C19

OPTIMIZACION Y AJUSTE DE PARAMETROS MEDIANTE EL METODO SIMPLEX (Nelder-Mead)

José Luis Martínez González, Universidad de Valladolid
jlumartinez@hotmail.com

Resumen

EcosimPro es un excelente lenguaje de simulación con un gran potencial y versatilidad. Aprovechando su potencial se ha implementado el método Simplex no-lineal, usando EL, para optimizar cualquier tipo de función objetivo. Se mostrarán varios ejemplos: simulación de un reactor químico discontinuo (batch) donde se optimiza la temperatura de refrigerante para minimizar la concentración de reactivos o el tiempo de operación. Pero el ejemplo quizá más útil para los usuarios de Ecosim es la simulación de una columna de desorción. En esta se dispone de datos experimentales obtenidos en un planta piloto. El método simplex se usa en este caso para ajustar los parámetros del modelo.

Este último caso es de gran utilidad ya que permite usar cualquier simulación realizada con Ecosim para ajustar los parámetros y así poder validar el modelo dinámico.

Palabras claves: optimización; ajuste de parámetros; método simplex; restricciones; discretización; diferencias finitas; EDP

1 INTRODUCCION

El método Simplex no lineal (Nelder- Mead) es un método heurístico de búsqueda de mínimos de cualquier función N-dimensional. Para ello a partir de un punto inicial estimativo , el método busca en el hiperespacio paramétrico aquellos valores de estos que minimizan la función objetivo. Se basa en fundamentos geométricos; a partir de la estimación inicial, y conocidos el numero de parámetros a optimizar, N, el algoritmo construye el poliedro mas sencillo en ese hiperespacio paramétrico: el poliedro posee N+1 vértices donde se evalúa la función objetivo y se decide que nuevos valores de los parámetros se ajustarán mejor al objetivo prefijado. Por ejemplo: para ajustar dos parámetros , el algoritmo construye un triángulo , en cuyos vértices se evalúa la función objetivo y según sus valores el poliedro se va moviendo y deformando para buscar el óptimo.

Los métodos Simplex son muy efectivos, especialmente con un gran número de parámetros. Para un problema con dos parámetros a ajustar (ver figura inferior), un Simplex normal es un triángulo equilátero, que empezando desde los valor estimados $b_i^{(0)}$, es llevado al punto de la superficie de F (función objetivo) en el espacio de los parámetros. Durante la búsqueda , F es evaluado en cada uno de los tres vértices, y el triángulo es reflejado justamente al lado opuesto del mayor valor de F. Si no hay una reducción en F después de la reflexión, entonces el Simplex se reduce de tamaño para permitir una localización más precisa del mínimo.

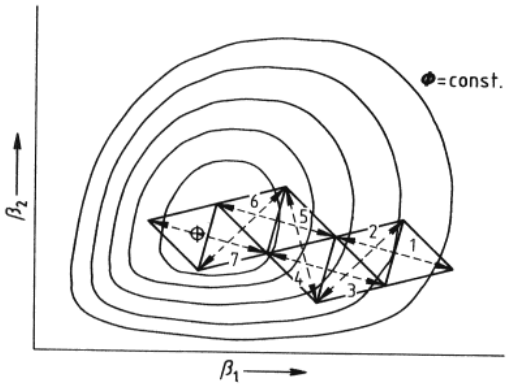


Figura: Representación del Método Simplex. Los números representan el numero de reflexión

Simplex con geometría variable se obtienen con una expansión después de una reflexión satisfactoria ,o con una contracción tras una reflexión desacertada. Un ejemplo viene dado en la siguiente figura para un ajuste biparamétrico. El algoritmo de **reflexión** es:

$$b_{ir} = \left(1 + \gamma_r\right) b_{ic} - \gamma_r\, b_{imax}$$

donde b_{ir} , $b_{i\ max}$, y b_{ic} son las coordenadas del punto reflejado, de el punto con la mayor suma de desviaciones al cuadrado (peor punto), y del punto medio del lado opuesto a este respectivamente (centroide); γ_r es un factor de reflexión.

