CAPÍTULO IV

ELEMENTOS PARA LA OPTIMIZACIÓN DE FUNCIONES DE DISTRIBUCIÓN

4.1 Introducción a la programación

Programadores analistas, ingenieros, gerentes y planeadores son confrontados tradicionalmente por problemas que necesitan solución. Los problemas muchas veces involucran llegar a un diseño óptimo, asignar recursos escasos o reducidos, planear operaciones industriales o encontrar la trayectoria de un cohete. En el pasado, un rango amplio de soluciones era considerado aceptable. Afortunadamente, el gran y rápido crecimiento de las habilidades computacionales se ha agregado substancialmente en el uso de desarrollo de técnicas.

La aplicación de las matemáticas y del método científico a las operaciones militares (el despliegue del radar y la administración de un convoy, las operaciones de bombardeo, de ataque contra submarinos y colocación de minas), se les llamo investigación de operaciones. En la actualidad este término quiere decir un planteamiento científico a la toma de decisiones, que busca determinar cómo diseñar y operar mejor un sistema, normalmente bajo condiciones que requieren la asignación de escasos recursos.

Otro aspecto que ha estimulado el uso de aproximaciones sistemáticas a solución de problemas es el rápido incremento en el tamaño y la complejidad de problemas como resultado del crecimiento tecnológico desde la Segunda Guerra Mundial. La programación se encuentra dividida en dos partes:

- La programación Lineal (PL) y,
- La programación no lineal (PNL).

Esta tesis cuenta con una función la cual se pretende optimizar que es la Función de Máxima Verosimilitud y por lo tanto se trabajará con la programación no lineal.

4.1.1 Programación No Lineal

La Programación No Lineal se caracteriza porque la función matemática que se optimiza es no lineal. Es necesario decir que un problema de Programación No Lineal puede ser indicado como un problema de maximización y las restricciones de desigualdad de una forma mayor que. Bazaraa y Sherali (1993).

Una función no lineal es aquella que tiene involucradas variables en la expresión matemática que pueden ser cuadráticas, polinomiales, exponenciales, etc. Y un ejemplo de una de ellas es la siguiente:

$$x_1^2 \log x_2 + \cos^2 x_2 + x_3^3 + x_4 = 12$$

Un problema de Programación No Lineal se puede expresar de la siguiente forma: Bazaraa y Sherali (1993).

$$\begin{aligned} &\text{Max o Min} && f(x) \\ &\text{Sujeto a} && \begin{matrix} g_i(x) \le 0 \\ h_i(x) = 0 \end{matrix}, \quad \forall i \\ & && x \in X \end{aligned}$$

Donde: *f(x)* es usualmente llamada función objetivo o función de criterio.
g(x) es cada una de las *i* restricciones llamadas de desigualdad.
h(x) es cada una de las *i* restricciones llamadas de igualdad.
x es un subconjunto de variables que pertenecen al conjunto *X*.

Un vector x $\in$ X que satisface a todas las restricciones es llanada una solución factible del problemas. El conjunto de todas y cada una de las soluciones de la región factible. Programar un problema no lineal, es encontrar un punto factible $\bar{x}$ cada que $f(x) \geq f(\bar{x})$ para cada punto factible *x* en el caso de minimizar. Cada uno de los puntos $\bar{x}$ es llamado solución óptima o simplemente solución del problema. Si existe más de un óptimo,

existen referencias como soluciones óptimas alternativas. En un problema de Programación No Lineal la solución óptima no tiene que ser un punto extremo de la región factible. Una región factible para el problema de Programación No Lineal es el conjunto de puntos $x_1, x_2, \ldots x_n$ que satisfacen las i restricciones del $f(x)$. Para lograr el objetivo de esta tesis y obtener la solución óptima fue necesario utilizar un algoritmo de búsqueda multi-direccional denominado Nelder Mead que se describe en la siguiente sección.

4.2 Algoritmo de búsqueda multi-direccional Nelder Mead

El algoritmo Nelder Mead simplex fue publicado por primera vez en 1965, es un método de búsqueda directo para minimización multi-direccional irrestricto, trabaja bastante bien para problemas estocásticos. Se basa evaluando una función en los vértices de un simplex, luego de una manera iterativa se realiza el acotamiento del simplex encontrando mejores puntos mediante algunos saltos deseados. El propósito de este algoritmo de búsqueda multi-direccional es minimizar una función no lineal sin restringir, de tal manera que se vayan construyendo una secuencia de mejores vértices para que de esta manera se encuentren valores cada vez más pequeños de la función que se este evaluando.

4.2.1 Método de búsqueda

Una de las metodologías propuestas para la utilización del algoritmo de Nelder Mead la hace Kuester y Mize (1973) en su libro ″Optimization Techniques with Fortran″ y se usará en el programa computacional desarrollado para obtener la optimización de la función Hockey Stick.

1. Un punto de inicio x_1, es seleccionado.

2. Se construye el simplex consistiendo del punto de inicio y los adicionales puntos siguientes:

$$x_j = x_1 + \xi_j \qquad \textbf{(4.1)}$$

Donde:

$j = 2, 3, ..., N+1.$

ξ_j es determinado por la siguiente tabla:

j	$\xi_{1,j}$	$\xi_{2,j}$	• • •	$\xi_{N-1,j}$	$\xi_{N,j}$
2	p	q	...	q	q
3	q	p	...	q	q
.	.	.	...	.	.
N	q	q	...	p	q
N+1	q	q	...	q	p

Tabla 4.1 Ubicación de los valores ξ_j . (Kuester, 1973)

N es el número total de variables.

a es la longitud del simplex.

$$p = \frac{a}{N\sqrt{2}}(\sqrt{N+1} + N - 1)$$

$$q = \frac{a}{N\sqrt{2}}(\sqrt{N+1} - 1)$$

3. Una vez que el simplex es formado, el algoritmo comienza calculando los valores de la función objetivo en cada uno de los vértices que están formados por los puntos calculados de acuerdo a la ecuación 4.1. El peor punto (el valor más alto de la función objetivo) es reemplazado por un nuevo punto, en cambio el mejor punto donde se obtiene el mejor resultado de la función, se realiza la búsqueda que optimice la función mediante el proceso de reflexión, el cual 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 Torczon (1989) y su

reporte técnico de la universidad de Rice, Texas; esto puede ser visto como la reflexión del simplex original a través del mejor vértice para originar el nuevo simplex. La figura 4.1 lo muestra gráficamente.

