Optimización

Introducción a la Estadística Computacional

Fernando Massa; Ramón Álvarez-Vaz

Setiembre 2022

FCEA FACULTAD DE CIENCIAS ECONÓMICAS Y DE ADMINISTRACIÓN
IESTA INSTITUTO DE ESTADÍSTICA
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Esquema de la Presentación

máx $f(x)$

La idea de éstas sesiones será introducir algunos algoritmos que permitan determinar numérica y eficientemente en qué valor de x que maximiza una cierta función $f(x)$.

máx $f(x)$

La idea de éstas sesiones será introducir algunos algoritmos que permitan determinar numérica y eficientemente en qué valor de x que maximiza una cierta función $f(x)$. Comenzaremos las exposiciones en un ámbito univariado tal como se hizo en las clases de *raíces de ecuaciones no lineales*, para luego generalizar los problemas a casos en dimensión $k \geq 2$.

máx $f(x)$

La idea de éstas sesiones será introducir algunos algoritmos que permitan determinar numérica y eficientemente en qué valor de x que maximiza una cierta función $f(x)$. Comenzaremos las exposiciones en un ámbito univariado tal como se hizo en las clases de *raíces de ecuaciones no lineales*, para luego generalizar los problemas a casos en dimensión $k \geq 2$.

Pese a que todos los métodos que veremos se emplearán para maximizar funciones, los mismos son fácilmente adaptables para minimizar funciones ya que:

$$\text{mín}\, f(x) = \text{máx}\,(-f(x))$$

máx $f(x)$

La idea de éstas sesiones será introducir algunos algoritmos que permitan determinar numérica y eficientemente en qué valor de x que maximiza una cierta función $f(x)$. Comenzaremos las exposiciones en un ámbito univariado tal como se hizo en las clases de *raíces de ecuaciones no lineales*, para luego generalizar los problemas a casos en dimensión $k \geq 2$.

Pese a que todos los métodos que veremos se emplearán para maximizar funciones, los mismos son fácilmente adaptables para minimizar funciones ya que:

$$\text{mín}\, f(x) = \text{máx}\,(-f(x))$$

Los algoritmos que se cubrirán son:

1. Sección áurea (golden section).
2. Ascenso más rápido (steepest ascent).
3. Método de Newton.
4. Método de quasi-Newton (BFGS).

máx $f(x)$

La idea de éstas sesiones será introducir algunos algoritmos que permitan determinar numérica y eficientemente en qué valor de x que maximiza una cierta función $f(x)$. Comenzaremos las exposiciones en un ámbito univariado tal como se hizo en las clases de *raíces de ecuaciones no lineales*, para luego generalizar los problemas a casos en dimensión $k \geq 2$.

Pese a que todos los métodos que veremos se emplearán para maximizar funciones, los mismos son fácilmente adaptables para minimizar funciones ya que:

$$\text{mín}\, f(x) = \text{máx}\,(-f(x))$$

Los algoritmos que se cubrirán son:

1. Sección áurea (golden section).
2. Ascenso más rápido (steepest ascent).
3. Método de Newton.
4. Método de quasi-Newton (BFGS).

Todos los métodos que desarrollaremos se caracterizan por ser de caracter iterativo por lo cual, orientaremos la programación al uso de estructuras de control y funciones en R.

CASO GENERAL

Diremos que $f : \mathbb{R} \to \mathbb{R}$ es una función continua y que x^* es su máximo global si se cumple que $f(x^*) > f(x) \quad \forall x \neq x^*$. El máximo será local si la desigualdad solo se cumple en un entorno de x^*.

CASO GENERAL

Diremos que $f : \mathbb{R} \to \mathbb{R}$ es una función continua y que x^* es su máximo global si se cumple que $f(x^*) > f(x) \quad \forall x \neq x^*$. El máximo será local si la desigualdad solo se cumple en un entorno de x^*.

Según lo aprendido en los cursos de cálculo, las condiciones suficientes para que x^* sea un máximo local son:

CASO GENERAL

Diremos que $f : \mathbb{R} \to \mathbb{R}$ es una función continua y que x^* es su máximo global si se cumple que $f(x^*) > f(x) \quad \forall x \neq x^*$. El máximo será local si la desigualdad solo se cumple en un entorno de x^*.

Según lo aprendido en los cursos de cálculo, las condiciones suficientes para que x^* sea un máximo local son:

- $f'(x^*) = 0$

CASO GENERAL

Diremos que $f : \mathbb{R} \to \mathbb{R}$ es una función continua y que x^* es su máximo global si se cumple que $f(x^*) > f(x) \quad \forall x \neq x^*$. El máximo será local si la desigualdad solo se cumple en un entorno de x^*.

Según lo aprendido en los cursos de cálculo, las condiciones suficientes para que x^* sea un máximo local son:

- $f^{'}(x^*) = 0$
- $f^{''}(x^*) < 0$

CASO GENERAL

Diremos que $f : \mathbb{R} \to \mathbb{R}$ es una función continua y que x^* es su máximo global si se cumple que $f(x^*) > f(x) \quad \forall x \neq x^*$. El máximo será local si la desigualdad solo se cumple en un entorno de x^*.

Según lo aprendido en los cursos de cálculo, las condiciones suficientes para que x^* sea un máximo local son:

- $f^{'}(x^*) = 0$
- $f^{''}(x^*) < 0$

Si adicionalmente se cumple que $f(x)$ es una función cóncava, entonces x^* es un máximo global

Ejemplo 1

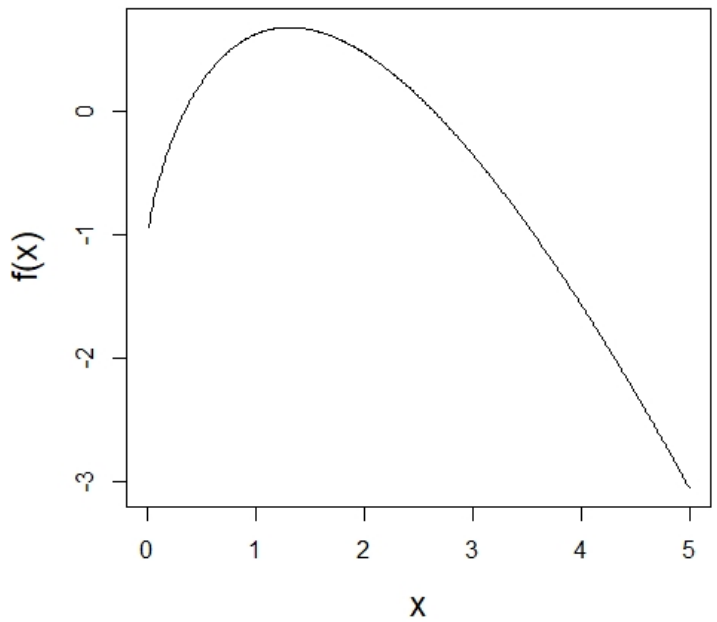
Caso general

Supongamos que nos interesa encontrar la raíz de la ecuación $f(x) = x - x\log(x) - e^{-x}$. La figura siguiente muestra que el máximo pertenece al intervalo $(0, 5)$.

Ejemplo 1

Caso general

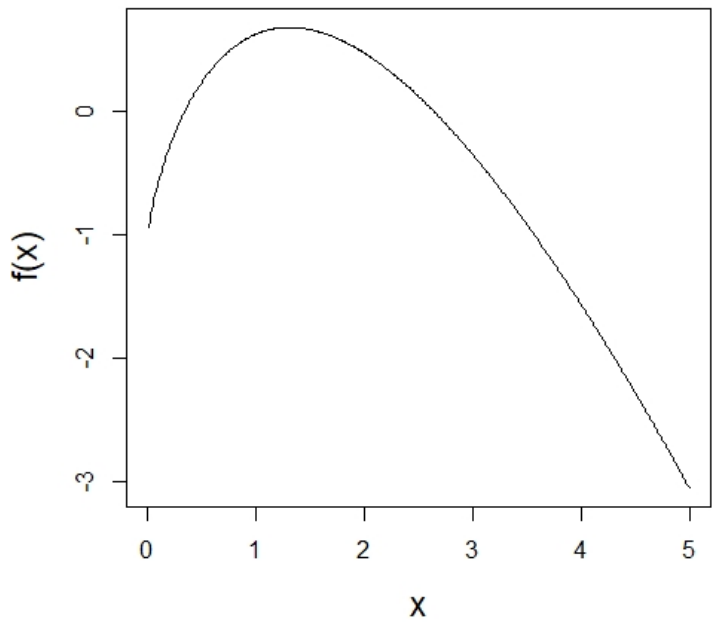
Supongamos que nos interesa encontrar la raíz de la ecuación $f(x) = x - x\log(x) - e^{-x}$. La figura siguiente muestra que el máximo pertenece al intervalo $(0,5)$.



Ejemplo 1

Caso general

Supongamos que nos interesa encontrar la raíz de la ecuación $f(x) = x - x\log(x) - e^{-x}$.
La figura siguiente muestra que el máximo pertenece al intervalo $(0, 5)$.



¿El máximo será local o global?

Generalidade para GS

La manera en la que trabaja este algoritmo es la misma que utiliza el algoritmo de *bisección* para encontrar raíces de funciones, refinando el intervalo donde se encuentra el máximo hasta que la amplitud de ese intervalo sea lo suficientemente pequeña.

Generalidade para GS

La manera en la que trabaja este algoritmo es la misma que utiliza el algoritmo de *bisección* para encontrar raíces de funciones, refinando el intervalo donde se encuentra el máximo hasta que la amplitud de ese intervalo sea lo suficientemente pequeña.
De los algoritmos que veremos es el único que no hace ningún supuesto respecto de $f^{'}(x)$.