Centroide: $$b_{ic} = \frac{1}{N} \sum_{\substack{j=1 \\ j \neq imax}}^{N+1} b_j$$

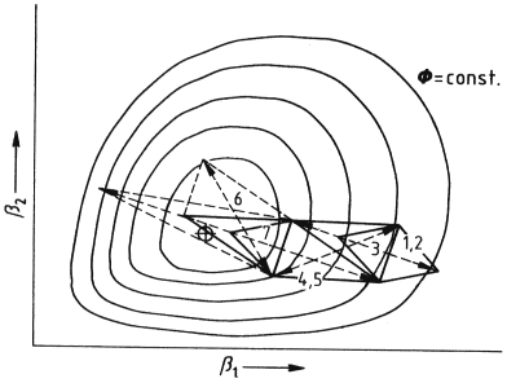


Figura: Representación del Método Simplex con geometría variable. Los números representan los números de reflexión, expansión o contracción

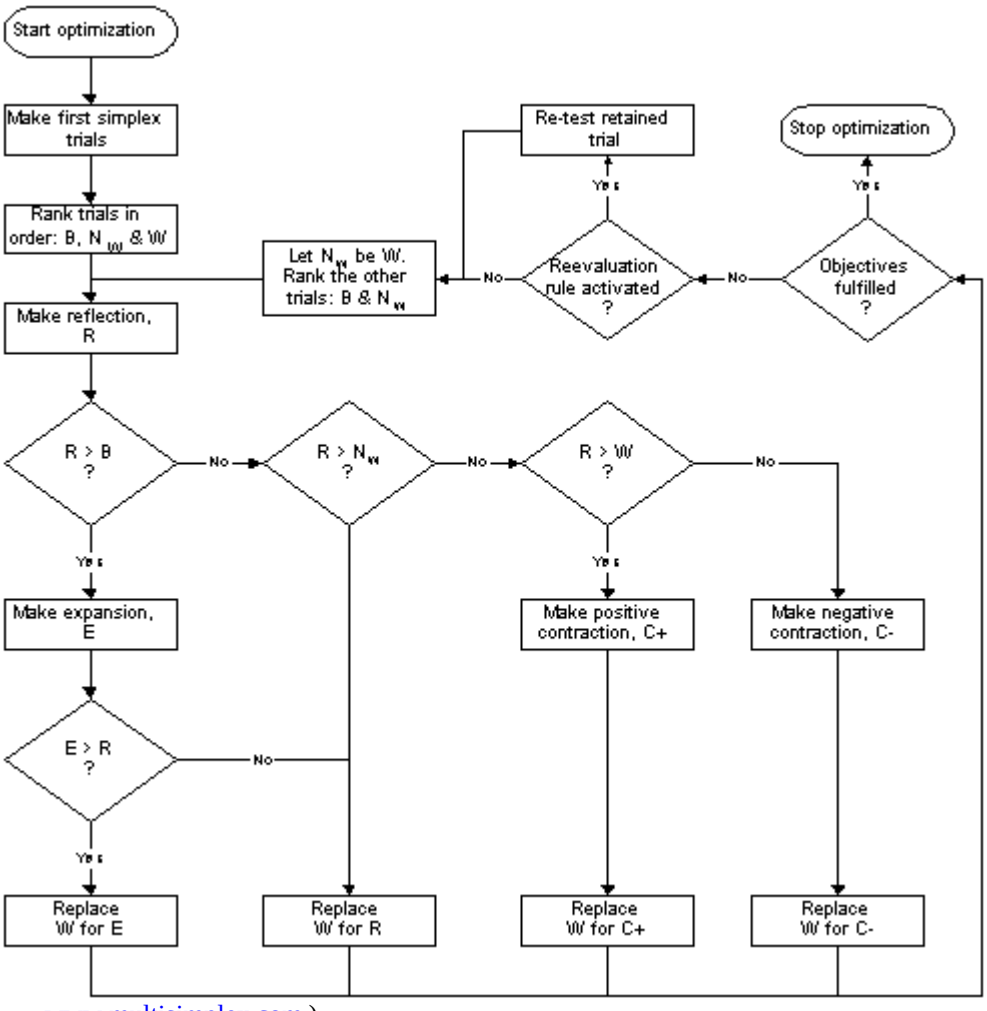
$$b_{ie} = \gamma_e \, b_{ir} + \left(1 - \gamma_e\right) b_{ic}$$

donde b_{ie} son las coordenadas del punto expandido y γ_e es un factor de expansión. Después de una expansión o reflexión insatisfactoria, la **contracción** se consigue con el siguiente algoritmo:

$$b_{ik} = \left(1 - \gamma_k\right) b_{ic} + \gamma_k \, b_{imax}$$

donde b_{ik} y γ_k tienen significado análogo a los usados en reflexión o expansión.

