Universidad Nacional Autónoma de México

Escuela Nacional de Estudios Profesionales Acatlán

EL ALGORITMO DE SHUBERT PARA OPTIMIZACION GLOBAL EN UNA LINEA

T E S I S

Que para obtener el título de:

A C T U A R I O

P r e s e n t a :

FERNANDO VEGA SAENZ

México, D. F. 1980

M-0037497

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Agradezco con toda mi alma

A Jesús y Bertha, quienes desde antes de mi nacimiento me han dado lo mejor de sí mismos. Por su cariño, dedicación y compañía a través de toda mi vida.

A mis asesores en la realización de este trabajo, Gilberto Calvillo y Eduardo Godoy, por su guía, paciencia y comprensión.

A mis compañeros del Banco de México, por su valioso apoyo intelectual y moral.

Y a todas las personas que quiero y con las cuales he compartido anhelos, alegrías y desilusiones.

FERNANDO VEGA SAENZ.

ENEP ACATLAN
COORDINACION DEL PROGRAMA
DE INGENIERIA Y ACTUARIA

CAI-C-0218-79

SR. FERNANDO VEGA SAENZ
Alumno de la Carrera de Actuario
P r e s e n t e

De acuerdo a su solicitud presentada con fecha 28 de junio de 1979, me complace notificarle que esta Coordinación tuvo a bien asignarle el siguiente tema de tesis: -- "El método de Shubert para optimización global en una linea", el cual se desarrollará como sigue:

1. Introducción, propósitos de la tesis.
2. El problema de programación no lineal.
3. El método de Shubert.
4. Estructura de datos y algoritmos de ordenamiento.
5. Estructura desarrollada para el método de Shubert.
6. Algunos ejemplos y aplicaciones del método de Shubert.
7. Apéndice: listado del algoritmo.

Asimismo fue designado como Asesor de Tesis el señor- M. en C. José Eduardo Gabriel Godoy Escoto, profesor de esta Escuela.

Ruego a usted tomar nota que en cumplimiento de lo especificado en la Ley de Profesiones, deberá prestar servicio social durante un tiempo mínimo de seis meses como requisito básico para sustentar examen profesional, así como de la disposición de la Dirección General de Servicios Escolares en el sentido de que se imprima en lugar visible - de los ejemplares de la tesis el título del trabajo realizado. Esta comunicación deberá imprimirse en el interior- de la tesis.

A t e n t a m e n t e

"POR MI RAZA HABLARA EL ESPIRITU"
Sta. Cruz Acatlán, Edo. de México a 13 de agosto de 1979

ING. IGNACIO LIZARRAGA G.,
Coordinador del Programa
de Ingeniería y Actuaría.

I N D I C E

Página

INTRODUCCION

En todas las actividades que realiza el ser humano, ya sea en política, en la industria, o bien en la vida diaria de cada persona, es necesario tomar decisiones sobre qué acciones se deben efectuar para alcanzar los objetivos deseados con la máxima utilidad. Donde utilidad se refiere a menor tiempo, mayor diferencia beneficio-costo, menor contaminación del medio ambiente, o cualquier otra medida de evaluación de los resultados que se desean alcanzar(*).

Debido a que la selección del conjunto óptimo de acciones en algunos problemas es sumamente compleja, es por lo que el asunto se ha estudiado científicamente. De aquí el creciente desarrollo de la aplicación de modelos matemáticos en diversos campos.

Varias revistas se editan periódicamente divulgando las investigaciones que, de este tipo de problemas, se realizan en el mundo(**).

Son interesantes y muy variados los problemas que se tratan y los procedimientos que se utilizan para resolverlos. Por ejemplo, en el cuidado del medio ambiente se ha escrito acerca del "uso de programación entera mixta en la evaluación de políticas alternativas para combatir la contaminación del aire" [11], o bien, "Tratamiento de la contaminación de un lago visto como un problema de control óptimo" [21]; en el área financiera se encuentran importantes

---

(*) El problema de la evaluación de la utilidad y del análisis de cursos de acción, se estudia en la formulación de la teoría de decisiones. Consultar caps. I, II de [12].

(**) En especial las que se clasifican dentro del área denominada investigación de operaciones, por ej. [17], [23], [24].

aplicaciones como "Sistema de planeación de la cartera de un banco" [6] ; dentro de la estadística es posible solucionar problemas tales como encontrar los estimadores de máxima verosimilitud para una familia de distribuciones uniparamétricas; y aplicaciones inesperadas en áreas como es la investigación bio-médica, "Optimización en la modelación de las señales eléctricas emitidas por el tubo digestivo" [20] .

Como se observa, la importancia que ha adquirido el uso de la optimización en el tratamiento de problemas de diversa índole es ampliamente reconocida.

Las etapas que usualmente se siguen al tratar los problemas a través de modelos matemáticos son las siguientes :

a) Planteamiento del Problema. Esto es, reconocer los objetivos que se desean alcanzar, los medios de que se dispone, sus limitaciones, y toda información que pueda ayudar a la resolución del problema.

b) Formulación del Modelo. El propósito es buscar una representación simbólica del problema, para que pueda ser tratado matemáticamente.

c) Explotación del Modelo. Es la aplicación de diversas herramientas, como son las matemáticas, el análisis numérico, etc. que permiten mediante el modelo obtener cierta información elaborada a partir de la información básica.

d) Interpretación de la Solución. O sea, hallar lo que representan los resultados del modelo en la realidad. A veces este proceso es inmediato, pero otras requiere de un cuidadoso análisis.

En esta modelación de los problemas surge la teoría de la programación matemática, área en que se ubica el estudio presentado en esta tesis.

El problema general que estudia la programación matemática es el de hallar los valores extremos de una función real definida en un subconjunto de puntos del espacio euclidiano de n dimensiones, llamado el dominio.

De acuerdo a las características de la función y a las del dominio, el problema se ha clasificado y estudiado por separado, dando lugar a áreas como "programación lineal", "optimización combinatoria", etc.

El análisis matemático y la computación han contribuído a, que en algunos casos, se encuentren los valores extremos de una función, o bien, se aproximen satisfactoriamente. Sin embargo, en muchos otros, el problema sigue insoluble, o los métodos de solución son ineficientes. Tal es el caso de la optimización global, esto es, hallar el valor máximo absoluto cuando existen máximos relativos(*).

El caso más simple de este problema, se obtiene cuando el dominio es un segmento de recta. Este caso ha sido tratado por Bruno O. Shubert (ver [25]), y constituye el tema principal de esta tesis.

Estudiar el problema de una variable es importante, pues en el problema de optimización no lineal con varias variables, existen métodos, como el del gradiente reducido que usan optimizaciones de este tipo como parte fundamental del procedimiento (ver [3] o [19]); o en otros, el problema multivariado se aproxima mediante

---

(*) Para definición precisa ver pág. 110, Cap. VI de [22]. En el capítulo I de la presente tesis se trata este punto.

una transformación a un problema univariado (ver [8]).

El objetivo central de la presente tesis es elaborar una implantación eficiente del algoritmo de Shubert para un computador. El desarrollo se expone en cuatro capítulos.

En el primer capítulo se trata la representación matemática del problema de programación no lineal, enfocando el caso univariado, sus características y los algoritmos más importantes que han sido diseñados para resolverlo.

En el segundo capítulo se expone el método de Shubert para optimización global univariada, atendiendo a sus particularidades.

En el tercer capítulo se describen las estructuras de datos utilizadas en la implantación del algoritmo de Shubert y sus características. Además se presenta el desarrollo de la implantación.

En el último capítulo se presentan aplicaciones, ejemplos y conclusiones de la implantación del algoritmo de Shubert. Se muestran los resultados de las pruebas efectuadas.

Se anexan dos Apéndices. En el primero, se da una guía de operación para la utilización de la implantación realizada; en el segundo, se explican y listan los elementos simbólicos de las subrutinas elaboradas. Estas se encuentran documentadas con abundantes comentarios.