Generalidade para GS

La manera en la que trabaja este algoritmo es la misma que utiliza el algoritmo de *bisección* para encontrar raíces de funciones, refinando el intervalo donde se encuentra el máximo hasta que la amplitud de ese intervalo sea lo suficientemente pequeña.
De los algoritmos que veremos es el único que no hace ningún supuesto respecto de $f^{'}(x)$.

Los siguientes argumentos son análogos a lo expuestos en el algoritmo de *bisección* y se invita al lector a compararlos.

Generalidade para GS

La manera en la que trabaja este algoritmo es la misma que utiliza el algoritmo de *bisección* para encontrar raíces de funciones, refinando el intervalo donde se encuentra el máximo hasta que la amplitud de ese intervalo sea lo suficientemente pequeña.
De los algoritmos que veremos es el único que no hace ningún supuesto respecto de $f^{'}(x)$.

Los siguientes argumentos son análogos a lo expuestos en el algoritmo de *bisección* y se invita al lector a compararlos.

Particularidades para GS

- $f(x)$ es continua en el intervalo (x_i,x_d) y sea x_m un punto perteneciente al interior de dicho intervalo.
- $f(x_i) < f(x_m)$ y $f(x_d) < f(x_m)$

Entonces existe un máximo x^* dentro del intervalo (x_i, x_d)

Generalidade para GS

La manera en la que trabaja este algoritmo es la misma que utiliza el algoritmo de *bisección* para encontrar raíces de funciones, refinando el intervalo donde se encuentra el máximo hasta que la amplitud de ese intervalo sea lo suficientemente pequeña.
De los algoritmos que veremos es el único que no hace ningún supuesto respecto de $f^{'}(x)$.

Los siguientes argumentos son análogos a lo expuestos en el algoritmo de *bisección* y se invita al lector a compararlos.

Particularidades para GS

- $f(x)$ es continua en el intervalo (x_i,x_d) y sea x_m un punto perteneciente al interior de dicho intervalo.
- $f(x_i) < f(x_m)$ y $f(x_d) < f(x_m)$

Entonces existe un máximo x^* dentro del intervalo (x_i, x_d)

El modo en el que trabaja el algoritmo es ir "refinando" el intervalo sucesivamente de modo de ir actualizando los límites x_i y x_d.

Implementando GS en R

Veamos como sería un posible pseudocódigo en R.

IMPLEMENTANDO GS EN R

Veamos como sería un posible pseudocódigo en R.

```
# se inicializan algunas variables
# tolerancia, iterador, x_i, x_d, x_m amplitud

while(condicion){
   si x_d - x_m > x_m - x_i
   {
      se elije x_k en el intervalo (x_m, x_d)
      si f(x_k) > f(x_m)
      {
         x_i<-x_m
         x_m<-x_k
      } si no x_d <-x_k
   }
   si no
   {
      se elije x_k en el intervalo (x_i, x_m)
      si f(x_k) > f(x_m)
      {
         x_d<-x_m
         x_m<-x_k
      } si no x_i <-x_k
   }
   # por ultimo se actualiza la condicion y se incrementa el iterador
}
```

IMPLEMENTANDO GS EN R

Veamos como sería un posible pseudocódigo en R.

```
# se inicializan algunas variables
# tolerancia, iterador, x_i, x_d, x_m amplitud

while(condicion){
   si x_d - x_m > x_m - x_i
   {
      se elije x_k en el intervalo (x_m, x_d)
      si f(x_k) > f(x_m)
      {
         x_i<-x_m
         x_m<-x_k
      } si no x_d <-x_k
   }
   si no
   {
      se elije x_k en el intervalo (x_i, x_m)
      si f(x_k) > f(x_m)
      {
         x_d<-x_m
         x_m<-x_k
      } si no x_i <-x_k
   }
   # por ultimo se actualiza la condicion y se incrementa el iterador
}
```

¿Y cómo se elige x_k?

Elección de x_k (a)

En la práctica x_k podría ser elegido de cualquier manera, sin embargo, puede demostrarse que con el siguiente método la velocidad en la que se reduce la amplitud del intervalo (x_i, x_d) es la óptima.

Elección de x_k (a)

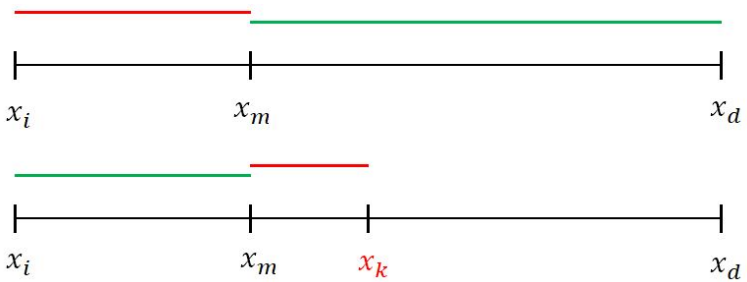
En la práctica x_k podría ser elegido de cualquier manera, sin embargo, puede demostrarse que con el siguiente método la velocidad en la que se reduce la amplitud del intervalo (x_i, x_d) es la óptima.

Sean x_i, x_m y x_d los puntos definidos anteriormente y consideremos el caso en el que x_m está más cerca de x_i que de x_d.

Elección de x_k (a)

En la práctica x_k podría ser elegido de cualquier manera, sin embargo, puede demostrarse que con el siguiente método la velocidad en la que se reduce la amplitud del intervalo (x_i, x_d) es la óptima.

Sean x_i, x_m y x_d los puntos definidos anteriormente y consideremos el caso en el que x_m está más cerca de x_i que de x_d.



La forma en la que se seleccionaría x_k es de tal forma que:

$$\frac{x_m - x_i}{x_k - x_m} = \frac{x_d - x_m}{x_m - x_i} = \rho$$

Elección de x_k (2)

A partir de la relación anterior puede determinarse x_k como $x_m + \frac{x_d - x_m}{1+\rho}$

Elección de x_k (2)

A partir de la relación anterior puede determinarse x_k como $x_m + \frac{x_d - x_m}{1+\rho}$

Siendo $\rho = \frac{1+\sqrt{5}}{2}$

Elección de x_k (2)

A partir de la relación anterior puede determinarse x_k como $x_m + \frac{x_d - x_m}{1+\rho}$

Siendo $\rho = \frac{1+\sqrt{5}}{2}$

A partir de este esquema se puede demostrar (y esto puede convertirse en un ejercicio) que la amplitud del intervalo (x_i, x_d) entre la iteración n y la $n-1$ decrece en un factor $\frac{\rho}{1+\rho} \approx 0{,}618$

Elección de x_k (2)

A partir de la relación anterior puede determinarse x_k como $x_m + \frac{x_d - x_m}{1+\rho}$

Siendo $\rho = \frac{1+\sqrt{5}}{2}$

A partir de este esquema se puede demostrar (y esto puede convertirse en un ejercicio) que la amplitud del intervalo (x_i, x_d) entre la iteración n y la $n-1$ decrece en un factor $\frac{\rho}{1+\rho} \approx 0{,}618$

Mediante un argumento análogo puede demostrarse lo mismo en el caso de que x_m esté más cerca de x_d que de x_i. En cuyo caso, la manera de seleccionar x_k sería $x_m - \frac{x_m - x_i}{1+\rho}$

Ejemplo 2

$f(x) = x - x\log(x) - e^{-x}$

Ahora que comprendemos completamente como funciona este algoritmo, busquemos el máximo de la ecuación anterior.

El código de R podría ser

Ejemplo 2

$f(x) = x - x\log(x) - e^{-x}$

Ahora que comprendemos completamente como funciona este algoritmo, busquemos el máximo de la ecuación anterior.

El código de R podría ser

- ¿Qué significa que x_m se inicialice como *NULL*?
- Determine el máximo de $f(x)$ empleando $x_i = 1$ y $x_d = 4$.
- ¿Cuántas iteraciones necesitó?
- Modifique levemente el algoritmo de modo tal que x_k sea elejido como el punto medio entre x_m y x_d (o x_i y x_m). LLame a la función modificada *gs2*.
- Con este nuevo algoritmo, vuelva a determinar el máximo y compare los resultados.
- Vuelva a modificar el script de modo tal que x_k sea elejido de manera aleatoria entre x_m y x_d (o x_i y x_m). LLame a la función modificada *gsr*.
- Con este nuevo algoritmo, vuelva a determinar el máximo y compare los resultados.

Generalidades para SA

A partir de este punto comenzaremos a pensar en resolver problemas de maximización de funciones cuyo dominio sea $\mathbb{R}^k$. Para emplear los algoritmos que veremos de aquí en adelante haremos el supuesto de que $f(x)$ es una función continua con derivadas parciales de primer y segundo orden están bien definidas alrededor de su máximo x^*.

Generalidades para SA

A partir de este punto comenzaremos a pensar en resolver problemas de maximización de funciones cuyo dominio sea $\mathbb{R}^k$. Para emplear los algoritmos que veremos de aquí en adelante haremos el supuesto de que $f(x)$ es una función continua con derivadas parciales de primer y segundo orden están bien definidas alrededor de su máximo x^*.
Los algoritmos que veremos en las próximas sesiones pertenecen a la familia de *búsqueda lineal* (*line search*). Estos métodos proponen una recursión donde para crear el iterando x_{n+1} a partir de x_n se llevan a cabo dos pasos:

Generalidades para SA

A partir de este punto comenzaremos a pensar en resolver problemas de maximización de funciones cuyo dominio sea $\mathbb{R}^k$. Para emplear los algoritmos que veremos de aquí en adelante haremos el supuesto de que $f(x)$ es una función continua con derivadas parciales de primer y segundo orden están bien definidas alrededor de su máximo x^*.
Los algoritmos que veremos en las próximas sesiones pertenecen a la familia de *búsqueda lineal* (*line search*). Estos métodos proponen una recursión donde para crear el iterando x_{n+1} a partir de x_n se llevan a cabo dos pasos:

- Se designa una dirección de búsqueda p_n.
- Sobre la dirección p_n se "da" una paso de longitud α_n.

