CAPÍTULO IV

Elementos para la optimización de funciones de distribución

CAPÍTULO IV

ELEMENTOS PARA LA OPTIMIZACIÓN DE FUNCIONES DE DISTRIBUCIÓN

4.1 Introducción a la Programación

La programación ha tenido un desarrollo exponencial como resultado del gran crecimiento tecnológico que se presentó a partir Segunda Guerra Mundial. Los problemas a los que se fue enfrentando la humanidad incrementaron su complejidad y tamaño. La continua competencia por encontrar la solución óptima tomando en cuenta las restricciones de los recursos contribuyó también en el desarrollo de esta disciplina.

El tipo de programación que se utiliza para la resolución de un problema depende de la naturaleza del mismo. Los tipos de programación que existen son:

- Programación Lineal (PL).
- Programación No Lineal (PNL).

La Función de Máxima Verosimilitud que se pretende optimizar en esta tesis es no lineal, por lo que se procederá a trabajar sobre la segunda.

4.1.1 Descripción de la PNL

La PNL se caracteriza porque la función matemática que se pretende optimizar es no lineal (Bazaraa y Shetty, 1943). Se pueden presentar los siguientes casos:

- La función a optimizar es no lineal y las restricciones son lineales.

- Las restricciones son un conjunto de igualdades no lineales y la función objetivo es lineal.

Una función no lineal es aquella en la cual cuando menos una de las variables involucradas en la expresión matemática es no lineal (cuadráticas, polinomiales, exponenciales, etc.) Un ejemplo de una expresión no lineal es el siguiente:

$$x_1^2 \log x_2 + \text{sen}^2 x_2 + x_3^3 + x_4 = 5$$

En general, un problema de PNL se expresa de la siguiente forma (Winston, 1991):

Max (o min) $z=f(x_1,x_2,\ldots,x_n)$

s.a. $g_1(x_1,x_2,\ldots,x_n)(\leq = \geq)b_1$

$g_2(x_1,x_2,\ldots,x_n)(\leq = \geq)b_2$

.

.

$g_m(x_1,x_2,\ldots,x_n)(\leq = \geq)b_m$

Donde: $f(x_1,x_2,\ldots,x_n)$ Función objetivo del PNL.

$g_m(x_1,x_2,\ldots,x_n)(\leq = \geq)b_m$ Restricciones del PNL.

Es importante mencionar que en los problemas de PNL la función objetivo y/o las restricciones tienen que ser ecuaciones no lineales.

Un problema de PNL puede no satisfacer las suposiciones de Proporcionalidad y de Aditividad, como un programa lineal lo hace. La resolución de cualquier problema de programación tiene que encontrarse en la región factible de solución. Se conoce como región factible de solución al conjunto de puntos que satisfacen las m restricciones establecidas. Una solución óptima de un PNL puede ser un punto $\bar{x}$ en la región factible. En un problema de maximización, este punto debe satisfacer la siguiente restricción: $f\left(\bar{x}\right) \geq f\left(x\right)$, tal que $\bar{x} \geq x$; en el caso de minimizar, como se lleva a cabo en cualquier problema de optimización, la restricción se invierte. En un programa de PNL la solución óptima no tiene que ser un punto extremo de la región factible.

Para lograr el objetivo de esta tesis y obtener la solución óptima fue necesario utilizar un algoritmo de búsqueda multidireccional denominado Nelder Mead que a continuación se describe:

4.2 Algoritmo de búsqueda multidireccional "Nelder Mead"

El propósito del algoritmo de búsqueda es minimizar una función no lineal sin restringir, de tal manera que se vaya construyendo una secuencia de mejores vértices para que de esta manera se encuentren valores, cada vez más pequeños, de la función que se esté evaluando.

Para desarrollar este algoritmo es necesario utilizar el método simples (Winston, 1994). El valor de los puntos n tiene que estar dentro de la región factible del problema.

4.2.1 Método de búsqueda.

Kuester, James L. y Mize (1973) en su libro "Optimization Techniques with Fortran" propone una metodología para la utilización del Algoritmo Nelder Mead en el programa computacional desarrollado para obtener la optimización de la función Tina de Baño.

1. Se selecciona un punto inicial $\underline{x}_1$.
2. Se construye el simplex inicial incluyendo el punto $\underline{X}_1$ y los puntos adicionales $\underline{\boldsymbol{x}}_{j,i}$, tal que:

$$x_j = \underline{x}_1 + \underline{\boldsymbol{x}}_{j,i} \qquad (4.1)$$

Donde: j=1,2,....,N+1.

i=1,2,…,N

La siguiente tabla explica la forma en que son establecidos los valores $\underline{\boldsymbol{x}}_{j,i}$:

Tabla 4.1. Ubicación de los valores $\underline{\boldsymbol{x}}_{j,i}$

j \ i	$\underline{\boldsymbol{x}}_{1,j}$	$\underline{\boldsymbol{x}}_{2,j}$	…	$\underline{\boldsymbol{x}}_{N-1,j}$	$\underline{\boldsymbol{x}}_{i,j}$
1	p	q	…	q	q
2	q	p	…	q	q
⋮	⋮	⋮		⋮	⋮
N	q	q	…	p	q
N+1	q	q	…	q	p

