C19

PARAMETER OPTIMISATION AND ADJUSTMENT BY THE SIMPLEX METHOD (Nelder-Mead)

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

Abstract

EcosimPro is an excellent simulation tool with great potential and versatility. Taking advantage of its potential, the EL language was used to implement the non-linear simplex method in order to optimise any type of objective function. Some examples will be given: the simulation of a discontinuous chemical reactor (batch) in which the coolant temperature is optimised to minimise the concentration of reagents or the operating time. But perhaps the most useful example for EcosimPro users is the simulation of a desorption column. Experimental data retrieved from a pilot plant was used in this simulation. In this case, the simplex method was used to adjust the parameters of the model.

The latter case is very useful because any simulation carried out with EcosimPro can be used to adjust the parameters and therefore the dynamic model can be validated.

Key words: optimisation; parameter adjustment; simplex method; restrictions; discretisation; finite differences; PDS

1 INTRODUCTION

The non-linear simplex method (Nelder-Mead) is a heuristic method of searching for the minimums of any N-dimensional function. Starting from a preliminary approximate point, the method searches the parametric hyperspace for values which minimise the objective function. It works on a geometrical basis; starting with the preliminary estimate and a known number of parameters to be optimised, N, the algorithm constructs the simplest polyhedron in the parametric hyperspace: the polyhedron has N+1 vertices where the objective function is evaluated and it is decided which new values of the parameters will better adjust to the pre-established objective. For example, to adjust two parameters the algorithm constructs a triangle on the vertices of which the objective function is evaluated and, based on the values, the polyhedron moves and distorts as it searches for the optimum.

The simplex methods are very effective, especially with a large number of parameters. For a problem with two parameters to be adjusted (see the figure below), a normal simplex is an equilateral triangle which, starting with the estimated values $b_i^{(0)}$, is carried to the point of the F (objective function) surface in the space of the parameters. During the search, F is evaluated in each of the three vertices, and the triangle is reflected on exactly the opposite side to the greatest value of F. If after the reflection there is no reduction in F, then the size of the simplex reduces to allow a more precise location of the minimum.

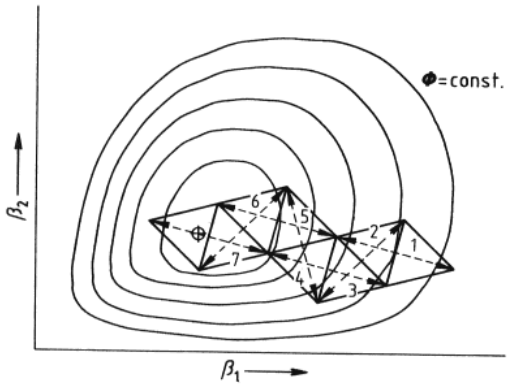


Figure: Representation of the Simplex Method. The numbers represent the reflection numbers

Simplex methods with variable geometry are obtained with expansion after satisfactory reflection, or with contraction after erroneous reflection. An example of a biparametric adjustment is shown in the figure below. The **reflection** algorithm is:

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

where b_{ir}, $b_{i\,max}$, and b_{ic} are the coordinates of the reflected point, of the point with the largest sum of deviations to the square (worst point), and of the centre point of the side opposite to this one, respectively (centroid); γ_r is a reflection factor.

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

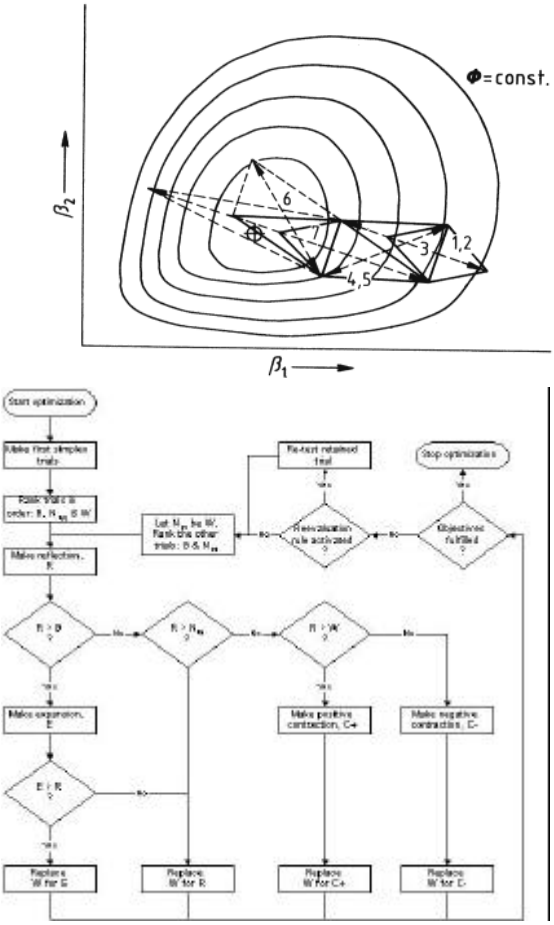


Figure: Representation of the Simplex Method with Variable Geometry. The numbers represent the reflection, expansion or contraction numbers

