Algoritmos de optimización para funciones con ruido

Introducción:

Los algoritmos que hemos visto hasta el momento no pueden ser implementados si el gradiente de la función objetivo no está disponible analíticamente. Vamos a presentar algunos métodos de optimización que no necesitan la evaluación de gradientes. Los algoritmos que presentamos son útiles para funciones que son perturbaciones de funciones regulares.

- **Filtrado implícito**: Es el algoritmo de máximo descenso con el gradiente aproximado mediante diferencias.
- **Nelder-Mead**: Es un algoritmo de búsqueda directa que meramente compara valores funcionales; los valores de la función objetivo son tomados en un conjunto de puntos muestreados (símplices) y son usados para continuar el proceso.
- **Búsqueda multidireccional**: Utiliza símplices para aproximar el mínimo, mediante contracciones, expansiones y reflexiones.
- **Hooke-Jeeves**: Busca las direcciones de descenso mediante movimientos exploratorios a través de las direcciones coordenadas.

A - Filtrado implícito

El algoritmo de filtrado implícito fue diseñado para problemas en que la función objetivo es una perturbación de alta frecuencia y baja amplitud de un problema regular simple. Es un método de mayor descenso o un método cuasi-Newton en diferencias finitas, en el que el paso de discretización se ajusta a medida que la optimización progresa. De este modo el algoritmo filtra implícitamente la perturbación de alta frecuencia.

Implementación del algoritmo

Para $x \in R^n$ y $h \neq 0$ el gradiente en diferencias finitas progresivas de f con paso h en ese punto está dado por:

$$(\nabla_h^p f(x))_i = \frac{f(x+he_i) - f(x)}{h}$$

donde e_i es el vector unitario en la i-ésima dirección coordenada y $(\nabla_h^p f(x))_i$ denota la i-ésima componente del gradiente en diferencias.

La iteración básica del método de mayor descenso en diferencias finitas esta dada por:

$$x_+ = x_c - \lambda \nabla_h f(x_c)$$

donde x_c representa el iterante actual y x_+ el nuevo iterante.

Se usa una búsqueda de línea de Armijo y se exige que se cumpla la condición de decrecimiento suficiente:

$$f(x - \lambda \nabla_h f(x)) - f(x) < -\alpha\lambda \left\| \nabla_h f(x) \right\|^2$$

para algún $\alpha > 0$ dado.

El algoritmo se termina cuando

$$\left\| \nabla_h f(x) \right\| \leq \tau h$$

para algún $\tau > 0$, cuando se supera un máximo fijado de iteraciones o cuando la búsqueda de línea falla.

El **algoritmo de filtrado implícito** ejecuta repetidas veces el algoritmo de mayor descenso en diferencias reduciendo h después de cada terminación de este.

Métodos cuasi-Newton

El comportamiento del filtrado implícito mejora mucho usando una aproximación cuasi-Newton para el hessiano.
La iteración cuasi_newton en diferencias finitas es:

$$x_+ = x_c + d_c$$

donde $d_c = -H_c^{-1}\nabla_h f(x_c)$ para H_c aproximación cuasi_newton del Hessiamo de f.

El test para decrecimiento suficiente es, en este caso:

$$f(x+\lambda d_c) - f(x) < -\alpha\lambda\nabla_h f(x)^t d_c$$

Se actualiza la aproximación del hessiano con una fórmula BFGS, y si esta no es definida positiva se reemplaza por la matriz identidad.

En este caso el **algoritmo de filtrado implícito** ejecuta repetidas veces el algoritmo cuasi-Newton en diferencias reduciendo h después de cada terminación de este.

B - Método de Nelder-Mead

El algoritmo símplex Nelder-Mead es un método de búsqueda directa para la minimización sin restricciones de funciones multidimensionales. Desde su primera publicación en 1965, este algoritmo se ha convertido en uno de los métodos más ampliamente usados para la optimización no lineal sin restricciones, especialmente en el campo de la química, ingeniería química y la medicina. A pesar de ser ampliamente usado, apenas se han probado de forma explícita resultados teóricos del algoritmo Nelder-Mead

El método Nelder-Mead trata de minimizar una función escalar no lineal de *n* variables usando sólo valores de la función, sin obtener ninguna información de la derivada (ni implícita ni explícitamente). En el análisis de este algoritmo utilizaremos mucho un concepto que a continuación pasamos a definir:

<u>Definición</u>: *Definimos símplex n-dimensional como una figura geométrica en dimensión n de volumen no nulo, que es la envolvente convexa de n+1 puntos. Esto es, dados los puntos* $a_1, a_2, ..., a_{n+1}$*, el símplex será*

$$\Delta = \left\{ x \in R^n / x = \sum_{i=1}^{n+1} \lambda_i \cdot a_i \quad \text{con} \quad 0 \le i \le n+1 \; ; \; 0 \le \lambda_i \le 1 \; ; \; \sum_{i=1}^{n+1} \lambda_i = 1 \right\}$$

Cada iteración de este método comienza con un símplex, especificado por sus *n+1* vértices y los valores de la función asociados. Tras calcular uno o más puntos de prueba y evaluar la función en dichos puntos, la iteración dará como resultado un nuevo símplex, de tal manera que los valores de la función en los vértices satisfagan de alguna forma una condición de descenso con respecto al símplex anterior.

Entre sus ventajas cabe destacar:

- Experimenta grandes mejoras durante las primeras iteraciones, lo que lo hace muy válido, por ejemplo, para procesos de control industrial donde simplemente se pretende encontrar valores de parámetros que mejoren el rendimiento de una máquina.

- El método Nelder-Mead tiende a necesitar substancialmente menos evaluaciones de función que las otras alternativas, lo que lo hace muy útil en aquellas aplicaciones donde la evaluación de la función sea muy cara o consuma mucho tiempo. Aunque la precisión alcanzada no es la deseable, la rentabilidad de su utilización prevalece sobre las carencias teóricas de este algoritmo.

- Es un algoritmo fácil de programar y sus etapas son sencillas de explicar.

Implementación del algoritmo

Caso n-dimensional

El funcionamiento de este algoritmo se basa en la construcción de una sucesión de símplices para aproximarse al punto óptimo. Para definir de forma completa el método deben especificarse cuatro parámetros:

- ρ : Coeficiente de reflexión.
- χ : Coeficiente de expansión.
- γ : Coeficiente de contracción.
- σ : Coeficiente de encogimiento o reducción (shrink).

que deben satisfacer:

$$\rho > 0 \ ; \ \chi > 1 \ ; \ 0 < \gamma < 1 \ ; \ 0 < \sigma < 1$$

(en general se usa: $\rho = 1$; $\chi = 2$; $\gamma = 1/2$; $\sigma = 1/2$)

Al principio de la **iteración k**, disponemos de un símplex no degenerativo Δ_k, que viene dado por sus (*n+1*) vértices, cada uno de los cuales es un punto de R^n. La iteración k empezará ordenando y etiquetando estos vértices como

$$x_1, x_2, ..., x_{n+1}$$

de tal manera que

$$f_1 \leq f_2 \leq ... \leq f_{n+1}$$

donde

$$f_i = f\left(x_i\right)$$

La iteración genera (n+1) vértices que definen un nuevo y diferente símplex para la próxima iteración, esto es $\Delta_{k+1} \neq \Delta_k$. Como nosotros buscamos minimizar f, nos referiremos a x_1 como el mejor vértice y a x_{n+1} como el peor.

Una vez ordenados los puntos se plantean distintos casos:

a.- <u>Reflexión</u>: Se calcula el punto de reflexión

$$x_r = \bar{x} + \rho \cdot \left(\bar{x} - x_{n+1} \right) = \left(1 + \rho \right) \cdot \bar{x} - \rho \cdot x_{n+1}$$

donde

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

es el centroide de los n mejores puntos del símplex de la iteración actual. Si a continuación evaluamos $f_r = f\left(x_r\right)$ tenemos que

Si $f_1 \leq f_r < f_n$, aceptamos el punto reflejado x_r y termina la iteración.

b.- <u>Expansión</u>: Si $f_r < f_1$, calculamos el punto de expansión

$$x_e = \bar{x} + \chi \cdot \left(x_r - \bar{x} \right) = \bar{x} + \rho \cdot \chi \cdot \left(\bar{x} - x_{n+1} \right) = \left(1 + \rho \cdot \chi \right) \cdot \bar{x} - \rho \cdot \chi \cdot x_{n+1}$$

y evaluamos $f_e = f(x_e)$. Si $f_e < f_r$, aceptamos x_e y finaliza la iteración; en otro caso aceptamos x_r y termina la iteración.