Generalidades para SA

A partir de este punto comenzaremos a pensar en resolver problemas de maximización de funciones cuyo dominio sea $\mathbb{R}^k$. Para emplear los algoritmos que veremos de aquí en adelante haremos el supuesto de que $f(x)$ es una función continua con derivadas parciales de primer y segundo orden están bien definidas alrededor de su máximo x^*.
Los algoritmos que veremos en las próximas sesiones pertenecen a la familia de *búsqueda lineal* (*line search*). Estos métodos proponen una recursión donde para crear el iterando x_{n+1} a partir de x_n se llevan a cabo dos pasos:

- Se designa una dirección de búsqueda p_n.
- Sobre la dirección p_n se "da" una paso de longitud α_n.

$$x_{n+1} = x_n + \alpha_n p_n \tag{1}$$

Generalidades para SA

A partir de este punto comenzaremos a pensar en resolver problemas de maximización de funciones cuyo dominio sea $\mathbb{R}^k$. Para emplear los algoritmos que veremos de aquí en adelante haremos el supuesto de que $f(x)$ es una función continua con derivadas parciales de primer y segundo orden están bien definidas alrededor de su máximo x^*.
Los algoritmos que veremos en las próximas sesiones pertenecen a la familia de *búsqueda lineal* (*line search*). Estos métodos proponen una recursión donde para crear el iterando x_{n+1} a partir de x_n se llevan a cabo dos pasos:

- Se designa una dirección de búsqueda p_n.
- Sobre la dirección p_n se "da" una paso de longitud α_n.

$$x_{n+1} = x_n + \alpha_n p_n \tag{1}$$

Particularidades para SA

La efectividad de estos algoritmos recae sobre efectuar decisiones adecuadas en la elección de la dirección de búsqueda (p_n) y la longitud del paso (α_n).

p_n

A la hora de seleccionar el vector p_n que nos permita "movernos" de x_n a x_{n+1} podemos plantear un desarrollo de Taylor de primer orden de $f(x_{n+1})$ alrededor de x_n.

p_n

A la hora de seleccionar el vector p_n que nos permita "movernos" de x_n a x_{n+1} podemos plantear un desarrollo de Taylor de primer orden de $f(x_{n+1})$ alrededor de x_n.

$$\begin{aligned} f(x_{n+1}) &= f(x_n) + \nabla f(x_n)^T (x_{n+1} - x_n) + o(||x_{n+1} - x_n||) \\ &\approx f(x_n) + \alpha_n \nabla f(x_n)^T p_n \end{aligned}$$

p_n

A la hora de seleccionar el vector p_n que nos permita "movernos" de x_n a x_{n+1} podemos plantear un desarrollo de Taylor de primer orden de $f(x_{n+1})$ alrededor de x_n.

$$\begin{aligned} f(x_{n+1}) &= f(x_n) + \nabla f(x_n)^T (x_{n+1} - x_n) + o(||x_{n+1} - x_n||) \\ &\approx f(x_n) + \alpha_n \nabla f(x_n)^T p_n \end{aligned}$$

Si pretendemos que el algoritmo busque un máximo, necesitamos que $f(x_{n+1}) > f(x_n)$. Por ende, resulta natural pensar que $p_n = \nabla f(x_n)$ ya que de esta manera:

p_n

A la hora de seleccionar el vector p_n que nos permita "movernos" de x_n a x_{n+1} podemos plantear un desarrollo de Taylor de primer orden de $f(x_{n+1})$ alrededor de x_n.

$$\begin{aligned} f(x_{n+1}) &= f(x_n) + \nabla f(x_n)^T (x_{n+1} - x_n) + o(||x_{n+1} - x_n||) \\ &\approx f(x_n) + \alpha_n \nabla f(x_n)^T p_n \end{aligned}$$

Si pretendemos que el algoritmo busque un máximo, necesitamos que $f(x_{n+1}) > f(x_n)$. Por ende, resulta natural pensar que $p_n = \nabla f(x_n)$ ya que de esta manera:

$$f(x_{n+1}) \approx f(x_n) + \alpha_n ||\nabla f(x_n)|| \tag{2}$$

p_n

A la hora de seleccionar el vector p_n que nos permita "movernos" de x_n a x_{n+1} podemos plantear un desarrollo de Taylor de primer orden de $f(x_{n+1})$ alrededor de x_n.

$$\begin{aligned} f(x_{n+1}) &= f(x_n) + \nabla f(x_n)^T (x_{n+1} - x_n) + o(||x_{n+1} - x_n||) \\ &\approx f(x_n) + \alpha_n \nabla f(x_n)^T p_n \end{aligned}$$

Si pretendemos que el algoritmo busque un máximo, necesitamos que $f(x_{n+1}) > f(x_n)$. Por ende, resulta natural pensar que $p_n = \nabla f(x_n)$ ya que de esta manera:

$$f(x_{n+1}) \approx f(x_n) + \alpha_n ||\nabla f(x_n)|| \tag{2}$$

Debido a que $||\nabla f(x_n)|| > 0$ el algoritmo "asciende" en cada iteración y deja de ascender cuando $||\nabla f(x_n)|| = 0$

Longitud del paso (1)

α_n

Ya que contamos con una dirección de búsqueda, el algoritmo consiste en la siguiente recursión.

Longitud del paso (1)

α_n

Ya que contamos con una dirección de búsqueda, el algoritmo consiste en la siguiente recursión.

$$x_{n+1} = x_n + \alpha_n \nabla f(x_n) \tag{3}$$

Longitud del paso (1)

α_n

Ya que contamos con una dirección de búsqueda, el algoritmo consiste en la siguiente recursión.

$$x_{n+1} = x_n + \alpha_n \nabla f(x_n) \tag{3}$$

El último paso consiste en seleccionar la longitud del paso α_n.

Longitud del paso (1)

α_n

Ya que contamos con una dirección de búsqueda, el algoritmo consiste en la siguiente recursión.

$$x_{n+1} = x_n + \alpha_n \nabla f(x_n) \tag{3}$$

El último paso consiste en seleccionar la longitud del paso α_n.

La forma en la que el método selecciona el valor de α en cada iteración es la siguiente. Dada una dirección de búsqueda, se elige el valor de α que maximiza a $f(x)$ en dicha dirección, es decir:

Longitud del paso (1)

α_n

Ya que contamos con una dirección de búsqueda, el algoritmo consiste en la siguiente recursión.

$$x_{n+1} = x_n + \alpha_n \nabla f(x_n) \tag{3}$$

El último paso consiste en seleccionar la longitud del paso α_n.

La forma en la que el método selecciona el valor de α en cada iteración es la siguiente. Dada una dirección de búsqueda, se elige el valor de α que maximiza a $f(x)$ en dicha dirección, es decir:

$$\alpha_n = \max_{\alpha} g(\alpha) = \max_{\alpha} f(x_n + \alpha \nabla f(x_n)) \tag{4}$$

α_n

Ya que contamos con una dirección de búsqueda, el algoritmo consiste en la siguiente recursión.

$$x_{n+1} = x_n + \alpha_n \nabla f(x_n) \tag{3}$$

El último paso consiste en seleccionar la longitud del paso α_n.

La forma en la que el método selecciona el valor de α en cada iteración es la siguiente. Dada una dirección de búsqueda, se elige el valor de α que maximiza a $f(x)$ en dicha dirección, es decir:

$$\alpha_n = \max_{\alpha} g(\alpha) = \max_{\alpha} f(x_n + \alpha \nabla f(x_n)) \tag{4}$$

Si el valor de α seleccionado es lo suficientemente pequeño (cero), consideraremos que ya no se puede progresar en dicha dirección, entonces hemos alcanzado el máximo. En caso de que $\alpha > 0$ entonces, por lo visto anteriormente se tiene que $f(x_{n+1}) > f(x_n)$.

Longitud del paso (2)

α_n

Si bien estamos en un contexto de optimización multivariado, la elección de la longitud del paso puede verse como un problema de maximización univariado.

Longitud del paso (2)

α_n

Si bien estamos en un contexto de optimización multivariado, la elección de la longitud del paso puede verse como un problema de maximización univariado.
En este sentido, el algoritmo se enfrenta a una decisión:

- Se desea obtener un valor de α que permita maximizar $g(\alpha)$
- Pero sin pasar mucho tiempo obteniendo dicho valor.

Longitud del paso (2)

α_n

Si bien estamos en un contexto de optimización multivariado, la elección de la longitud del paso puede verse como un problema de maximización univariado.
En este sentido, el algoritmo se enfrenta a una decisión:

- Se desea obtener un valor de α que permita maximizar $g(\alpha)$
- Pero sin pasar mucho tiempo obteniendo dicho valor.

Una alternativa sería emplear el método de sección áurea (GS) para determinar el valor de α. Por este motivo, los algoritmos suelen llevar a este paso de manera *inexacta*. Para eso es necesario definir las condiciones de Wolfe.

Longitud del paso (2)

α_n

Si bien estamos en un contexto de optimización multivariado, la elección de la longitud del paso puede verse como un problema de maximización univariado.
En este sentido, el algoritmo se enfrenta a una decisión:

- Se desea obtener un valor de α que permita maximizar $g(\alpha)$
- Pero sin pasar mucho tiempo obteniendo dicho valor.

Una alternativa sería emplear el método de sección áurea (GS) para determinar el valor de α. Por este motivo, los algoritmos suelen llevar a este paso de manera *inexacta*. Para eso es necesario definir las condiciones de Wolfe.