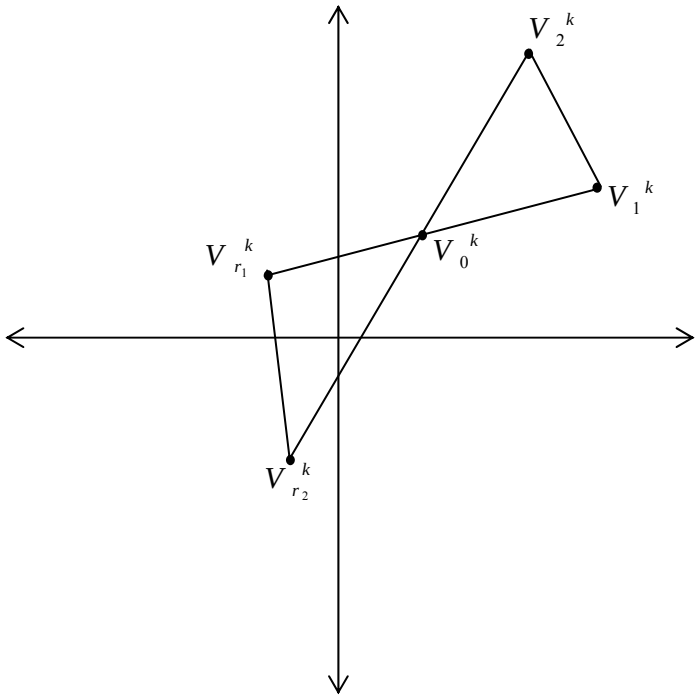

Figura 4.1 El simplex original y su reflejo. Torczon (1989).

En la Figura 4.1 se representa el simplex original y su reflejo. V_0^k representa el mejor vértice en cada una de estas *n* direcciones. La longitud de cada paso es igual al largo del borde que determinó la dirección de búsqueda, y $V_1^k, V_2^k, ..., V_n^k$ son los 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 evaluando, 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 de nueva cuenta vuelve a ser evaluada la función y en caso de que se obtenga un mejor resultado, este se toma de base para seguir realizando los distintos procedimientos de búsqueda considerados por el algoritmo. Los puntos reflejados se calculan de la siguiente forma:

$$X_{i,j}(reflejado) = \overline{X}_{i,c} + \alpha(\overline{X}_{i,c} - X_{i,j}(peor)) \qquad \begin{matrix} i = 1,...,N \\ j = 1,2,...,N+1 \end{matrix}$$

Donde: α es una constante positiva.

$\overline{X}_{i,c}$ son los puntos que son tomados como referencia para realizar la búsqueda multi direccional. Estos puntos son calculados con la siguiente ecuación:

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

K = N+1

$X_{i,j}(peor)$ es el punto cuyo valor de la función resulta ser el mayor.

4. Si el vértice reflejado al ser evaluado en la función objetivo resulta ser el peor valor de la función de los puntos actuales, se procede a realizar la búsqueda mediante el procedimiento de contracción. El algoritmo contrae el simplex, evalúa las funciones y compara con el valor de la función que fue tomada como referencia. De esta manera se realiza una nueva búsqueda de los valores de los parámetros que minimicen el valor de la función. La Figura 4.2 lo muestra gráficamente.

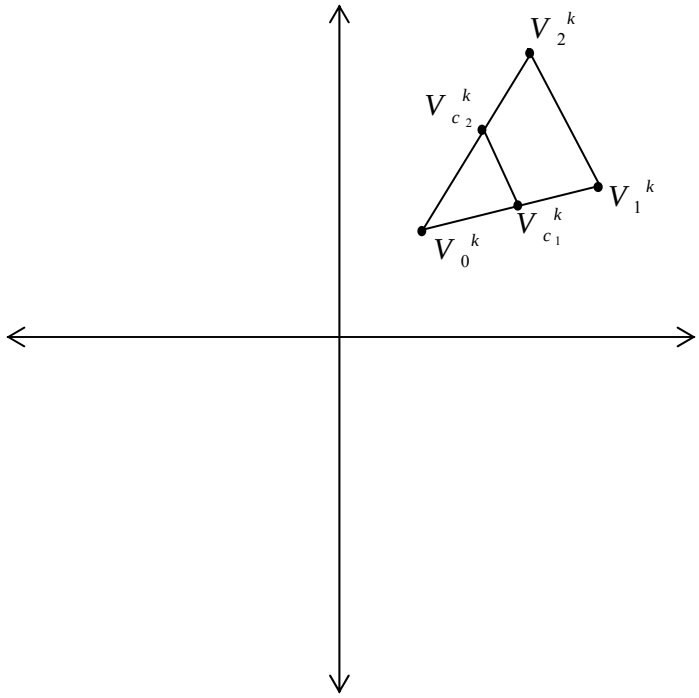

Figura 4.2 El simplex original y su contracción. Torczon (1989).