c.- <u>Contracción</u>:

Si $f_r \geq f_n$, se realiza una contracción entre $\bar{x}$ y el mejor punto de entre x_r y x_{n+1}. La contracción puede ser hacia dentro o hacia fuera:

c.1 Contracción hacia fuera: Tiene lugar cuando $f_n \leq f_r < f_{n+1}$, esto es, cuando x_r es estrictamente mejor que x_{n+1}. Calculamos

$$x_c = \bar{x} + \gamma \cdot (x_r - \bar{x}) = \bar{x} + \rho \cdot \gamma \cdot (\bar{x} - x_{n+1}) = (1 + \rho \cdot \gamma) \cdot \bar{x} - \rho \cdot \gamma \cdot x_{n+1}$$

y evaluamos $f_c = f(x_c)$

Si $f_c \leq f_r$, aceptamos x_c y finaliza la iteración; en otro caso nos vamos a **d** (shrink).

c.2 Contracción hacia adentro: Tiene lugar cuando $f_r \geq f_{n+1}$. Calculamos

$$x_{cc} = \bar{x} - \gamma \cdot \left(\bar{x} - x_{n+1} \right) = \left(1 - \gamma \right) \cdot \bar{x} + \gamma \cdot x_{n+1}$$

y evaluamos $f_{cc} = f\left(x_{cc} \right)$

Si $f_{cc} < f_{n+1}$, aceptamos x_{cc} y finaliza la iteración; en otro caso nos vamos a **d** (shrink).

d.- <u>Shrink</u>: Se evalúa f en los puntos

$$y_i = x_1 + \sigma \cdot \left(x_i - x_1 \right) \text{ , con } i = 2,...,n+1$$

Los vértices (desordenados) para la próxima iteración serán $x_1, y_2, ..., y_{n+1}$.

Caso bidimensional

Explicaremos el funcionamiento del algoritmo para una iteración cualquiera. Los tres vértices x_1, x_2 y x_3 de cada símplex (trián son ordenados según el valor que toma la función objetivo en cada uno de ellos,

$$f\left(x_1\right) \leq f\left(x_2\right) \leq f\left(x_3\right)$$

y el peor vértice, en este caso x_3 , es reemplazado por un nuevo punto de la forma

$$x(\nu) = (1+\nu)\cdot \bar{x} + \nu \cdot x_3$$

donde $\bar{x}$ es el centroide de la envolvente convexa del conjunto $\{x_1, x_2\}$, esto es, el punto medio del segmento formado por ambos puntos. El valor de ν se selecciona de la secuencia $-1 < \nu_d < 0 < \nu_c < \nu_b < \nu_a$ (valores típicos de estos parámetros son $\nu_d = -0.5$, $\nu_c = 0.5$, $\nu_b = 1$ y $\nu_a = 2$) mediante las reglas mostradas en el siguiente algoritmo:

Mientras $f(x_3) - f(x_1)$ no sea lo suficientemente pequeño, calcularemos $x(\nu_b)$ y $f_b = f\left(x(\nu_b)\right)$ y :

a) Si $f_b < f(x_1)$, calcularemos $f_a = f\left(x(\nu_a)\right)$.

Si $f_a < f_b$, sustituiremos x_3 por $x(\nu_a)$; de no ser así, sustituiremos x_3 por $x(\nu_b)$.

b) Si $f(x_1) \le f_b < f(x_2)$, sustituiremos x_3 por $x(\nu_b)$ e iremos a **f)**.

c) Si $f(x_2) \le f_b < f(x_3)$, calcularemos $f_c = f\left(x(\nu_c)\right)$.

Si $f_c \le f_b$, sustituiremos x_3 por $x(\nu_c)$ e iremos a **f)**; en otro caso iremos a **e)**.

