Trabajo 2

Alejandro García Montoro

7 de abril de 2016

Modelos lineales

Ejercicio 1 - Gradiente descendente

Apartado 1.a

Apartado 1.a.1

La función $E(u,v) = (ue^v - 2ve^{-u})^2$ tiene las siguientes derivadas parciales:

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

luego su gradiente es

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

Apartado 1.a.2

Para implementar el algoritmo del gradiente descendente necesitamos definir la función gradiente siguiente:

```
# Devuelve el valor del gradiente de E en el punto (u,v)
E.gradiente <- function(punto){
  u <- punto[1]
  v <- punto[2]

  coeff <- 2*(u*exp(v) - 2*v*exp(-u))

  return(coeff * c(exp(v) + 2*v*exp(-u), u*exp(v) - 2*exp(-u)))
}
```

```
E <- function(punto){
  u <- punto[1]
  v <- punto[2]

  return((u*exp(v) - 2*v*exp(-u))^2)
}

gradienteDescendente <- function(f, f.gradiente, w_0 = c(1,1), eta = 0.1, tol = 1e-14, maxIter = 100){
  w <- w_0
  iteraciones <- 0
```

```
  while(f(w) >= tol && iteraciones < maxIter){
    direccion <- -f.gradiente(w)
    w <- w + eta*direccion
    iteraciones <- iteraciones + 1
  }

  return(list("Pesos"=w, "Iter"=iteraciones))
}
```

Ejecutamos la función anteriormente implementada:

```
resultado.E <- gradienteDescendente(E, E.gradiente)
```

Comprobamos así que el algoritmo ha ejecutado 10 iteraciones hasta obtener un valor de $E(u, v)$ inferior a 10^{-14}.

Apartado 1.a.3

De la ejecución anterior podemos ver también que los valores de u y v obtenidos tras las 10 iteraciones son los siguientes:

$$u = 0.0447363$$
$$v = 0.0239587$$

Apartado 1.b

Sea ahora la función $f(x, y) = x^2 + 2y^2 + 2\sin(2\pi x)\sin(2\pi y)$, cuyo gradiente es:

$$\nabla f(x, y) = (2x + 4\pi \sin(2\pi y) cos(2\pi x), 4y + 4\pi \sin(2\pi x) cos(2\pi y))$$

Apartado 1.b.1

Definimos las funciones f y ∇f en R:

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

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

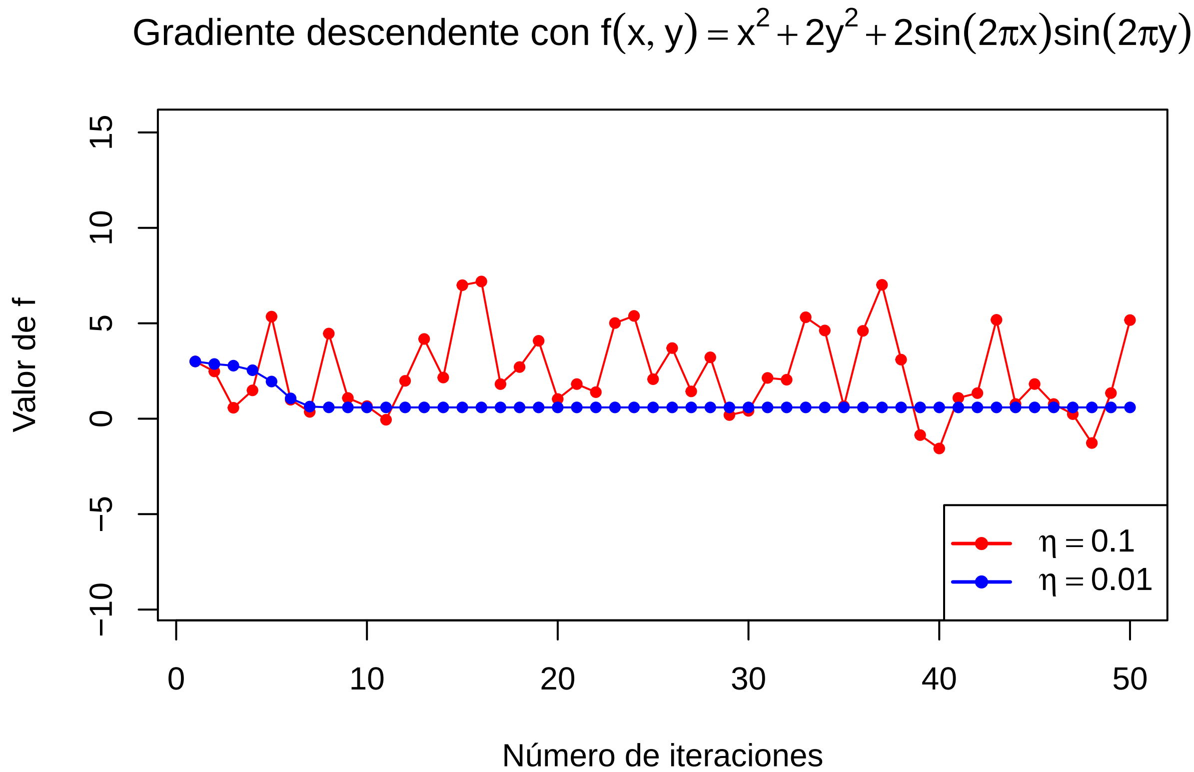
```
# Devuelve el valor del gradiente de f en el punto (x,y)
f.gradiente <- function(punto){
  x <- punto[1]
  y <- punto[2]

  f.grad.x <- 2*x + 4*pi*sin(2*pi*y)*cos(2*pi*x)
  f.grad.y <- 4*y + 4*pi*sin(2*pi*x)*cos(2*pi*y)

  return(c(f.grad.x, f.grad.y))
}
```