(Kuester et al., 1973)

Donde N : Número total de variables.

a : Longitud del simplex.

p : $\frac{a}{N\sqrt{2}}\left(\sqrt{N+1}+N-1\right)$

q : $\frac{a}{N\sqrt{2}}\left(\sqrt{N+1}-1\right)$

3. Una vez que el simplex ha sido formado, el algoritmo comienza calculando los valores de la función en cada uno de los vértices que están formados por los puntos calculados de acuerdo a la ecuación 4.1. Estos puntos son sometidos a evaluación y en el vértice en el cual se obtiene el mejor resultado de la función, se realiza la búsqueda que optimice la función mediante el procedimiento de reflexión. Este método de búsqueda consiste en que los segmentos formados por cada vértice tienden a reflejarse sobre el vértice en el que se obtuvo la mejor evaluación de la función (x_1). Según el Reporte Técnico 90-07 de la Universidad de Rice, Washington; esto puede ser visto como la reflexión del simplex original a través del mejor vértice para originar el nuevo simplex. A continuación se presenta gráficamente en la figura 4.1.

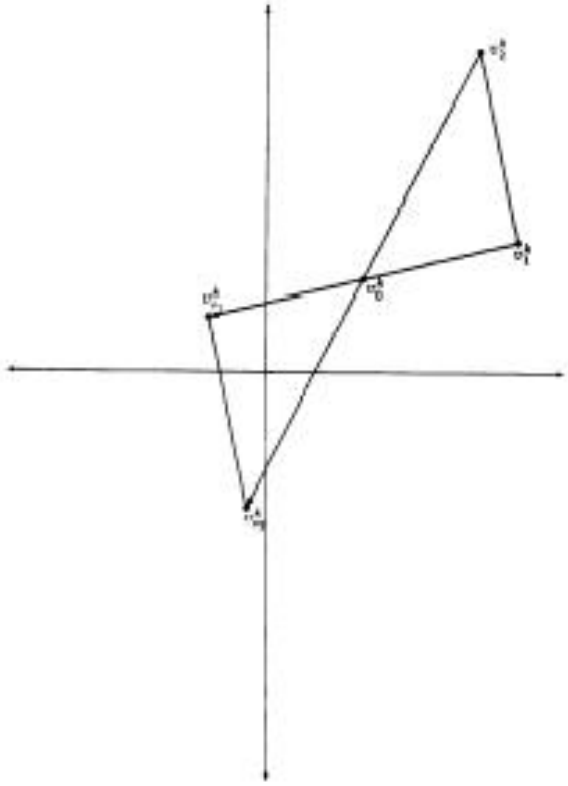

Figura 4.1 El simplex original y su reflexión (Torckson,1990).

Donde: v_0^k : Mejor vértice.

$v_1^k, v_2^k, ..., v_n^k$: Vértices sometidos a evaluación de la función a minimizar.

El simplex formado por los vértices v_1^k, v_2^k,...., v_n^k y evaluado, se refleja en el vértice donde resultó el menor valor de la función, v_0^k, generando un nuevo simplex al reflejar los puntos. Ahí vuelve a ser evaluada la función y en caso de que se

obtenga un mejor resultado, es decir, que el valor de la función en alguno de los nuevos vértices sea mejor que el vértice que se tomó como referencia en la iteración anterior, este punto se toma de base para seguir realizando los distintos procedimientos de búsqueda considerados por este algoritmo. Es importante mencionar que los ángulos y las longitudes de los arcos se conservan al llevarse a cabo la reflexión.

Los puntos reflejados se calculan de la siguiente forma:

$$\overline{x_{i,j}}(reflejado) = \overline{x_{i,c}} + \alpha\left(\overline{x_{i,c}} - x_{i,j}(peor)\right) \qquad \begin{matrix} i=1,2,...,N \\ j=1,2,\ldots N+1 \end{matrix}$$

Donde : α : Constante positiva.

$\overline{x_{i,c}}$: Son los puntos que son tomados como referencia para realizar la búsqueda multidireccional. Estos puntos son calculados con la siguiente ecuación:

$$\overline{x_{i,c}} = \frac{1}{k-1}\left[\sum_{i=1}^{k} x_{i,j} - x_{i,j}(peor)\right], \qquad \begin{matrix} i=1,2,....,N \\ j=1,2,\ldots,N+1 \end{matrix}$$

k=N+1

$x_{i,j}(peor)$: Punto cuyo valor de la función resulta ser el mayor.

4. En caso de que el valor de la función del vértice reflejado sea mayor que el del vértice anterior, se procede a realizar la búsqueda mediante el procedimiento de contracción. El algoritmo simplemente contrae el simplex, evalúa las funciones y compara con el valor de la función que fue tomada como referencia (el mejor vértice). De esta forma se realizará una nueva búsqueda de los valores de los parámetros que minimicen el valor de la función. Gráficamente se puede ver en la figura 4.2.

