Subrutinas para la resolución de ecuaciones de una variable

Pablo Santamaría

v0.3 (Noviembre 2011)

1. Introducción

En general, las raíces de una ecuación no lineal $f(x) = 0$ no pueden ser obtenidas por fórmulas explícitas cerradas, con lo que no es posible obtenerlas en forma exacta. De este modo, para resolver la ecuación nos vemos obligados a obtener soluciones aproximadas a través de algún método numérico.

Estos métodos son *iterativos*, esto es, a partir de una o más aproximaciones iniciales para la raíz, generan una sucesión de aproximaciones $x_0, x_1, x_2, \ldots$ que esperamos convergan al valor de la raíz α buscada. El proceso iterativo se continúa hasta que la aproximación se encuentra próxima a la raíz dentro de una tolerancia $\epsilon > 0$ preestablecida. Como la raíz no es conocida, dicha proximidad, medida por el error absoluto $|x_{n+1} - \alpha|$, no puede ser computada. Sin un conocimiento adicional de la función $f(x)$ o su raíz, el mejor criterio para detener las iteraciones consiste en proceder hasta que la desigualdad

$$\frac{|x_{n+1} - x_n|}{|x_{n+1}|} < \epsilon$$

se satisfaga, dado que esta condición *estima* en cada paso el error relativo. Ahora bien, puede ocurrir en ciertas circunstancias que la desigualdad anterior nunca se satisfaga, ya sea por que la sucesión de aproximaciones diverge o bien que la tolerancia escogida no es razonable. En tal caso el método iterativo no se detiene nunca. Para evitar este problema consideramos además un número máximo de iteraciones a realizarse. Si este número es excedido entonces el problema debe ser analizado con más cuidado.

¿Cómo se escoge un valor correcto para las aproximaciones iniciales requeridas por los métodos? No existe una respuesta general para esta cuestión. Para el método de bisección es suficiente conocer un intervalo que contenga la raíz, pero para el método de Newton, por ejemplo, la aproximación tiene que estar suficientemente cercana a la raíz para que funcione [1]. En cualquier caso primeras aproximaciones iniciales para las raíces pueden ser obtenidas graficando la función $f(x)$.

En las siguientes secciones presentamos implementaciones de los métodos numéricos usuales como subrutinas FORTRAN. Con el fin de proporcionar subrutinas de propósito general, las mismas tienen entre sus argumentos a la función $f(x)$ involucrada, la cual pude ser entonces implementada por el usuario como un subprograma `FUNCTION` con el nombre que quiera, pero siempre debe asegurarse de declararla como `EXTERNAL` en el programa que realiza la llamada a la subrutina. Otros argumentos que requieren estas subrutinas son los valores para las aproximaciones iniciales que necesite el método, una tolerancia para la aproximación final de la raíz y un número máximo de iteraciones. La raíz aproximada es devuelta en otro de los argumentos. Dado que el método puede fallar utilizamos una variable entera como clave de error para advertir al programa principal. Si dicha clave es igual a cero, entonces el método funcionó correctamente y el valor devuelto es la raíz aproximada dentro de la tolerancia preescrita. Si la clave de error es, en cambio, distinta

[1] De hecho, efectuando algunas iteraciones del método de bisección podemos obtener una buena aproximación para iniciar el método de Newton.

de cero, entonces ocurrió un error. La naturaleza del error dependerá del método, pero un error posible en todos ellos es que el número máximo de iteraciones fue alcanzado.

Espero que este apunte les resulte útil, y como siempre, cualquier duda o sugerencia envíenla por *e-mail* a `pablo@fcaglp.unlp.edu.ar`.

2. Método de bisección

El método de bisección comienza con un intervalo $[a, b]$ que contiene a la raíz. Entonces se computa el punto medio $x_0 = (a + b)/2$ del mismo y se determina en cual de los dos subintervalos $[a, x_0]$ o $[x_0, b]$ se encuentra la raíz analizando el cambio de signo de $f(x)$ en los extremos. El procedimiento se vuelve a repetir con el nuevo intervalo así determinado.