CAPITULO I

EL PROBLEMA DE PROGRAMACION NO LINEAL

Como se mencionó en la introducción, la idea central de esta tesis se refiere al desarrollo de una implantación en computadora de un método para resolver una determinada clase de problemas. El objetivo de este capítulo es introducir al lector con el tipo de problema al que nos referimos y dar un breve resumen de los métodos más usuales que han sido diseñados para resolverlos. Todo esto para posteriormente presentar la metodología que nos interesa, conocida como "Algoritmo de Shubert", debido a su autor.

I.1 FORMULACION (*)

Sea f una función definida en un conjunto $\Omega \subseteq R^n$ y que toma valores en $\mathbb{R}^1$.

Se dice que f tiene un máximo relativo (o local) en x° si existe $\varepsilon > 0$ tal que $f(x^\circ) \geq f(x)$ $\forall\, x \in \Omega$ y $\| x - x^\circ \| < \varepsilon$.

f tiene un máximo global en x° si $f(x^\circ) \geq f(x)$ $\forall\, x \in \Omega$

(*) Es fácil encontrar una extensa variedad de bibliografía sobre este aspecto, entre otros [14], [16], [22].
Se sobreentiende que el lector tiene al menos una ligera idea sobre el tema, por lo que se mencionan sólo conceptos básicos.

Se dice que f tiene un mínimo local o global si la función h = -f tiene un máximo local o global respectivamente.

Al proceso de encontrar los puntos máximos o mínimos se le denomina optimizar la función objetivo.

El problema general de programación matemática se describe como encontrar el óptimo global de una función real definida sobre $\mathbb{R}^n$.

Usualmente el conjunto Ω es definido mediante un conjunto de restricciones de la forma

$$CI_i \leq g_i(x) \leq CS_i \qquad \forall_i = 1, 2, \ldots, m$$

Donde CI_i, CS_i establecen cotas inferiores y superiores para los valores de la función $g_i: \mathbb{R}^n \to \mathbb{R}^1$, $i = 1, \ldots, m$

Como ya fue mencionado, dependiendo de las características que presenten las funciones f y g_i, $i = 1, 2, \ldots, m$, se han clasificado las distintas ramas de la programación matemática.

El caso al que se refiere la presente tesis es el de optimización no lineal en una variable sin restricciones, o sea, f es no lineal y $\Omega = [a, b] \subset R^1$.

I.2 ALGORITMO. EVALUACION DE EFICIENCIA

La idea de "algoritmo" esta íntimamente asociada con el proceso lógico que se sigue en la resolución de un problema. Sin embargo, a pesar de que la palabra proviene desde la historia antigua

de las matemáticas (*), no es sino hasta la primera mitad del siglo XX con la concepción de la máquina de Turing cuando el concepto es formalmente definido (**).

Debido a que esta definición se basa en la construcción de una máquina ideal y a que el análisis profundo del tema no entra en el campo de estudio del presente trabajo, nos limitaremos a la idea intuitiva que se desprende de este concepto.

Se entiende como algoritmo a una secuencia finita de instrucciones totalmente definidas para resolver un problema.

Cabe aclarar que el hecho de que la secuencia de instrucciones sea finita no implica que el proceso que generen lo sea, ni que su total definición necesariamente de lugar a un proceso no aleatorio.

El amplio desarrollo técnico, especialmente en computación ha permitido la implantación de algoritmos cada vez más complicados. Sin embargo, es deseable dada la capacidad finita de un computador y su uso más frecuente en todas las áreas, que los algoritmos utilizados sean eficientes.

A continuación se presentan dos conceptos, que originalmente fueron creados en el estudio de sucesiones, y son utilizados en la evaluación de eficiencia de algoritmos.

Se mencionan estos dos debido a su relación con el trabajo expuesto en esta tesis.

(*) Ver Knuth [18].

(**) Es entonces cuando se cuestiona todo el aspecto filosófico de la existencia de algoritmos para resolver problemas y la posibilidad de crear un modelo de máquina de Turing para cada uno. Ver [26].

I.3 DOS CONCEPTOS EN LA EVALUACION DE ALGORITMOS

I.3.1 Notación O

Definición (*): Dadas dos sucesiones $\{a_n\} \geq 0$, $\{b_n\} \geq 0$ $\forall n$ se escribe

$a_n = O(b_n)$ (se lee "a_n es de orden b_n")

Si existe $M > 0$ tal que $a_n \leq Mb_n$ $\forall n$

Esta definición significa que para cada valor de n la sucesión $\{Mb_n\}$ constituye una cota superior a $\{a_n\}$.

Este concepto es muy útil cuando el número de operaciones elementales que necesita un algoritmo para resolver un problema varía de acuerdo a los datos de entrada en cada caso específico estando, sin embargo, acotado superiormente por un valor que es función del tamaño del problema.

Entendiéndose como "tamaño del problema" a una función de la cardinalidad del conjunto de variables que utiliza el algoritmo.

Ejemplo:

Sea $A \equiv (a_{ij})$ una matriz de tamaño m x m. Se desea construir otra matriz $B \equiv (b_{ij})$, de igual dimensión, asociada a A mediante la relación.

(*) La Definición fue tomada de [2] y se restringió al caso cuando $\{a_n\} \geq 0$ y $\{b_n\} \geq 0$, que es el que nos interesa.

$$b_{ij} = \begin{cases} f(a_{ij}) & \text{si } a_{ij} > c \\ 1 & \text{si } a_{ij} \leq c \end{cases}$$

Supóngase que en la evaluación de f se realizan n operaciones elementales, y que no se consideran las acciones de asignación ni comparación.

Este proceso es de $O(m^2)$ y la cota superior al número de operaciones es nm^2.

Si además se conoce que la matriz A es generada de tal manera que la probabilidad de que una entrada sea mayor que la constante c es:

$$P\{a_{ij} > c\} = \frac{1}{m} \qquad \forall i = 1, 2, \ldots, m \quad \forall j = 1, 2, \ldots, m$$

Esto implica que la variable aleatoria "número de entradas mayores a c" se distribuye binomialmente con media m. En este caso es posible evaluar el número promedio de operaciones al realizar repetidamente el proceso: nm.

En la práctica se utilizan el análisis del peor caso y del caso promedio para evaluar algoritmos dependiendo de las circunstancias del problema.

I.3.2 Orden y Razón De Convergencia

Definición (*): Sea $\{r_k\}_{k=0}^{\infty}$ una sucesión que converge a r^*. El orden de convergencia de $\{r_k\}$ es el máximo valor positivo p - tal que

(*) Presupone conocimiento del concepto de convergencia y de límte superior de convergencia para sucesiones.

$$0 \leq \lim_{k \to \infty} \sup \frac{|r_{k+1} - r^*|}{|r_k - r^*|^p} = \beta < \infty$$

β se define como la razón de convergencia.

Este concepto es ampliamente utilizado en el análisis numérico donde la generalidad de los métodos para hallar la solución a un problema son iterativos y convergen cuando el número de iteraciones tiende a infinito. Aquí los valores obtenidos en cada iteración pueden ser vistos bajo el concepto de sucesión.

Ejemplo:

Sea la sucesión r_k definida como $r_k = a^k$ donde $0 < a < 1$, que converge a cero.

Si obtenemos

$$\lim_{k \to \infty} \frac{|a^{k+1} - 0|}{|a^k - 0|^p} = \lim_{k \to \infty} \frac{a^{k+1}}{a^{pk}}$$

Se observa que $p = 1$ es el máximo valor para el cual el límite esta acotado. Por lo tanto, esta sucesión es de orden de convergencia igual a 1 con razón de convergencia a. (valor del límite cuando $p = 1$).

Si la sucesión $\{r_k\}$ converge a r^* y

$$\lim_{k \to \infty} \frac{|r_{k+1} - r^*|}{|r_k - r^*|} = \beta < 1$$

se dice que converge linealmente. En el caso extremo cuando $\beta = 0$, la sucesión converge superlinealmente. De aquí, cualquier sucesión cuyo orden de convergencia sea mayor que la unidad, converge superlinealmente.

