Trabajo2

Gustavo Rivas Gervilla

1. MODELOS LINEALES

1. 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 $E(u,v)$.

Calculamos en primer lugar las derivadas parciales de la función respecto a de las variables u y v, que serán las dos componentes del gradiente de la función:

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

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

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.

```
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
    }
  }

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

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

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 que avanzan las iteraciones del algoritmo, algo que se nos pide en el siguiente apartado.

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

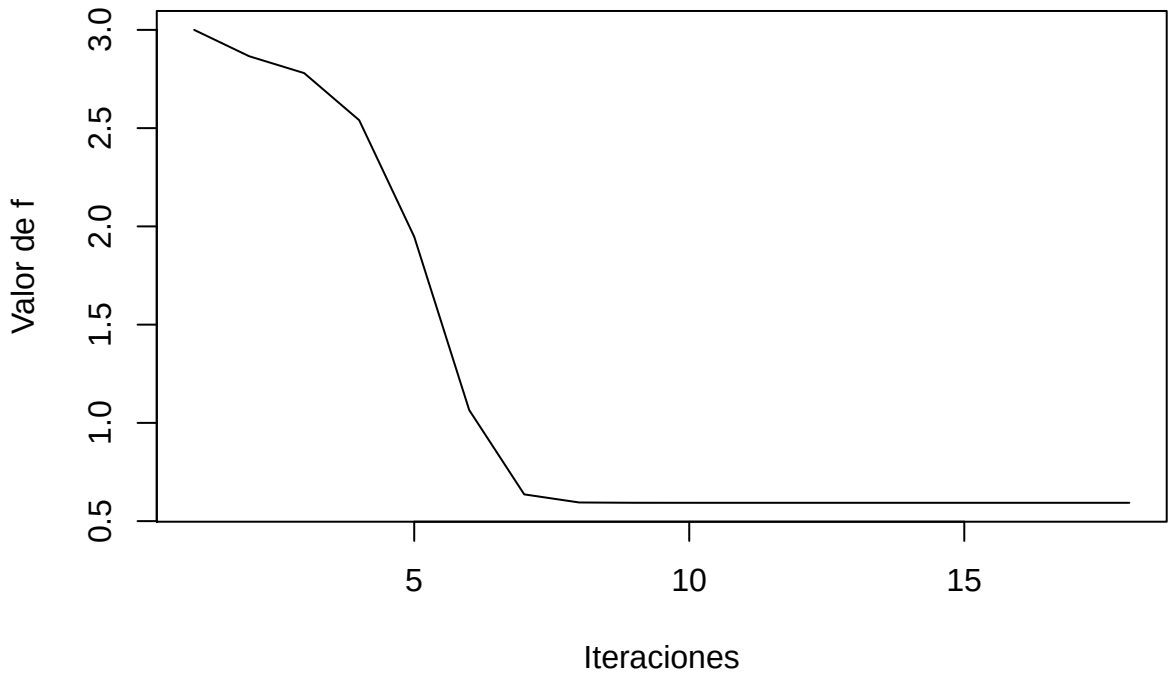
1) Usar gradiente descendente para minimizar esta función. Usar como valores iniciales $x_0 = 1$, $y_0 = 1$, la tasa de aprendizaje $\eta = 0.01$ y un máximo de 50 iteraciones. Generar un gráfico de cómo desciende el valor de la función con las iteraciones. Repetir el experimento pero usando $\eta = 0.1$, comentar las diferencias.

Definimos la función y su gradiente igual que antes:

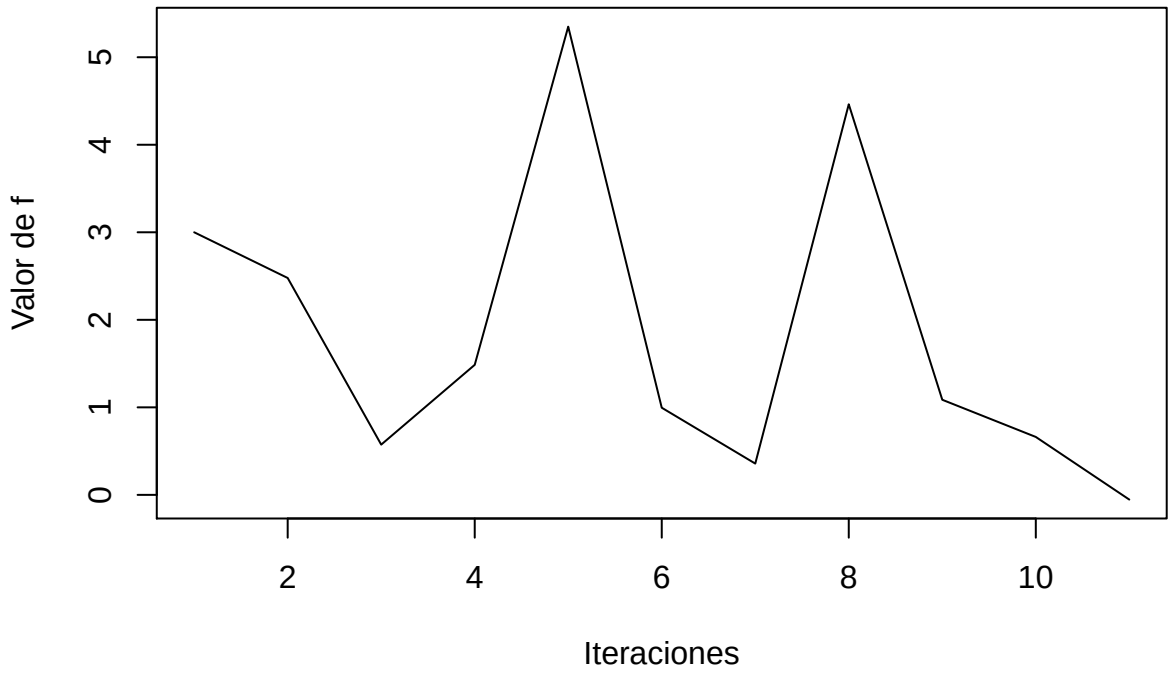
```
f <- function(X){
  x = X[1]
  y = X[2]