Es claro que la raíz es acotada en cada paso por el intervalo así generado y que una estimación del error cometido en aproximar la raíz por el punto medio de dicho intervalo es igual a la mitad de la longitud del mismo. Esta estimación es utilizada, en la siguiente implementación del método, como criterio de paro para la sucesión de aproximaciones.

```
      SUBROUTINE BISECCION(F,XA,XB,N,TOL,RAIZ,ICLAVE)
*     ------------------------------------------------
*     METODO DE BISECCION para encontrar una solución
*     de f(x)=0 dada la función continua f en el intervalo
*     [a,b] donde f(a) y f(b) tienen signos opuestos.
*     ------------------------------------------------
*     Bloque de identificación de argumentos
*     ------------------------------------------------
*     F      = Función que define la ecuación    (arg entrada)
*     XA,XB  = Extremos del intervalo            (arg entrada)
*     N      = Número máximo de iteraciones      (arg entrada)
*              Número de iteraciones realizadas  (arg salida )
*     TOL    = Tolerancia para el error absoluto (arg entrada)
*     RAIZ   = Aproximación de la solución       (arg salida )
*     ICLAVE = Clave de éxito                    (arg salida )
*              0  = éxito
*             -1  = no se puede proceder (f(x) de
*                   igual signo en a y b)
*              1  = iteraciones excedidas
*     ------------------------------------------------
*     Bloque de declaración de tipo
*     ------------------------------------------------
      IMPLICIT NONE
      INTEGER N,ICLAVE
      DOUBLE PRECISION F,XA,XB,TOL,RAIZ
*     ------------------------------------------------
      INTEGER I
      DOUBLE PRECISION ERROR,A,B
*     ------------------------------------------------
*     Bloque de procesamiento
*     ------------------------------------------------
      A = XA
      B = XB
      ICLAVE = 0
      IF (F(A)*F(B).GT.0.0D0) THEN
          ICLAVE = -1
          RETURN
      ENDIF
      DO I=1,N
```

```
            ERROR = (B-A)*0.5D0
            RAIZ  = A + ERROR
            IF (ERROR.LT.TOL) THEN
                N = I
                RETURN
            ENDIF
            IF (F(A)*F(RAIZ).GT.0.0D0) THEN
                A = RAIZ
            ELSE
                B = RAIZ
            ENDIF
        ENDDO
        ICLAVE = 1
        RETURN
        END
*       ------------------------------------------------
```

3. Método *regula falsi*

El método de *regula falsi* (o método de posición falsa), procede en forma semejante al método de bisección analizando el cambio de signo de la función, pero en lugar de tomar el punto medio del intervalo, considera como siguiente aproximación la abscisa del punto de intersección de la recta que pasa por los dos puntos $(x_n, f(x_n)), (x_{n-1}, f(x_n - 1))$ obtenidos en las aproximaciones anteriores, con el eje x. Este método, al igual que bisección, siempre converge (en tanto la función $f(x)$ sea continua), pero distinto a éste, falla en dar un intervalo pequeño en el cual se encuentre contenida la raíz.

```
        SUBROUTINE REGULA_FALSI(F,XA,XB,N,TOL,RAIZ,ICLAVE)
*       ------------------------------------------------
*       METODO DE POSICION FALSA (REGULA FALSI) (o método
*       de las cuerdas) para encontrar una solución de
*       f(x)=0 dada la función continua f en el intervalo
*       [a,b] donde f(a) y f(b) tienen signos opuestos
*       ------------------------------------------------
*       Bloque de identificación de argumentos
*       ------------------------------------------------
*       F      = Función que define la ecuación   (arg entrada)
*       XA,XB  = Extremos del intervalo           (arg entrada)
*       N      = Número máximo de iteraciones     (arg entrada)
*                Número de iteraciones realizadas (arg salida )
*       TOL    = Tolerancia para el error relativo(arg entrada)
*       RAIZ   = Aproximación de la solución      (arg salida )
*       ICLAVE = Clave de éxito                   (arg salida )
*                0 = éxito
*               -1 = no se puede proceder (f(x) de
*                    igual signo en a y b)
*                0 = iteraciones excedidas
*       ------------------------------------------------
*       Bloque de declaración de tipo
*       ------------------------------------------------
        IMPLICIT NONE
        INTEGER N,ICLAVE
        DOUBLE PRECISION F,XA,XB,TOL,RAIZ
*       ------------------------------------------------
        INTEGER I
        DOUBLE PRECISION A, B, FA, FB
        DOUBLE PRECISION FR, RAIZ0
```

```
*       ------------------------------------------------
*       Bloque de procesamiento
*       ------------------------------------------------
        ICLAVE = 0
        A      = XA
        B      = XB
        FA     = F(A)
        FB     = F(B)
        IF (FA*FB.GT.0D0) THEN
            ICLAVE = -1
            RETURN
        ENDIF
        RAIZ0 = 0D+0
        DO I=1,N
            RAIZ  = A - FA*(B-A)/(FB-FA)
            IF (ABS((RAIZ-RAIZ0)/RAIZ).LT.TOL) THEN
               N = I
               RETURN
            ENDIF
            FR = F(RAIZ)
            IF (FA*FR.GT.0.0D0) THEN
                A   = RAIZ
                FA  = FR
            ELSE
                B   = RAIZ
                FB  = FR
            ENDIF
            RAIZ0 = RAIZ
        ENDDO
        ICLAVE = 1
        RETURN
        END
*       ------------------------------------------------
```