Condiciones de Wolfe

- Condición de ascenso suficiente (condición de Armijo)
- Condición de curvatura

CONDICIÓN DEL ASCENSO SUFICIENTE

Postula que:

$$f(x_n + \alpha p_n) \geq f(x_n) + c_1 \alpha \nabla f(x_n)^T p_n \tag{5}$$

CONDICIÓN DEL ASCENSO SUFICIENTE

Postula que:

$$f(x_n + \alpha p_n) \geq f(x_n) + c_1 \alpha \nabla f(x_n)^T p_n \tag{5}$$

Siendo c_1 una constante perteneciente al intervalo $(0,1)$, típicamente 0,0001. Esta condición impone que los valores aceptables de α son aquellos donde $f(x_{n+1})$ está por encima de la recta con pendiente $c_1 \nabla f(x_n)^T p_n$

CONDICIÓN DEL ASCENSO SUFICIENTE

Postula que:

$$f(x_n + \alpha p_n) \geq f(x_n) + c_1 \alpha \nabla f(x_n)^T p_n \tag{5}$$

Siendo c_1 una constante perteneciente al intervalo $(0,1)$, típicamente 0,0001. Esta condición impone que los valores aceptables de α son aquellos donde $f(x_{n+1})$ está por encima de la recta con pendiente $c_1 \nabla f(x_n)^T p_n$

CONDICIÓN DE LA CURVATURA

Se encarga de descartar valores demasiado pequeños de α.

CONDICIÓN DEL ASCENSO SUFICIENTE

Postula que:

$$f(x_n + \alpha p_n) \geq f(x_n) + c_1 \alpha \nabla f(x_n)^T p_n \tag{5}$$

Siendo c_1 una constante perteneciente al intervalo $(0,1)$, típicamente 0,0001. Esta condición impone que los valores aceptables de α son aquellos donde $f(x_{n+1})$ está por encima de la recta con pendiente $c_1 \nabla f(x_n)^T p_n$

CONDICIÓN DE LA CURVATURA

Se encarga de descartar valores demasiado pequeños de α.

$$\nabla f(x_n + \alpha_n p_n)^T p_n \leq c_2 \nabla f(x_n)^T p_n \tag{6}$$

CONDICIÓN DEL ASCENSO SUFICIENTE

Postula que:

$$f(x_n + \alpha p_n) \geq f(x_n) + c_1 \alpha \nabla f(x_n)^T p_n \tag{5}$$

Siendo c_1 una constante perteneciente al intervalo $(0,1)$, típicamente 0,0001. Esta condición impone que los valores aceptables de α son aquellos donde $f(x_{n+1})$ está por encima de la recta con pendiente $c_1 \nabla f(x_n)^T p_n$

CONDICIÓN DE LA CURVATURA

Se encarga de descartar valores demasiado pequeños de α.

$$\nabla f(x_n + \alpha_n p_n)^T p_n \leq c_2 \nabla f(x_n)^T p_n \tag{6}$$

Siendo $c_2 > c_1$, típicamente $c_2 = 0{,}9$. Esta condición impone que el gradiente de $f(x_n + \alpha_n p_n)$ no sea demasiado "empinado" con respecto al gradiente de $f(x_n)$

Backtracking

Una manera de encontrar un posible valor de α es mediante el algoritmo de *backtracking*.

Backtracking

Una manera de encontrar un posible valor de α es mediante el algoritmo de *backtracking*. El algoritmo comienza con un valor inicial α_{max} y lo *contrae* iterativamente hasta que se cumplen las condiciones de Wolfe.

BACKTRACKING

Una manera de encontrar un posible valor de α es mediante el algoritmo de *backtracking*. El algoritmo comienza con un valor inicial α_{max} y lo *contrae* iterativamente hasta que se cumplen las condiciones de Wolfe.

Paso 1 (*backtracking*)

BACKTRACKING

Una manera de encontrar un posible valor de α es mediante el algoritmo de *backtracking*. El algoritmo comienza con un valor inicial α_{max} y lo *contrae* iterativamente hasta que se cumplen las condiciones de Wolfe.

Paso 1 (*backtracking*)

```
# se inicializan algunas variables
while(f(xn+alfa*pn) <= f(xn)+alfa*c1*df(xn)*pn){
   alfa <- tau*alfa
   # contraccion de alfa
}
```

BACKTRACKING

Una manera de encontrar un posible valor de α es mediante el algoritmo de *backtracking*. El algoritmo comienza con un valor inicial α_{max} y lo *contrae* iterativamente hasta que se cumplen las condiciones de Wolfe.

Paso 1 (*backtracking*)

```
# se inicializan algunas variables
while(f(xn+alfa*pn) <= f(xn)+alfa*c1*df(xn)*pn){
   alfa <- tau*alfa
   # contraccion de alfa
}
```

Paso 2 (*forwardtracking*)

BACKTRACKING

Una manera de encontrar un posible valor de α es mediante el algoritmo de *backtracking*. El algoritmo comienza con un valor inicial α_{max} y lo *contrae* iterativamente hasta que se cumplen las condiciones de Wolfe.

Paso 1 (*backtracking*)

```
# se inicializan algunas variables
while(f(xn+alfa*pn) <= f(xn)+alfa*c1*df(xn)*pn){
   alfa <- tau*alfa
   # contraccion de alfa
}
```

Paso 2 (*forwardtracking*)

```
# se inicializan algunas variables
while(df(xn+alfa*pn)*pn > c2*df(xn)*pn){
   alfa_anterior <- alfa/tau
   dif   <- alfa_anterior - alfa
   alfa <- alfa + dif*(1-tau2^2)
   # expansion de alfa
}
```

Siendo τ y τ_2 los parámetros de contracción y expansión especificiados por el usuario (que típicamente adoptan los valores 0,5 y 0,8).

UN EJEMPLO NUEVO

Hoy si cambiamos de ejemplo. Consideremos la siguiente función:

$$f(x_1, x_2) = \sin(x_1^2/2 - x_2^2/4)\cos(2x_1 - e^{x_2})$$

UN EJEMPLO NUEVO

Hoy si cambiamos de ejemplo. Consideremos la siguiente función:

$$f(x_1, x_2) = \sin(x_1^2/2 - x_2^2/4)\cos(2x_1 - e^{x_2})$$

Considerando que la mejor dirección de ascenso es $p_n = \nabla f(x_1, x_2)$ necesitaremos el vector de derivadas parciales de $f(x_1, x_2)$.

UN EJEMPLO NUEVO

Hoy si cambiamos de ejemplo. Consideremos la siguiente función:

$$f(x_1, x_2) = \sin(x_1^2/2 - x_2^2/4)\cos(2x_1 - e^{x_2})$$

Considerando que la mejor dirección de ascenso es $p_n = \nabla f(x_1, x_2)$ necesitaremos el vector de derivadas parciales de $f(x_1, x_2)$.

$$\begin{aligned} \frac{d}{dx_1} f(x_1, x_2) &= x_1 \cos(x_1^2/2 - x_2^2/4)\cos(2x_1 - e^{x_2}) - 2\sin(x_1^2/2 - x_2^2/4)\sin(2x_1 - e^{x_2}) \\ \frac{d}{dx_2} f(x_1, x_2) &= -\cos(x_1^2/2 - x_2^2/4)\cos(2x_1 - e^{x_2}) + e^{x_2}\sin(x_1^2/2 - x_2^2/4)\sin(2x_1 - e^{x_2}) \end{aligned}$$

UN EJEMPLO NUEVO

Hoy si cambiamos de ejemplo. Consideremos la siguiente función:

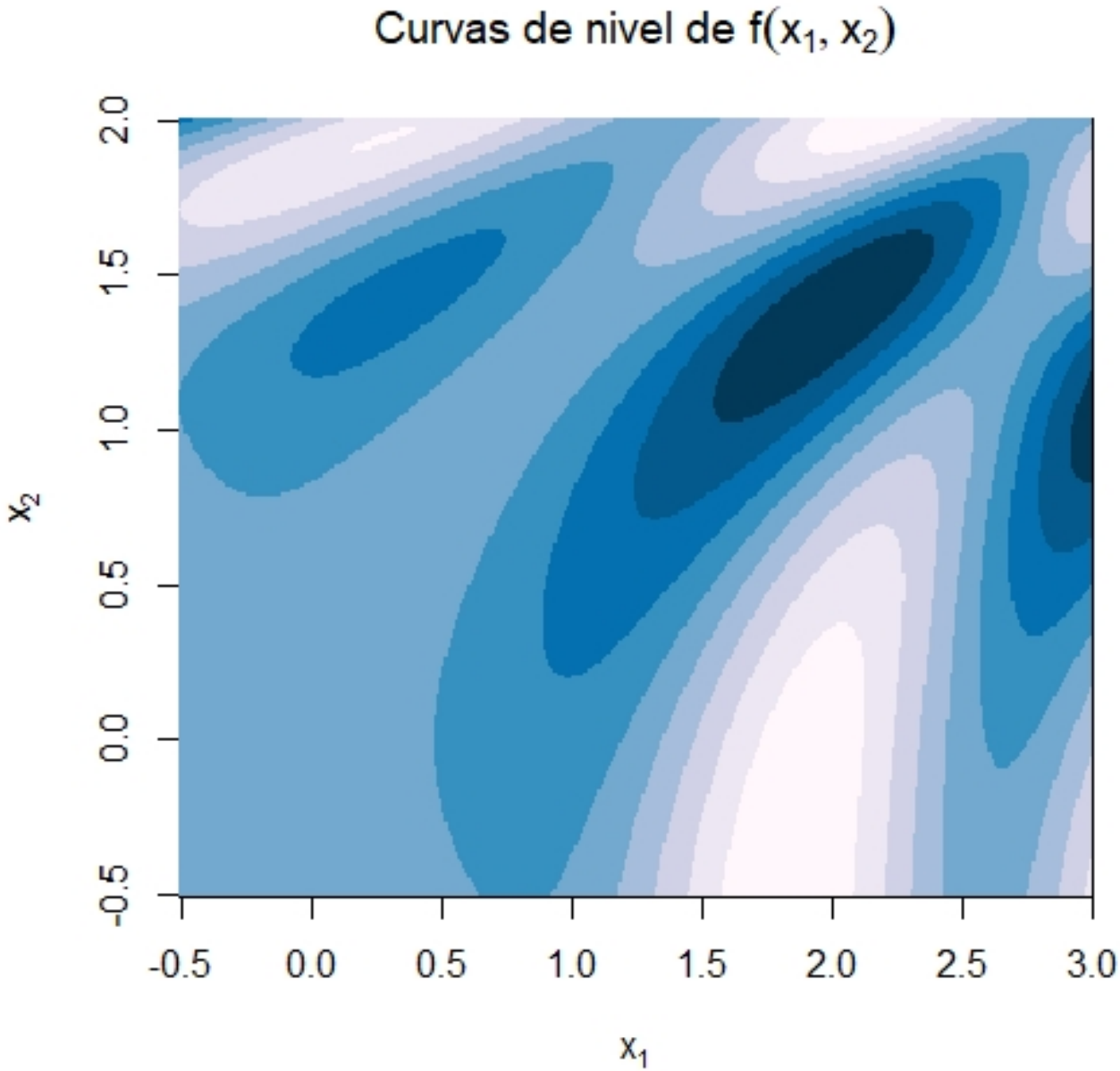
$$f(x_1, x_2) = \sin(x_1^2/2 - x_2^2/4)\cos(2x_1 - e^{x_2})$$