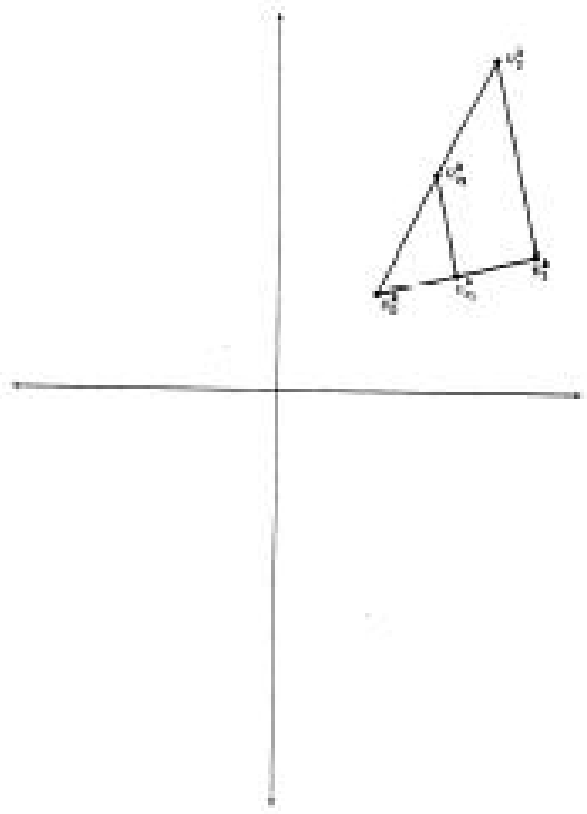

Figura 4.2 El simplex original con su reflexión y contracción. (Torckson, 1990)

Para poder seguir realizando la búsqueda mediante este procedimiento, es necesario que el valor de la función de algunos de los vértices que se originan al contraer el simplex sean menores al valor de la función del vértice que se tomó como referencia.

En este algoritmo, los valores de las funciones de los vértices que son sometidos a la contracción son calculados de la siguiente forma:

$$x_{i,j}(contraído) = \overline{x}_{i,c} - \beta\left(\overline{x}_{i,c} - x_{i,j}(peor)\right) \qquad \begin{matrix} i=1,2,...,N \\ j=1,2,\ldots,N+1 \end{matrix}$$

Donde β toma un valor entre 0 y 1.

En caso de que algún valor de la función correspondiente a la reflexión de algún vértice sea mejor comparado con el resultado obtenido en el peor vértice; el punto contraído es calculado a partir del punto reflejado. Es decir:

$$x_{i,j}(contraído) = \overline{x}_{i,c} - \beta\left(\overline{x}_{i,c} - x_{i,j}(reflejada)\right) \qquad i=1,2,...,N$$

En caso de que al comparar el valor de la función calculada en cada vértice contraído con el vértice que resultó ser tomado como referencia (el mejor evaluado en la iteración anterior) no se optimice la función, la distancia entre el vértice de referencia y los puntos ya contraídos se reducen a la mitad. Esto se puede expresar a través de la siguiente ecuación:

$$x_{i,j}(nuevo) = \frac{\left(x_{i,j}(mejor) + x_{i,j}(viejo)\right)}{2} \qquad \begin{matrix} \text{i=1,2,...,N} \\ \text{j=1,2,…,N+1} \end{matrix}$$

Donde : $x_{i,j}(mejor)$: Vértices cuya longitud de arco fue contraída en la iteración anterior.

$x_{i,j}(viejo)$: Vértice cuyo valor de función resultó ser el menor en la iteración anterior.

5. En caso de que el valor de la función del vértice reflejado sea menor que el del vértice cuya función tenía el mejor valor en la iteración anterior, se procede a realizar la búsqueda mediante el procedimiento de expansión. Los valores de los vértices expandidos son calculados de la siguiente forma:

$$x_{i,j}(\text{exp } andido) = \overline{x}_{i,c} - g\left(x_{i,j}(reflejada) - \overline{x}_{i,c}\right) \qquad \begin{matrix} \text{i=1,2,...,N} \\ \text{j=1,2,…,N+1} \end{matrix}$$

A continuación se muestra gráficamente:

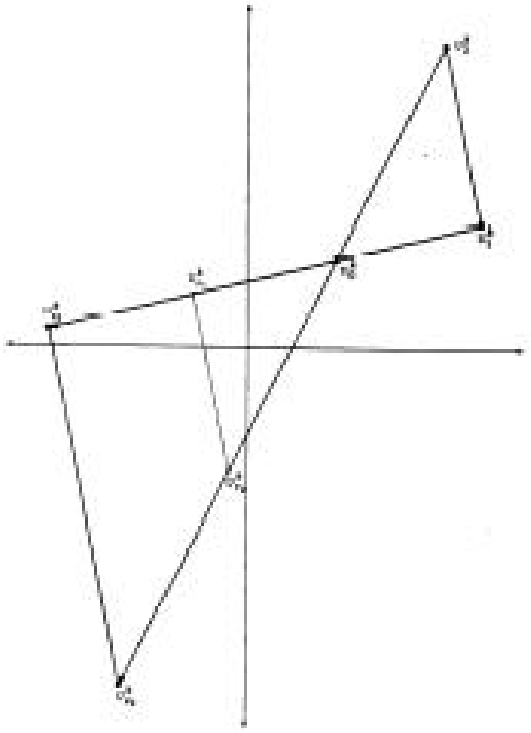

Figura 4.3 El simplex original con su reflexión y expansión. (Torckson, 1990)

El diagrama que se presenta a continuación resume la lógica que sigue el Algoritmo Nelder Mead para optimizar las funciones de distribución que sean programadas.