Para poder obtener el gráfico solicitado tenemos que modificar la función del gradiente descendente, de manera que devuelva un vector con el valor de la función en las sucesivas iteraciones y el criterio de parada sea ahora únicamente el número de iteraciones:

```
gradienteDescendente.log <- function(f, f.gradiente, w_0 = c(1,1), eta = 0.1, maxIter = 50){
  w <- w_0
  valorIter <- f(w)


  while(length(valorIter) < maxIter){
    direccion <- -f.gradiente(w)
    w <- w + eta*direccion
    valorIter <- c(valorIter, f(w))
  }

  return(list("Pesos"=w, "Iter"=length(valorIter)-1, "Valores"=valorIter))
}
```

Generamos ahora los datos y el gráfico con la función modificada:

```
# Ejecutamos el experimento con ambos parámetros
resultado.f.0.01 <- gradienteDescendente.log(f, f.gradiente, eta = 0.01)
resultado.f.0.1 <- gradienteDescendente.log(f, f.gradiente, eta = 0.1)

# Generamos los dos plots
plot(resultado.f.0.1$Valores, col = 'red', pch = 20, asp=1, type='o',
     main=expression(paste("Gradiente descendente con ",
                           f(x,y) == x^2 + 2*y^2 + 2*plain(sin)(2*pi*x)*plain(sin)(2*pi*y))),
     xlab="Número de iteraciones", ylab="Valor de f")
points(resultado.f.0.01$Valores, col = 'blue', pch = 20, asp=1, type='o')

# Añadimos leyenda de los datos dibujados.
legend("bottomright", c(expression(eta == 0.1), expression(eta == 0.01)),
       bg="white", col=c("red", "blue"), lty=1, pch=20, lwd=1.75)
```



Apartado 1.b.2

Redefinimos la función, de nuevo, para que devuelva también un vector de los puntos por los que pasa. El criterio de parada sigue siendo el número máximo —cincuenta en este caso— de iteraciones:

```
gradienteDescendente.log2 <- function(f, f.gradiente, w_0 = c(1,1), eta = 0.1, maxIter = 50){
  w <- w_0
  variables.x <- w[1]
  variables.y <- w[2]
  valores <- f(w)


  while(length(valores) < maxIter){
    direccion <- -f.gradiente(w)
    w <- w + eta*direccion
    variables.x <- c(variables.x, w[1])
    variables.y <- c(variables.y, w[2])
    valores <- c(valores, f(w))
  }

  return(list("Pesos"=w, "Iter"=length(valores)-1,
              "Res"=data.frame(X = variables.x, Y = variables.y, Value = valores)))
}
```

Generamos ahora los datos desde los datos iniciales indicados y tomamos los mínimos

```r
# Definimos una función que, dado p, devuelve el valor mínimo del gradiente descendente
# con punto inicial = (p,p) y dónde se alcanza este:
minimoGradiente <- function(p){
  valores <- gradienteDescendente.log2(f, f.gradiente, c(p, p))$Res
  unlist(valores[which.min(valores$Value),])
}

# Definimos los puntos iniciales
puntosIniciales <- c(0.1, 1, -0.5, 1)

# Usamos la funcion minimoGradiente sobre cada uno de los puntos anteriores
tablaMinimos <- lapply(puntosIniciales, minimoGradiente)

# Convertimos la lista devuelta en un data frame y ponemos nombres a columnas y filas
tablaMinimos <- data.frame(matrix(unlist(tablaMinimos), nrow=4, byrow=T))
colnames(tablaMinimos) <- c("X", "Y", "Valor")
rownames(tablaMinimos) <- c("(0.1, 0.1)", "(1, 1)", "(-0.5, 0.5)", "(-1, -1)")

# Generamos la tabla solicitada
kable(tablaMinimos, digits=5)
```

	X	Y	Valor
(0.1, 0.1)	0.20646	-0.17155	-1.59489
(1, 1)	0.26879	-0.31392	-1.55870
(-0.5, 0.5)	0.28629	-0.33151	-1.39647
(-1, -1)	0.26879	-0.31392	-1.55870