Para seguir haciendo la búsqueda por medio de la contracción, es necesario que el valor de la función de los vértices que se originaron contrayendo el simplex sean más pequeños al valor de la función en el vértice que se tomó como referencia, es decir, si el punto reflejado fue el peor valor en la función objetivo de los puntos actuales, un punto contraído es localizado y se calcula de la siguiente forma:

$$X_{i,j}(contraido) = \overline{X}_{i,c} - \beta(\overline{X}_{i,c} - X_{i,j}(peor)) \qquad \begin{matrix} i = 1,...,N \\ j = 1,2,...,N+1 \end{matrix}$$

Donde: β toma un valor entre 0 y 1

En el caso de que el punto reflejado sea mejor que el peor punto, un punto contraído es calculado para el punto reflejado y se calcula de la siguiente manera:

$$X_{i,j}(contraido) = \overline{X}_{i,c} - \beta(\overline{X}_{i,c} - X_{i,j}(reflejado)) \qquad \begin{matrix} i = 1,...,N \\ j = 1,2,...,N+1 \end{matrix}$$

Ahora la función es evaluada en el punto contraído. Si no existe una mejora, 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 expresión:

$$X_{i,j}(nuevo) = \frac{X_{i,j}(mejor) + X_{i,j}(viejo)}{2} \qquad \begin{matrix} i = 1,...,N \\ j = 1,2,...,N+1 \end{matrix}$$

Donde: $X_{i,j}(mejor)$ son los vértices cuya longitud de arco fue contraída en la iteración anterior.

$X_{i,j}(viejo)$ son los vértices cuyo valor de la función resultó ser el menor en la iteración anterior.

5. Si la función en el punto reflejado es menor que le del vértice cuya función tenía el mejor valor en la iteración anterior, procede a realizar la búsqueda mediante el proceso de expansión y por lo tanto un punto expandido es calculado y se realiza de la siguiente forma:

$$X_{i,j}(\exp andido) = \overline{X}_{i,c} + \gamma(X_{i,j}(reflejado) - \overline{X}_{i,c}) \qquad \begin{matrix} i = 1,...,N \\ j = 1,2,...,N+1 \end{matrix}$$

El simplex original con el simplex reflejado y su expansión se muestra gráficamente en la Figura 4.3.

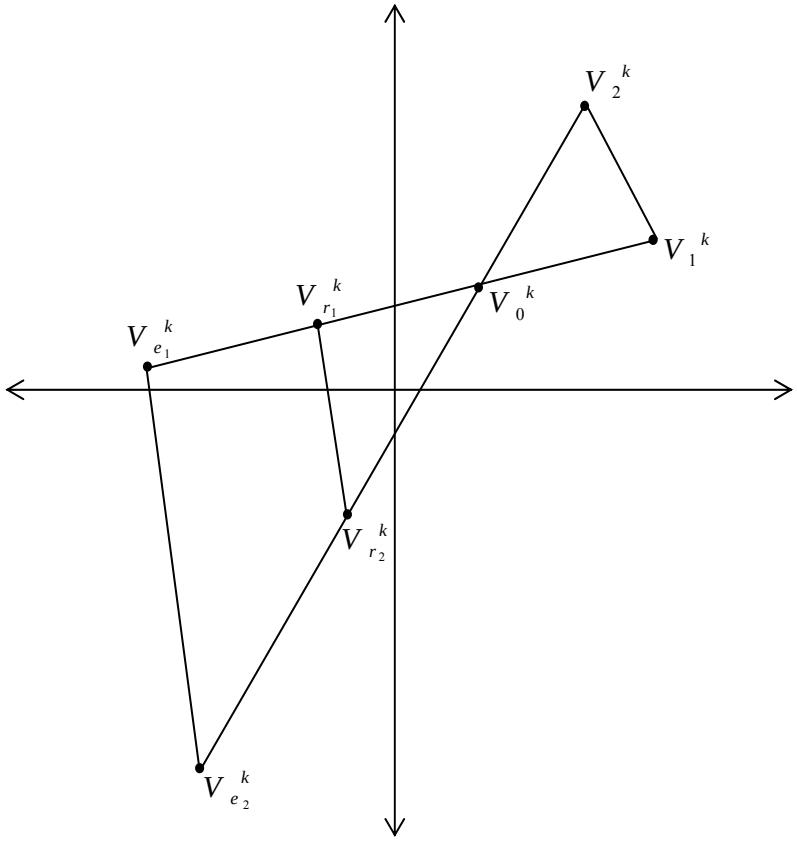

Figura 4.3 El simplex original, el simplex reflejado y su expansión. Torczon (1989).

La Figura 4.4 muestra un resumen de la lógica que sigue el algoritmo de Nelder Mead para optimizar las funciones de distribución que sean programadas.