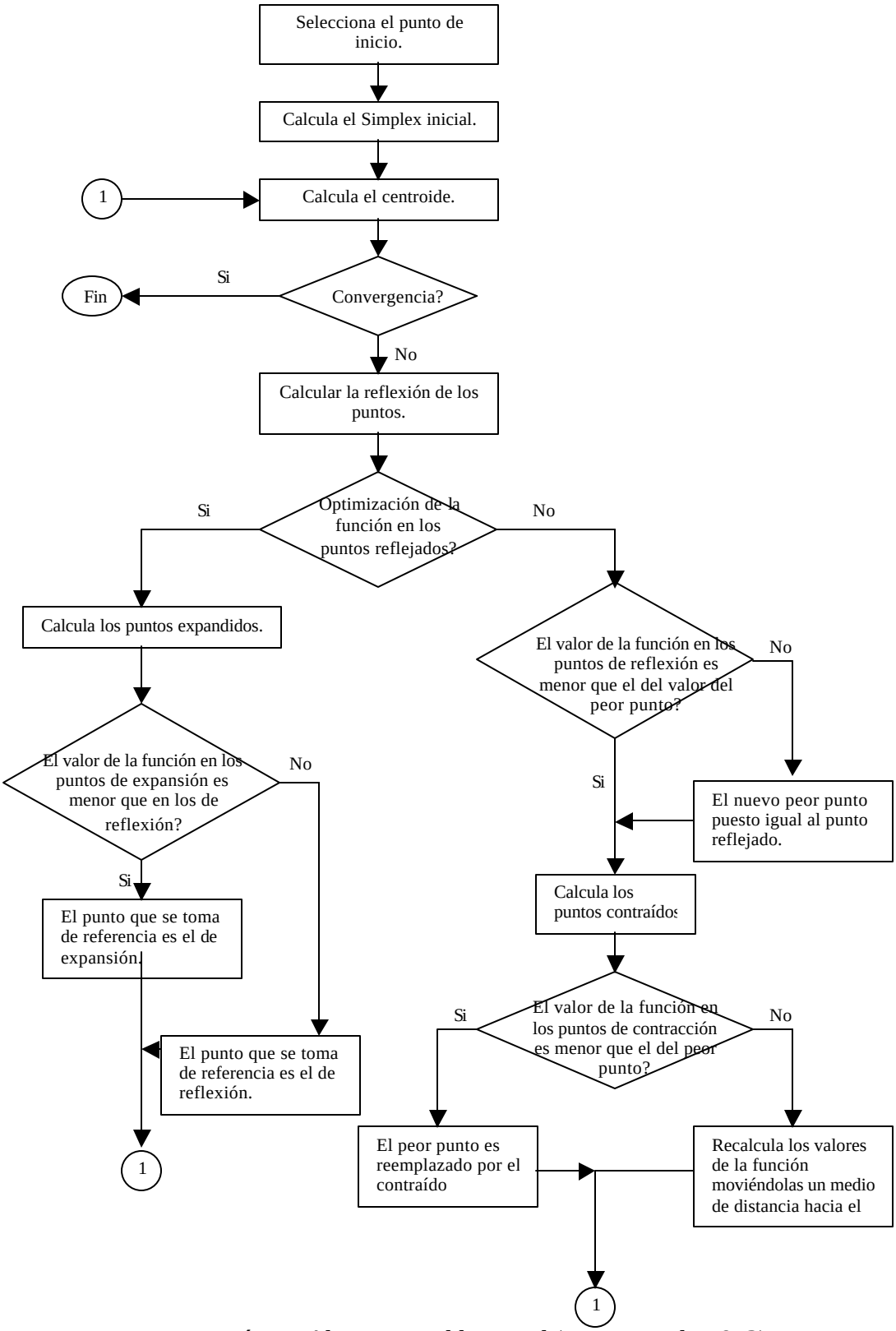

Figura 4.4. Diagrama Lógico. Algoritmo Nelder Mead (Kuester et al., 1973)

El lenguaje que se utilizó para la programación y posteriormente para la simulación de este algoritmo fue Fortran. Por esta razón a continuación se describirán algunos aspectos del programa que cooperaron con la realización de este trabajo.

4.3 Lenguaje de Programación Fortran

El lenguaje estructurado que se utilizará para la realización de los programas correspondientes a esta tesis será Fortran (FORmula TRANslator (McCracken, 1977)); ya que es un lenguaje orientado a la computación matemática, gracias a la precisión que ofrece en sus cálculos.

El Fortran es un lenguaje que tiene algunos elementos básicos comunes a todos los leguajes; tiene un conjunto de caracteres, un pequeño vocabulario en inglés y una sintaxis.