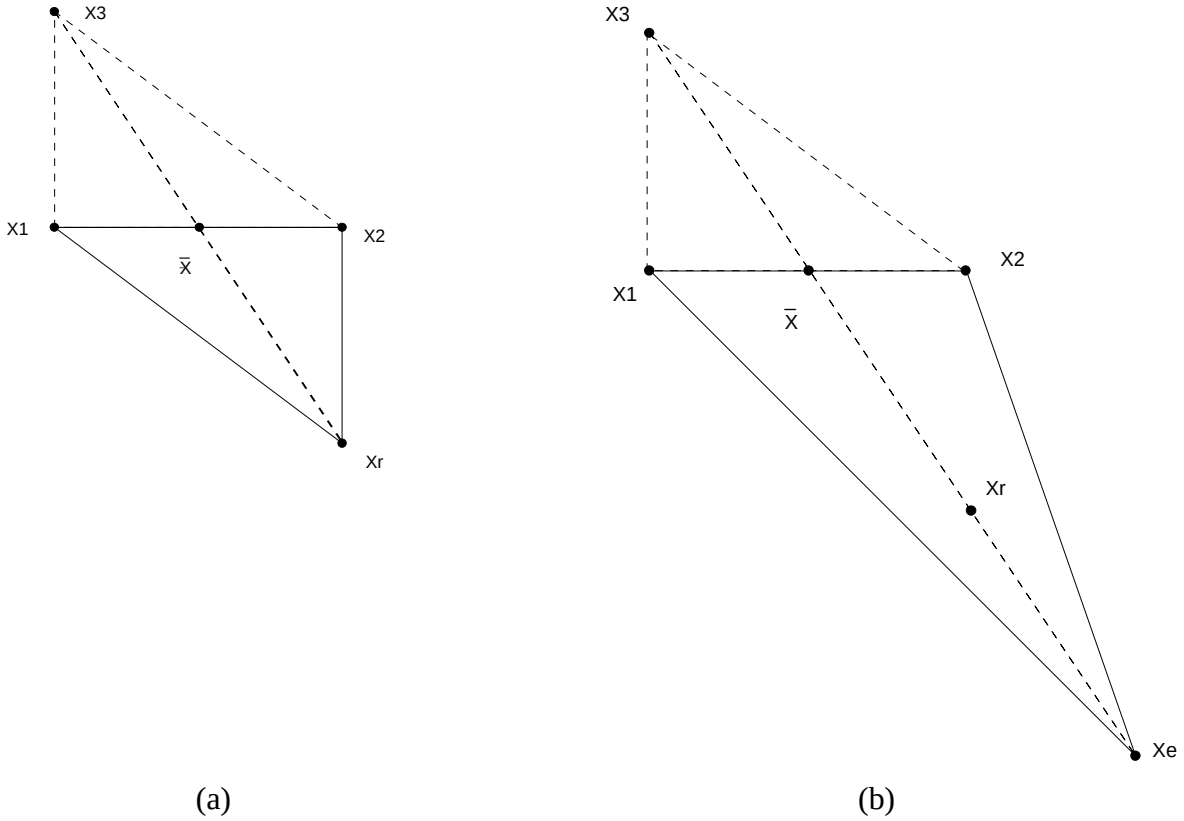
d) Si $f(x_3) \le f_b$, calcularemos $f_d = f\left(x(\nu_d)\right)$.

Si $f_d \le f(x_3)$, sustituiremos x_3 por $x(\nu_d)$ e iremos a **f)**; en otro caso iremos a **e)**.

e) Asignar $x_2 = x_1 + (x_2 - x_1)/2$ y $x_3 = x_1 + (x_3 - x_1)/2$.

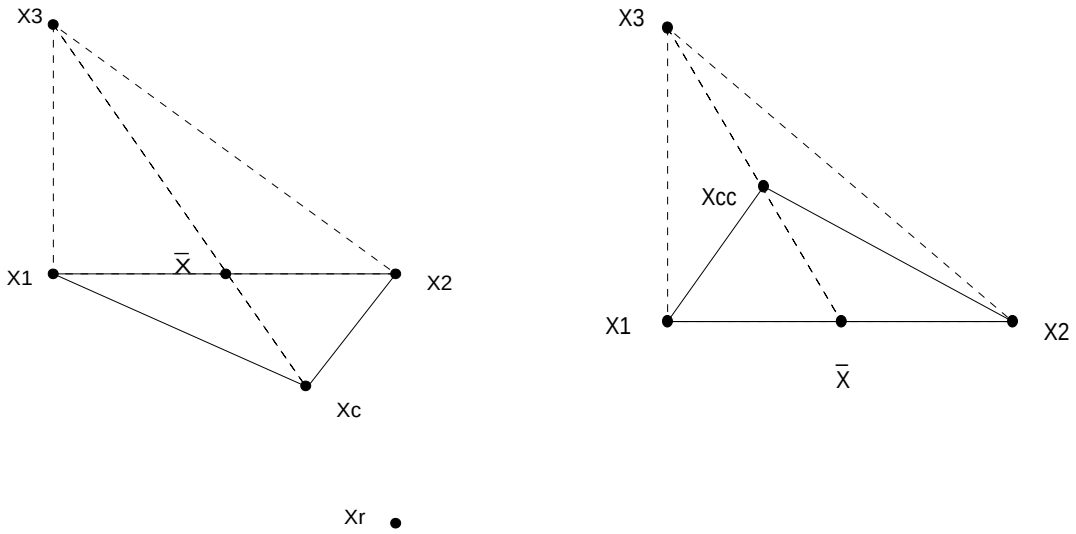
f) Evaluar la función en los nuevos vértices y ordenarlos nuevamente según los valores obtenidos.