I.4 FUNCIONES DE LIPSCHITZ

El método de Shubert sirve para resolver problemas de optimización no lineal en una variable. El algoritmo requiere que la función objetivo cumpla con la propiedad de Lipschitz. A continuación se define esta propiedad y algunos resultados relativos.

Se dice que la función $f: S \subseteq R^m \longrightarrow R^n$ cumple la propiedad de Lipschitz o es una función de Lipschitz si

Para toda $x_1, x_2 \in S$ Existe $C \in R$ tal que

$$||f(x_1) - f(x_2)|| \leq C||x_1 - x_2||.$$

En lo siguiente se denotará a $\mathcal{L}_C(S)$ como el conjunto de funciones sobre S que cumplen con la propiedad antes mencionada, dónde a C se le denomina constante de Lipschitz.

Algunas propiedades que se satisfacen entre funciones de Lipschitz son las siguientes(*):

(P1) Sea $f \in \mathcal{L}_{c_1}(S)$: $S \subseteq R^m \longrightarrow R^n$ y

$g \in \mathcal{L}_{c_2}(S)$: $S \subseteq R^m \longrightarrow R^n$ entonces

Para toda $\alpha, \beta \in R$ $\alpha f + \beta g \in \mathcal{L}_{|\alpha|c_1+|\beta|c_2}(S)$.

(*) Las demostraciones son sencillas, de cualquier forma pueden consultarse en [8]

(P2) Sea $f \in \mathcal{L}_C(S)$: $S \subseteq R^1 \longrightarrow R^1$ entonces

$$|f| \in \mathcal{L}_C(S)$$

(P3) Sea $f \in \mathcal{L}_{C_1}(S)$: $S \subseteq R^m \longrightarrow T \subseteq R^n$ y

$g \in \mathcal{L}_{C_2}(T)$: $T \subseteq R^n \longrightarrow R^k$ entonces

$$f \circ g \in \mathcal{L}_{C_1 C_2}$$

(P4) Sea f: $S \subseteq R^m \dashrightarrow R^n$. Si S es compacto y $f \in C^1(S)$ (*), entonces

$$f \in \mathcal{L}_C(S) \quad \text{para algún } C.$$

(P5) Sea $f_i \in \mathcal{L}_{C_i}(S)$: $S \subseteq R^m \longrightarrow R^n$, $i=1,\ldots,k$ un conjunto de funciones de Lipschitz.

Sea $f^\circ = \max \{f_i: i=1,\ldots,k\}$

$f_\circ = \min \{f_i: i=1,\ldots,k\}$ entonces

$$f^\circ \in \mathcal{L}_C(S) \quad \text{y}$$

$$f_\circ \in \mathcal{L}_C(S)$$

donde $C = \max \{C_i: i=1,\ldots,k\}$.

I.5 METODOS MAS USUALES PARA RESOLVER EL PROBLEMA DE OPTIMIZACION NO LINEAL EN UNA VARIABLE.

El problema descrito en I.1, cuando se restringe al caso univariado, parece ser trivial. Se han diseñado diversos métodos para resolverlo utilizando propiedades muy variadas de las funciones objetivo, sin embargo ninguno de estos se considera eficiente en todos los casos.

(*) $C^1(S)$ denota las funciones cuya primer derivada es continua sobre S.

Sin observar detenidamente el asunto, parece ser que la mejor forma de resolver el problema de optimización univariado sin restricciones es analíticamente. Esto es, encontrar los puntos críticos de la función objetivo mediante los ceros de su derivada (si la tiene) y analizarlos para deducir cuales de ellos son puntos óptimos.

En los problemas que se sucitan en la práctica es muy usual que la función objetivo no sea derivable; que su representación matemática sea muy compleja, y requiera mucho tiempo analizarla; o bien sea necesario obtener los puntos óptimos de un conjunto de problemas y el trabajo que esto representa puede ser excesivo. Por esta razón se han ideado métodos que, aprovechando las ventajas del procesamiento de información en computadoras ayudan a resolver el problema. Entre los más sencillos están aquellos cuya filosofía es servir de apoyo al análisis matemático de la función objetivo. Por ejemplo, si la función es unimodal, el cero de su derivada es fácil de determinar utilizando métodos como el de Newton-Raphson o Regula Falsi [9]; o bien quizás sea conveniente estimar la derivada de la función objetivo mediante aproximaciones sucesivas.

Usualmente en paquetes técnicos de optimización son utilizados algoritmos cuya eficiencia es reconocida, pero sólo obtienen puntos óptimos relativos.

Entre los algoritmos de optimización local se destacan los de búsqueda secuencial. Con este nombre se conocen aquellos que generan una sucesión de puntos, donde la obtención de un nuevo punto depende de los anteriores; además teoricamente, la sucesión converge al óptimo si el número de términos tiende a infinito.

Entre estos algoritmos están la búsqueda por partición simétrica en dos puntos, método muy sencillo requiriendo unicamente continuidad en la función objetivo; la búsqueda Fibonacci, dónde se

minimiza el tamaño del intervalo de incertidumbre dependiendo de la tolerancia al error y de la cantidad de evaluaciones de la función fijadas de antemano; o bien la búsqueda de la razón dorada, cuya eficiencia es un poco menor al anterior, pero más sencillo de implantar.

El problema de optimización en una linea se complica cuando se desea encontrar el punto óptimo global en un intervalo. Se han diseñado técnicas diversas, entre ellas cabe mencionar a la más elemental, conocida como grid search cuya idea es evaluar la función en puntos equidistantes en el intervalo de análisis y la estimación del óptimo se obtiene con la máxima evaluación de la función obtenida. Si además la función cumple la propiedad de Lipschitz, esto da la distancia necesaria entre los puntos para obtener un error no mayor que una tolerancia predeterminada [10].

Entre los algoritmos de búsqueda secuencial para optimización global están el de Shubert, que será descrito en el siguiente capítulo con más detalle por ser el tema fundamental de esta tesis; y el de Evtushenko [10], que al igual que el anterior explota la propiedad de Lipschitz de la función objetivo.

Los métodos probabilísticos, por lo general no exigen muchas propiedades en la función analizada. El criterio de este tipo de algoritmos es generar una muestra aleatoria de puntos del dominio de la función objetivo, que permite hallar un intervalo dónde con una determinada probabilidad se localiza el óptimo. Además se garantiza que dicha probabilidad tiende a uno al incrementar el tamaño de la muestra.

La desventaja de este tipo de métodos es la incertidumbre en los resultados al verse como una probabilidad. Entre los autores que han trabajado en esta área es posible mencionar a R. Wets - F. Solís [28], Boender et al [4] o Chichinadze [10] entre otros.

La metodología del análisis de intervalos aplicada a problemas de optimización global parece dar buenos resultados, sin embargo exige que la función sea doblemente derivable. En [15] Hansen prevee presentar posteriormente el método modificado para tratar una clase más general de funciones. El procedimiento de esta metodología es descartar subintervalos, para hacer más pequeña la región de incertidumbre, analizándola más detalladamente hasta obtener intervalos-solución.

Para terminar con este breve resumen de métodos de optimización global, es conveniente citar los de Branin [7] , Goldstein y Price [13] y muy particularmente el de Velasco-Levy y Montalvo [27]. Estos métodos han sido diseñados para resolver problemas de varias variables y frecuentemente son muy ineficientes cuando se particularizan al caso univariado. El método de Velasco-Levy y Montalvo quizás pudiera aplicarse con éxito a este caso, aunque los resultados reportados en [27] no son muy prometedores.

CAPITULO II

EL ALGORITMO DE SHUBERT

En este capítulo se presenta el algoritmo de Shubert en su descripción original, así como los rasgos más característicos de su desarrollo.

Este algoritmo fue diseñado para resolver el problema de optimización global en una variable, requiriendo unicamente que la función objetivo cumpla con la propiedad de Lipschitz y que la constante que la determina sea conocida.