4.3.1 Operaciones Básicas

Los lenguajes de programación pueden realizar tres tipos de operaciones básicas principalmente:

- Operaciones de entrada / salida: Leer e Imprimir.
- Manejo y comparación de datos: Sumar, restar, multiplicar, dividir, negar, asignar y comparar.
- Operaciones de control: Transferencia, ejecución condicional y parada.

4.3.2 Estructura general de la pantalla

La estructura general de Fortran en la pantalla del editor de programas se presenta a continuación en la Figura 4.5:

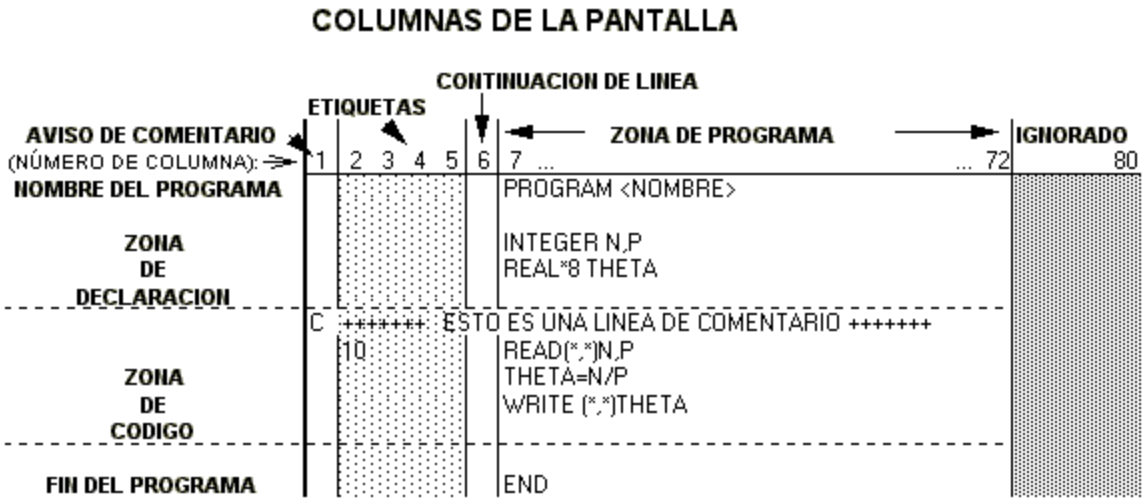

Figura 4.5 Estructura general de la pantalla de Fortran (http://personal.redestb.es)

De acuerdo al número de columna en el que se esté trabajando se pueden identificar las siguientes zonas:

- 1ª columna: se escribe una C en el caso de que queramos introducir una línea de comentarios. En caso contrario, esta línea estará siempre en blanco.
- Las columnas 2 a 5: sirven para especificar etiquetas. Las etiquetas son nombres que se pueden dar a las líneas para hacer referencia a ellas posteriormente.
- 6ª columna**:** sirve para introducir un carácter que indique al compilador que la línea anterior continúa en ésta. Esto es necesario cuando una línea supera en longitud la columna 72.

- Las columnas 7 a 72: definen la zona en la cual se sitúa el código FORTRAN. En estas columnas se escriben las líneas de las instrucciones de ejecución del programa.
- Después de la columna 72 el compilador ignora el contenido.

Las líneas de instrucción se pueden dividir en cuatro zonas principalmente:

- 1ª zona: Es la línea donde se escribe el nombre del programa. Está formada por la sentencia PROGRAM seguida del nombre del programa.
- 2ª zona**:** Es donde se declaran las variables, constantes y estructuras que tendrá el programa. Esta zona es de definición y por tanto, no se pueden situar sentencias de otro tipo.
- 3ª zona**:** Es en la que se sitúa el cuerpo del programa. Aquí se define el proceso que daremos a los datos.
- 4ª zona**:** determina el final del programa. Está constituida por la sentencia END, y separa el programa principal de los distintos subprogramas a que pueda hacer referencia éste. Esta línea es imprescindible y única. Dos sentencias END en un solo programa provocan un error en el compilador.

4.3.3 Errores comunes de programación:

La sintaxis de Fortran incluye reglas gramaticales que señalan que combinaciones de caracteres son admisibles y cuáles están prohibidas. Para encontrar los errores cuando se está programando en Fortran se recurre al comando Debugging, para así lograr detectar y corregir los errores de un programa.

Un programa puede tener tres tipos de errores:

- Errores de sintaxis**,** los cuales se deben a deficiencias en la escritura o captura del programa.
- Errores de ejecución, los cuales generalmente no son detectados por el compilador; estos son errores de programación.
- Errores lógicos, los cuales suceden cuando se obtienen resultados no esperados.

4.3.4 Tipo de datos.

Fortran emplea 5 tipos de datos:

- Lógico: Este dato sólo puede ser verdadero o falso. Es como un interruptor que puede estar encendido o apagado.
- Entero: Se trata de números sin decimales.
- Real: Números de cualquier tipo.
- Complejo**:** Se trata de un vector de dos componentes reales. Cada una de ellas representa la parte real e imaginaria del complejo.
- Carácter: son los caracteres alfanuméricos, numéricos, y símbolos que un ordenador puede representar.