Figure: Simplex Algorithm (source www.multisimplex.com)

The **expansion** algorithm is:

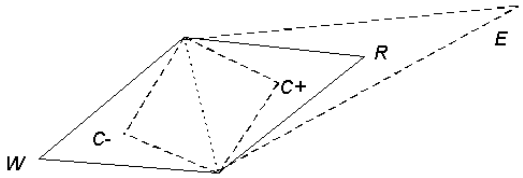
$$b_{ie} = \gamma_e \, b_{ir} + \left(1 - \gamma_e\right) b_{ic}$$

where b_{ie} are the coordinates of the expanded point and γ_e is an expansion factor. After an unsatisfactory expansion or reflection, the **contraction** is obtained with the following algorithm:

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

where b_{ik} and γ_k have meanings which are analogous to those used in reflection or expansion.

The search algorithm for new points is summarised in the previous figure (where it is expressed as a maximisation algorithm and not as a minimisation algorithm like the one that is implemented[1] !!); B corresponds to the point with the best objective function value, W to the point with the worst objective function value, N_w the second worst point. R reflection; E expansion, C+ and C- positive and negative contraction, respectively, and $\overline{C}$ the centroid.



$$R = \overline{C} + \alpha(\overline{C} - W)$$

$$E = \overline{C} + \gamma(\overline{C} - W)$$

$$C+ = \overline{C} + \beta^{+}(\overline{C} - W)$$

$$C- = \overline{C} - \beta^{-}(\overline{C} - W)$$

Advantages of the simplex method:

- It is a heuristic method. It works on a geometrical basis and does not need to use derivatives of the objective function.
- It is extremely efficient, even for adjusting a large number of parameters.
- It can be used with very sinuous objective functions, so in the first iterations it makes a wider search for the minimum and easily avoids the location of local minimums.
- Easy to implement and easy to use, it is nonetheless highly efficient.

[1] It is the same algorithm, but the way to select which is the best and worst point changes because in the minimisation the best point is that of the smallest objective function and in the maximisation it is the greatest objective function.

Disadvantages of the simplex method:

- It converges more slowly than other methods because it requires a greater number of iterations.

2 OPTIMISATION AND USE OF THE ALGORITHM

The algorithm has been programmed in the form of an EcosimPro experiment. The simulation that has been carried out is called from the experiment and the parameters indicated are optimised.

The simulation data output function must be called "y" (REAL y) and the adjustment parameters must form an array of real values called tau (REAL tau[npara]), where "npara" must take a determined whole number (for example, three parameters to be optimised, etc).

2.1 SIMULATION OF A DISCONTINUOUS CHEMICAL REACTOR (BATCH) AND OPTIMISATION OF OPERATING PARAMETERS

An exothermic reactor has been simulated. Establishing the balances of heat and energy, we obtain:

$$\frac{dc}{dt} = -r(c,T)$$

$$\frac{dT}{dt} = J \cdot r(c,T) - f_k \cdot (T - Tr(t))$$

$$r(c,T) = A \cdot e^{-\frac{E}{T+273.15}} \cdot c^n$$

where the factors "J" and "f_k" represent the coefficients which take into account the generation of heat by reaction and the transmission of heat between the reactor and the coolant, respectively.

In this example the parameter to be optimised is the *temperature of the coolant*. A search will be made for the value of this temperature so that:

A. The final concentration of the reagents is minimised after a fixed time

B. The operating time is minimised to attain a determined conversion of reagents

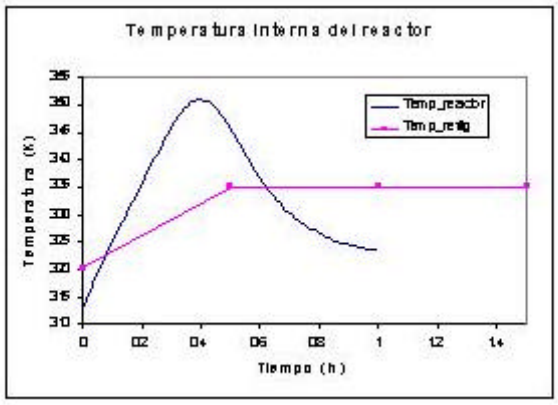
Some restrictions[2] have been imposed on the optimisation:

1. The temperature of the coolant (physical restriction external to the system) must be greater than 310 K and less than 335 K

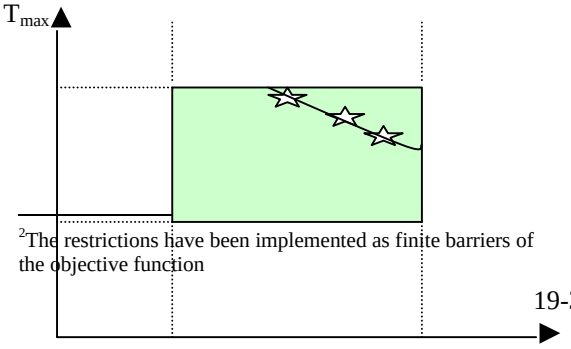
2. The temperature of the reactor must at no time go beyond the simulation of the interval 309 – 351 K (restrictions internal to the system)

During the first instants of the reaction (C_o = 6; T_o = 313) a great amount of heat is generated because the concentration of reagents is very high and therefore the kinetics is very fast. As these are consumed, so the reaction slows down and the reactor reaches lower temperatures. It is therefore necessary to regulate the coolant temperature so that the imposed restrictions are not exceeded and, at the same time, the cooling operation is capable of contributing as much heat as possible and thus accelerate the speed of reaction.