Figura: Algoritmo Simplex (Fuente

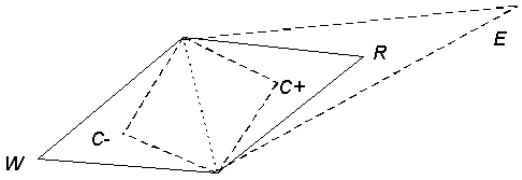


www.multisimplex.com)

El algoritmo de búsqueda de los nuevos puntos se resume en la figura previa (donde esta expresado como algoritmo de maximización, y

El algoritmo de **expansión** es:

no de minimización como el implementado[1] !!); B corresponde al punto con mejor valor de la función objetivo (best), W al punto con peor (worst), N_w el segundo peor punto. R reflexion; E, expansión, C+ y C-, contraccion positiva y negativa respectivamente, y $\overline{C}$ el centroide.



$$R = \overline{C} + \alpha(\overline{C} - W)$$
$$E = \overline{C} + \gamma(\overline{C} - W)$$
$$C+ = \overline{C} + \beta^{+}(\overline{C} - W)$$
$$C- = \overline{C} - \beta^{-}(\overline{C} - W)$$

Ventajas del método Simplex:

- Es un método heurístico. Se basa en consideraciones geométricas y no requiere el uso de derivadas de la función objetivo.
- Es de gran eficacia incluso para ajustar gran numero de parámetros.
- Se puede usar con funciones objetivo muy sinuosas pues en las primeras iteraciones busca el mínimo mas ampliamente y evita caer en mínimos locales fácilmente.
- Es fácil de implementar y usar, y sin embargo tiene una alta eficacia.

Desventajas del método simplex:

- Converge mas lentamente que otros métodos pues requiere mayor numero de iteraciones.

2 OPTIMIZACION Y USO DEL ALGORITMO

El algoritmo se ha programado en forma de un experimento de EcosimPro. Desde el experimento se llama a la simulación realizada y optimiza los parámetros indicados.

La función de datos de salida de la simulación debe llamarse " y " (REAL y) y los parámetros de ajuste deberán formar un array de valores reales llamado tau (REAL tau[npara]) donde "npara" debe tomar un valor determinado y entero (pej. tres parámetros a optimizar....)

2.1.- Simulación de un reactor químico discontinuo (batch) y optimización de parámetros de operación

Se ha simulado un reactor exotérmico. Estableciendo los balances de materia y energía, estos quedan:

$$\frac{dc}{dt} = -r(c,T)$$
$$\frac{dT}{dt} = J \cdot r(c,T) - f_k \cdot (T - Tr(t))$$
$$r(c,T) = A \cdot e^{-\frac{E}{T+273.15}} \cdot c^{n}$$

donde los factores "J " y " f_k " representan los coeficientes que dan cuenta de la generación de calor por reacción y la transmisión de calor entre el reactor y el refrigerante respectivamente.

En este ejemplo el parámetro a optimizar es la *temperatura del refrigerante*. Se buscará el valor de esta temperatura para que:

A) Se minimice la concentración final de los reactivos tras un tiempo fijado

B) Se minimice el tiempo de operación para alcanzar una conversión de reactivos determinada

Se han impuesto una restricciones[2] a la optimización:

1- La temperatura del refrigerante (restricción física externa al sistema) debe ser mayor de 310 K y menor de 335 K

2- La temperatura del reactor no debe salir en ningún momento de la simulación del intervalo 309 - 351 K (restricciones interna al sistema)