4.3.5 Elementos de programación

- Constantes: Las constantes son cantidades numéricas o alfanuméricas (cadenas de caracteres) que no cambian su valor durante el proceso

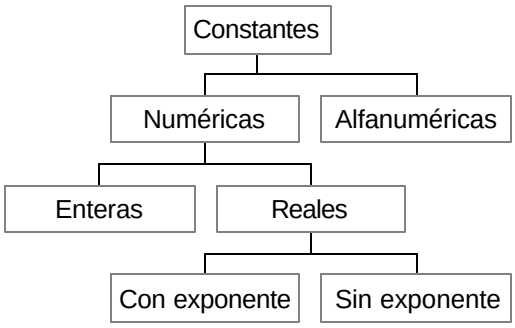

Figura 4.6. Clasificación general de las constantes

Las constantes enteras son cantidades numéricas sin punto decimal. El número máximo para una constante entera varía de computadora a computadora. En este tipo de constantes es válido omitir el signo +.

Las constantes reales sin exponente son cantidades con punto decimal. Las constantes reales con exponente tienen el significado de números multiplicados por una potencia de 10. La forma general de estas últimas es *a*E*x*. Donde a es una constante real sin exponente y x es una constante entera (con o sin signo). En este tipo de constantes no son válidas letras, comas o dobles signos a excepción de un punto decimal y de la letra E (notación exponencial). El menor y mayor valor, así como el epsilón depende también de la computadora que se maneje.

Las constantes de tipo alfanumérico son aquellas cadenas de combinaciones de números, letras y caracteres especiales encerrados entre comillas.

- Variables: Son aquellos nombres alfanuméricos, que tomarán distintos valores en el proceso. La longitud de una variable (su tamaño máximo en caracteres) se recomienda limitarla a 5 caracteres. En las variables no es válido incluir caracteres especiales como /, +, %, etc. En la Figura 4.7 se muestran los tipos de variables que maneja Fortran.

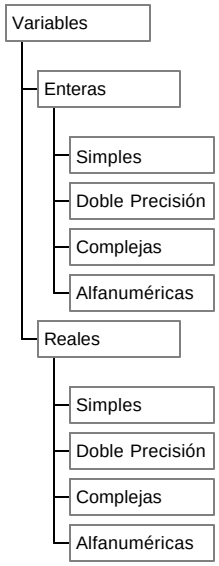

Figura 4.7. Clasificación general de las variables.

A continuación se enuncia una descripción general de cada uno de los tipos de variables.

- Entera simple. Sólo recibe valores enteros y debe empezar su nombre con alguna de las letras: I, J, K, L, M, N.
- Real simple. Es aquélla que recibe valores reales o que se le asignarán valores que deseamos tengan punto decimal. Su nombre puede empezar con cualquiera de las letras de la A a la H y de la O a la Z.

- De doble precisión. Una variable entera o una real puede admitir valores numéricos de doble precisión (se almacena en dos palabras de computadora), siempre y cuando se incluya una declaración DOUBLE PRECISION al principio del programa. La escritura para las constantes de doble precisión es la misma que se explicó con anterioridad, con excepción de las constantes reales con exponente en las cuales se sustituye la letra E por una letra D.
- Complejas. Una variable entera o real puede recibir valores numéricos complejos (del tipo a+bi), siempre y cuando se incluya una declaración COMPLEX al principio del programa.
- Alfanuméricas. Una variable entera o real puede adquirir valores alfanuméricos (letras y símbolos) mediante lectura o asignación.

4.3.6 Operadores y sus prioridades

Las operaciones básicas en FORTRAN se presentan en la tabla 4.2:

Tabla 4.2. Simbología de los principales operadores

**	Elevación a una potencia.
*	Multiplicación
/	División
+	Suma
-	Resta

Elaboración propia.

La jerarquía de las operaciones es:

1. Elevación a potencia.
2. Multiplicaciones y/o divisiones.
3. Sumas y restas.

En caso de existir en una expresión varias operaciones de igual jerarquía, se ejecutarán en el orden en el que aparecen en la expresión, de izquierda a derecha; excepto en el caso de varias elevaciones a determinada potencia que son ejecutadas de derecha a izquierda. Es posible modificar dicha jerarquía con el uso de paréntesis; primero se realizan las operaciones dentro del paréntesis más interno y así sucesivamente y siempre de izquierda a derecha. Estos símbolos nos permiten operar tanto con constantes como con variables.

- Expresiones: Una expresión aritmética es una relación matemática entre constantes y variables, constantes y funciones, constantes o variables o funciones, por medio de los operadores aritméticos y el uso de paréntesis (Friedman y Koffman, 1984).

  El uso de paréntesis cambia notablemente el significado de las ecuaciones. En la escritura de expresiones en FORTRAN, no es válido emplear dos operadores juntos, es necesario separarlos por medio de paréntesis. Un ejemplo de una expresión correcta es: A**(-2).