  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 11
## Valor de f alcanzado -0.05388005
## w obtenido 0.3881225 0.6516421
```



Lo que ocurre es que la tasa de aprendizaje nos define la longitud de los "saltos" que damos de un iteración del algoritmo a otra, es decir, el gradiente nos manda la dirección en la que nos movemos pero la tasa de aprendizaje es la que nos dice cuánto nos movemos en dicha dirección. Entonces cuando la tasa de aprendizaje es grande, $\eta = 0.1$, la longitud del "salto" es tan grande que nos pasamos del mínimo, es decir, pasamos a una zona de la función con un valor mayor del que ya estábamos, entonces aunque la convergencia con una tasa de aprendizaje menor es más lenta, es más segura al tener menor probabilidad de tener este efecto.

Lo que si observamos es que, en esta ocasión, llegamos a un valor por debajo del establecido en un menor número de iteraciones puesto que el dar "saltos" de mayor longitud nos ha llevado a una "zona" de la función de valores menores y hemos alcanzado un valor bajo más rápido, pero pensemos por ejemplo en la función x^2 que tiene sólo un mínimo en el cero y que quisiéramos alcanzar dicho mínimo, entonces quizás con una tasa de aprendizaje tan grande no llegaríamos nunca a alcanzarlo.

2) Obtener el valor mínimo y los valors de las variables que lo alcanzan cuando el punto de inicio se fija: (0.1,0.1), (1,1), (-0.5, -0.5), (-1,-1). Generar una tabla con los valores obtenidos. ¿Cuál sería su conclusión sobre la verdadera dificultad de encontrar el mínimo global de una función arbitraria?

```
## Num de iters empleado 4
## Valor de f alcanzado -0.0009552139
## w obtenido 0.005266095 -0.002392069

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

## Num de iters empleado 6
## Valor de f alcanzado -0.01244002
## w obtenido -0.5803234 -0.3839919

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

Pto. inicial	Mín. alcanzado	Iteraciones Empleadas	Valor de las variables
(0.1,0.1)	-0.0009552139	4	(0.005266095, -0.002392069)
(1,1)	0.5932694	18	(1.21807, 0.7128119)
(-0.5,-0.5)	-0.01244002	6	(-0.5803234, -0.3839919)
(-1,-1)	0.5932694	18	(-1.21807, -0.7128119)

Por como funciona nuestro algoritmo y los criterios de parada que hemos establecido y que hemos mencionado anteriormente tenemos que la principal dificultad es la de partir de un buen punto inicial ya que podemos quedar atrapados en un mínimo local o en una zona de poca variabilidad por eso en los puntos (1,1) y (-1,-1) pese a alcanzar un valor bastante elevado de la función el algoritmo para puesto que la variabilidad entre los distintos valores de la función en iteraciones consecutivas en muy pequeña.

2. Coordenada descendente En este ejercicio compararemos la eficiencia de la técnica de optimización de "coordenada descendente" usando la misma función del ejercicio 1.1a. En cada iteración tenemos dos pasos a lo largo de dos coordenadas. En el Paso-1 nos movemos a lo largo de la coordenada u para reducir el error (suponer que se verifica una aproximación de primer orden como en gradiente descendente), y en el Paso-2 es para reevaluar y movernos a lo largo de la coordenada v para reducir el error (hacer la misma hipótesis que en el Paso-1). Usar una tasa de aprendizaje $\eta = 0.1$.