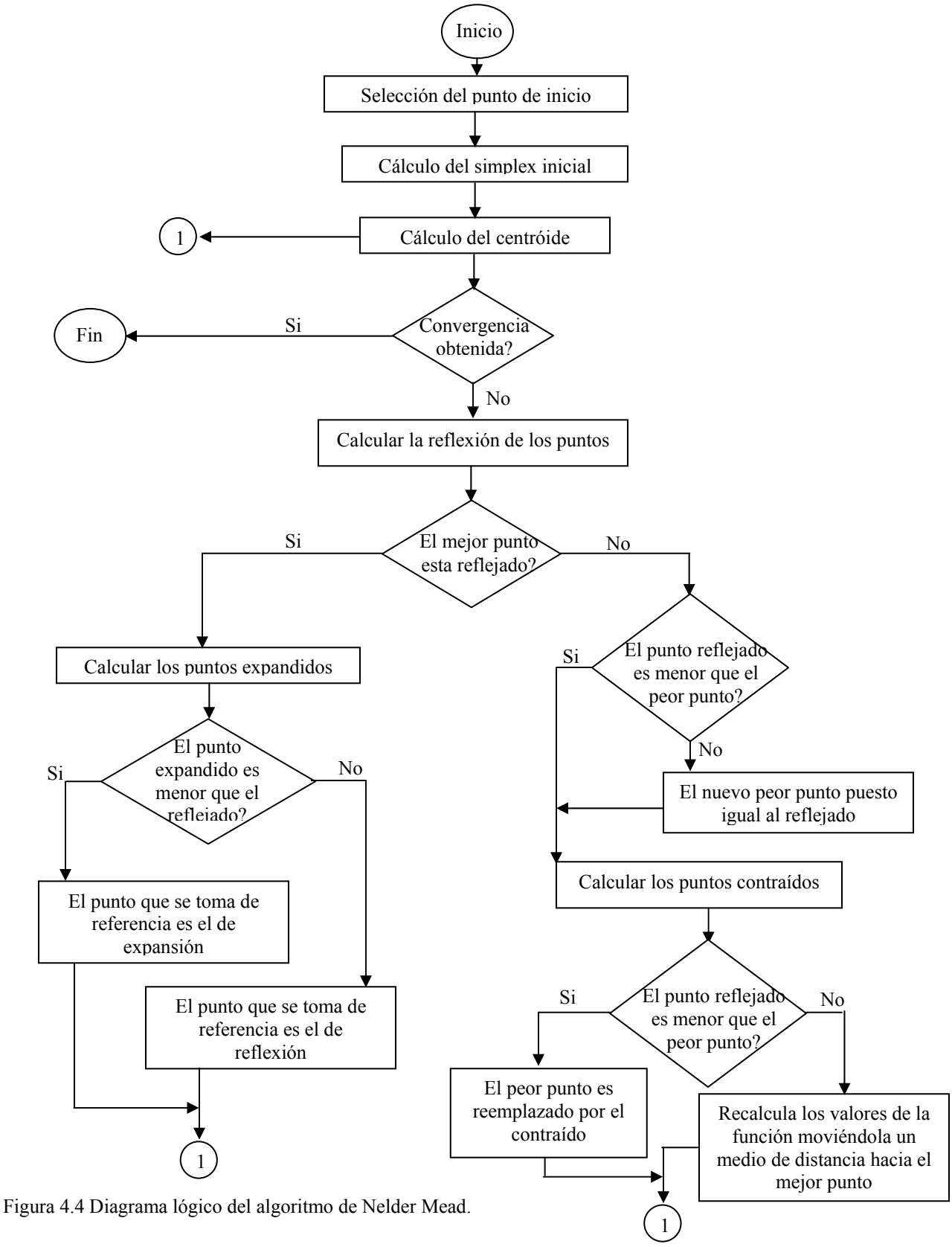

Figura 4.4 Diagrama lógico del algoritmo de Nelder Mead.

El lenguaje que se utilizará para la programación de la Función de Máxima Verosimilitud de la distribución Hockey Stick y posteriormente para la simulación de este algoritmo Nelder Mead será Fortran. Por esta razón se procede a describir algunos aspectos del lenguaje que cooperaron con la elaboración de esta tesis.

4.3 Lenguaje de programación Fortran

El empleo de las computadoras por la sociedad es cada día mayor debido a las enormes ventajas que estas reportan al trabajo, mejorando la productividad y el rendimiento en un alto número de actividades. Es por esto que resulta imprescindible para los profesionales de un gran número de áreas, un conocimiento informático mínimo como complemento a su formación académica.

El lenguaje estructurado que se utilizará para la realización de los programas correspondientes a esta tesis será Fortran. Fortran (FORmula TRANslator) es un lenguaje de propósito general que principalmente se encuentra orientado a la computación matemática gracias a la precisión que ofrece en sus cálculos. Fortran fue el primer lenguaje de programación de alto nivel. El desarrollo de Fortran se inició en la década de 1950 en IBM y ha habido muchas versiones desde entonces. Este lenguaje tiene algunos elementos básicos comunes a todos los lenguajes; tiene un conjunto de caracteres, un pequeño vocabulario en ingles y una sintaxis.

4.3.1 Estructura general de la pantalla

Las antiguas computadoras no disponían de teclado para la introducción de programas. Estos, se codificaban en tarjetas mediante perforaciones, y se introducían en un lector que las traducía a números (una especie de braille informático). Fortran tiene una estructura general en la pantalla del editor de programas, la cual se muestra en la Figura 4.5 (Basada en un monitor de 80 columnas).

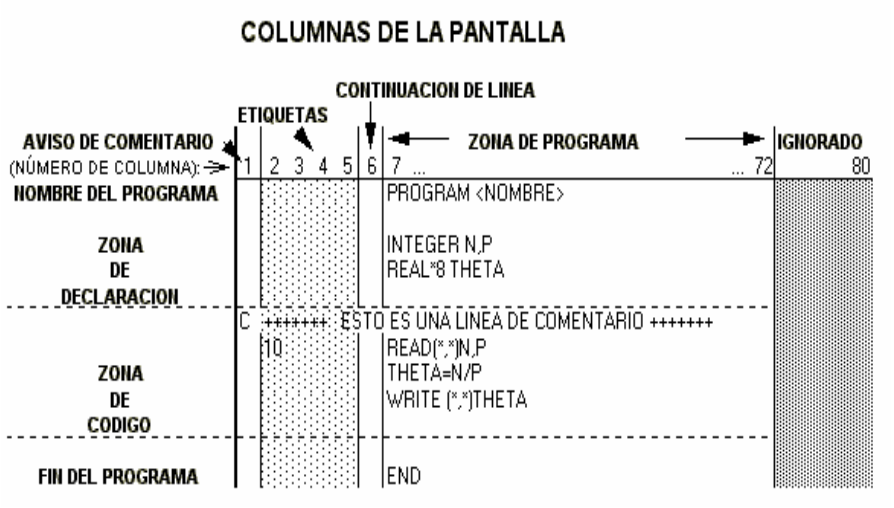

Figura 4.5 Estructura general de la pantalla del editor de programas de Fortran.