4. Método de Newton–Raphson

El método de Newton comienza con una aproximación inicial x_0 dada, a partir de la cual se genera la sucesión de aproximaciones x_1, x_2, ..., siendo x_{n+1} la abscisa del punto de intersección del eje x con la recta tangente a $f(x)$ que pasa por el punto $(x_n, f(x_n))$. Esto conduce a la fórmula de iteración

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad n = 1, 2, \ldots$$

Nuestra implementación de la subrutina correspondiente requiere que se pase también como argumento no sólo la funcion $f(x)$, sino también su derivada $f'(x)$, la cual debe ser implementada por el usuario, al igual que $f(x)$, como una `FUNCTION`.

```
        SUBROUTINE NEWTON(F,DF,XX0,N,TOL,RAIZ,ICLAVE)
*       ------------------------------------------------
*       METODO DE NEWTON-RAPHSON para encontrar una
*       solución de f(x)=0 dada la función derivable
*       f y una aproximación inicial x0.
*       ------------------------------------------------
*       Bloque de identificación de argumentos
*       ------------------------------------------------
*       F       = Función que define la ecuación  (arg entrada)
*       DF      = Función derivada de F           (arg entrada)
```

```
*       XX0     = Aproximación inicial            (arg entrada)
*       N       = Número máximo de iteraciones    (arg entrada)
*                 Número de iteraciones realizadas(arg salida )
*       TOL     = Tolerancia del error relativo   (arg entrada)
*       RAIZ    = Estimación de la solución       (arg salida)
*       ICLAVE  = Clave de éxito                  (arg salida)
*                 0 = éxito
*                 1 = iteraciones excedidas
*       ------------------------------------------------
*       Bloque de declaración de tipo
*       ------------------------------------------------
        IMPLICIT NONE
        INTEGER N, ICLAVE
        DOUBLE PRECISION F, DF, XX0, TOL, RAIZ
*       ------------------------------------------------
        INTEGER I
        DOUBLE PRECISION X0
*       ------------------------------------------------
*       Bloque de procesamiento
*       ------------------------------------------------
        ICLAVE = 0
        X0     = XX0
        DO I=1,N
            RAIZ  = X0 - F(X0)/DF(X0)
            IF (ABS((RAIZ-X0)/RAIZ).LT.TOL) THEN
               N = I
               RETURN
            ENDIF
            X0 = RAIZ
        END DO
        ICLAVE = 1
        RETURN
        END
*       ------------------------------------------------
```

5. Método de Newton para ecuaciones algebraicas

En el caso particular en que $f(x)$ es un polinomio, el método de Newton puede ser eficientemente implementado si la evaluación de $f(x_n)$ (y su derivada) es realizada por el método iterativo de Horner. En efecto, supongamos que $f(x)$ es un polinomio de grado m:

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_m x^m,$$

la evaluación de $f(x_n)$ por la regla de Horner procede computando

$$\begin{cases} b_m = a_m \\ b_k = a_k + b_{k+1} x_n \quad k = m-1, \ldots, 0 \end{cases}$$

siendo, entonces

$$b_0 = f(x_n),$$

en tanto que $f'(x_n)$ es computada haciendo

$$\begin{cases} c_m = b_m \\ c_k = b_k + c_{k+1} x_n \quad k = m-1, \ldots, 1 \end{cases}$$

siendo, entonces

$$c_1 = f'(x_n).$$

El método de Newton se reduce así a

$$x_{n+1} = x_n - \frac{b_0}{c_1}$$

El procedimiento resultante se conoce a menudo como método de Birge–Vieta.