Resumiendo, la etapa **a)** expande el triángulo; la **b)** refleja el peor vértice; los pasos **c)** y **d)** reducen la superficie del triángulo modificando tan sólo el peor vértice y el paso **e)** reduce el área contrayendo los dos peores vértices.



(a) (b)

Figura 1: Transformación de los símplices en dimensión 2. (a) *tras una reflexión*, (b) *tras una expansión.*



(a) (b)

Figura 2: Transformación de los símplices en dimensión 2. (a) *tras una contracción hacia afuera*, (b) *tras una contracción hacia adentro.*

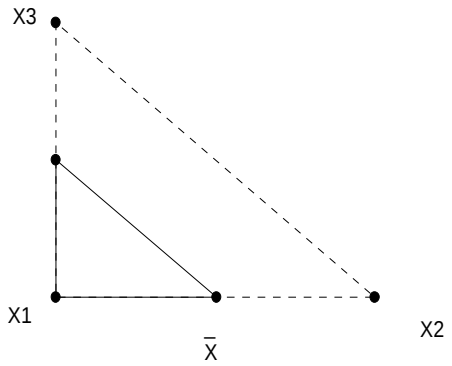


Figura 3: Transformación de los símplices en dimensión 2 después de un shrink.

Si bien el algoritmo Nelder-Mead no garantiza la convergencia en el caso general, puesto que puede estancarse en un punto no óptimo, en la práctica este método presenta unas buenas propiedades de convergencia para dimensiones bajas.

C - Búsqueda multidireccional

Una de las maneras de corregir el posible mal condicionamiento en el algoritmo de Nelder-Mead es exigir que el número de condición de los símplices sea acotado. Los métodos de búsqueda multidireccional aseguran esto haciendo cada símplex congruente con el anterior. De manera análoga al método de Nelder Mead, se eligen parámetros de expansión y contracción (valores usuales son $\nu_e = 2$, $\nu_c = 0.5$) . Dado un simplex inicial S, un algoritmo de búsqueda multidireccional es el siguiente:

1) Se ordenan y etiquetan los vértices de S como

$$x_1, x_2, \ldots, x_{n+1}$$

de tal manera que

$$f_1 \leq f_2 \leq \ldots \leq f_{n+1}$$

donde

$$f_i = f\left(x_i\right)$$

2) Se establece un contador de evaluaciones de función, FCONT=n+1.

3) Mientras $f(x_{n+1}) - f(x_1) > \tau$,

a) Reflexión: Si FCONT=MÁXIMO, entonces ACABAR; en otro caso:

Para $j = 2,...,n+1: r_j = x_1 - (x_j - x_1)$; Calcular $f(r_j)$; FCONT=FCONT+1.

Si $f(x_1) > \min_j \{ f(r_j) \}$ ir a 3b), si no, ir a 3c).

b) Expansión:

i. Para $j = 2,...,n+1: e_j = x_1 - \nu_e (x_j - x_1)$; Calcular $f(e_j)$;

FCONT=FCONT+1.

ii. Si $\min_j \{ f(r_j) \} > \min_j \{ f(e_j) \}$ entonces

Para $j = 2,...,n+1: x_j = e_j$

en otro caso

Para $j = 2,...,n+1: x_j = r_j$

iii. Ir a 3d).