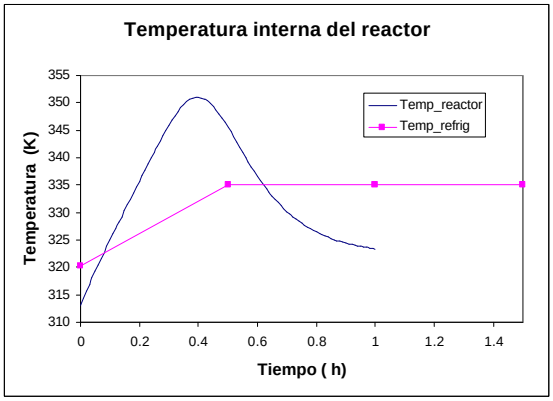
En los primeros instantes de la reacción (C_o = 6; T_o = 313) se genera una gran cantidad de

[1] El algoritmo es el mismo pero cambia la manera de seleccionar cual es el mejor y peor punto, dado que en el minimización el mejor punto es el de menor función objetivo y en maximización es el de mayor función objetivo.

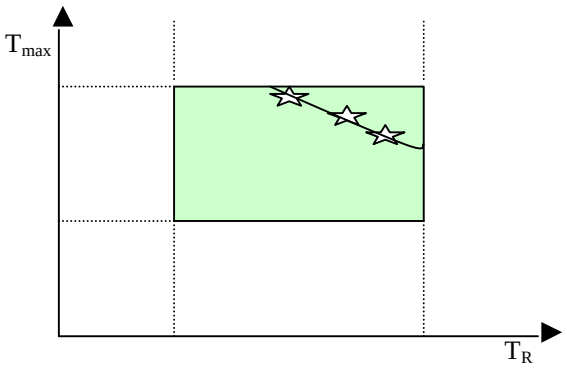
[2] Las restricciones se han implementado como barreras infinitas de la función objetivo

calor debido a que la concentración de reactivos es muy alta y por tanto la cinética es muy rápida. A medida que estos se consumen la reacción se va ralentizando y el reactor adquiere temperaturas menores. Por tanto es necesario regular la temperatura del refrigerante para que no se superen las restricciones impuestas y a la vez esta refrigeración sea capaz de aportar el máximo calor posible para así acelerar la velocidad de reacción.

Se recalculó la temperatura de refrigerante optima cada intervalo de 0.5 h. Los resultados se muestran a continuación:



Como se observa, la temperatura del reactor que se obtiene ante la sucesión optima de temperaturas del refrigerante que encuentra el algoritmo nunca sobrepasa la restricción de temperatura máxima del reactor (351 K). El algoritmo es capaz de encontrar una sucesión de valores de la temperatura del refrigerante que dé una solución factible para las restricciones impuestas.



2.2 .- Ajuste de parámetros de una columna de desorción de fluidos supercríticos

La planta piloto opera en los laboratorios del departamento de Ingeniería Química de la Universidad de Valladolid. Su objetivo es la desorción de sustancias contaminantes que impregnan suelos expuestos a agentes químicos peligrosos (regeneración de suelos). Para ello se introduce un flujo de CO_2 supercrítico que actúa de disolvente de estas sustancias. Las desorbe de las partículas de suelo arenoso y las arrastra fuera del sistema.

El modelo que representa este sistema es un modelo en EDP. Para un sistema isotérmico se demuestra que los balances de materia a lo largo de toda la columna cargada con de suelo contaminado son:

Balance al contaminante en la fase supercrítica

$$e\frac{\partial c}{\partial t}+\frac{u}{L}\frac{\partial c}{\partial z}=-k_g\cdot a\cdot(c-c^*)$$

Cond. inicial (t=0) C = 0
Cond. contorno en z =0 $C=C_{entrada}=0$

Balance al contaminante en la fase sólida

$$(1-e)\frac{\partial c_s}{\partial t}=k_g\cdot a\cdot(c-c^*)$$

Cond inicial (t = 0) $C_s = C_{so}$

Equilibrio de desorción en la intefase

$$c^*=h\cdot c_s$$

La ecuación en derivadas parciales se discretizó usando el método de diferencias finitas (con nueve puntos internos de discretización → C_1 .....C_9):

$$\frac{\partial c}{\partial z}\approx\frac{c_i-c_{i-1}}{\Delta z}$$

Este es un método muy eficaz y de gran sencillez de implementar usando Ecosimpro:

```
COMPONENT col_desorcion (INTEGER n=9)

DATA
        --Ajustar: h & kg
        REAL tau[2]= { 0.15 ,0.0015}
        .....
        ...

DECLS

        REAL  cliq[n+1]
        REAL  csol[n+1]
        REAL  y
        ...

INIT
```