Veamos la implementación que hemos hecho del algoritmo:

```
coordDesc <- function(f, gradf, eta, w0, tol, max_iter = 1000000) {
  w = w0
  n_iters = 1

  repeat{
    w_ant = w
    f_ant = f(w)
    n_iters = n_iters + 1

    w[1] = w[1] - eta*gradf(w)[1]
    w[2] = w[2] - eta*gradf(w)[2]

    if (f(w) < tol || n_iters == max_iter || d(w_ant, w) <= tol || abs(f(w) - f_ant) <= tol) break

  }

  list(w, f(w))
}
```

a) ¿Qué valor de la función $E(u, v)$ se obtiene después de 15 iteraciones completas (i.e. 30 pasos)?

El valor que obtenemos es: 0.1478295.

b) Establezca una comparación entre esta técnica y la técnica de gradiente descendente.

Lo que hemos hecho es llamar al gradiente descendente con los mismo parámetros que los empleados para la la llamada a Coordenada Descendente y los resultados obtenido son los siquientes:

```
gradDesc(E, gradE, 0.1, c(1,1), 10^{-14}, 15)
```

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

con lo cual como podemos ver la convergencia es mucho más rápida pues emplea menos de 15 iteraciones antes de parar además de alcanzar un valor mucho menor de f. Es lógico que la convergencia sea más rápida ya que estamos cambiando ambas componentes del vector de pesos a la vez.

3. Método de Newton Implementar el algoritmo de Newton y aplicarlo a la función $f(x, y)$ dada en el ejercicio 1b. Desarolle los mismo experimentos usando los mismo 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.

Lo que hacemos antes de definir el método es definir la matriz Hessiana de la función ya que este método hace uso de ella, así tenemos el siguiente código:

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

  matrix(c(2-8*pi*pi*sin(2*pi*y)*sin(2*pi*x), 8*pi*pi*cos(2*pi*y)*cos(2*pi*x),
           8*pi*pi*cos(2*pi*x)*cos(2*pi*y), 4-8*pi*pi*sin(2*pi*x)*sin(2*pi*y)),
         2,2)
}

Newton <- function(f, gradf, Hf, eta, w0, tol, max_iter = 1000000, dibujar = FALSE){
  w = w0
  n_iters = 1
  valores_f = NULL

  repeat {
    w_ant = w
    f_ant = f(w)

    valores_f = c(valores_f, f_ant)
    w = w-t(eta*ginv(Hf(w))%*%gradf(w))
    n_iters = n_iters + 1
```

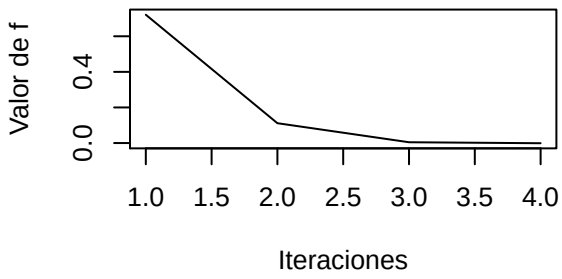
```
    if (f(w) < tol || n_iters == max_iter || d(w_ant, w) <= tol || abs(f(w) - f_ant) <= tol){
      valores_f = c(valores_f, f(w))
      break
    }
  }