Primero se empezará a hablar de la estructura de las columnas donde cada una de ellas, desde la 1 hasta la 72 tiene un sentido que a continuación se describe:

- **Columna 1:** esta columna es reservada para una *C* en el caso de que se quiera especificar que esa línea esta destinada a un comentario. El comentario puede aparecer en cualquier lugar del programa. El colocarlos en el lugar preciso incrementan la legibilidad del código.
- **Columnas 2 a la 5:** estas columnas se encuentran reservadas para el caso de que se necesite poner una etiqueta para hacer referencia a ellas posteriormente.
- **Columna 6:** es la columna de continuación. Ocasionalmente una sentencia no cabe en una sola línea, es decir, supera la columna 72. Se puede dividir la sentencia en dos o más líneas, y usar la marca de continuación en la posición 6, colocando cualquier símbolo distinto de un espacio en blanco o un 0.
- **Columnas 7 a la 72:** estas columnas definen la zona en la cual se sitúan las sentencias del código, se escriben las líneas de las instrucciones de ejecución del programa.
- **Columna 73 en adelante:** el compilador ignora su contenido, es decir, no se usan en modo interactivo. Estas tenían sentido cuando se trabajaba con tarjetas perforadas.

Un programa de Fortran por lo general consiste de un programa principal o *main* y posiblemente varios subprogramas. Por el momento se considerara que todas las sentencias están en el programa principal; los subprogramas se revisarán más adelante. De acuerdo a la Figura 4.5 las líneas de instrucción se pueden dividir en cuatro zonas principales de acuerdo a la organización del código las cuales se describen a continuación:

- **Nombre del programa:** esta es la primera zona y en ella se encuentra el nombre del programa que esta precedido por la sentencia *PROGRAM*.
- **Zona de declaración:** esta que es la segunda zona, es donde se hace la declaración de las constantes, variables y estructuras que tendrá el programa. Esta zona es de definición y por lo tanto, no se pueden situar sentencias de otro tipo.
- **Zona de código:** la tercera zona como su nombre lo indica es en la cual se sitúa el cuerpo del programa y es aquí donde se define el proceso que se dará a los datos.
- **Fin del programa:** la cuarta zona determina que es aquí donde finaliza el programa. Esta constituida por la sentencia *END* y además separa el programa principal o el *main* de los distintos subprogramas a que pueda hacer referencia éste.

4.3.2 Operaciones básicas

Existen tres tipos de operaciones que los lenguajes de programación pueden realizar y estas operaciones son:

- **Operaciones de entrada y salida:** una parte importante del cualquier programa de cómputo es manejar la entrada y la salida de datos, es decir, leerlos e imprimirlos.
- **Manejo y comparación de datos:** se refiere a la suma, resta, multiplicación, división, negación, asignación y comparación de datos.

- **Operaciones de control:** son operaciones de transferencia, ejecución condicional y parada.

4.3.3 Tipos de datos

Fortran emplea 5 tipos de datos que se describirán a continuación en esta sección.

- **Entero:** este tipo de datos se trata de números enteros, es decir, sin decimales.
- **Real:** los números reales son de cualquier tipo.
- **Complejos:** estos números constan de dos partes: una real y una imaginaria y están conformados en un vector de dos componentes reales.
- **Lógicos:** los valores que puede tomar una variable de tipo lógico son: Verdadero y Falso, obtenidos por medio de operaciones de álgebra booleana. Se pueden interpretar como un interruptor que puede estar encendido o apagado.
- **Alfanumérico:** este módulo se usa para representar datos literales que puedan contener caracteres alfanuméricos, numéricos y especiales. Es usado para producir títulos, encabezados, etc.

Los datos enteros, reales y complejos son usados para representar valores numéricos obtenidos mediante operaciones aritméticas.

4.3.4 Elementos de programación

En los tipos de datos, excluyendo el alfanumérico el lenguaje permite la definición de constantes y variables.

Las constantes son valores invariables durante todo el programa, es decir, el valor de una constante no se cambia durante la corrida de un programa, mientras que el valor de una variable si puede cambiarse. En Fortran las constantes se escriben literalmente. Si la constante es un entero, su valor se escribe usando dígitos decimales; si la constante es lógica su valor se escribe usando las palabras *true* o *false* dependiendo de si se quiere

que asuma un valor verdadero o falso respectivamente. Se definen al principio del programa como un nombre y su valor y además pueden ser usadas más tarde referenciándolas. La Figura 4.6 muestra la clasificación general de las constantes.

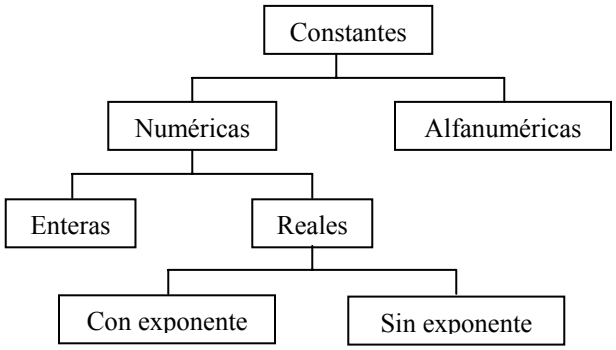

Figura 4.6 Clasificación general de las constantes.