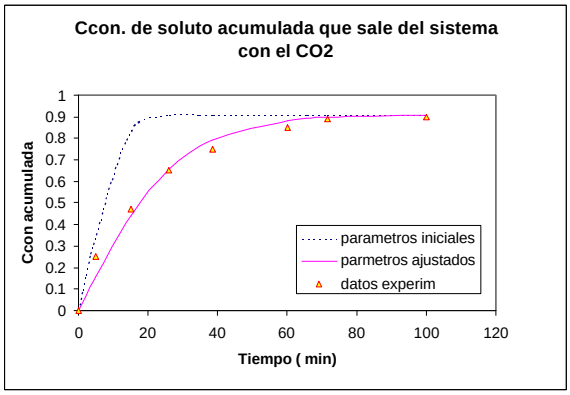
```
        FOR (i IN 2,n+1)
               cliq[i]=cliq_ini
               csol[i]=csol_ini
        END FOR

        y=0

CONTINUOUS

deltaz=1/(n+1)

cliq[1]=c_entrada -- Cond contorno z=0
csol[1]=csol[2]   -- aproximacion ...

--Discretizacion por diferencias
--finitas de los balances de materia

EXPAND_BLOCK ( i IN 2,n+1)

e*cliq[i]'+u/L*(cliq[i]-cliq[i-1])
/deltaz= - tau[2]*av*(cliq[i]-
tau[1]*csol[i])   -- Ccon en el liquido

(1-e)*csol[i]'=tau[2]*av*(cliq[i]-
tau[1]*csol[i])   -- Ccon el el solido

END EXPAND_BLOCK

-- Acumulacion de cliq a la salida

y'=cliq[n+1]

END COMPONENT
```

El ajuste de parámetros en este caso se redujo a sólo dos: el coeficiente de trasferencia de materia (k_g) y el equilibrio de solubilidad (h). No obstante el método programado esta diseñado para poder usarse con cualquier función multiparamétrica sin mayores problemas

En la siguiente gráfica se muestran los datos experimentales , la curva inicial de ajuste (valor inicial del método de optimización) , y la curva final que se obtiene para la concentración de contaminante acumulado que abandona la columna de desorción junto con el CO_2 supercrítico.

Valores de los parámetros :

h = 0.1149
k_g = 0.00152 min^{-1}



Datos experimentales:

Tiempo (min)	Ccon acum.
0	0
5	0.25
15	0.47
26	0.65
38.5	0.75
60	0.85
71.5	0.89
100	0.9

3 CONCLUSIONES

Se ha demostrado la validez y fiabilidad del algoritmo implementado. El código puede ser usado fácilmente por cualquier usuario de Ecosimpro, dado que solo requiere la implementación de la función objetivo a optimizar en cada caso concreto, y además esta escrito en EL con lo que no es necesario ningún conocimiento de lenguajes de programación externos. El algoritmo está escrito para trabajar con cualquier numero de parámetros y su inserción dentro de una simulación de Ecosim es inmediata.

Agradecimientos

A Pedro Cobas por toda la ayuda prestada para aprender Ecosim; y especialmente a Cesar de Prada por todo lo que me ha enseñado en el campo de la simulación y control de procesos.

Referencias

Nelder, J. A., Mead, R. A simplex method for function minimization. Computer Journal 7(1965) 308-313.

Åberg, E. R., Gustavsson, A. G. T. Design and evaluation of modified simplex methods. Analytica Chimica Acta 144(1982) 39-53.

Betteridge, D., Wade, A. P., Howard, A. G. Reflections on the modified simplex - II. Talanta 32(1985):8B 723-734.