El algoritmo lo describió Bruno O. Shubert originalmente [25] de la siguiente forma

Sea $f \in \mathcal{L}_C(S)$: $S = [a,b] \subset R^1 \longrightarrow R^1$

Sea ϕ el valor máximo que toma f en S, y Φ el conjunto de abscisas donde f alcanza el óptimo, esto es

$$\phi = \max_{x \in S} f(x)$$

$$\Phi = \{x: x \in S, f(x)=\phi\}$$

Se define recursivamente la secuencia de puntos $x_0, x_1, x_2, \ldots$ en S como sigue

$$x_0 \in S$$

$$x_{n+1} \text{ tal que } F_n(x_{n+1}) = M_n \tag{1}$$

donde $$M_n = \max_{x \in S} F_n(x) \tag{2}$$

y $$F_n(x) = \min_{k=0,1,..,n} \{f(x_k) + C|x-x_k|\} \tag{3}$$

Shubert demuestra que si n tiende a infinito, $f(x_n)$ y M_n tienden ambas a ϕ; que el $\inf_{x \in \phi} |x-x_n|$ tiende a cero y que además el método converge en el peor caso aritméticamente, esto es, x_i es $O(1/n)$. Esto sucede cuando la función objetivo es una constante.

La idea fundamental del algoritmo es acotar superiormente la curva representada por la función objetivo, mediante una función lineal por pedazos. Esta función lineal por pedazos cambia en cada iteración y está descrita por la relación (3).

El método establece que

$$F_i(x) \geq F_{i+1}(x) \geq f(x) \quad \text{Para todo } x \in [a,b] \tag{4}$$

y que la sucesión de puntos máximos $\{M_n\}$ de $F_n(x)$ converge al valor máximo de $f(x)$ conforme n tiende a infinito.

Para probar que (4) se satisface, obsérvese en primer lugar que los mínimos locales de $F_n(x)$ se ubican en los puntos $x_i : i=1,..,n$,

en donde se ha evaluado la función objetivo y que $F_n(x_i) = f(x_i)$ para $i = 1,\ldots,n$. Además obsérvese que para cualquier otro punto $x \in [a,b]$ se tiene que

$$|f(x) - f(x_i)| \leq C\,|x - x_i| \qquad \text{Para toda } i.$$

y por lo tanto

$$f(x) \leq f(x_i) + C\,|x - x_i| \qquad \text{Para toda } i.$$

y consecuentemente

$$f(x) \leq \min_{i=1,..,n} \{f(x_i) + C\,|x - x_i|\} = F_n(x).$$

La Fig. II.1 representa un ejemplo de las primeras cinco iteraciones del algoritmo. Las funciones $F_i(x)$: $i=0,..,3$ se grafican con linea punteada y $F_4(x)$ con linea continua. Además se señalan los puntos M_i: $i=0,..,4$ y la secuencia x_i: $i=1,..,5$.

Existen varias maneras de tener totalmente definida una función lineal por pedazos. En el caso de $F_n(x)$ la situación es simple pues el valor de la pendiente de cada segmento de recta es el mismo sólo que alternando su signo. Para manipular esta función lineal por pedazos conviene representarla por el conjunto de puntos D_n donde posee máximos relativos, esto es

$$D_n = \{(t_i,z_i):\ (t_i,z_i) \in F_n(x) \text{ y Existe } \delta > 0 \text{ tal que } (t_i,z_i) > F_n(x) \text{ si } \delta > |t_i - x| > 0\ \}$$

De esta forma, tomando el $\max\ z_i$: $i=1,\ldots,n$ se obtiene el valor M_n mencionado en (2).

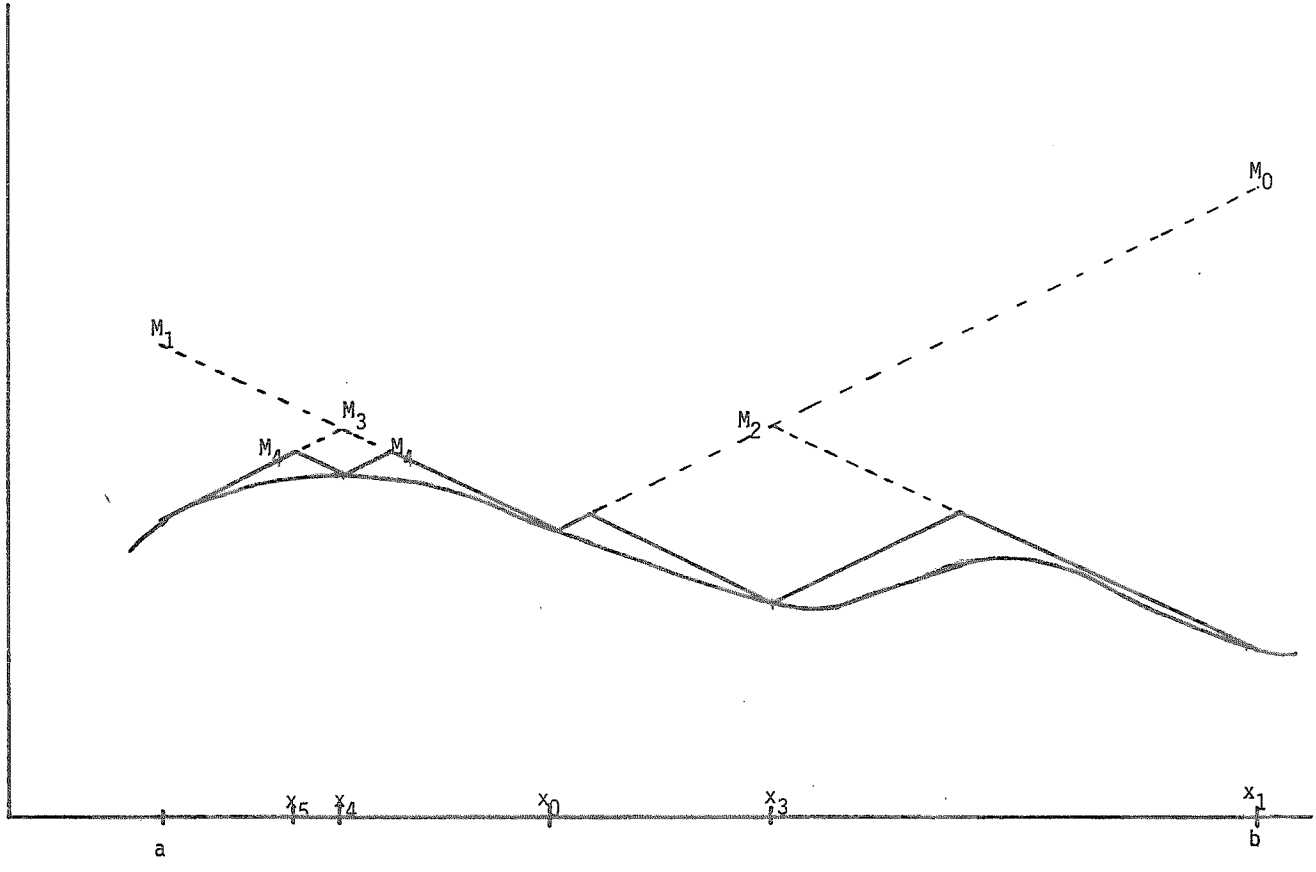


Fig. II.1

Esta representación de $F_n(x)$ es conveniente ya que podemos construir facilmente D_{n+1}, y así representar F_{n+1}, de la siguiente manera:

Sea $x_{n+1} = t_k$ donde k es tal que $z_k = M_n$, entonces D_{n+1} se obtiene descartando el punto (t_k, z_k) de D_n y añadiendo dos nuevos puntos (t_{n+1}, z_{n+1}), (t_{n+2}, z_{n+2}) provenientes de las intersecciones de las rectas

$y = c(x-t_k) + z_k$ con $y = -c(x-t_k) + f(x_k)$

y

$y = -c(x-t_k) + z_k$ con $y = c(x-t_k) + f(x_k)$

respectivamente. Es fácil demostrar que

$$t_{n+1} = t_k - \frac{z_k - f(x_k)}{2c}$$

$$t_{n+2} = t_k + \frac{z_k - f(x_k)}{2c}$$

$$z_{n+1} = z_{n+2} = \frac{z_k + f(x_k)}{2} .$$

En la práctica el algoritmo termina cuando

$$\left| \frac{M_n - \phi_n}{\phi_n} \right| < \varepsilon$$

donde $\phi_n = \max_{k=0,..,n} \{f(x_k)\}$

y 'ε' es un valor pequeño que denota una tolerancia al error permitido.