- Las constantes enteras son cantidades numéricas sin punto decimal.
- 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 es *a*E*x*. Donde *a* es una constante real sin exponente y *x* es una constante entera positiva o negativa.
- Las constantes alfanuméricas son aquellas cadenas de combinaciones de números, letras y caracteres especiales encerrados entre comillas.

Muchas veces es necesario definir en un programa un lugar en el cual se pueda introducir un valor para modificarlo a medida que avanza el programa. En este caso, se define una variable. Las variables se representan por nombres. Durante la operación del programa el nombre es la dirección donde se encuentra un dato en el almacenamiento de la computadora. El uso del nombre de una variable en un programa es equivalente a solicitar el contenido actual de la dirección de almacenamiento así nombrada y usar este contenido como un valor para llevar a cabo un cálculo dado. La Figura 4.7 muestra una clasificación general de las variables.

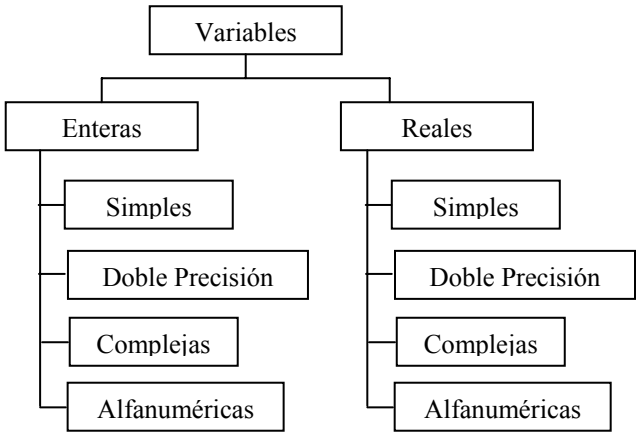

Figura 4.7 Clasificación general de las variables.

- Las variables enteras simples solo reciben valores enteros y debe empezar su nombre con alguna de las letras: I, J, K, L, M, N.
- Las variables reales simples son aquellas que reciben valores reales o que se les asignarán valores que se desea que tengan punto decimal. Su nombre puede empezar con alguna de las letras: A a la H y de la O a la Z.
- La variable de doble precisión puede ser una variable entera o real siempre y cuando se incluya una declaración *DOUBLE PRECISION* al inicio del programa.
- Las variables complejas pueden ser enteras o reales y puede recibir valores numéricos complejos (del tipo a $+b_i$) siempre y cuando se incluya una declaración *COMPLEX* en el inicio del programa.
- Las variables alfanuméricas pueden ser enteras o reales y pueden tomar valores alfanuméricos mediante lectura o asignación.

4.3.5 Los operadores y sus prioridades

Se llaman operadores a todos aquellos símbolos que realizan una operación determinada del argumento antecedente con el consecuente. En la expresión a la derecha del operador de asignación puede ser válida cualquier combinación de constantes, variables, paréntesis

y operadores aritméticos o lógicos. Los operadores son de distintos tipos y clases según sus argumentos.

- Los operadores aritméticos o matemáticos son los conocidos símbolos de suma, resta, multiplicación, etc. Se muestran en la Tabla 4.2.

Símbolo	Operador
+	Suma
-	Resta
*	Multiplicación
/	División
**	Potencia

Tabla 4.2 Símbolos de los principales operadores matemáticos.

- El operador de cadena es exclusivo para cadenas de caracteres, une el argumento antecedente y el consecuente en una sola cadena.

*Cadena*1 // *cadena*2

- Los operadores relacionales son operadores de resultado lógico. Esto significa que devuelven verdadero o falso según el resultado de la expresión que evalúan. La Tabla 4.3 muestra estos operadores.

Símbolo	Operador
.EQ.	Igualdad
.NE.	Desigualdad
.LT.	Menor que
.GT.	Mayor igual
.LE.	Menor o igual que
.GE.	Mayor o igual que

Tabla 4.3 Símbolos de los operadores relacionales.

- Los operadores lógicos se basan en la aritmética del lenguaje binario, en el cual, sólo existen dos valores, 0 o 1, falso o verdadero. Sus argumentos son, por lo tanto, valores lógicos.

A menudo algunas operaciones aritméticas son combinadas en una sola expresión. En una expresión es importante conocer el orden de cada una de las operaciones que serán evaluadas. Fortran ha establecido una serie de reglas de jerarquía o de orden en cada operación evaluada dentro de una expresión. Las reglas de orden para los operadores matemáticos o aritméticos de Fortran generalmente siguen las reglas normales de álgebra. El orden en cada operación aritmética es evaluada de la siguiente forma, Chapman (2004):

- El contenido de todos los paréntesis son evaluados primero, empezando por el que se encuentra más dentro de la expresión y evaluando hacia afuera de la expresión.
- Todos los exponentes son evaluados comenzando de derecha a izquierda.
- Todas las multiplicaciones y divisiones son evaluadas de izquierda a derecha.
- Las sumas y restas son evaluadas de izquierda a derecha.

4.3.6 Subprogramas

Muchas veces en un programa es necesario repetir un mismo proceso varias veces en distintas partes de éste. Para evitar el tener que repetir el mismo código varias veces, el programador tiene la opción de abstraer esa parte del programa en un módulo. Este módulo es independiente del programa principal y puede ser llamado por éste cuando se requiera. El lenguaje Fortran permite crear módulos independientes a los programas de dos tipos distintos: funciones y subrutinas.

- Una función es un bloque de código con un nombre de variable. En el programa principal se hace referencia a él como si se tratara de una variable más.