Considerando que la mejor dirección de ascenso es $p_n = \nabla f(x_1, x_2)$ necesitaremos el vector de derivadas parciales de $f(x_1, x_2)$.

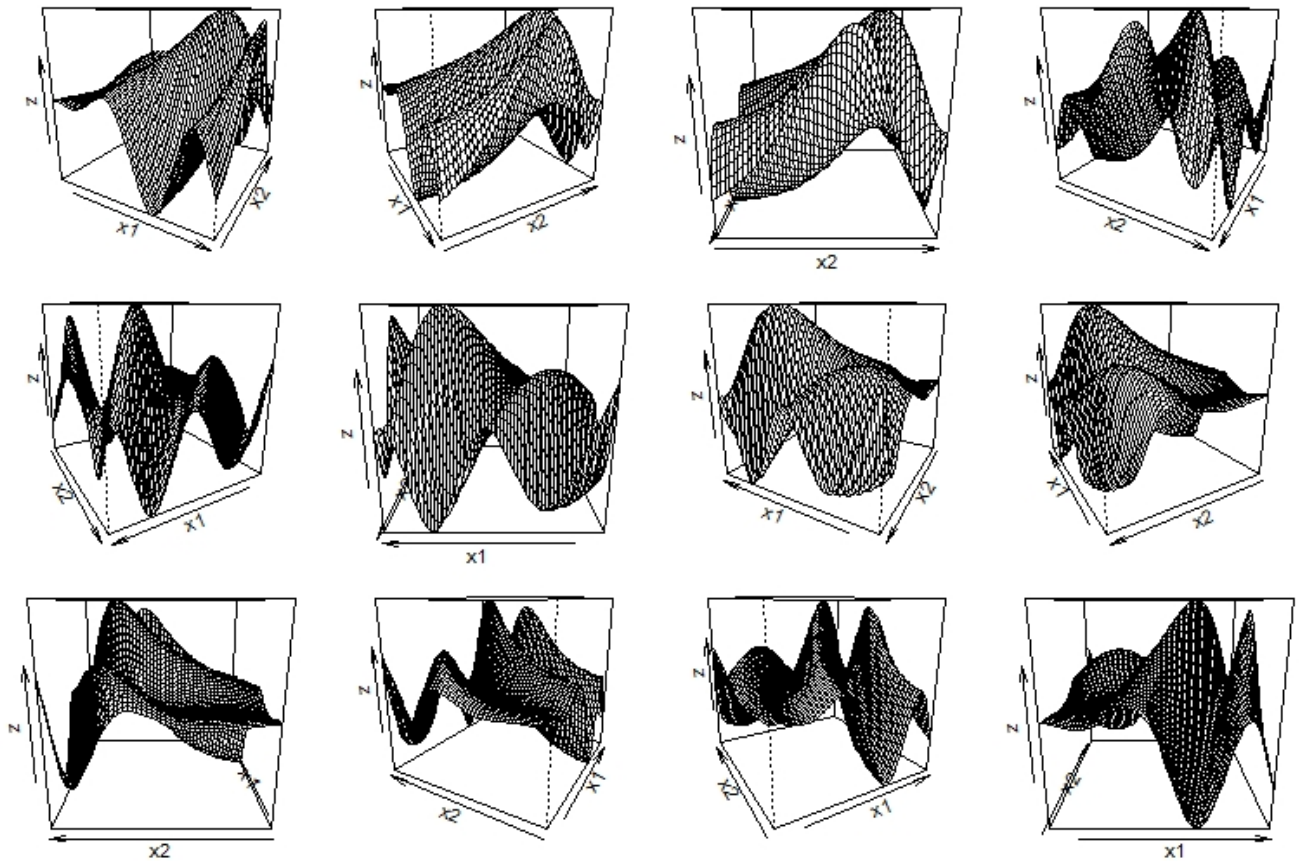
$$\begin{aligned} \frac{d}{dx_1} f(x_1, x_2) &= x_1 \cos(x_1^2/2 - x_2^2/4)\cos(2x_1 - e^{x_2}) - 2\sin(x_1^2/2 - x_2^2/4)\sin(2x_1 - e^{x_2}) \\ \frac{d}{dx_2} f(x_1, x_2) &= -\cos(x_1^2/2 - x_2^2/4)\cos(2x_1 - e^{x_2}) + e^{x_2}\sin(x_1^2/2 - x_2^2/4)\sin(2x_1 - e^{x_2}) \end{aligned}$$

- Ejecutar las líneas de código del script *f1.R* para visualizar la función $f(x_1, x_2)$ mediante curvas de nivel y gráficos tridimendionales (ver próximas diapositivas).
- En el script *ascenso.R* se encuentran las funciones necesarias para ejecutar el algoritmo de ascenso más rápido empleando backtracking. Ejecutarlo para encontrar el máximo de f.

Curvas de nivel



Y algunos gráficos de $f(x_1, x_2)$ en 3 dimensiones.



La función Banana de Rosenbrock

Si, Ud. leyó bien el nombre.

LA FUNCIÓN BANANA DE ROSENBROCK

Si, Ud. leyó bien el nombre.

$$f(x_1, x_2) = -100(x_2 - x_1^2)^2 - (1 - x_1)^2$$

LA FUNCIÓN BANANA DE ROSENBROCK

Si, Ud. leyó bien el nombre.

$$f(x_1, x_2) = -100(x_2 - x_1^2)^2 - (1 - x_1)^2$$

- Comprobar analíticamente que la función de Rosenbrock tiene un máximo en ....
- Construir en R una función que se llame banana que tenga como único argumento un vector con 2 elementos y que retorne un valor.
- Construir en R una función que se llame dbanana que tenga como único argumento un vector con 2 elementos y que retorne un vector con 2 elementos (el gradiente).
- Maximizar la función empleando como punto de arranque $x_0 = (1{,}2; 1{,}2)$ y luego inicialice el algoritmo desde $x_0 = (-1{,}2; 1)$.

Generalidades de MN

Cuando estudiamos métodos para aproximar raíces, vimos como el método de Newton-Raphson aceleraba el orden de convergencia del algoritmo de iteración de punto fijo IPF. Ahora que optimizamos funciones (univariadas o multivariadas), veremos que el MN acelera el orden de convergencia del ascenso más rápido.

Generalidades de MN

Cuando estudiamos métodos para aproximar raíces, vimos como el método de Newton-Raphson aceleraba el orden de convergencia del algoritmo de iteración de punto fijo IPF. Ahora que optimizamos funciones (univariadas o multivariadas), veremos que el MN acelera el orden de convergencia del ascenso más rápido.

Particularidades de MN

La forma en la que se gesta el método es a partir de un desarrollo de Taylor de segundo orden, por lo que además de contar con la información contenida en el gradiente de la función a maximizar, también será necesario contar con la *Hessiana* de f.

Dirección de búsqueda (1)

p_k

En el ascenso más rápido vimos que la recursión era de la forma:

Dirección de búsqueda (1)

p_k

En el ascenso más rápido vimos que la recursión era de la forma:

$$x_{n+1} = x_n + \alpha_n p_n$$

Dirección de búsqueda (1)

p_k

En el ascenso más rápido vimos que la recursión era de la forma:

$$x_{n+1} = x_n + \alpha_n p_n$$

La elección de $\nabla f(x)$ como elección de ascenso fue llevada a cabo a partir de un desarrollo de Taylor de primer orden de $f(x_{n+1})$ alrededor de x_n.

Dirección de búsqueda (1)

p_k

En el ascenso más rápido vimos que la recursión era de la forma:

$$x_{n+1} = x_n + \alpha_n p_n$$

La elección de $\nabla f(x)$ como elección de ascenso fue llevada a cabo a partir de un desarrollo de Taylor de primer orden de $f(x_{n+1})$ alrededor de x_n.

Si el mismo desarrollo es de segundo orden se tiene que:

p_k

En el ascenso más rápido vimos que la recursión era de la forma:

$$x_{n+1} = x_n + \alpha_n p_n$$

La elección de $\nabla f(x)$ como elección de ascenso fue llevada a cabo a partir de un desarrollo de Taylor de primer orden de $f(x_{n+1})$ alrededor de x_n.