  En el caso de exponentes se recomienda que éstos sean enteros ya que una expresión como X**2.0 se evalúa por series, mientras que X**2 se calcula como el producto X*X.

4.3.7 Subprogramas.

Ojeda, R. (2003) define los subprogramas como elementos realizables que le permiten a un programa fraccionarse en tareas de programación específicas. Un subprograma cuenta con las siguientes características:

- Es posible compilarlo por separado del programa que lo utiliza.
- Sus variables pueden ser locales y globales.
- Requiere de un llamado del programa en el cuál el subprograma esta siendo utilizado.
- Mediante instrucciones especiales se lleva a cabo la transferencia de datos entre el programa y el subprograma correspondiente.

Los tipos de subprogramas que se pueden utilizar en Fortran son los siguientes:

- Funciones intrínsecas: Son subprogramas que forman parte de el compilador de el lenguaje, de manera que basta únicamente hacer referencia a la función deseada dentro del programa para que la computadora ejecute la serie de instrucciones correspondientes a dicha función. Para que estas funciones sean utilizadas es necesario escribir el nombre de la función seguida de una parte que se encuentra encerrada dentro del paréntesis, la cual recibe el nombre de argumento. Este puede ser una constante o variable numérica, o bien una expresión matemática. A continuación se presenta una lista de las algunas funciones que son comúnmente utilizadas:

Tabla 4.3. Principales funciones en Fortran

Función	Descripción
SIN(X)	Seno trigonométrico de X
COS(X)	Coseno trigonométrico de X
ATAN(X)	Ángulo cuya tangente es X
ABS(X)	Valor absoluto de X
EXP(X)	Exponencial de X
ALOG(X)	Logaritmo natural de X
SQRT(X)	Raíz cuadrada de X
TAN(X)	Tangente de X
COT(X)	Cotangente de X
ALOG10(X)	Logaritmo en base 10 de X
TANGH(X)	Tangente hiperbólico de X
IMIX(X)	Conversión de tipo real a entero
FLOAT(I)	Conversión de tipo entero a real
RANDOM(X)	Generador de números pseudo aleatorios

Elaboración propia.

En todos los casos X es una variable, constante o expresión aritmética real. "I" es una variable o expresión entera.

- Subprogramas externos escritos por el usuario: Son escritos paralelamente con el programa principal que los utiliza. Este tipo de subprograma se subdivide en:

  A. Función. Unidad independiente que devuelve un valor al ser llamada por el programa principal o por otros programas.

  B. Subrutina. Este subprograma es también una unidad independiente, a la cual se le asigna un nombre y argumentos. La subrutina altera el valor de cuando

menos una de sus variables incluidas el argumento que le corresponde. Para llamar a una subrutina se emplea la instrucción CALL.

```
CALL AREA(A,B,C)
```

Los argumentos permiten enviar valores de variables a la subrutina y también recibir resultados de ella para cada variable enviada.

```
Subroutine area (x,y,z)
x=0.5*y*z
Return
End
```

Evidentemente la variable X del subprograma corresponde a la variable A del programa que llama; Y con B y Z con la variable C. Es definitivamente necesario respetar el modo de las variables; el nombre de éstas puede cambiarse al gusto.

- Subprogramas externos en bibliotecas. Estos son las bibliotecas con las que cuenta el compilador de Fortran.

4.3.8 Instrucciones de uso común

- Read: Es la proposición que controla las operaciones de entrada, es decir, la forma general de una lectura de datos.

READ(N,M) variable1, variable2 <----Conjunto de variables
| | separadas por comas
| |
| ---------Número del formato con el que va a
| leer las variables de la lista.
|
----------Número de la unidad de lectura,
es una constante entera que varía.

- Write: Es una proposición que produce la información de salida que usa el programador, es decir una salida de datos o por archivo.

WRITE(N,M) variable1, variable2 <----variables separadas por comas
| |
| |
| ---------Número de formato
|
----------Número asignado a la unidad de salida.

- Input y Print: Son casos particulares de read y write. Estas instrucciones son utilizadas para transferir información desde o hacia un teletipo. Estas instrucciones no necesitan una lista de control como en el caso de read y write. Asimismo, el formato puede ser el predeterminado por el compilador Fortran.

  10 INPUT A,B

  20 PRINT A,B

  ó

  10 INPUT X

  20 PRINT,'EL VALOR DE X ES',X

- Proposición DO: La instrucción iterativa en Fortran es la instrucción DO. El DO equivale en principio a los siguientes elementos: un contador, una pregunta y una transferencia de control. La forma de esta instrucción es la siguiente:

  DO $n_i=n_1,n_2,n_3$

  Instrucciones

  n CONTINUE

  Donde: n: Es un número positivo que corresponde con la última instrucción ejecutable en el ámbito del DO.

i: Es una variable de tipo entero que variará de su valor inicial n_1 a su valor final n_2, en incrementos de valor n_3. Si el incremento es unitario se puede omitir n_3.

n_1, n_2, n_3 : Son variables o constantes de tipo entero, sin embargo, algunas versiones de compiladores de Fortran permiten valores fraccionarios o variables reales.