Cuando el algoritmo termina, queda un subconjunto de puntos D_n' tal que

$$D_n \supseteq D_n' = \{(t_j,z_j): (t_j,z_j) \in D_n \ , \ \phi_n < z_j \leq M_n\}.$$

Si se intersecta la recta $y = \phi_n$ con

$$y = c(x-t_j) + z_j \qquad y \qquad y = -c(x-t_j) + z_j$$

donde $(t_j,z_j) \in D_n'$, esto genera un conjunto de subintervalos de la forma

$$\left(t_j - \frac{z_j - \phi_n}{c} \ , \ t_j + \frac{z_j + \phi_n}{c} \right) \quad \text{con } (t_j,z_j) \in D_n'.$$

a los cuales Shubert llama intervalos de incertidumbre. Es claro por construcción, que el óptimo global debe estar en algunos de estos intervalos. Shubert prueba que cuando n tiende a infinito la unión de estos intervalos tiende a Φ.

Lo que Shubert propone es agrupar estos intervalos si la distancia que los separa es menor a una tolerancia pequeña. Obteniendo de esta forma intervalos más grandes, pero un número menor de ellos, lo cual facilita el análisis de los resultados.

El siguiente es un resumen del método de Shubert suficien-

temente conciso como para ser programado en un computador:

Parámetros de entrada

$S = [a,b]$ = intervalo de análisis de la función objetivo.

x_0 = semilla con la que inicia el algoritmo. $a < x_0 < b$

f = función objetivo, y su constante de Lipschitz C.

ε = valor pequeño de tolerancia al error en la estimación del valor máximo de la función objetivo.

δ = valor pequeño de tolerancia al agrupar los intervalos de incertidumbre.

(1) $i = 0$

(2) $F_i(x) = \min_{k=0,..,i} \{f(x_k) + C\,|x - x_k|\}$

(3) $M_i = \max_{x \in S} F_i(x)$

(4) $\phi_i = \max_{k=0,..,i} f(x_k)$

(5) Si $\left|\frac{M_i - \phi_i}{\phi_i}\right| < \varepsilon$ ir a (9)

(6) $x_{i+1} = x_m$ tal que $F_i(x_m) = M_i$

(7) $i = i + 1$

(8) ir a (2)

(9) Obtener los intervalos $Ij = \left(t_j - \frac{z_j - \phi_n}{C},\ t_j + \frac{z_j - \phi_n}{C}\right)$ tal que $0 \leqslant j \leqslant n$ y $\phi_n < z_j \leqslant M_i$.

(10) Ordenar I_j tal que $t_{j+1} > t_j$ Para toda j.

(11) Revisar I_j Para toda j, y si $\left|t_{j+1} - \frac{z_{j+1} - \phi_i}{C} - t_j - \frac{z_j - \phi_i}{C}\right| < \delta$

crear el nuevo intervalo

$$I_j' = \left(t_j - \frac{z_j - \phi_i}{C} \;,\; t_{j+1} + \frac{z_{j+1} - \phi_i}{C} \right)$$

Valores de salida

ϕ_i = máxima evaluación de la función objetivo.
x_ϕ = abscisa dónde la función tomó el valor ϕ_i.
I_j' Para toda j = intervalos de incertidumbre.

CAPITULO III

IMPLANTACION DEL ALGORITMO DE SHUBERT

III.1 ESTRUCTURAS DE DATOS UTILIZADAS

El algoritmo de Shubert ya ha sido programado anteriormente(*). Sin embargo la idea de efectuar una nueva implantación surgió al observar que existe una estructura de datos, conocida como heap, con propiedades interesantes y que se adapta muy bien a la filosofía de este algoritmo.

Para la descripción de la implantación realizada es necesario definir algunos conceptos básicos relacionados con el tema. En lo que sigue se presupone que el lector está familiarizado con los conceptos elementales de listas, colas, pilas y arreglos en el estudio de estructuras de datos en el proceso de información en computadoras(**)

(*) Teniendo como objetivo el método en sí [25] y como herramienta en otro estudio [8].

(**) Todos estos conceptos se definen en cualquier libro de estructuras de datos en el diseño de algoritmos computacionales, ver por ejemplo [1] o [18].

III.1.1 Arboles. Conceptos Relacionados (*)

Se le llama Gráfica G a una terna ordenada

$$G \equiv (V(G), A(G), \Psi_G)$$

que consiste de un conjunto no vacío de vértices o nodos V(G), un conjunto de aristas o arcos A(G), disjunto del conjunto V(G), y una función de incidencia Ψ_G que le asocia a cada elemento de A(G) dos elementos (no necesariamente distintos) de V(G).

Sean $V(G) = \{v_1, v_2, \ldots, v_n\}$

$A(G) = \{a_1, a_2, \ldots, a_m\}$

Usualmente Ψ_G se describe como un conjunto de ternas $(v_i\ a_j\ v_k)$ o pares $(v_i\ v_k)$ que especifican la asociación entre los vértices v_i y v_k, donde en el primer caso se concede importancia al arco que los relaciona. En lo que sigue se utilizará la segunda nomenclatura.

Se diferencía arco de arista entendiendo que el primero lleva implícita una dirección, esto es, el par o terna que lo definen están ordenados.

Si existe en Ψ_G una secuencia $(v_1v_2),(v_2v_3),\ldots,(v_{k-1}v_k)$ se dice que G tiene un camino de v_1 a v_k de longitud k-1.

Ciclo es un camino cuyos vértices inicial y final son el mismo, además los vértices y aristas interiores son distintos.

(*) Se exponen sólo los conceptos que se consideran necesarios para comprender los temas posteriores. Para mayor información ver [5] o [18].

Gráfica Acíclica es aquella que no posee ningún ciclo. Gráfica Conexa es aquella en la que para todo par de sus vértices existe al menos un camino.

Con los conceptos anteriores es posible definir Arbol como una gráfica acíclica conexa.

A continuación se presentan conceptos que se refieren a relaciones entre nodos de árboles.

Dado un árbol A y un nodo específico $r \in V(A)$; que llamaremos la raíz de A, podemos orientar A de forma tal que para cada nodo v_j exista a lo más un arco en A de la forma $(v_i v_j)$ y que no exista ninguno de la forma $(v_k r)$. Al árbol A con tal orientación le llamaremos un árbol enraízado en r. En lo que sigue sólo se considerarán árboles enraízados.

Si existe en A el arco (v_i, v_j), v_i el el padre de v_j y v_j es un hijo de v_i. Si existe un camino que una v_i con v_j, v_i es ascendiente de v_j, y v_j es descendiente de v_i.

Hoja es un nodo sin descendientes

Arbol Binario es aquel donde cada vértice tiene a lo más dos hijos. Si además estos hijos tienen una etiqueta que los identifica, se les llama hijo izquierdo y derecho respectivamente. En todo árbol binario etiquetado es posible definir para cada vértice v_i un subárbol (posiblemente vacío) izquierdo y otro derecho, cuyas raíces son respectivamente los nodos hijos izquierdo y derecho de v_i.