Si el mismo desarrollo es de segundo orden se tiene que:

$$f(x_{n+1}) = f(x_n) + \nabla f(x_n)^T (x_{n+1} - x_n) + \frac{1}{2}(x_{n+1} - x_n)^T H_{f(x_n)}(x_{n+1} - x_n) + r_n$$

p_k

En el ascenso más rápido vimos que la recursión era de la forma:

$$x_{n+1} = x_n + \alpha_n p_n$$

La elección de $\nabla f(x)$ como elección de ascenso fue llevada a cabo a partir de un desarrollo de Taylor de primer orden de $f(x_{n+1})$ alrededor de x_n.

Si el mismo desarrollo es de segundo orden se tiene que:

$$f(x_{n+1}) = f(x_n) + \nabla f(x_n)^T (x_{n+1} - x_n) + \frac{1}{2}(x_{n+1} - x_n)^T H_{f(x_n)} (x_{n+1} - x_n) + r_n$$

Donde $\nabla f(x_n)$ y $H_{f(x_n)}$ son el gradiente y la Hessiana de f respectivamente.

p_k

En el ascenso más rápido vimos que la recursión era de la forma:

$$x_{n+1} = x_n + \alpha_n p_n$$

La elección de $\nabla f(x)$ como elección de ascenso fue llevada a cabo a partir de un desarrollo de Taylor de primer orden de $f(x_{n+1})$ alrededor de x_n.

Si el mismo desarrollo es de segundo orden se tiene que:

$$f(x_{n+1}) = f(x_n) + \nabla f(x_n)^T (x_{n+1} - x_n) + \frac{1}{2}(x_{n+1} - x_n)^T H_{f(x_n)} (x_{n+1} - x_n) + r_n$$

Donde $\nabla f(x_n)$ y $H_{f(x_n)}$ son el gradiente y la Hessiana de f respectivamente.
Si sustituimos $(x_{n+1} - x_n)$ por $\alpha_n p_n$ obtenemos:

p_k

En el ascenso más rápido vimos que la recursión era de la forma:

$$x_{n+1} = x_n + \alpha_n p_n$$

La elección de $\nabla f(x)$ como elección de ascenso fue llevada a cabo a partir de un desarrollo de Taylor de primer orden de $f(x_{n+1})$ alrededor de x_n.

Si el mismo desarrollo es de segundo orden se tiene que:

$$f(x_{n+1}) = f(x_n) + \nabla f(x_n)^T (x_{n+1} - x_n) + \frac{1}{2}(x_{n+1} - x_n)^T H_{f(x_n)} (x_{n+1} - x_n) + r_n$$

Donde $\nabla f(x_n)$ y $H_{f(x_n)}$ son el gradiente y la Hessiana de f respectivamente.
Si sustituimos $(x_{n+1} - x_n)$ por $\alpha_n p_n$ obtenemos:

$$f(x_{n+1}) = f(x_n) + \alpha_n \nabla f(x_n)^T p_n + \frac{\alpha^2}{2} p_n^T H_{f(x_n)} p_n + r_n$$

Dirección de búsqueda (2)

Este última expresión, a la que denotaremos por $\psi(p_n)$ debe verse como una función de la dirección p_n. Nuestro objetivo es determinar la dirección que maximiza dicha función. Por lo tanto procedemos a derivarla e igualarla a cero.

Dirección de búsqueda (2)

Este última expresión, a la que denotaremos por $\psi(p_n)$ debe verse como una función de la dirección p_n. Nuestro objetivo es determinar la dirección que maximiza dicha función. Por lo tanto procedemos a derivarla e igualarla a cero.

$$\begin{aligned} \frac{d}{dp_n}\psi(p_n) &= \frac{d}{dp_n}\left[f(x_n) + \alpha_n \nabla f(x_n)^T p_n + \frac{\alpha_n^2}{2} p_n^T H_{f(x_n)} p_n\right] \\ &= \alpha_n \nabla f(x_n)^T + \alpha_n^2 H_{f(x_n)} p_n \end{aligned}$$

Dirección de búsqueda (2)

Este última expresión, a la que denotaremos por $\psi(p_n)$ debe verse como una función de la dirección p_n. Nuestro objetivo es determinar la dirección que maximiza dicha función. Por lo tanto procedemos a derivarla e igualarla a cero.

$$\begin{aligned} \frac{d}{dp_n}\psi(p_n) &= \frac{d}{dp_n}\left[f(x_n) + \alpha_n \nabla f(x_n)^T p_n + \frac{\alpha_n^2}{2} p_n^T H_{f(x_n)} p_n\right] \\ &= \alpha_n \nabla f(x_n)^T + \alpha_n^2 H_{f(x_n)} p_n \end{aligned}$$

Al igualar a cero esta expresión se obtiene que $p_n = -\frac{1}{\alpha_n} H^{-1}_{f(x_n)} \nabla f(x_n)$. De donde se puede despreciar el factor $1/\alpha_n$.

Dirección de búsqueda (2)

Este última expresión, a la que denotaremos por $\psi(p_n)$ debe verse como una función de la dirección p_n. Nuestro objetivo es determinar la dirección que maximiza dicha función. Por lo tanto procedemos a derivarla e igualarla a cero.

$$\begin{aligned} \frac{d}{dp_n}\psi(p_n) &= \frac{d}{dp_n}\left[f(x_n) + \alpha_n \nabla f(x_n)^T p_n + \frac{\alpha_n^2}{2} p_n^T H_{f(x_n)} p_n\right] \\ &= \alpha_n \nabla f(x_n)^T + \alpha_n^2 H_{f(x_n)} p_n \end{aligned}$$

Al igualar a cero esta expresión se obtiene que $p_n = -\frac{1}{\alpha_n} H^{-1}_{f(x_n)} \nabla f(x_n)$. De donde se puede despreciar el factor $1/\alpha_n$.

Luego, si volvemos al desarrollo original e introducimos esta dirección, se obtiene la siguiente expresión:

Dirección de búsqueda (2)

Este última expresión, a la que denotaremos por $\psi(p_n)$ debe verse como una función de la dirección p_n. Nuestro objetivo es determinar la dirección que maximiza dicha función. Por lo tanto procedemos a derivarla e igualarla a cero.

$$\begin{aligned} \frac{d}{dp_n}\psi(p_n) &= \frac{d}{dp_n}\left[f(x_n) + \alpha_n \nabla f(x_n)^T p_n + \frac{\alpha_n^2}{2} p_n^T H_{f(x_n)} p_n\right] \\ &= \alpha_n \nabla f(x_n)^T + \alpha_n^2 H_{f(x_n)} p_n \end{aligned}$$

Al igualar a cero esta expresión se obtiene que $p_n = -\frac{1}{\alpha_n} H_{f(x_n)}^{-1} \nabla f(x_n)$. De donde se puede despreciar el factor $1/\alpha_n$.

Luego, si volvemos al desarrollo original e introducimos esta dirección, se obtiene la siguiente expresión:

$$f(x_{n+1}) \approx f(x_n) - \nabla f(x_n)^T H_{f(x_n)} \nabla f(x_n)$$

Dirección de búsqueda (2)

Este última expresión, a la que denotaremos por $\psi(p_n)$ debe verse como una función de la dirección p_n. Nuestro objetivo es determinar la dirección que maximiza dicha función. Por lo tanto procedemos a derivarla e igualarla a cero.

$$\begin{aligned} \frac{d}{dp_n}\psi(p_n) &= \frac{d}{dp_n}\left[f(x_n)+\alpha_n\nabla f(x_n)^T p_n+\frac{\alpha_n^2}{2}p_n^T H_{f(x_n)}p_n\right] \\ &= \alpha_n\nabla f(x_n)^T+\alpha_n^2 H_{f(x_n)}p_n \end{aligned}$$

Al igualar a cero esta expresión se obtiene que $p_n = -\frac{1}{\alpha_n}H^{-1}_{f(x_n)}\nabla f(x_n)$. De donde se puede despreciar el factor $1/\alpha_n$.

Luego, si volvemos al desarrollo original e introducimos esta dirección, se obtiene la siguiente expresión:

$$f(x_{n+1}) \approx f(x_n) - \nabla f(x_n)^T H_{f(x_n)}\nabla f(x_n)$$

Si deseamos que $f(x_{n+1}) > f(x_n)$, se necesita que $\nabla f(x_n)^T H_{f(x_n)}\nabla f(x_n) \leq 0$, lo cual sucede cuando $H_{f(x_n)}$ es *definida negativa*

Dirección de búsqueda (3)

De esta manera, la recursión del método de Newton sería:

Dirección de búsqueda (3)

De esta manera, la recursión del método de Newton sería:

$$x_{n+1} = x_n - \alpha_n H^{-1}_{f(x_n)} \nabla f(x_n)$$

Dirección de búsqueda (3)

De esta manera, la recursión del método de Newton sería:

$$x_{n+1} = x_n - \alpha_n H^{-1}_{f(x_n)} \nabla f(x_n)$$

Observe como la misma es un análogo vectorial del método de Newton-Raphson donde ahora se busca anular el gradiente de $f(x)$. Sin embargo, esto último, junto con la última observación de la diapositiva anterior, tiene repercusiones.

Dirección de búsqueda (3)

De esta manera, la recursión del método de Newton sería:

$$x_{n+1} = x_n - \alpha_n H^{-1}_{f(x_n)} \nabla f(x_n)$$

Observe como la misma es un análogo vectorial del método de Newton-Raphson donde ahora se busca anular el gradiente de $f(x)$. Sin embargo, esto último, junto con la última observación de la diapositiva anterior, tiene repercusiones.

Mediante este algoritmo, se busca encontrar al vector x^* de modo que $\nabla f(x^*) = 0$,

Dirección de búsqueda (3)

De esta manera, la recursión del método de Newton sería:

$$x_{n+1} = x_n - \alpha_n H^{-1}_{f(x_n)} \nabla f(x_n)$$

Observe como la misma es un análogo vectorial del método de Newton-Raphson donde ahora se busca anular el gradiente de $f(x)$. Sin embargo, esto último, junto con la última observación de la diapositiva anterior, tiene repercusiones.