Nuestra implementación en la siguiente subrutina pasa los coeficientes del polinomio en un arreglo A de $(M+1)$ componentes, siendo M el grado del polinomio (valor que también es requerido como argumento). Para simplificar el tratamiento de los subíndices, el arreglo es declarado con el índice inferior 0, no 1, de manera que $A(0) = a_0, A(1) = a_1, \ldots, A(M) = a_m$.

```
      SUBROUTINE BIRGE_VIETA(A,M,XX0,TOL,N,RAIZ,ICLAVE)
*     ------------------------------------------------
*     METODO DE BIRGE-VIETA para resolver ECUACIONES
*     ALGEBRAICAS:
*
*                     P (x) = 0
*
*     donde P es un polinomio de grado m de coeficientes
*     reales; basado en el método de Newton-Raphson
*     implementando el esquema de Horner para la evalua-
*     ción del polinomio y su derivada.
*     ------------------------------------------------
*     Bloque de identificación de argumentos
*     ------------------------------------------------
*     A       = Arreglo unidimensional de 0 a M que
*               contiene los coeficientes del polinomio
*                                           (arg entrada)
*     XX0     = Aproximación inicial        (arg entrada)
*     M       = Grado del polinomio         (arg entrada)
*     N       = Número máximo de iteraciones (arg entrada)
*               Número de iteraciones realizadas(arg salida )
*     TOL     = Tolerancia error relativo   (arg entrada)
*     RAIZ    = Estimación de la solución   (arg salida )
*     ICLAVE  = Clave de éxito              (arg salida )
*               0 = éxito
*               1 = iteraciones excedidas
*     ------------------------------------------------
*     Bloque de declaración de tipo
*     ------------------------------------------------
      IMPLICIT NONE
      INTEGER M, N
      DOUBLE PRECISION A(0:M)
      DOUBLE PRECISION XX0,TOL,RAIZ
      INTEGER ICLAVE
*     ------------------------------------------------
      INTEGER I, J
      DOUBLE PRECISION X0,B,C
*     ------------------------------------------------
      ICLAVE = 0
      X0     = XX0
      DO I=1,N
*         ------------------------
*         Esquema de Horner
*         ------------------------
          B = A(M)
```

```
            C = A(M)
            DO J=M-1,1,-1
                B = B*X0+A(J)
                C = C*X0+B
            ENDDO
            B = B*X0+A(0)
*           ------------------------
*           Método de Newton
*           ------------------------
            RAIZ  = X0 - B/C
            IF (ABS((RAIZ-X0)/RAIZ).LT.TOL) THEN
               N = I
               RETURN
            ENDIF
            X0 = RAIZ
         END DO
         ICLAVE = 1
         RETURN
         END
```

6. Método de la secante

El método de la secante procede a partir de *dos* aproximaciones iniciales obteniendo la aproximación x_{n+1} como la abscisa del punto de intersección del eje x con la recta secante que pasa por los puntos $(x_{n-1}, f(x_{n-1}))$ y $(x_n, f(x_n))$. La fórmula de iteración es entonces

$$x_{n+1} = x_n - f(x_n)\frac{(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}, \quad n = 2, 3, \ldots$$