Dos casos particulares de árboles binarios fueron utilizados en la implantación del algoritmo de Shubert. Antes de describir los conviene mencionar que en la información representada por los vér-

tices de los árboles existe una relación de orden, esto es

Se dice que existe un orden parcial ($\preccurlyeq$) entre los elementos de un conjunto S si satisfacen las siguientes propiedades

Sean $x,y,z \in S$

i) si $x \preccurlyeq y$, $y \preccurlyeq z$ entonces $x \preccurlyeq z$

ii) si $x \preccurlyeq y$, $y \preccurlyeq x$ entonces $x = y$

iii) $x \preccurlyeq x$

la notación $x \preccurlyeq y$ se lee x precede o es equivalente a y. Y $x \prec y$ significa x precede estrictamente a y.

Un orden total en S es un orden parcial que puede ser establecido para cualquier par de elementos de S.

ej:
Si se considera en la Fig. III.1 que la relación $a_i \preccurlyeq a_j$ se expresa como $a_i \longrightarrow a_j$, la figura III.1 (a) representa un orden parcial y III.1 (b) un orden total. Como se observa (b) implica (a) y no inversamente.

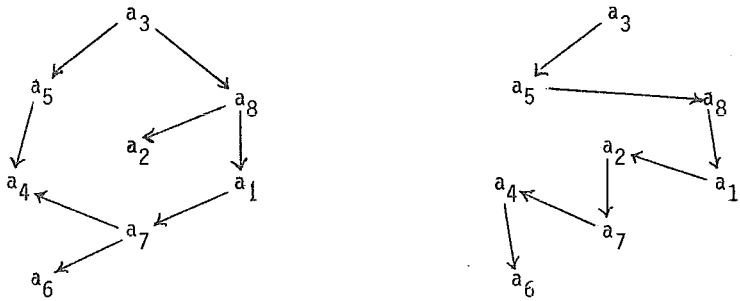


Fig. III.1

III.1.2 Arbol Binario en Inorder

Un árbol binario en inorder es un árbol binario A para el cual se ha definido un orden total en los nodos del mismo y que puede describirse como sigue:

(1) Sean $u,v \in V(A)$

$u \preceq v$ $\Longleftrightarrow$ Existe un subárbol de A con raíz p tal que se da uno de los tres casos

i) u = p y v está en el subárbol derecho de p ó

ii) v = p y u está en el subárbol izquierdo de p ó

iii) u está en el subarbol izquierdo de p y v en el derecho.

Los nodos de un árbol con esta estructura se pueden recorrer en orden creciente mediante el siguiente algoritmo que se define recursivamente

Recorrido en In order

- Se recorre en inorder el subárbol izquierdo de la raíz.
- Se visita la raíz.
- Se recorre en inorder el subárbol derecho de la raíz.

La importancia de esta estructura en computación radica en el hecho de que sirve para ordenar (es decir listar de acuerdo a un orden) un conjunto de elementos. La idea básica es construir un árbol binario donde cada nodo representa un elemento del conjunto y el orden entre estos satisface la estructura mencionada en (1). Teniendo el árbol de esta manera, basta con recorrerlo en inorder para tener ordenado el conjunto.

III.1.3 Estructura Heap

Un Heap es un árbol binario H para el cual se define un orden parcial entre sus nodos de la siguiente manera:

(2) Sean $u,v \in V(H)$, si v es descendiente de $u \Longrightarrow u \preccurlyeq v$.

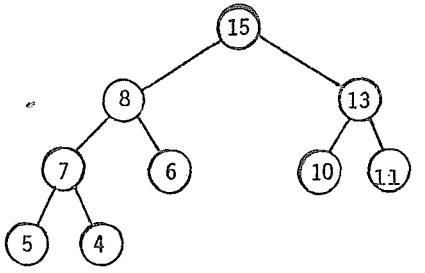
En la práctica, por facilidad de manejo se construye el heap etiquetando sus vértices con los números naturales de la siguiente manera

(3) i) la raíz del árbol tiene como etiqueta al 1.

ii) para todo nodo del árbol con etiqueta "i" su hijo izquierdo tiene como etiqueta a "2i" y el derecho a "2i + 1".

iii) Si existe la etiqueta "j" existen todas las etiquetas "i" tal que $i < j$, $i,j \in N$.

La importancia de esta estructura consiste en que si se representa un conjunto ordenado S mediante el heap H, se tiene en la raíz de H al mayor elemento de S. Además el heap etiquetado como se describe en (3) es muy fácil de manejar en memoria de una computadora cuando la etiqueta de un vértice representa la posición en un arreglo (ver ej. en Fig. III.2)



Ejemplo de un heap con 9 vértices.

i	A[i]
1	15
2	8
3	13
4	7
5	6
6	10
7	11
8	5
9	4

Representación del heap adjunto en un arreglo de 9 posiciones.

Fig. III.2

En la implantación de la metodología de Shubert, se utiliza un heap en el cual se cambia el valor de la raíz y se añade un nuevo nodo al final en cada iteración. Por esta razón es necesario verificar que se sigue manteniendo la estructura heap, y ésto es sencillo con los siguientes algoritmos:

- Nuevo valor en la raíz
  1) i = 1
  2) Se compara A[i] con A[2i] y A[2i+1], si es menor que alguno de ellos se intercambia con el mayor. En caso contrario termina.
  3) Si A[i] se intercambió con A[2i] entonces i = 2i. En caso contrario i = 2i+1
  4) ir a (2).

- inserción de un nodo al final del arreglo en la posicion n+1
  1) i = $\lfloor n+1/2 \rfloor$ (*)
  2) Se compara A[i] con A[2i] y A[2i+1], si es menor que alguno de ellos se intercambia con el mayor. En caso contrario termina.
  3) i = $\lfloor i/2 \rfloor$
  4) ir a (2).

Como se observa la única diferencia entre estos dos algoritmos es que el primero revisa a los descendientes de la raíz y el segundo a los ascendientes del último nodo.

(*) $\lfloor x \rfloor$ significa el máximo entero inferior a x.

III.2 DESCRIPCION DE LA IMPLANTACION DEL ALGORITMO DE SHUBERT (*)

Para facilitar la descripción de la implantación del algoritmo de Shubert, ésta se dividió en 8 etapas

1) Utilización del heap
2) Inicialización del algoritmo
3) Generación de nuevos nodos
4) Conservación del heap
5) Estadísticas del heap
6) Selección de intervalos de incertidumbre
7) Criterios de Terminación
8) Impresión de resultados

En lo que sigue de este capítulo se describirá cada etapa.

III.2.1 Utilización del heap

La estructura heap se utilizó para almacenar al conjunto de puntos D_n mencionado en la descripción del algoritmo de Shubert. Los valores z_i fueron utilizados para establecer el orden entre los nodos del árbol binario. De esta manera, si se preserva la estructura heap en cada iteración del algoritmo, la raíz contendrá al valor 'M_n' ya descrito.

En el algoritmo de Shubert, al comenzar la n+1 ésima iteración, se generan dos nuevos puntos (t_{n+1}, z_{n+1}), (t_{n+2}, z_{n+2}). Añadiéndolos en el heap con los almacenados anteriormente, se logra la representación de la nueva función lineal por pedazos $F_{n+1}(x)$.

(*) En esta sección se utilizarán conceptos y nomenclatura definidos anteriormente en este trabajo.

Para almacenarlos se utilizaron las propiedades de que $z_{n+1} = z_{n+2}$ y de que t_{n+1}, t_{n+2} son contiguos al ordenar crecientemente t_i: i=1,...,n+2. Debido a esto ambos puntos pueden ocupar un sólo nodo del heap y su representación en un arreglo sólo requiere tres posiciones:

$$Y[i] = z_{n+1} = z_{n+2}$$
$$XI[i] = t_{n+1}$$
$$XD[i] = t_{n+2}$$

A los valores contenidos en la i-ésima posición de los arreglos XI, XD se les llama abscisa izquierda y derecha respectivamente del i-ésimo nodo.