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

}
```

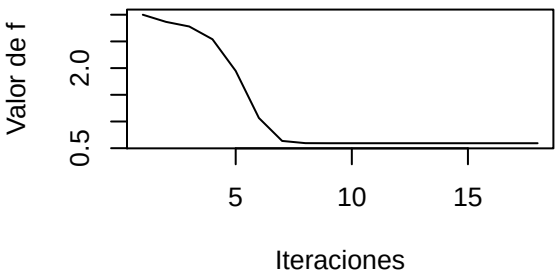
```
## Num de iters empleado 4
## Valor de f alcanzado -0.0009552139
## w obtenido 0.005266095 -0.002392069
```

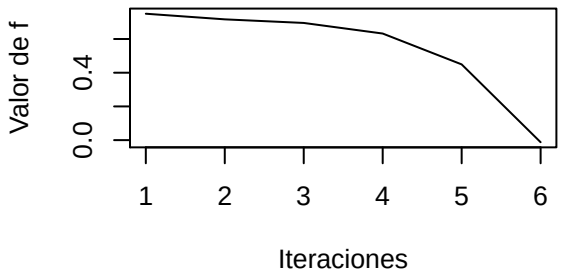
```
## Num de iters empleado 18
## Valor de f alcanzado 0.5932694
## w obtenido 1.21807 0.7128119
```

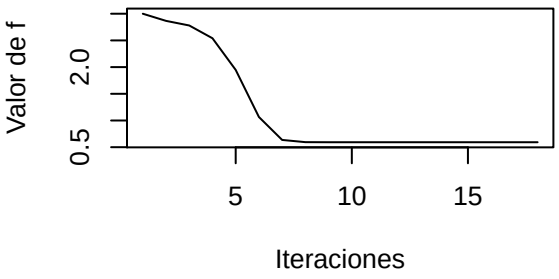
```
## Num de iters empleado 6
## Valor de f alcanzado -0.01244002
## w obtenido -0.5803234 -0.3839919
```

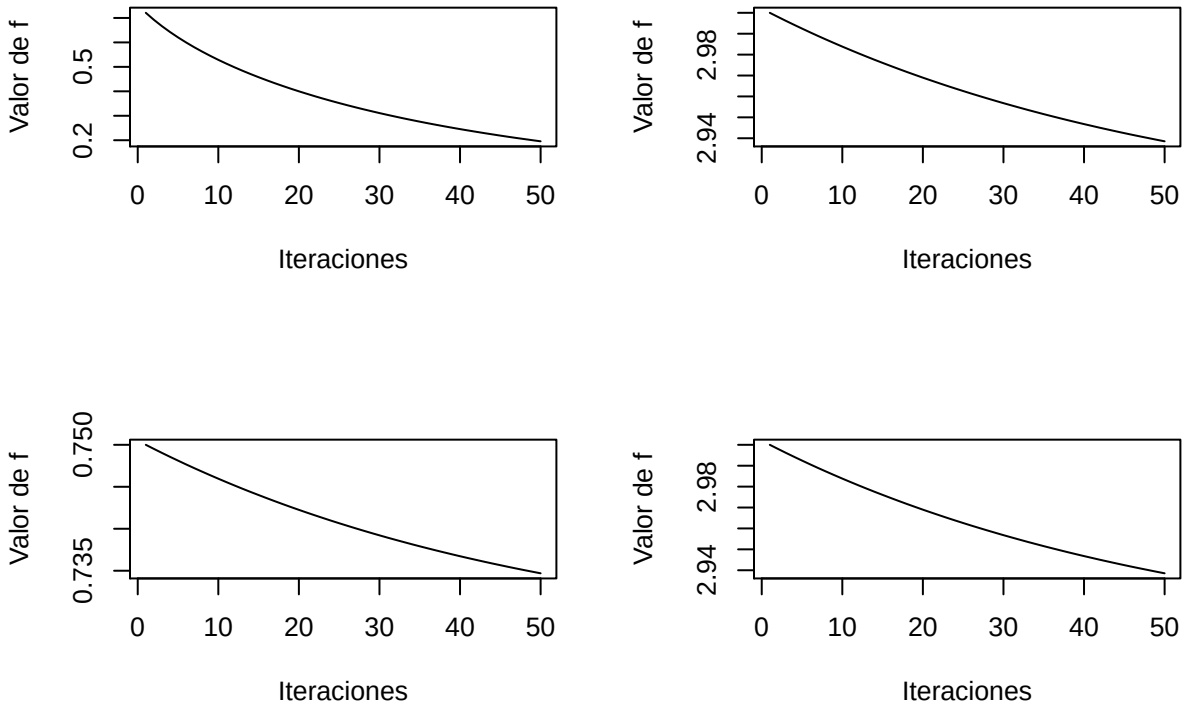
```
## Num de iters empleado 18
## Valor de f alcanzado 0.5932694
## w obtenido -1.21807 -0.7128119
```











REGULARIZACIÓN Y SELECCIÓN DE MODELOS

1. 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 coordenadas muestreando de forma independiente una $(N)(0, 1)$. 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 ϵ_n es un ruido que sigue también una $(N)(0, 1)$ 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 $0.05/N$.

a) Para $N \in \{d + 15, d + 25, ..., d + 115\}$ calcular los errores $e_1, ..., e_N$ de validación cruzada y E_{cv}.

b) Repetir el experimento 10^3 veces, anotando el promedi y la varianza de e_1, e_2 y E_{cv} en todos los experimentos.

c) ¿Cuál debería 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.

Lo que ocurre es que para cada e_i (que al fin y al cabo sólo se diferencian en qué elemento quitamos del conjunto de datos, elementos generados aleatoriamente) se da que $\mathbb{E}_{\mathcal{D}}[e_i] = \bar{E}_{out}(N-1)$, es decir, el promedio de e_i nos da la media del error fuera de la muestra de la función estimada cuando estamos aprendiendo con conjuntos de datos de tamaño $N - 1$, entonces como esto se da para todos los e_i también se dará para su media, es decir, el E_{cv} y entonces lo que ocurre es que estos promedios serán muy similares entre sí como podemos ver en los resultados de los experimentos.

d) ¿Qué es lo que contribuye a la varianza de los valores de e_1?

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

f) 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 encontrarse que N_{eff} está cercano a N.

g) 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.