Mediante este algoritmo, se busca encontrar al vector x^* de modo que $\nabla f(x^*) = 0$, pero

Dirección de búsqueda (3)

De esta manera, la recursión del método de Newton sería:

$$x_{n+1} = x_n - \alpha_n H^{-1}_{f(x_n)} \nabla f(x_n)$$

Observe como la misma es un análogo vectorial del método de Newton-Raphson donde ahora se busca anular el gradiente de $f(x)$. Sin embargo, esto último, junto con la última observación de la diapositiva anterior, tiene repercusiones.

Mediante este algoritmo, se busca encontrar al vector x^* de modo que $\nabla f(x^*) = 0$, pero nada garantiza que el valor $f(x^*)$ sea un máximo, o un mínimo, ni siquiera un punto de inflexión.

Dirección de búsqueda (3)

De esta manera, la recursión del método de Newton sería:

$$x_{n+1} = x_n - \alpha_n H^{-1}_{f(x_n)} \nabla f(x_n)$$

Observe como la misma es un análogo vectorial del método de Newton-Raphson donde ahora se busca anular el gradiente de $f(x)$. Sin embargo, esto último, junto con la última observación de la diapositiva anterior, tiene repercusiones.

Mediante este algoritmo, se busca encontrar al vector x^* de modo que $\nabla f(x^*) = 0$, pero nada garantiza que el valor $f(x^*)$ sea un máximo, o un mínimo, ni siquiera un punto de inflexión.

Como se señaló anteriormente, la forma de lograr que el algoritmo converja a un máximo es asegurándonos de que la Hessiana sea definida negativa. Entonces, cuando en los libros se ve la frase:

"... debe elegir un punto de arranque cercano al máximo..."

lo que se está diciendo, es que por lo menos debemos asegurarnos de que $H_{f(x_0)}$ sea definida negativa.

RETOMANDO LA FUNCIÓN TRIGONOMÉTRICA

Cuando trabajamos con el ascenso más rápido, maximizamos la función:

$$f(x_1, x_2) = \sin(x_1^2/2 - x_2^2/4)\cos(2x_1 - e^{x_2})$$

y comenzando el algoritmo en el punto $(0{,}1; 0{,}3)$ llegamos al valor máximo luego de 45 iteraciones.

RETOMANDO LA FUNCIÓN TRIGONOMÉTRICA

Cuando trabajamos con el ascenso más rápido, maximizamos la función:

$$f(x_1, x_2) = \sin(x_1^2/2 - x_2^2/4)\cos(2x_1 - e^{x_2})$$

y comenzando el algoritmo en el punto $(0{,}1; 0{,}3)$ llegamos al valor máximo luego de 45 iteraciones.

Empleando el código del script optimizacion.R:

- Ejecutar el método de Newton a partir del mismo punto de arranque y "vea que pasa".
- Utilizar la función *chequeo* para determinar si la Hessiana en $(0{,}1; 0{,}3)$ es definida positiva.
- Antes descartar el método de Newton, inicializarlo desde $(2{,}5; 1{,}5)$ y comparlo con el método del ascenso más rápido, iniciando este último desde el mismo punto.
- ¿Cómo podría seleccionar adecuadamente un "buen" punto de arranque?

GENERALIDADES PARA QR

A diferencia del método de Newton, estos métodos sólo requieren de evaluaciones de la función a minimizar y su gradiente. La idea principal es que al ir registrando cambios en el gradiente a través de las iteraciones, es posible aproximar la Hessiana de $f(x)$. Si bien el orden de convergencia resultante no es cuadrático, se alcanza un orden de convergencia superlineal.

GENERALIDADES PARA QR

A diferencia del método de Newton, estos métodos sólo requieren de evaluaciones de la función a minimizar y su gradiente. La idea principal es que al ir registrando cambios en el gradiente a través de las iteraciones, es posible aproximar la Hessiana de $f(x)$. Si bien el orden de convergencia resultante no es cuadrático, se alcanza un orden de convergencia superlineal.

PARTICULARIDADES PARA QR

A la hora de comparar esta familia de métodos con los dos anteriores, el orden de convergencia es superior al del ascenso más rápido (que es de orden lineal) pero menor al método de Newton. Sin embargo, al no tener que evaluar la Hessiana, puede llegar a ser más veloz que este último.

B_n

En el método de Newton vimos que la recursión era de la forma:

B_n

En el método de Newton vimos que la recursión era de la forma:

$$x_{n+1} = x_n - \alpha_n H^{-1}_{f(x_n)} \nabla f(x_n)$$

B_n

En el método de Newton vimos que la recursión era de la forma:

$$x_{n+1} = x_n - \alpha_n H^{-1}_{f(x_n)} \nabla f(x_n)$$

La idea de los métodos de *QN* es aproximar la Hessiana por lo que estos métodos son de la forma.

B_n

En el método de Newton vimos que la recursión era de la forma:

$$x_{n+1} = x_n - \alpha_n H^{-1}_{f(x_n)} \nabla f(x_n)$$

La idea de los métodos de *QN* es aproximar la Hessiana por lo que estos métodos son de la forma.

$$x_{n+1} = x_n + \alpha_n B_n^{-1} \nabla f(x_n)$$

B_n

En el método de Newton vimos que la recursión era de la forma:

$$x_{n+1} = x_n - \alpha_n H^{-1}_{f(x_n)} \nabla f(x_n)$$

La idea de los métodos de *QN* es aproximar la Hessiana por lo que estos métodos son de la forma.

$$x_{n+1} = x_n + \alpha_n B_n^{-1} \nabla f(x_n)$$

La actualización de la aproximación de la Hessiana entre las iteraciones n y $n+1$ parte de la siguiente idea:

$$B_{n+1}(x_{n+1} - x_n) = \nabla f(x_{n+1}) - \nabla f(x_n)$$

B_n

En el método de Newton vimos que la recursión era de la forma:

$$x_{n+1} = x_n - \alpha_n H^{-1}_{f(x_n)} \nabla f(x_n)$$

La idea de los métodos de *QN* es aproximar la Hessiana por lo que estos métodos son de la forma.

$$x_{n+1} = x_n + \alpha_n B_n^{-1} \nabla f(x_n)$$

La actualización de la aproximación de la Hessiana entre las iteraciones n y $n+1$ parte de la siguiente idea:

$$B_{n+1}(x_{n+1} - x_n) = \nabla f(x_{n+1}) - \nabla f(x_n)$$

Si se renombra $s_n = x_{n+1} - x_n$ y $y_n = \nabla f(x_{n+1}) - \nabla f(x_n)$, se obtiene la llamada *ecuación secante*

$$B_{n+1} s_n = y_n$$

B_n

La aproximación B_n resulta de encontrar una solución a la ecuación secante. Sin embargo esta ecuación admite infinitas soluciones debido a los $k(k+1)/2$ grados de libertad en B_n.

B_n

La aproximación B_n resulta de encontrar una solución a la ecuación secante. Sin embargo esta ecuación admite infinitas soluciones debido a los $k(k+1)/2$ grados de libertad en B_n.

La primera solución presentada por Davidon en la década de los cincuenta es:

$$B_{n+1} = \left(I - \frac{1}{y_n^T s_n} y_n s_n^T\right) B_n \left(I - \frac{1}{y_n^T s_n} s_n y_n^T\right) + \frac{y_n y_n^T}{y_n^T s_n}$$

B_n

La aproximación B_n resulta de encontrar una solución a la ecuación secante. Sin embargo esta ecuación admite infinitas soluciones debido a los $k(k+1)/2$ grados de libertad en B_n.

La primera solución presentada por Davidon en la década de los cincuenta es:

$$B_{n+1} = \left(I - \frac{1}{y_n^T s_n} y_n s_n^T\right) B_n \left(I - \frac{1}{y_n^T s_n} s_n y_n^T\right) + \frac{y_n y_n^T}{y_n^T s_n}$$

Luego, la inversa de esta matriz es empleada en la recursión para obtener x_{n+1} a partir de x_n. Sin embargo, en vez de actualizar B_n y luego invertirla, es posible actualizar la inversa.

$$H_{n+1} = H_n - \frac{H_n y_n y_n^T H_n}{y_n^T H_n y_n} + \frac{s_n s_n^T}{y_n^T s_n}$$

De esta manera queda definido el método DFP (Davidon-Fletcher-Powell). A modo de ejercicio, obtenga este resultado aplicando la fórmula de Sherman-Morrison-Woodbury

Rango de la actualización

Las dos fórmulas de la diapositiva anterior son de rango dos, en el sentido de que la modificación de B_n (o de H_n) es a partir de dos matrices de rango uno.

RANGO DE LA ACTUALIZACIÓN

Las dos fórmulas de la diapositiva anterior son de rango dos, en el sentido de que la modificación de B_n (o de H_n) es a partir de dos matrices de rango uno.
Observe como estas fórmulas:

- incorporan la información observada del cambio en el gradiente.
- mantienen la simetría
- garantizan que si B_n (o H_n) es definida negativa, entoces B_{n+1} (o H_{n+1}) también lo es. (¿podría probarlo?)

RANGO DE LA ACTUALIZACIÓN

Las dos fórmulas de la diapositiva anterior son de rango dos, en el sentido de que la modificación de B_n (o de H_n) es a partir de dos matrices de rango uno.
Observe como estas fórmulas:

- incorporan la información observada del cambio en el gradiente.
- mantienen la simetría
- garantizan que si B_n (o H_n) es definida negativa, entoces B_{n+1} (o H_{n+1}) también lo es. (¿podría probarlo?)

Un segundo método, surge de imponer la ecuación secante a la inversa de B_n:

$$H_{n+1} = \left(I - \frac{1}{y_n^T s_n} s_n y_n^T\right) H_n \left(I - \frac{1}{y_n^T s_n} y_n s_n^T\right) + \frac{s_n s_n^T}{y_n^T s_n}$$

Siendo este es el método *BFGS* (Broyden-Fletcher-Goldfarg-Shanno).