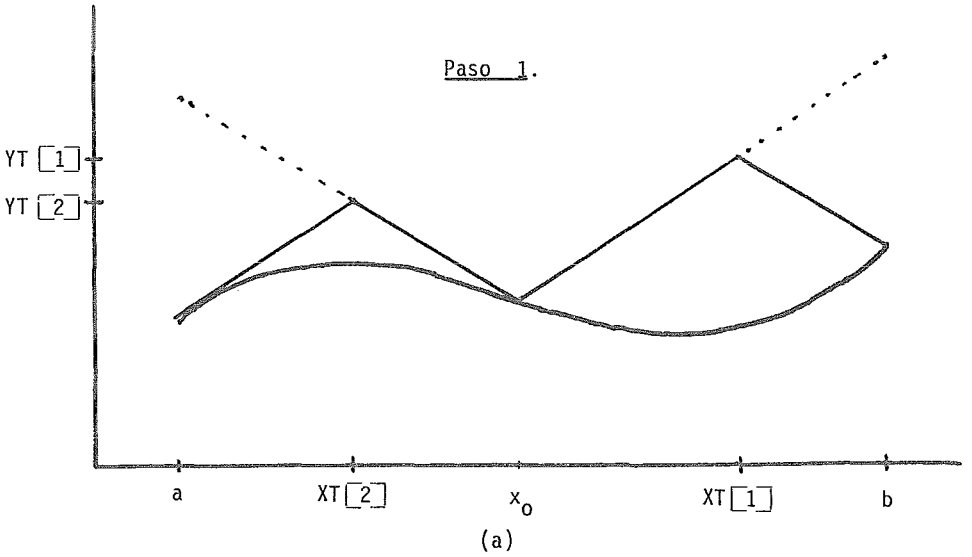
III.2.2 Inicialización del Algoritmo

En la sección anterior se menciona que en cada nueva iteración del algoritmo se generan dos nuevos puntos para definir a la siguiente función lineal por pedazos. Estos puntos se caracterizan por ser iguales en su ordenada, por lo cual es posible representarlos mediante un sólo nodo en el heap.

Sin embargo, esta propiedad no se satisface al inicio del algoritmo (Ver Fig. III.3 (a), (b)). Debido a esto, en la implantación la primera iteración consistió en dos pasos para dejar la función lineal por pedazos representada mediante dos nodos del heap, cada uno con su respectiva abscisa izquierda y derecha.

Es obvio que los puntos generados en el paso 1 de la primera iteración se obtienen mediante las siguientes relaciones

$$XT[1] = \frac{cb + f(b) + cx_0 - f(x_0)}{2c}$$



(a)

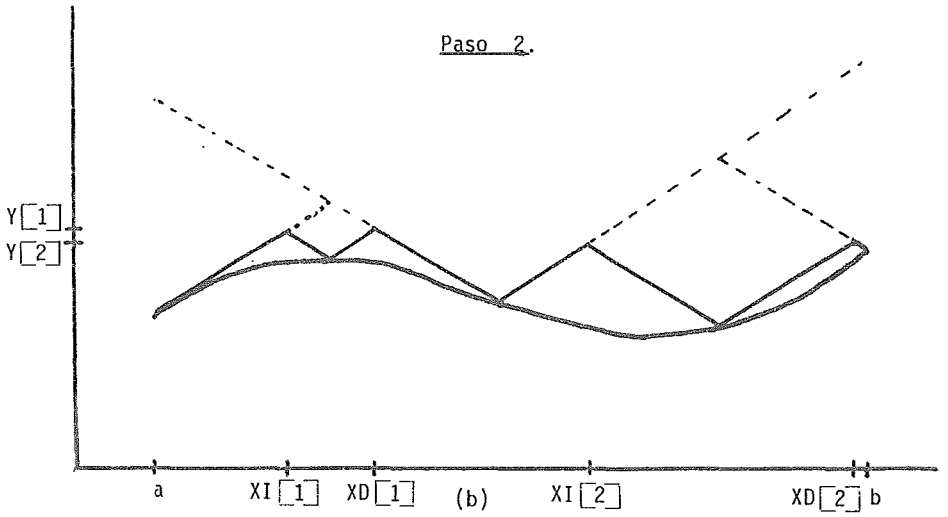


Fig. III.3

$$YT[1] = \frac{cb + f(b) - cx_0 - f(x_0)}{2c}$$

$$XT[2] = \frac{cx_0 + f(x_0) + ca - f(a)}{2}$$

$$YT[2] = \frac{cx_0 + f(x_0) - ca + f(a)}{2}$$

Donde c = constante de Lipschitz
a,b = cotas del intervalo de análisis
x_0 = punto inicial del algoritmo
f(x) = valor de la evaluación de la función objetivo en x.

Los resultados de estas relaciones se representaron en los arreglos XT[i], YT[i], i=1,2. Queriendo expresar con la terminación 'T' que unicamente son utilizados en el paso 1 de la primera iteración y por tanto temporales.

A partir de XT, YT se obtienen los puntos del paso 2 de la primera iteración mediante las siguientes relaciones

$$XI[j] = \frac{2c\ XT[j] - YT[j] + f(XT[j])}{2c} \qquad j = 1,2$$

$$XD[j] = \frac{2c\ XT[j] + YT[j] + f(XT[j])}{2c} \qquad j = 1,2$$

$$Y[j] = f(XT[j]) + \frac{YT[j] - f(XT[j])}{2} \qquad j = 1,2$$

Estas últimas fórmulas son las utilizadas a partir de la siguiente iteración para generar un nuevo nodo del heap.

III.2.3 Generación de Nuevos Nodos

El proceso que se describe en esta sección supone que los arreglos Y, XI, XD mantienen la estructura heap con el orden establecido por los valores almacenados en Y[i]. Con esto, en cada iteración Y[1] contiene al valor 'M_n' y XI[1], XD[1] las dos abscisas donde la función lineal por pedazos alcanza dicho valor.

Por generación de nuevos nodos deberá entenderse el cálculo de los nuevos puntos que definen a la nueva función lineal por pedazos en cada iteración. Como ya se ha visto, esta función en la j-ésima iteración se define mediante el conjunto D_j, y en la siguiente iteración el conjunto D_{j+1} se obtiene como

$$D_{j+1} = \{(t_i,z_i): (t_i,z_i) \in (D_j \cup N_{j+1})\} - \{(t_k,z_k)\}$$

Donde

$$N_{j+1} = \{(t_j,z_j): t_j = t_k \pm \frac{z_k - f(x_k)}{2c},\ z_j = \frac{z_k + f(x_k)}{2}\}$$

y 'k' es tal que $z_k = M_j$.

Operativamente, desde la segunda iteración existen al menos dos puntos (t_k,z_k), pero puede suceder que existan más. En este caso se toma algún par de ellos (por comodidad los que se representan en la raíz del heap), y en la siguiente iteración otros dos,etc.

En la implantación realizada, se substituye la raíz por el mayor (en su ordenada) de los nuevos puntos y el menor se añade al final del arreglo que representa al heap. Además se deja la opción al usuario para que se descarte al menor de los nuevos puntos si su ordenada es menor al máximo valor alcanzado en las evaluaciones de la función objetivo.

III.2.4 Conservación del Heap

En la implantación del algoritmo de Shubert, como fue mencionado en la sección anterior, los vértices nuevos que se generan en cada iteración se almacenan uno en la raíz y el otro al final del árbol. Con esto, todos los demás nodos conservan la estructura heap y sólo es necesario reacomodar los dos nuevos vértices.

En la implantación existe una subrutina que realiza este reacomodamiento (ver apéndice B) basada en los algoritmos presentados en la sección III.1.3. Sólo necesita un parámetro de entrada que le indica si debe revisar los descendientes de la raíz o los ascendientes del último vértice.

III.2.5 Estadísticas del Heap

En la planeación de la implantación por realizar, se pensó que la cantidad de nodos que forman el heap frecuentemente sería lo suficientemente grande como para hacer ineficiente su manejo. Ante este problema se comenzó a buscar un criterio mediante el cual en la implantación se decidiera si era necesario hacer una clasificación de los nodos y desechar los que fuesen menores al valor máximo alcanzado en las evaluaciones anteriores de la función objetivo.