Himmenblau, D. M. & Bischoff, K.B. Análisis y Simulacion de Procesos. Ed. Reverté

Nonlinear Analysis in Chemical Engineering. Finlayson, B.A. Ed. Prentice-Hall
URL: http://www.multisimplex.com

ANEXO: Código del método simplex con EL

En cada caso particular hay que adecuar la función objetivo a los requisitos de cada problema. Pero el resto del algoritmo es totalmente válido. Debe definirse el numero de parámetros a optimizar (INTEGER npara =&&). Se debe definir un punto inicial de búsqueda para los parámetros (REAL xo[&&]={...,....,..., ,....,...}, donde "&&" es el numero concreto de parámetros a optimizar. El tamaño de todas las matrices y vectores debe definirse en la sección declarativa del experimento, por tanto sus dimensiones deben darse con números concretos (sin hacer referencia a valores como "npara =4, pej.", sino directamente el numero de parámetros (pej. una matriz de dimensiones N+1 quedará declarada como REAL valores[4+1,4+1])

Otras constantes deben darse para cada optimización en concreto (iteraciones máximas, condiciones de ruptura del algoritmo y factores de escala ....) .A continuación se muestra el código del simplex aplicado al ajuste de los parámetros de la simulación de la columna de desorción:

```
USE MATH
EXPERIMENT exp1 ON col_desorcion.simplex

-- IMPLEMENTACION DEL METODO SIMPLEX PARA MINIMIZACION de errores
--Calculo de parametros

DECLS

       REAL peor                "Valor de la F.O. en  el  peor punto"
       REAL peor2               "Valor de la F.O. en el 2º peor punto"
       REAL mejor               "Valor de la F.O. en  el  mejor punto"
       INTEGER pw               "Posicion del peor punto en la matriz de puntos"
       INTEGER pb               "Posicion del mejor punto en la matriz de puntos"
       INTEGER pb_prev          "Posicion del mejor en la iteracion anterior"
       INTEGER pw_prev          "Posicion del peor  en la iteracion anterior"
       INTEGER p       = 1
       REAL    obj     = 0.
       REAL obj_prev   = 0.
       INTEGER iter=0
       REAL suma_valores=0
       REAL tfinal
       REAL variac
       REAL mejora_obj  =100.
       REAL mejora_para =100.
       REAL para1
       REAL para2
       REAL para11
       REAL para22

                    -- VALORES A AJUSTAR EN CADA PROBLEMA:

       INTEGER ndatos = 8  " Numero de datos"
       INTEGER npara  = 2

       REAL
datos[8,2]={{0,0},{5,0.25},{15,0.47},{26,0.65},{38.5,0.75},{60,0.85},{71.5,0.89},{100,0.
90}}
       --datos experimentales: variacion de "y" con el tiempo:{ti, yi}
       REAL  xo[2]= {0.25,0.015} --Valores inicial de los parametros

       REAL y_ini= 0 --Condicion inicial para la funcion objetivo "y"
```

```
        -- Para estimar los primeros puntos del simplex inicial
        REAL     f_escala= 1         REAL    alfa    = 1.
        REAL    beta_pos= 0.5
        REAL    beta_neg=-0.5
        REAL    gamma   = 2.
        INTEGER iter_max=6*3
        REAL cond_ruptura_obj=1.
        REAL cond_ruptura_para=1.
        REAL tolerancia_rup=0.001

--Matrices y vectores:ajustar dimens.
        REAL valores[2+1,2+1]
        REAL valores_prev[2+1,2+1]
        REAL pref[2+1]
        REAL pcont[2+1]
        REAL pext[2+1]
        REAL centroide[2+1]

-------------------------------


INIT    -- set initial values FOR variables
         -- Dynamic variables
   y = y_ini

BOUNDS    -- set expressions FOR boundary variables: v = f(t,...)



BODY
   REPORT_TABLE("reportAll", " * ")
   TIME = 0
   TSTOP = 1.001*datos[ndatos,1]
   tfinal=TSTOP
   CINT = TSTOP
-------------------------------------------------------------------------------
--La matriz " valores " almacena el valor de los n+1 puntos  =j
--En la columna inicial(i=1) guarda el valor de la funcion obj. pasa ese
--punto=n+1 dimensiones