The optimum coolant temperature was recalculated at intervals of 0.5 h. The results are shown in the following figure:



As we can see, the reactor temperature obtained in the light of the optimum sequence of coolant temperatures found by the algorithm never exceeds the maximum temperature restriction of the reactor (351 K). The algorithm is capable of finding a series of coolant temperature values to offer a feasible solution to the restrictions imposed.



[2]The restrictions have been implemented as finite barriers of the objective function

T_R

2.2 ADJUSTMENT OF PARAMETERS OF A DESORPTION COLUMN OF SUPERCRITICAL FLUIDS

The pilot plant operates in the laboratories of the Chemical Engineering Department of the University of Valladolid. It is designed to desorb the pollutant substances that impregnate soil exposed to hazardous chemical agents (regeneration of soil). To this end, a flow of supercritical CO_2 is introduced, which acts as a solvent of these substances. It desorbs them from the sandy particles and carries them out of the system.

The model representing this system is a model in PDS. For an isothermal system it can be shown that the balances of matter throughout the column loaded with the polluted soil are:

Balance of pollutant in the supercritical phase

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

Prelim. cond. (t=0) C = 0
Boundary cond. in z =0 $C=C_{entrada}=0$

Balance of pollutant in the solid phase

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

Initial condition (t = 0) $C_s = C_{so}$

Desorption equilibrium at the inter-phase

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

The equation in partial derivatives was discretised using the method of finite differences (with new internal points of discretisation → C_1 .....C_9):

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

This is a very efficient method which is very easy to implement with the use of EcosimPro:

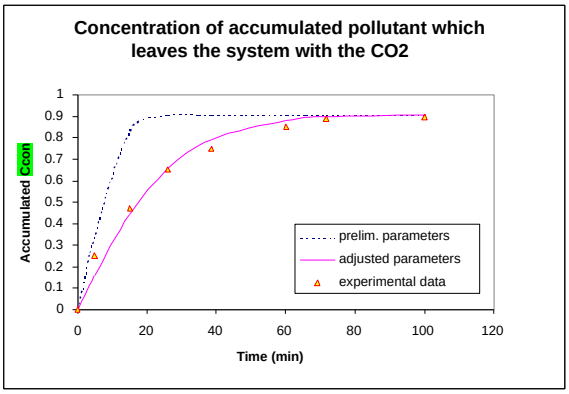
```
COMPONENT col_desorcion (INTEGER n=9)

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
```

The adjustment of parameters in this case was reduced to only two: the coefficient of transfer of matter (k_g) and the equilibrium of solubility (h). However, the method programmed is designed for use with any multiparametric function without undue difficulty.

The following graph shows the experimental data, the preliminary adjustment curve (preliminary value of the optimisation method), and the final curve obtained for the concentration of accumulated pollutant that leaves the desorption column together with the supercritical CO_2.

Values of the parameters:

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



Experimental data:

Time (min)	Accumulated Concentration
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 CONCLUSIONS

We have demonstrated the validity and reliability of the algorithm which was implemented. The code can easily be used by any EcosimPro user, because all that is required is the implementation of the objective function to be optimised in each particular case, and what's more it is written in EL so no knowledge of external programming languages is necessary. The algorithm is written so that it will work with any number of parameters and it can be immediately inserted into an EcosimPro simulation.

Acknowledgements

I wish to thank Pedro Cobas for all the help he gave me when I was learning about EcosimPro; and special thanks to César de Prada for all that he has taught me in the field of simulation and process control.

References

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

ATTACHMENT: Simplex method code written in EL

In each particular case, the objective function has to be adjusted to the requirements of each problem. But the rest of the algorithm is completely valid. The number of parameters to be optimised must be defined (INTEGER npara =&&). A starting point to search for the parameters must be defined (REAL xo[&&]={...,...,..., ,...,...}, where "&&" is the actual number of parameters to be optimised. The size of all the matrices and vectors must be defined in the declaration section of the experiment, and therefore their dimensions must be given with concrete numbers without making reference to values like "npara =4, pej.", but directly indicating the number of parameters (for example, a matrix with the dimensions N+1 will be declared as REAL values[4+1,4+1])

Other constants must be given for each particular optimisation (maximum iterations, rupture conditions of the algorithm and scale factors, etc). The following is the code of a simplex method applied to the adjustment of the parameters in the simulation of the desorption column:

```
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
```