4.3.9 Instrucciones de transferencia de control.

- Go to: La instrucción de transferencia de control GO TO n, donde n es un número entero positivo, al ejecutarse altera la secuencia de ejecución del programa desviándola a la instrucción numerada con n. Por ejemplo: GO TO 98

- If: Se utiliza para preguntar por el valor de una expresión condicional. Su forma general es:

  If (e1.operador relación al.e2) instrucción

  Donde: e1 y e2 son las expresiones que se van a comparar. Los operadores relacionales son los siguientes:

  .LT. Menor que (<)

  .LE. Menor o igual (<=)

  .EQ. Igual (=)

  .GE. Mayor o igual (>=)

  .GT. Mayor que (>)

En las expresiones lógicas se utilizan, además de los operadores relacionales, los operadores lógicos:

.OR. o

.AND. y

.NOT. no

4.4 Simulación

Una de las herramientas estadística que fue necesaria para alcanzar el objetivo de esta tesis, fue la simulación. Al igual que la programación, la simulación en un principio llegó a ser un proceso complicado y muy tardado; sin embargo, con la llegada de las computadoras las técnicas de simulación pudieron ser aplicadas a problemas relacionados a diferentes áreas de la ciencia y la tecnología sin necesidad de la inversión de tantos recursos y gran tiempo.

La simulación consiste en la reproducción del comportamiento de un sistema de procesos por medio de la utilización de muestras aleatorias de una población. Algunos de los muchos beneficios en los que el uso de la simulación puede ayudar son:

- Estimación de parámetros.
- Identificación de cuellos de botella.
- Evaluación del uso de recursos.

Como se mencionó anteriormente, para poder hacer uso de la simulación es necesario tener una muestra aleatoria de una población. El Teorema de la Transformación Inversa que a continuación se explica hizo posible la obtención de tal requerimiento.

4.4.1 Teorema de la Transformación Inversa

Es posible generar números aleatorios de cualquier distribución mediante el Teorema de la Transformación Inversa, utilizando la distribución Uniforme (0,1). El teorema, mencionado en Burguete, Tamborero y Morales, enuncia lo siguiente:

> "Toda distribución de una variable continua X puede transformarse en la densidad uniforme. Debido a:

$$f(y)\begin{cases} = 1 & 0 < y < 1_a \\ = 0 & d.o.m \end{cases}$$

Es posible igualar la Función de Distribución Acumulativa de la distribución de la cual se deseen generar los números aleatorios con la variable aleatoria "y".

$$y=G(x)$$

Donde: G(x) es la distribución acumulativa de x.

Para generar los números aleatorios es necesario despejar x aplicando la función inversa de la distribución acumulativa de x, $G^{-1}(y)$:

$$x=G^{-1}(y).$$

Este teorema se utilizó para generar los números aleatorios necesarios para llevar a cabo la optimización de la función de máxima verosimilitud. Como enuncia el teorema, se igualó la FDA en las tres etapas de la Tina de Baño a una variable "x" que se distribuye Uniforme (0,1). Posteriormente, se despeja la variable "t", para que por medio de la generación de números aleatorios entre 0 y 1 se pueda obtener el dato que conformará la muestra aleatoria necesaria para realizar esta tesis.

Siguiendo el Teorema de la Transformación Inversa se deben de igualar las Funciones de Distribución Acumulativa en cada etapa de la Tina de Baño, F(t), a una variable que se distribuya uniformemente (0,1), "x", resultando las siguiente ecuaciones:

$$F(x)\begin{cases} = 1-\exp\left(-\left(\frac{t}{ca}\right)^{m_a}\right) & 0 < t < t_a \\ = 1-\exp\left(-\lambda\left[t_a\left(\frac{1-m_a}{m_a}\right)+t\right]\right) & t_a < t < t_b \\ = 1-\exp\left(-\left[\lambda\left(t_a\left(\frac{1-m_a}{m_a}\right)+t_b\left(\frac{m_b-1}{m_b}\right)\right)+\left(\frac{t}{cb}\right)^{m_b}\right]\right) & t_b < t \end{cases} = x$$

Como enuncia el Teorema es necesario despejar t, variable de interés, por lo que se aplica la función logaritmo natural en cada una de las expresiones anteriores. Posteriormente se despejo la variable "t" para obtener la $F^{-1}(x)$.

$$t\begin{cases} = \left(-\ln(1-x)\right)^{1/m_a} * c_a & 0 < t < t_a \\ = \dfrac{-\ln(1-x)}{\lambda} - t_a\left(\dfrac{1-m_a}{m_a}\right) & t_a < t < t_b \\ = \left[-\ln(1-x) - \lambda\left[t_a\left(\dfrac{1-m_a}{m_a}\right) + t_b\left(\dfrac{m_b-1}{m_b}\right)\right]\right]^{1/m_b} * c_b & t_b < t \end{cases}$$

Estas son las expresiones que fueron utilizadas para la generación de la muestra aleatoria, necesaria para la resolución de esta tesis. En el siguiente Capítulo se explicará a detalle la metodología que se siguió para obtener la muestra que tiene una distribución Tina de Baño.