-------------------------------------------------------------------------------

--Construccion de la matriz de parametros y valores de la funcion objetivo:

FOR (j IN 2,npara+1)
   FOR (i IN 2,npara+1)
      IF (j==2) THEN  -- Pto. inicial xo
         valores[1,i]=xo[i-1]
      END IF

      IF (i==j) THEN         --   Nuevos ptos. iniciales
         valores[j,i]=xo[i-1]+(sqrt(npara+1.)+npara-1)/(npara*sqrt(2.))*f_escala*xo[i-1]
      ELSE
         valores[j,i]=xo[i-1]+(sqrt(npara+1.)-1)/(npara*sqrt(2.))*f_escala*xo[i-1]
      END IF
   END FOR
END FOR

FOR (j IN 1,npara+1)
        --Evaluacion de la Funcion objetivo:
        FOR (k IN 2,npara+1)
           tau[k-1]=valores[j,k]
          --Asignacion de valor de param.
        END FOR

        obj=0.
        p=1
        TIME =0
       y = y_ini

        WHILE(  p <=ndatos )
               INTEG_TO(datos[p,1],CINT)
               obj=obj + (datos[p,2]-y)**2
               p= p+1
        END WHILE
           --  F.O en los nuevos puntos
       valores[j,1]=obj
```

```
END FOR

-----------------------------------------------------------
variac= 2*tolerancia_rup

WHILE(  (iter<iter_max OR (variac > tolerancia_rup OR  (mejora_obj >cond_ruptura_obj
         AND (mejora_para > cond_ruptura_para )))) )

  FOR (i IN 1,npara+1)
       FOR(j IN 1,npara+1)
              valores_prev[i,j]=valores[i,j]
       END FOR
  END FOR
--------------------------------------
               --Elegir el mejor,el peor y el segundo peor punto de "valores"

   mejor=min(valores[1,1],valores[2,1] )   -- Menor valor de la F.O.
   peor=max(valores[1,1],valores[2,1] )   -- Peor valor de la F.O.
   FOR( i IN 2,npara )
        mejor=min(mejor,valores[i+1,1] )
        peor =max(peor,valores[i+1,1] )
   END FOR

   peor2=mejor

   pw=2
   pb=1
   pb_prev = pb
   pw_prev = pw

   FOR (i IN 1,npara+1)
         IF(abs(mejor-valores[i,1])<1e-20) THEN
            pb =i    -- Guarda la posicion del mejor punto
         END IF

         IF (abs(peor - valores[i,1])<1e-20) THEN
             pw =i  --Guarda la posicion (fila) del peor pto.
         ELSE
             peor2=max(peor2,valores[i,1] )  --- Segundo peor punto
         END IF
   END FOR

   PRINT("Función objetivo = $mejor")

   ---------------------------------------
   --Calculo del centroide-----------------

   FOR ( i IN 2,npara+1 )
        suma_valores=0
        FOR (j IN 1,npara+1)
               IF ( j != pw) THEN
                      suma_valores=suma_valores+valores[j,i]
               END IF
        END FOR
        centroide[i]=1/npara*suma_valores
   END FOR


   --Calculo del nuevo punto nuevo (reflexion) a partir del centroide y del peor
   FOR (i IN 2,npara+1)
       pref[i]=valores[pw,i]+(1+alfa)*(centroide[i]-valores[pw,i])
   END FOR
   ----Evaluacion de la Funcion objetivo:
   FOR (k IN 2,npara+1)
       tau[k-1]=pref[k] --Asignacion de valor de parametros
   END FOR

   obj=0.
   p=1
   TIME =0
   y = y_ini

   WHILE(  p <=ndatos )
        INTEG_TO(datos[p,1],CINT)
        obj=obj + (datos[p,2]-y)**2
        p= p+1
```

```
 END WHILE
 ---------
 pref[1]=obj

 ---------------------------------------

IF (pref[1] < mejor) THEN

 ---------Extension
  FOR (i IN 2,npara+1)
    pext[i]=valores[pw,i]+(1+gamma)*(centroide[i]-valores[pw,i])
  END FOR
      --Evaluacion de la Funcion objetivo:
     FOR (k IN 2,npara+1)
        tau[k-1]=pext[k] --Asignacion de valor de parametros
     END FOR

     obj=0.
     p=1
     TIME =0
     y = y_ini
     WHILE(  p <=ndatos )
        INTEG_TO(datos[p,1],CINT)
        obj=obj + (datos[p,2]-y)**2
        p= p+1
     END WHILE
     ---------
     pext[1]=obj

      ------------------
     IF (pext[1] < pref[1]) THEN
       FOR (i IN 1,npara+1)
                     valores[pw,i]=pext[i]
       END FOR
        PRINT("Extensión")
     ELSE
       FOR (i IN 1,npara+1)