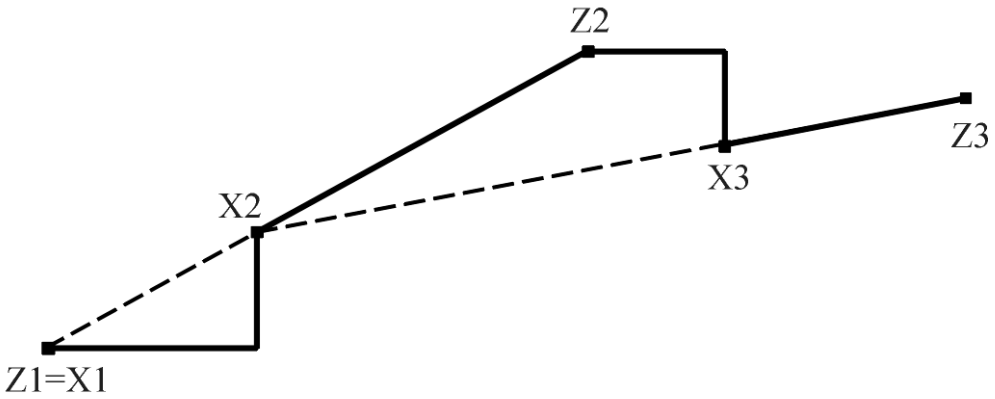
c) Contracción: $j = 2,...,n+1: x_j = x_1 + \nu_c (x_j - x_1)$; Calcular $f(x_j)$.

d) Ordenación: se ordenan los vértices de S de manera que se tenga: $f_1 \leq f_2 \leq ... \leq f_{n+1}$

D - Método de Hooke-Jeeves

El método de Hooke-Jeeves realiza dos tipos de búsqueda: una búsqueda exploratoria y una búsqueda de patrones. El procedimiento iterativo es el siguiente:

Dado un punto inicial x_1, se define $z_1 = x_1$. La búsqueda exploratoria a lo largo de las direcciones coordenadas produce un nuevo punto x_2. Después, la búsqueda de patrones a lo largo de la dirección $x_2 - x_1$ lleva a un nuevo punto z_2. Otra búsqueda exploratoria empezando en z_2 origina el punto x_3. La siguiente búsqueda de patrones se realiza a lo largo de la dirección $x_3 - x_2$, produciendo el nuevo punto z_3. El proceso se repite hasta la convergencia, reduciendo si es necesario el tamaño del patrón.



Un resumen del método es el siguiente:

1. **Paso de inicialización:** Sean $d_1,...,d_n$ las direcciones coordenadas. Sea $\varepsilon > 0$, un escalar que se usará para terminar el algoritmo. Además se elige un tamaño de paso $\Delta > \varepsilon$ y un factor de aceleración $\alpha > 0$. Se elige un punto inicial x_1 y se fija $y_1 = z_1 = x_1$, $k = j = 1$. Se va a la iteración general.

2. Iteración k:

i) Si $f(y_j + \Delta d_j) < f(y_j) : y_{j+1} = y_j + \Delta d_j$; ir a paso ii). En caso contrario:

- si $f(y_j - \Delta d_j) < f(y_j) : y_{j+1} = y_j - \Delta d_j$; ir a paso ii)
- si $f(y_j - \Delta d_j) \geq f(y_j) : y_{j+1} = y_j$; ir a paso ii)

ii) Si $j < n$, reemplazar j por $j+1$ y repetir paso i). En otro caso:

- si $f(y_{n+1}) < f(x_k)$ ir a paso iii)
- $f(y_{n+1}) \geq f(x_k)$ ir a paso iv)

iii) Sea $x_{k+1} = y_{n+1}$, y sea $y_1 = z_{k+1} = x_{k+1} + \alpha(x_{k+1} - x_k)$. Se reemplaza k por $k+1$, se hace $j = 1$ y se vuelve al paso i).

iv) Si $\Delta \leq \varepsilon$, PARAR; x_k es la solución. En otro caso, se reemplaza Δ por $\frac{\Delta}{2}$. Se hace $y_1 = z_{k+1} = x_k$ y $x_{k+1} = x_k$ y Se reemplaza k por $k+1$, se hace $j = 1$ y se repite el paso i).

Bibliografía básica:

- C. T. KELLEY : Iterative methods for optimization. SIAM, 1999.