Ejercicio 2

Ejercicio 2.1

Redefinimos, de nuevo, la función `gradienteDescendente` del primer apartado para realizar la técnica de *coordenada descendente*:

```r
coordenadaDescendente <- function(f, f.grad, w_0 = c(1,1), eta = 0.1, tol = 1e-14, maxIter = 50){
  w <- w_0
  iteraciones <- 0

  while(f(w) >= tol && iteraciones < maxIter){
    direccion.x <- c(-f.grad(w)[1], 0)
    w <- w + eta*direccion.x

    direccion.y <- c(0, -f.grad(w)[2])
    w <- w + eta*direccion.y

    iteraciones <- iteraciones + 1
  }

  return(list("Pesos"=w, "Iter"=iteraciones))
}
```

```
res.coordDesc <- coordenadaDescendente(E, E.gradiente, maxIter = 15)
```

De la ejecución anterior podemos ver que los valores de u y v obtenidos tras las 15 iteraciones son los siguientes:

$$u = 6.2970759$$
$$v = -2.852307$$

El valor alcanzado es además $f(6.2970759, -2.852307) = 0.1398138$.

Ejercicio 2.b

Como primera aproximación a la comparación, vamos a ejecutar ambos métodos sobre las funciones f y E con un máximo de 20 iteraciones cada uno, con el parámetro `tol` puesto a un valor virtualmente $-\infty$ para asegurarnos de que las iteraciones se realizan:

```
# Tomamos los resultados para f
comp.f.grad <- gradienteDescendente(f, f.gradiente, maxIter = 20, tol=-999999)
comp.f.coor <- coordenadaDescendente(f, f.gradiente, maxIter = 20, tol=-999999)

# Tomamos los resultados para E
comp.E.grad <- gradienteDescendente(E, E.gradiente, maxIter = 20, tol=-999999)
comp.E.coor <- coordenadaDescendente(E, E.gradiente, maxIter = 20, tol=-999999)

# Obtenemos el valor de f en el mínimo devuelto por ambos algoritmos
min.f.grad <- f(comp.f.grad$Pesos)
min.f.coor <- f(comp.f.coor$Pesos)

# Obtenemos el valor de E en el mínimo devuelto por ambos algoritmos
min.E.grad <- E(comp.E.grad$Pesos)
min.E.coor <- E(comp.E.coor$Pesos)
```

Podemos hacer una tabla rápida para comparar estos mínimos:

```
# Generamos un data frame con los datos obtenidos
tablaComp <- data.frame(c(min.f.coor, min.E.coor),
                        c(min.f.grad, min.E.grad))

colnames(tablaComp) <- c("Valor de f", "Valor de E")
rownames(tablaComp) <- c("Coordenada descendente", "Gradiente descendente")

# Generamos la tabla a partir del data frame
kable(tablaComp, digits=15)
```

	Valor de f	Valor de E
Coordenada descendente	2.5500174	1.814305
Gradiente descendente	0.1103108	0.000000

Ante estos datos parece que es claro que el método del gradiente descendente es mejor que el de coordenada descendente. Sin embargo, esta comparación no ha sido justa, ya que el método de coordenada está haciendo en realidad el doble de iteraciones que el de gradiente.

Así, para ser justos en la comparación, tenemos que ejecutar el método inicial de gradiente el doble de veces que el de la coordenada. Vamos a comparar entonces con 20 y 10 iteraciones, respectivamente, ambos métodos con las mismas funciones anteriores

```r
# Tomamos los resultados para f
comp.f.grad <- gradienteDescendente(f, f.gradiente, maxIter = 20, tol=-999999)
comp.f.coor <- coordenadaDescendente(f, f.gradiente, maxIter = 10, tol=-999999)

# Tomamos los resultados para E
comp.E.grad <- gradienteDescendente(E, E.gradiente, maxIter = 20, tol=-999999)
comp.E.coor <- coordenadaDescendente(E, E.gradiente, maxIter = 10, tol=-999999)

# Obtenemos el valor de f en el mínimo devuelto por ambos algoritmos
min.f.grad <- f(comp.f.grad$Pesos)
min.f.coor <- f(comp.f.coor$Pesos)

# Obtenemos el valor de E en el mínimo devuelto por ambos algoritmos
min.E.grad <- E(comp.E.grad$Pesos)
min.E.coor <- E(comp.E.coor$Pesos)
```

Rehacemos la tabla:

```r
# Generamos un data frame con los datos obtenidos
tablaComp <- data.frame(c(min.f.coor, min.E.coor),
                        c(min.f.grad, min.E.grad))

colnames(tablaComp) <- c("Valor de f", "Valor de E")
rownames(tablaComp) <- c("Coordenada descendente", "Gradiente descendente")

# Generamos la tabla a partir del data frame
kable(tablaComp, digits=15)
```

	Valor de f	Valor de E
Coordenada descendente	7.8390825	1.814305
Gradiente descendente	0.1926057	0.000000

Como vemos, en el caso de E no ha habido cambio, pero en el caso de f sí: el gradiente, en este caso, mucho más justo que el anterior, da unos valores muchísimo mejores, como era de esperar.

Ejercicio 3