RANGO DE LA ACTUALIZACIÓN

Las dos fórmulas de la diapositiva anterior son de rango dos, en el sentido de que la modificación de B_n (o de H_n) es a partir de dos matrices de rango uno.
Observe como estas fórmulas:

- incorporan la información observada del cambio en el gradiente.
- mantienen la simetría
- garantizan que si B_n (o H_n) es definida negativa, entoces B_{n+1} (o H_{n+1}) también lo es. (¿podría probarlo?)

Un segundo método, surge de imponer la ecuación secante a la inversa de B_n:

$$H_{n+1} = \left(I - \frac{1}{y_n^T s_n} s_n y_n^T\right) H_n \left(I - \frac{1}{y_n^T s_n} y_n s_n^T\right) + \frac{s_n s_n^T}{y_n^T s_n}$$

Siendo este es el método *BFGS* (Broyden-Fletcher-Goldfarg-Shanno).
El único detalle que queda por resolver de este método es la selección de H_0. Algunas alternativas son:

- Aproximar la Hessiana en x_0 e invertirla.
- Simplemente usar $H_0 = -I$

Actualización de rango uno

Tanto la actualización DFP como la BFGS son actualicaciones de rango dos, sin embargo, es posible obtener una actualización de rango uno. Esta es mas sencilla y requiere la mitad del tiempo empleada por las otras dos

Actualización de rango uno

Tanto la actualización DFP como la BFGS son actualicaciones de rango dos, sin embargo, es posible obtener una actualización de rango uno. Esta es mas sencilla y requiere la mitad del tiempo empleada por las otras dos

La fórmula para actualizar B_n es:

$$B_{n+1} = B_n + \frac{(y_n - B_n s_n)(y_n - B_n s_n)^T}{(y_n - B_n s_n)^T s_n}$$

Actualización de rango uno

Tanto la actualización DFP como la BFGS son actualicaciones de rango dos, sin embargo, es posible obtener una actualización de rango uno. Esta es mas sencilla y requiere la mitad del tiempo empleada por las otras dos

La fórmula para actualizar B_n es:

$$B_{n+1} = B_n + \frac{(y_n - B_n s_n)(y_n - B_n s_n)^T}{(y_n - B_n s_n)^T s_n}$$

Y aplicando la fórmula Sherman-Morrison obtenemos la actualización de la inversa.

$$H_{n+1} = H_n + \frac{(s_n - H_n y_n)(s_n - H_n y_n)^T}{(s_n - H_n y_n)^T y_n}$$

Actualización de rango uno

Tanto la actualización DFP como la BFGS son actualicaciones de rango dos, sin embargo, es posible obtener una actualización de rango uno. Esta es mas sencilla y requiere la mitad del tiempo empleada por las otras dos

La fórmula para actualizar B_n es:

$$B_{n+1} = B_n + \frac{(y_n - B_n s_n)(y_n - B_n s_n)^T}{(y_n - B_n s_n)^T s_n}$$

Y aplicando la fórmula Sherman-Morrison obtenemos la actualización de la inversa.

$$H_{n+1} = H_n + \frac{(s_n - H_n y_n)(s_n - H_n y_n)^T}{(s_n - H_n y_n)^T y_n}$$

Sin embargo, este método, conocido como SR1, tiene la desventaja de no garantizar que partiendo de B_n definida negativa, se obtenga que B_{n+1} sea definida negativa (¿podría probarlo?).

Código

Observe que para "transferir" estos algoritmos al lenguaje del R, solo debe modificar el las funciones *ascenso* o *newton* de la siguiente manera.

CÓDIGO

Observe que para "transferir" estos algoritmos al lenguaje del R, solo debe modificar el las funciones *ascenso* o *newton* de la siguiente manera.

En primer lugar deberá inicializar algunos objetos:

```
1     H  <- -diag(length(x0))
2     pn <- df0 <- -H%*%dFUN(x0)
3     df1<- dFUN(x1)
4     Id <- -H
```

CÓDIGO

Observe que para "transferir" estos algoritmos al lenguaje del R, solo debe modificar el las funciones *ascenso* o *newton* de la siguiente manera.

En primer lugar deberá inicializar algunos objetos:

```
1     H  <- -diag(length(x0))
2     pn <- df0 <- -H%*%dFUN(x0)
3     df1<- dFUN(x1)
4     Id <- -H
```

Luego deberá incluir el código de la actualización de H dentro del while.

```
1       sn<-x1-x0
2       yn<-df1-df0
3       rn<-1/sum(yn*sn)
4       H<-(Id-rn*sn%*%t(yn))%*%H%*%(Id-rn*yn%*%t(sn))+rn*sn%*%t(sn)
```

CÓDIGO

Observe que para "transferir" estos algoritmos al lenguaje del R, solo debe modificar el las funciones *ascenso* o *newton* de la siguiente manera.

En primer lugar deberá inicializar algunos objetos:

```
1     H  <- -diag(length(x0))
2     pn <- df0 <- -H%*%dFUN(x0)
3     df1<- dFUN(x1)
4     Id <- -H
```

Luego deberá incluir el código de la actualización de H dentro del while.

```
1         sn<-x1-x0
2         yn<-df1-df0
3         rn<-1/sum(yn*sn)
4         H<-(Id-rn*sn%*%t(yn))%*%H%*%(Id-rn*yn%*%t(sn))+rn*sn%*%t(sn)
```

Y luego actualizar todo lo demás.

Ejercicio 6

Comparación

En el script *optimización* se encuentran los algoritmos vistos hasta ahora, incluyendo una función llamada *bfgs.R*. Por otro lado, en el script *ejercicio6.R* se encuentra un conjunto de funciones para evaluar la función trigonométrica y la función de Rosenbrock, así como sus gradientes y Hessianas.
Compare el desempeño de los tres algoritmos partiendo desde distintos puntos de arranque.

Ejercicio 6

Comparación

En el script *optimización* se encuentran los algoritmos vistos hasta ahora, incluyendo una función llamada *bfgs.R*. Por otro lado, en el script *ejercicio6.R* se encuentra un conjunto de funciones para evaluar la función trigonométrica y la función de Rosenbrock, así como sus gradientes y Hessianas.
Compare el desempeño de los tres algoritmos partiendo desde distintos puntos de arranque.

Los otros

Basándose en el código de la función *bfgs* implemente dos funciones que le permitan comparar los algoritmos del ejercicio anterior con los métodos *DFP* y *SR*1.

Ejercicio 6

Comparación

En el script *optimización* se encuentran los algoritmos vistos hasta ahora, incluyendo una función llamada *bfgs.R*. Por otro lado, en el script *ejercicio6.R* se encuentra un conjunto de funciones para evaluar la función trigonométrica y la función de Rosenbrock, así como sus gradientes y Hessianas.
Compare el desempeño de los tres algoritmos partiendo desde distintos puntos de arranque.

Los otros

Basándose en el código de la función *bfgs* implemente dos funciones que le permitan comparar los algoritmos del ejercicio anterior con los métodos *DFP* y *SR*1.

Condición de parada (opcional)

- Modifique el objeto *cambio* en las funciones *bfgs* y *newton* de modo que el algoritmo se detenga cuando la norma del gradiente sea menor que la tolerancia. Esto le permitirá realizar una comparación más "justa" con el ascenso más rápido.
- Incluya un nuevo parámetro en las funciones que le permita elegir entre calcular el objeto *cambio* empleando la norma infinito o la norma euclideana. Deberá incluir una cláusula *if* dentro del while.

VEROSIMILITUD GAUSSIANA

En el script *aplicaciones.r* se encuentra un código que le permite simular n realizaciones de una variable aleatoria normal con media 40 y desvío 6.

- Plantee la función de log-verosimilitud y el score de la muestra.
- Escriba en R ambas funciones teniendo en cuenta que su único argumento deberá ser un vector de 2 parámetros.
- Intente maximizarlas con los algoritmos vistos en clase.
- Cuando solucione ls "inconvenientes", encuentre los estimadores máximo verosímiles.

VEROSIMILITUD GAUSSIANA

En el script *aplicaciones.r* se encuentra un código que le permite simular n realizaciones de una variable aleatoria normal con media 40 y desvío 6.

- Plantee la función de log-verosimilitud y el score de la muestra.
- Escriba en R ambas funciones teniendo en cuenta que su único argumento deberá ser un vector de 2 parámetros.
- Intente maximizarlas con los algoritmos vistos en clase.
- Cuando solucione ls "inconvenientes", encuentre los estimadores máximo verosímiles.

MÍNIMOS CUADRADOS

El set de datos *mtcars* contiene variables referentes a diferentes características de autos de la década de los setenta. Asuma que se desea predecir el rendimiento de los automóviles (mpg) a partir de una función lineal del peso (wt), el tiempo que tardan en recorrer un cuarto de milla ($qsec$) y una indicadora de que el auto cuente con transmisión automática (am).

$$mpg_i = \beta_0 + \beta_1 wt_i + \beta_1 qsec_i + \beta_2 am_i + \varepsilon_i$$

Encuentre los estimadores máximo verosímiles de los parámetros asumiendo $\varepsilon_i \sim N(0, \sigma^2)$. ¿Y si los errores tuvieran una dsitribución $t_\nu(0, \sigma)$?

¿Preguntas?

Muchas Gracias

Bibliografía

Jones, R. and, M. R. and Robinson, A. (2014). *Introduction to Scientific Programming and Simulation Using Centrality:*. The R Series. Taylor & Francis Group, 2nd edition.

Jorge Nocedal, S. W. (2006). *Numerical Optimization*. Springer series in operations research. Springer, 2nd ed edition.

Robert, C. and Cassella, G. (2010). *Introducing Monte Carlo Methods with Centrality:*. Springer.