El método de la secante, si bien no converge tan rápido como Newton, tiene la gran ventaja de no requerir la derivada de la función. Eso sí, ahora se necesitan *dos* aproximaciones iniciales para arrancar el método.

```
         SUBROUTINE SECANTE (F,XX0,XX1,N,TOL,RAIZ,ICLAVE)
*        ------------------------------------------------
*        ALGORITMO DE LA SECANTE para encontrar una solución
*        de f(x)=0, siendo f una función continua, dada las
*        aproximaciones iniciales x0 y x1.
*        ------------------------------------------------
*        Bloque de identificación de argumentos
*        ------------------------------------------------
*        F       = Función que define la ecuación  (arg entrada)
*        XX0,XX1 = Aproximaciones iniciales        (arg entrada)
*        N       = Número máximo de iteraciones    (arg entrada)
*                  Número de iteraciones realizadas(arg salida )
*        TOL     = Tolerancia del error relativo   (arg entrada)
*        RAIZ    = Estimación de la solución       (arg salida )
*        ICLAVE  = Clave de éxito                  (arg salida )
*                  0 = éxito
*                  1 = iteraciones excedidas
*        ------------------------------------------------
*        Bloque de declaración de tipo
*        ------------------------------------------------
         IMPLICIT NONE
         INTEGER N,ICLAVE
         DOUBLE PRECISION F,XX0,XX1,TOL,RAIZ
```

```
*       ------------------------------------------------
        INTEGER I
        DOUBLE PRECISION X0, X1, FX0, FX1
*       ------------------------------------------------
*       Bloque de procesamiento
*       ------------------------------------------------
        ICLAVE = 0
        X0     = XX0
        X1     = XX1
        FX0    = F(X0)
        FX1    = F(X1)
        DO I= 2,N
            RAIZ  = X1 - FX1*((X1-X0)/(FX1-FX0))
            IF (ABS((RAIZ-X1)/RAIZ).LT.TOL) THEN
                N = I
                RETURN
            ENDIF
            X0  = X1
            FX0 = FX1
            X1  = RAIZ
            FX1 = F(RAIZ)
        END DO
        ICLAVE = 1
        RETURN
        END
*       ------------------------------------------------
```

7. Método de Müller

El método de Müller empieza con *tres* aproximaciones iniciales x_0, x_1, x_2 y genera la sucesión de aproximaciones $x_3, x_4, \ldots$, siendo x_{n+1} la abscisa del punto de intersección de la parábola que pasa por los tres puntos $(x_n, f(x_n)), (x_{n-1}, f(x_{n-1})), (x_{n-2}, f(x_{n-2}))$. La determinación de x_{n+1} procede entonces resolviendo la ecuación cuadrática correspondiente (utilizando su expresión alternativa que resulta de "racionalizar el numerador", para minimizar los errores de redondeo) y escogiendo de los dos valores posibles el más próximo a x_n. Esto conduce al método iterativo

$$x_{n+1} = x_n - \frac{2c}{b + \text{sign}(b)\sqrt{b^2 - 4ac}}, \quad n = 2, 3, \ldots$$

donde

$$\begin{aligned} c &= f(x_n), \\ b &= \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}} + \frac{f(x_n) - f(x_{n-2})}{x_n - x_{n-2}} - \frac{f(x_{n-1}) - f(x_{n-2})}{x_{n-1} - x_{n-2}}, \\ a &= \frac{\dfrac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}} - \dfrac{f(x_{n-1}) - f(x_{n-2})}{x_{n-1} - x_{n-2}}}{x_n - x_{n-2}}. \end{aligned}$$