Generalmente, las funciones aparecen en sentencias de asignación y en expresiones a evaluar como una variable más.

- Las subrutinas son auténticos programas dentro de programas. Reciben y devuelven numerosos datos y resultados o incluso ninguno, es decir, podría ser que el proceso de la subrutina sea independiente del programa principal. Las subrutinas deben ser llamadas con una sentencia especial llamada *CALL*.

Los módulos se declaran mediante un nombre de variable. Los programas pueden llamar a estos módulos mediante su nombre. No es posible ejecutar un módulo por separado, sino que debe siempre estar asociado a un programa principal. Estos módulos cuentan con las siguientes características:

- Es posible compilarlos por separado del programa que los utilizan.
- Sus variables pueden ser locales o globales.
- Requieren de un llamado del programa en el cual el módulo esta siendo utilizado.
- Mediante instrucciones especiales se lleva a cabo la transferencia de datos entre el programa y el subprograma correspondiente.

Existen tres tipos distintos de funciones: de sentencia, intrínsecas y externas.

- **Funciones de sentencia:** se trata de funciones que se pueden definir en una sentencia del programa principal para calcular un determinado valor que se repite muchas veces durante el proceso.
- **Funciones intrínsecas:** estas funciones vienen definidas en el compilador y pueden ser llamadas mediante su nombre. Por ejemplo la función *SQRT*( ) que devuelve la raíz cuadrada del argumento.
- **Funciones externas:** son las funciones que se pueden definir para que las utilice el programa principal. Son los módulos propiamente dichos.
- **Funciones externas en bibliotecas:** estas son las bibliotecas con las que cuenta el compilador de Fortran.

4.3.7 Sentencias básicas

Las sentencias son aquellas instrucciones que se le dan al programa para que sean ejecutadas. Cabe mencionar que Fortran solo permite escribir una sentencia por línea, quedando restringido el utilizar varias de ellas en una misma línea. A continuación se presentan algunas de las sentencias básicas del Fortran.

- Al escribir un programa, su primera sentencia es *PROGRAM*, esta sentencia es la que le indica al programa el nombre al cual responderá y cuya sintaxis es simplemente:

PROGRAM nombre

Donde *nombre* es el nombre o denominación simbólica que quiere darse al programa que comienza a escribirse. Debe escribirse en primer lugar.

- Para indicar la terminación de la ejecución de un programa es necesario emplear la sentencia *STOP*. Un programa puede contener varias sentencias *STOP* o lo que es lo mismo, varios puntos de parada. Por ello aunque el uso de *n* no es obligatorio, es conveniente para poder saber donde realmente se ha detenido un programa, ya que la constante *n* puede ser visualizada en una unidad de salida, cuando se produce la parada. La sintaxis o forma general de esta sentencia es:

STOP n

Donde *n* es una serie de hasta cinco dígitos o bien una constante carácter.

- La última sentencia de un programa tiene que ser *END*. Esta sentencia equivale al final de un programa ejecutable e indica al compilador que ya no existen más sentencias en el programa fuente para ser convertidas en programa objeto.

END

Para resumir, se dirá que *STOP* produce una parada lógica y definitiva y sirve para detener un proceso; *END* sirve para indicar el final de una compilación.

- Por último, cualquier línea precedida por el símbolo *C* se entenderá como un comentario, y el compilador lo obviará a la hora de codificar el programa. Esto puede resultar útil, ya que puede hacer un programa más legible para personas que no sean el propio programador. Este símbolo se coloca en la primera columna, espacio destinado para este fin.

- La sentencia *CONTINUE* es ejecutable pero no genera instrucción alguna al ser traducida a código máquina por el compilador. Su uso más común es el siguiente: se emplea para colocar como última sentencia en un bucle. Otra aplicación es la de sustituir *CONTINUE* por una sentencia que anteriormente se hacia referencia. Su sintaxis es muy simple:

CONTINUE

- Para indicar una parada temporal en la ejecución de un programa se utiliza la sentencia *PAUSE*, cuya sintaxis o forma general es:

PAUSE n

Donde *n* es una serie de cinco dígitos como máximo o bien una constante carácter.

- La sentencia que controla la introducción de datos, es decir, la forma general de una lectura de datos. Para la lectura de elementos con *READ* las variables se separan por un blanco o por comas, si se ha de introducir una frase de tipo caracter, debe ir encerrado entre comillas. Si no hay suficientes datos para esta

instrucción se produce un error y el programa aborta. Se utiliza la palabra reservada *READ* con el formato siguiente:

READ (n, eti).

Donde *n* es el número que indica la unidad de entrada, colocando el valor * si no se quiere poner nada.

eti indica la etiqueta que va a definir un formato, si no se quiere poner nada se usa el valor *.

- Para la salida de datos se utilizan dos sentencias ambas igualmente válidas. La lista de variables pueden ser tanto variables, o constantes como expresiones de todo tipo. El cursor avanza a la siguiente línea. Este tipo de sentencias se suelen incluir antes de una sentencia *READ* y que tienen el siguiente formato:

PRINT *, Lista de variables (Separadas por comas).
WRITE (*,*), Lista de variables (Separadas por comas).

4.3.8 Instrucciones de transferencia de control

En los programas las instrucciones preparadas por un ordenador se ejecutan secuencialmente mientras no se disponga lo contrario, es decir, acabada la realización de una de ellas, comienza la ejecución de la siguiente.