                    valores[pw,i]=pref[i]
       END FOR
        PRINT("Reflexión")
     END IF   -- Final del IF de pext[1] < pref[1]
ELSE
 -- Se compara el punto reflejado con el segundo peor punto
  IF (pref[1]<peor2) THEN
    FOR ( i IN 1,npara+1)
        valores[pw,i]=pref[i]
    END FOR
    PRINT("Reflexión")
  ELSE
             -- Se compara el punto reflejado con el peor punto
    IF (pref[1] < peor) THEN
              ---------Contraccion positiva
              FOR ( i IN 2,npara+1)
              pcont[i]=valores[pw,i]+(1+beta_pos)*(centroide[i]-valores[pw,i])
              END FOR
              ----Evaluacion de la Funcion objetivo:
             FOR (k IN 2,npara+1)
                  tau[k-1]=pcont[k] --Asignacion de valor de parametros
             END FOR

             obj=0.
             p=1
            TIME =0
             y = y_ini

            WHILE(  p <=ndatos )
                 INTEG_TO(datos[p,1],CINT)
                 obj=obj + (datos[p,2]-y)**2
                p= p+1
             END WHILE
             ---------
             pcont[1]=obj
            ---------
            FOR (i IN 1,npara+1)
               valores[pw,i]=pcont[i]
            END FOR
            PRINT("Contracción positiva")
```

```
      ELSE
              --Contraccion negativa
             FOR (i IN 2,npara+1)
               pcont[i]=valores[pw,i]+(1+beta_neg)*(centroide[i]-valores[pw,i])
             END FOR
              --Evaluacion de la Funcion objetivo:
             FOR (k IN 2,npara+1)
                 tau[k-1]=pcont[k] --Asignacion de valor de parametros
             END FOR
             obj=0.
             p=1
             TIME =0
            y = y_ini
             WHILE(  p <=ndatos )

            INTEG_TO(datos[p,1],CINT)
            obj=obj + (datos[p,2]-y)**2
            p= p+1
          END WHILE
          ---------
          pcont[1]=obj

         --------------------
          FOR (i IN 1,npara+1)
               valores[pw,i]=pcont[i]
          END FOR
          PRINT("Contracción negativa")
      END IF  -- Final del IF de pref[1] < peor

   END IF   -- Final del IF de pref[1] < peor2

  END  IF    -- Final del IF  de pref[1] < mejor

  iter=iter+1

   -- Condiciones de ruptura del bucle while ....

  IF (iter ==1) THEN
         mejora_obj =2*cond_ruptura_obj
         mejora_para=2*cond_ruptura_para
         variac=2*tolerancia_rup
         obj_prev=mejor

  ELSE
         IF abs(mejor)>1e-20 THEN         -- Evita divisiones por cero
           mejora_obj=(100*(obj_prev-mejor)/mejor)
         ELSE
               mejora_obj= 2*cond_ruptura_obj  --(100*(obj_prev-mejor))
         END IF
          mejora_para=0.
          variac=0.
         FOR ( i IN 2,npara+1)
            IF abs(valores[pw,i])>1e-20 THEN            -- Evita divisiones por cero
                   mejora_para=mejora_para+abs(100*(valores[pw,i]-  \
                                       valores_prev[pw_prev,i])/valores[pw,i])
            ELSE
                   mejora_para=2*cond_ruptura_paramejora_para+abs(100*(valores[pw,i]- \
                                          valores_prev[pw_prev,i]))
            END IF
            variac=variac+abs(abs(valores[pb,i])-abs(valores[pw,i]))
         END FOR
         mejora_para=(1/npara)*mejora_para
--       variac=(1/npara)*variac
         obj_prev=mejor
  END IF

       --IMPRIME RESULTADOS FINALES
  PRINT("Función objetivo = $mejor ")

  para1=valores[pb,2]
  para11=valores[pb,3]

  para2=valores_prev[pw,2]
  para22=valores_prev[pw,3]

  PRINT("Iteración = $iter")
  PRINT("Porcentaje Mejora F.O. = $mejora_obj")
```

```
  PRINT("Porcentaje Mejora parametros = $mejora_para")

  PRINT("Punto mejor: $para1 , $para11    F.O =  $mejor  ")
  PRINT("Punto peor : $para2 , $para22     F.O =  $peor  ")


END WHILE

END EXPERIMENT
```