Nótese que si, en algún paso, el discriminante de la ecuación cuadrática es negativo entonces la aproximación será un número complejo (gráficamente, esto significa que la parábola no corta el eje x en ninguno de sus puntos). De este modo, si se implementa con aritmética compleja, el método aproximará raices complejas cuando sea apropiado. Sin por otro lado, sólo nos interesa encontrar las raíces reales, es claro que en esta situación necesitamos escoger otro valor para la aproximación x_{n+1}. Este valor, en nuestra implementación, será la ordenada del vértice de la parábola, el cual se obtiene, en la fórmula, tomando el discriminante igual a cero.

```
      SUBROUTINE MULLER(F,XX0,XX1,XX2,N,TOL,RAIZ,ICLAVE)
*     ------------------------------------------------
*     ALGORITMO DE MULLER para encontrar una solución
*     de f(x)=0, siendo f una función continua, dada las
*     aproximaciones iniciales x0, x1 y x2.
*     ------------------------------------------------
*     Bloque de identificación de argumentos
*     ------------------------------------------------
*     F           = Función que define la ecuación   (arg entrada)
*     XX0,XX1,XX2 = Aproximaciones iniciales         (arg entrada)
*     N           = Número máximo de iteraciones     (arg entrada)
*                   Número de iteraciones realizadas (arg salida )
*     TOL         = Tolerancia del error relativo    (arg entrada)
*     RAIZ        = Estimación de la solución        (arg salida )
*     ICLAVE      = Clave de éxito                   (arg salida )
*                   0 = éxito
*                   1 = iteraciones excedidas
*     ------------------------------------------------
*     Bloque de declaración de tipo
*     ------------------------------------------------
      IMPLICIT NONE
      DOUBLE PRECISION F,XX0,XX1,XX2,TOL,RAIZ
      INTEGER N,ICLAVE
*     ------------------------------------------------
      INTEGER I
      DOUBLE PRECISION X0,X1,X2
      DOUBLE PRECISION F0,F1,F2
      DOUBLE PRECISION DIF21,DIF20,DIF10
      DOUBLE PRECISION B,A,DELTA,DISCR
*     ------------------------------------------------
*     Bloque de procesamiento
*     ------------------------------------------------
      ICLAVE = 0
      X0     = XX0
      X1     = XX1
      X2     = XX2
      F0     = F(X0)
      F1     = F(X1)
      F2     = F(X2)
      DO I= 3,N
         DIF21 = (F2 - F1)/(X2-X1)
         DIF20 = (F2 - F0)/(X2-X0)
         DIF10 = (F1 - F0)/(X1-X0)
         A     = (DIF21 - DIF10)/(X2 - X0)
         B     = DIF21 + DIF20 - DIF10
         DISCR = B*B - 4.0D0*A*F2
         IF (DISCR.GT.0.D0) THEN
            DISCR = SQRT(DISCR)
         ELSE
            DISCR = 0.D0
         ENDIF
         IF (B.LT.0.D0) DISCR = - DISCR
         DELTA = -2.0D0*F2/(B+DISCR)
         RAIZ  = X2 + DELTA
         IF (ABS(DELTA).LE.TOL) THEN
             N = I
             RETURN
         ENDIF
```

```
            X0 = X1
            F0 = F1
            X1 = X2
            F1 = F2
            X2 = RAIZ
            F2 = F(RAIZ)
         END DO
         ICLAVE = 1
         RETURN
         END
C        ------------------------------------------------
```

8. Iteración de punto fijo

El método de punto fijo requiere que se reescriba la ecuación $f(x) = 0$ en la forma

$$x = \phi(x)$$

y entonces, a partir de una aproximación inicial x_0, se obtiene la sucesión de aproximaciones $x_1, x_2, ...$ según

$$x_{n+1} = \phi(x_n), \quad n = 1, 2, \ldots$$

En la siguiente implementación, es importante recordar que la función que es pasada por argumento es ahora $\phi(x)$ y no $f(x)$, función que debe ser implementada por el usuario como una `FUNCTION`.

```
         SUBROUTINE PUNTO_FIJO(FI,XX0,N,TOL,RAIZ,ICLAVE)
*        ------------------------------------------------
*        ALGORITMO DE PUNTO FIJO o DE APROXIMACIONES SUCESIVAS
*        para encontrar una solución de x=fi(x) dada una
*        aproximación inicial x0.
*        ------------------------------------------------
*        Bloque de identificación de argumentos
*        ------------------------------------------------
*        FI      = Función que define la ecuación   (arg entrada)
*        XX0     = Aproximación inicial             (arg entrada)
*        N       = Número máximo de iteraciones     (arg entrada)
*                  Número de iteraciones realizadas (arg salida )
*        TOL     = Tolerancia del error relativo    (arg entrada)
*        RAIZ    = Estimación de la solución        (arg salida )
*        ICLAVE  = Clave de éxito                   (arg salida )
*                  0 = éxito
*                  1 = iteraciones excedidas
*        ------------------------------------------------
*        Bloque de declaración de tipo
*        ------------------------------------------------
         IMPLICIT NONE
         INTEGER ICLAVE,N
         DOUBLE PRECISION FI,X0,TOL,RAIZ
*        ------------------------------------------------
*        Bloque de procesamiento
*        ------------------------------------------------
         ICLAVE = 0
         X0     = XX0
         DO I=1,N
             RAIZ  = FI(X0)
             IF (ABS((RAIZ-X0)/RAIZ).LT.TOL) THEN
```

```
              N = I
              RETURN
          ENDIF
          X0 = RAIZ
        END DO
        ICLAVE = 1
        RETURN
        END
*       ------------------------------------------------
```