- Existen situaciones donde nos interesa repetir una instrucción o ejecutar una instrucción que viene más abajo ignorando de esta manera las sentencias siguientes. Esto saltos se denominan bifurcaciones, y se realizan mediante la instrucción **GO TO** La sintaxis para el *GO TO* es la siguiente:

GO TO n

Donde *n* es el número o etiqueta de una sentencia ejecutable que aparecerá en el mismo programa, es una constante entera positiva y distinta de cero de valor $n < 99999$. La constante *n* representa el número de la siguiente instrucción a realizar. Este número puede escribirse entre las columnas 1 a 5 de la hoja de codificación.

- Otra sentencia que se emplea para realizar bifurcaciones, según ciertas condiciones, es la sentencia *IF*. El significado de esta instrucción es: si el valor de la expresión *e* es menor, igual o mayor de cero, el control se transfiere, respectivamente, a la sentencia n_1, n_2 o n_3 respectivamente. Responde a la sintaxis siguiente:

$$IF\ (e),\ n_1,\ n_2,\ n_3$$

Donde *e* es una expresión aritmética entera o real.

n_1, n_2 y n_3 son números o etiquetas de sentencias ejecutables, que se encuentran en el mismo programa.

- Para ejecutar repetidamente una serie de sentencias en bucle, tiene mucha utilidad el uso de una sentencia especial: la sentencia *DO*. Su sintaxis es:

$$DO\ n,\ i = m_1,\ m_2,\ m_3$$

Donde *n* es una constante entera que indica el número de una sentencia posterior a la *DO*, que se denomina sentencia terminal del bucle *DO*.

i es una variable entera.

m_1, m_2, m_3 son constantes o variables enteras, reales, o de doble precisión e indican un valor inicial, un valor final y un incremento. La variable m_3 puede omitirse si su valor es igual a 1.

4.4 Simulación

Una herramienta estadística que fue necesaria para la elaboración de esta tesis y alcanzar su objetivo fue la simulación. La simulación es la imitación de varios tipos de procesos o habilidades del mundo real. Estos procesos o habilidades son también llamados sistemas y con el propósito de estudiarlos científicamente muchas veces se tendrán que hacer suposiciones acerca de como trabajan. Estas suposiciones las cuales usualmente toman forma de relaciones lógicas o matemáticas constituyen un modelo de tal manera usado para tratar de comprender el comportamiento del sistema. Del mismo modo que la programación, la simulación en un principio llegó a ser un proceso muy complicado y muy tardado, sin embargo con las computadoras las técnicas de simulación pudieron ser aplicadas a problemas de diferentes áreas para evaluar modelos numéricos, y los datos son reunidos con el propósito de estimar las características del modelo deseadas sin necesidad de invertir gran cantidad de recursos y tiempo.

Algunas de las áreas de aplicación de la simulación son numerosas y diversas. A continuación se presenta una lista de algunos tipos particulares de problemas en los cuales la simulación ha encontrado ser una herramienta muy usada y poderosa.

- Estimando parámetros.
- Identificando cuellos de botella.
- Evaluando el uso de los recursos de cualquier sistema.
- Diseñando y analizando sistemas de manufactura.
- Evaluando sistemas de armas militares o los requerimientos logísticos.
- Diseñando y operando sistemas de transportes de aeropuertos, carreteras, puertos y metros.
- Evaluando diseños para organizaciones de servicios como: centros de llamadas, restaurantes de comida rápida, hospitales y correos entre otros.
- Determinando planes de acción para sistemas de inventarios.
- Etc.

4.4.1 Teorema de la transformación inversa

Mediante el teorema de la transformación inversa es posible generar números aleatorios de cualquier distribución, utilizando la distribución uniforme (0, 1). Este teorema se encuentra mencionado en Burguete, Tamborero y Morales (2003) y enuncia lo siguiente:

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

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

Por lo tanto, 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* para el caso de esta tesis será la Función de Distribución Acumulada de la distribución Hockey Stick.

$$y = G(x)$$

Donde: $G(x)$ es la Función de Distribución Acumulativa de *x*.

Los números aleatorios para que puedan ser generados es necesario despejar *x* aplicando inversa de la distribución acumulativa, como se muestra a continuación:

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

Para generar los números aleatorios necesarios para optimizar la función de máxima verosimilitud de la distribución Hockey Stick se utilizó este teorema. Como ya se enunció se iguala a una variable *y* que se distribuye uniforme (0, 1) la Función de Distribución Acumulativa en las dos etapas que conforman la distribución Hockey Stick. Luego se procede a despejar la variable *t*, para que se pueda obtener el dato que conformará la muestra aleatoria necesaria por medio de la generación de números aleatorios entre 0 y 1,

que son necesarios para la realización de esta tesis. Siguiendo el teorema de la transformación inversa resultan las siguientes ecuaciones:

$$F(t) = \left\{ \begin{array}{ll} 1 - e^{-\lambda t} & 0 < t \le t_a \\ 1 - e^{-\left[\lambda t_a \left(\frac{m-1}{m} \right) + \frac{t^m}{c^m} \right]} & t > t_a \\ 0 & d.o.m. \end{array} \right\} = y$$

Como ya se mencionó anteriormente es neceario despejar la variable *t*, la cual es la de interes, por lo que se aplica la función logaritmo natural en cada una de las expresiones anteriores. Posteriormente se despejó la variable *t* para obtener la $F^{-1}(x)$.

$$t = \left\{ \begin{array}{ll} -\frac{1}{\lambda}\ln(1-y) & 0 < t \le t_a \\ c\left[-\lambda t_a \left(\frac{m-1}{m} \right) - \ln(1-y) \right]^{\frac{1}{m}} & t > t_a \\ 0 & d.o.m. \end{array} \right.$$

Estas expresiones son las que serán usadas para la generación de la muestra aleatoria necesaria para la solución de esta tesis. La metodología que se sige para obtener la muestra que tiene una distribución Hockey Stick se explicará a detalle en el siguiente Capítulo.