w), "\n")
  cat("w obtenido", w, "\n")
}
```

Y los resultados que obtenemos son:

```
## Num de iters empleado 11
## Valor de f alcanzado 1.208683e-15
## w obtenido 0.04473629 0.02395871
```

Si nos fijamos en la función anterior vamos almacenando en *valores_f* los valores de la función que se van alcanzando con los distintos vectores de pesos, esto lo hacemos para poder dibujar una gráfica en la que se muestre cómo evoluciona el valor de la función a medida avanzan las iteraciones del algoritmo, algo que se nos pide en el siguiente apartado.

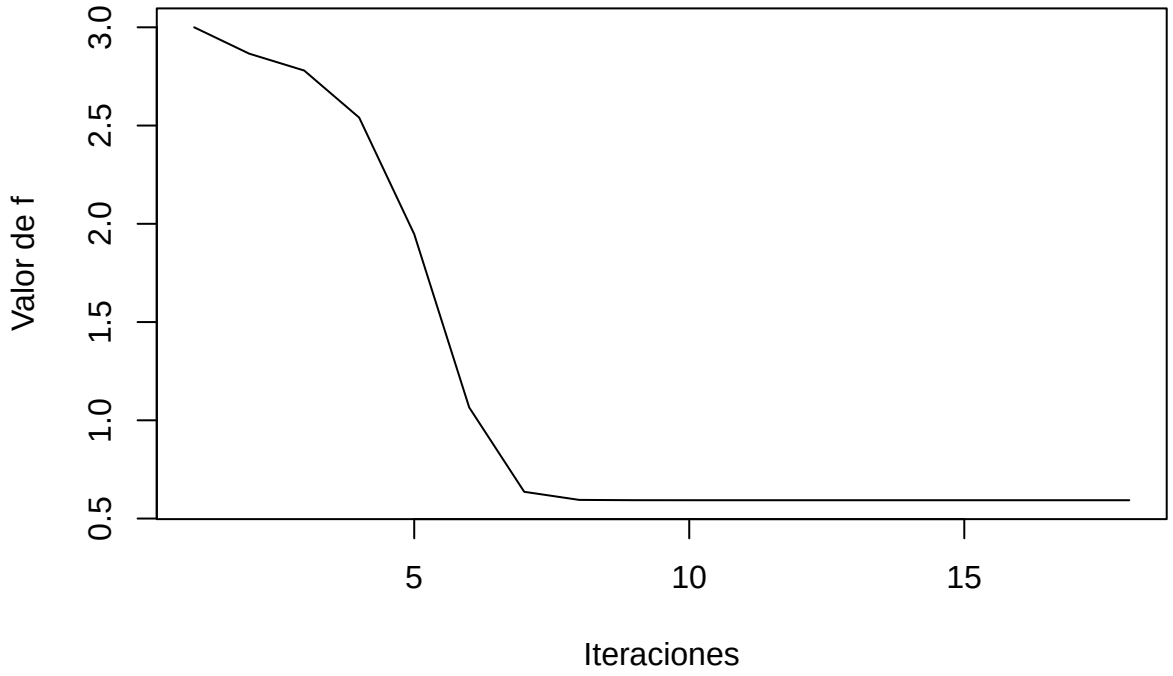
b) Considerar ahora la función $f(x, y) = x^2 + 2y^2 + 2sin(2\pi x)sin(2\pi y)$

1) Usar gradiente descendente para minimizar esta función. Usar como valores iniciales $x_0 = 1$, $y_0 = 1$, la tasa de aprendizaje $\eta = 0.01$ 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 $\eta = 0.1$, comentar las diferencias.

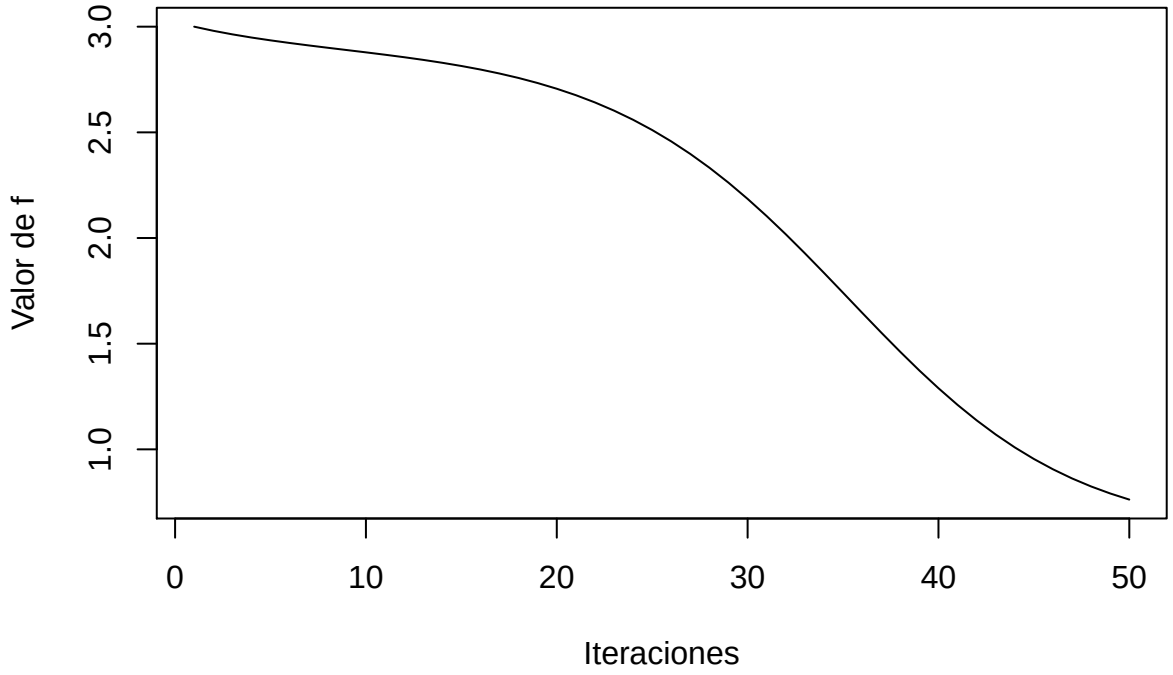
Definimos la función y su gradiente igual que antes:

```
f <- function(X){
  x = X[1]
  y = X[2]

  x**2+2*y**2+2*sin(2*pi*x)*sin(2*pi*y)
}

gradf <- function(X) {
  x = X[1]
  y = X[2]

  c(2*x + 4*pi*sin(2*pi*y)*cos(2*pi*x), 4*y + 4*pi*sin(2*pi*x)*cos(2*pi*y))
}
```

y pasamos a los resultados:




```
## Num de iters empleado 18
## Valor de f alcanzado 0.5932694
## w obtenido 1.21807 0.7128119
```




```
## Num de iters empleado 50
## Valor de f alcanzado 0.7623757
## w obtenido 1.171433 0.7579957
```