Al trabajar en esta idea se elaboró una subrutina que obtiene la media , varianza y desviación estándar de los valores almacenados en el heap. Sin embargo al realizar las pruebas de la implantación efectuada, se observó que era muy difícil encontrar en la práctica un tipo de problema que justificase hacer la clasificación (en los problemas resueltos, el árbol no alcanzó los 300 nodos). Tampoco se creyó conveniente desechar esta subrutina, pues las es-

tadísticas que genera podrían ser utilizadas con otro objetivo (por ejemplo evaluar la eficiencia del algoritmo). (*)

III.2.6 Selección de Intervalos de Incertidumbre

Al terminar el algoritmo en la n-ésima iteración, queda un subconjunto D_n' de D_n donde

$$D_n \supset D_n' = \{(t_i, z_i): \phi_n < z_i \leq M_n\}$$

Y es a partir de estos puntos como se generan los intervalos de incertidumbre descritos anteriormente. Para obtener estos intervalos se ordenan crecientemente los puntos $(t_i, z_i) \in D_n'$ respecto al valor t_i. Este ordenamiento se hace mediante el procedimiento que se conoce como sort bianrio, utilizando apuntadores para hacer más eficiente al algoritmo.

Este proceso se basa en la construcción de un árbol binario dónde el subarbol izquierdo de todo vértice contiene valores menores a este vértice, y el derecho, mayores. De esta forma al recorrer el árbol en inorder se tienen los valores ordenados crecientemente.

Ya teniendo las abscisas de los puntos que componen D_n' ordenadas crecientemente, se obtienen los intervalos correspondientes. Si estos distan en menos de una distancia 'δ' se unen formando un sólo intervalo.

(*) Si no se desea en el algoritmo, basta con quitar la subrutina 'ESTADS' y el área común 'MEDVAR' (ver apéndice B), sin que se altere el proceso normal.

III.2.7 Criterios de Terminación

En la implantación realizada, el algoritmo termina por alguna de las siguientes razones

i) Cuando el error relativo real al estimar el máximo es menor a una tolerancia 'ε' pequeña y predeterminada, esto es

$$\left| \frac{M_n - \phi_n}{\phi_n} \right| < \varepsilon.$$

ii) Cuando el algoritmo alcanza un número máximo de iteraciones permitidas.

iii) Si se han utilizado todas las posiciones del arreglo que representa al heap y ya no existe lugar donde insertar nuevos puntos.

nota: Si esto ocurre y se desea continuar el algoritmo deberá modificarse la dimensión del arreglo, definida por el parámetro 'NMAX' en las subrutinas. (ver listados en apéndice B)

iv) Si en algún momento $M_n < \phi_n$. Esto indica un error en la estimación de la constante de Lipschitz para la función objetivo.

III.2.8 Impresión de Resultados

En la implantación, se permite al usuario elegir la impresión mediante el valor de una variable de entrada.

Los resultados de un problema ejemplo presentados en las siguientes páginas contienen la máxima impresión. Los números entre llaves { } indican la impresión por cada valor de la variable de entrada 'IIMP'. (ver apéndice A).

Si IIMP =

0	No hay impresión.
1	Escribe un encabezado con los datos proporcionados por el usuario y la abscisa y ordenada de la mayor evaluación de la función objetivo.
2	Impresión (1) más los intervalos de incertidumbre.
3	Impresión (2) más el heap y estadísticas finales.
4	Impresión (2), además en cada iteración los nodos nuevos, valor máximo, la diferencia relativa en la evaluación del error, y la evaluación de la función objetivo en las nuevas abscisas izquierda y derecha.
5	Impresión (3) y (4).

```
          ***  ALGORITMO DE SHUBERT PARA OPTIMIZACION GLOBAL  ***


INTERVALO CONSIDERADO          (        .000000,       50.000000)
PUNTO INICIAL                                25.000000
CONSTANTE DE LIPSCHITZ                       17.500000
ERROR MAXIMO PERMITIDO EPSFUN                  .005000
ERROR MAXIMO PERMITIDO EPSINT                  .050000
NUMERO MAXIMO DE ITERACIONES                        50.
TIPO DE IMPRESION                                    5  <---
PARAMETRO IDESC                                      1
PARAMETRO IESTD                                      1


ITERACION NUMERO     1    NUMERO DE NODOS DEL ARBOL     2

NODO(S) GENERADO(S)
                  VALOR            ABSCISA IZQ          ABSCISA DER
                  335.509174             3.020554            16.201468
                  321.151936            32.978118            46.309421

VALORES DE LA FUNCION EVALUADA
IZQUIERDO=      260.048382   DERECHO=      196.024927
MAXIMO=(X,F(X))= (        3.020554,      260.048382)            ...(1)
NODO MAXIMO=Y(1)=      297.778778                               ...(2)
DIFERENCIA=(2)-(1)=       75.460793  DIFERENCIA RELATIVA= DIFERENCIA/(1) =        .290180

ITERACION NUMERO     2    NUMERO DE NODOS DEL ARBOL     3

NODO(S) GENERADO(S)
                  VALOR            ABSCISA IZQ          ABSCISA DER
                  297.778778               .864531             5.176577
                  265.767052             12.216203            20.186732

VALORES DE LA FUNCION EVALUADA
IZQUIERDO=      187.972805   DERECHO=      233.027548
MAXIMO=(X,F(X))= (        3.020554,      260.048382)            ...(1)
NODO MAXIMO=Y(1)=      277.089741                               ...(2)
DIFERENCIA=(2)-(1)=       61.103554  DIFERENCIA RELATIVA= DIFERENCIA/(1) =        .234970
```

{1}

{4}

CAPITULO IV

EJEMPLOS, APLICACIONES Y CONCLUSIONES

IV.1 EJEMPLOS Y APLICACIONES

La optimización en una linea puede tener gran variedad de aplicaciones. Además de los ejemplos que posteriormente se presentan es posible encontrar casos como los siguientes:

- Hallar estimadores de máxima verosimilitud o mínima χ^2 para una familia de distribuciones uniparamétricas.
- En lo que se denomina programación paramétrica, donde para cada valor de un parámetro se resuelve un problema dado de programación lineal, y se desea hallar el valor del parámetro que optimice a esa familia de problemas.
- En algoritmos para resolver problemas multivariados, donde se selecciona una dirección y sobre ésta se busca un óptimo, ya sea local [19] o global [3].

Los ejemplos que se presentan en este capítulo se seleccionaron con el fín de probar y evaluar la implantación realizada para el algoritmo de Shubert. Esta selección se hizo buscando diversidad en el comportamiento de las funciones objetivo.

En todas las pruebas, optimizar la función analíticamente sería un problema complejo o largo, además las constantes de Lipschitz respectivas no presentan problemas para calcularse.

A continuación se describen las funciones utilizadas, se anexan gráficas de las mismas y tablas con los resultados obtenidos. Cada ejemplo se ejecutó con varias opciones en los parámetros de entrada, los resultados que aquí se presentan se obtuvieron con los más adecuados para cada problema.

IV.1.1 Ejemplo 1

Este se refiere al problema considerado en el artículo de Shubert [25]. Se desea

$$\max z = \max \sum_{k=1}^{5} k \operatorname{sen}((k+1)x + k)$$

en el intervalo $-10 \leq x \leq 10$

La Fig. IV.1 muestra el comportamiento de esta función.

Como es sabido, la constante de Lipschitz puede ser evaluada como el valor máximo que toma la derivada de la función objetivo en el intervalo considerado, por lo que

$$\frac{dz}{dx} = \sum_{k=1}^{5} k \left(\cos\left[(k+1)\ x + k\right]\right)(k+1) \qquad (1.1)$$

además

$$-1 \leq \cos\left[(k+1)\ x + k\right] \leq 1 \qquad \text{Para todo } x \text{ y } k.$$

y como el producto $k\ (k+1)$ es positivo para todos los sumandos de (1.1), acotando superiormente la derivada se tiene que

$$\frac{dz}{dx} \leq \sum_{k=1}^{5} k\ (k+1) = 70$$

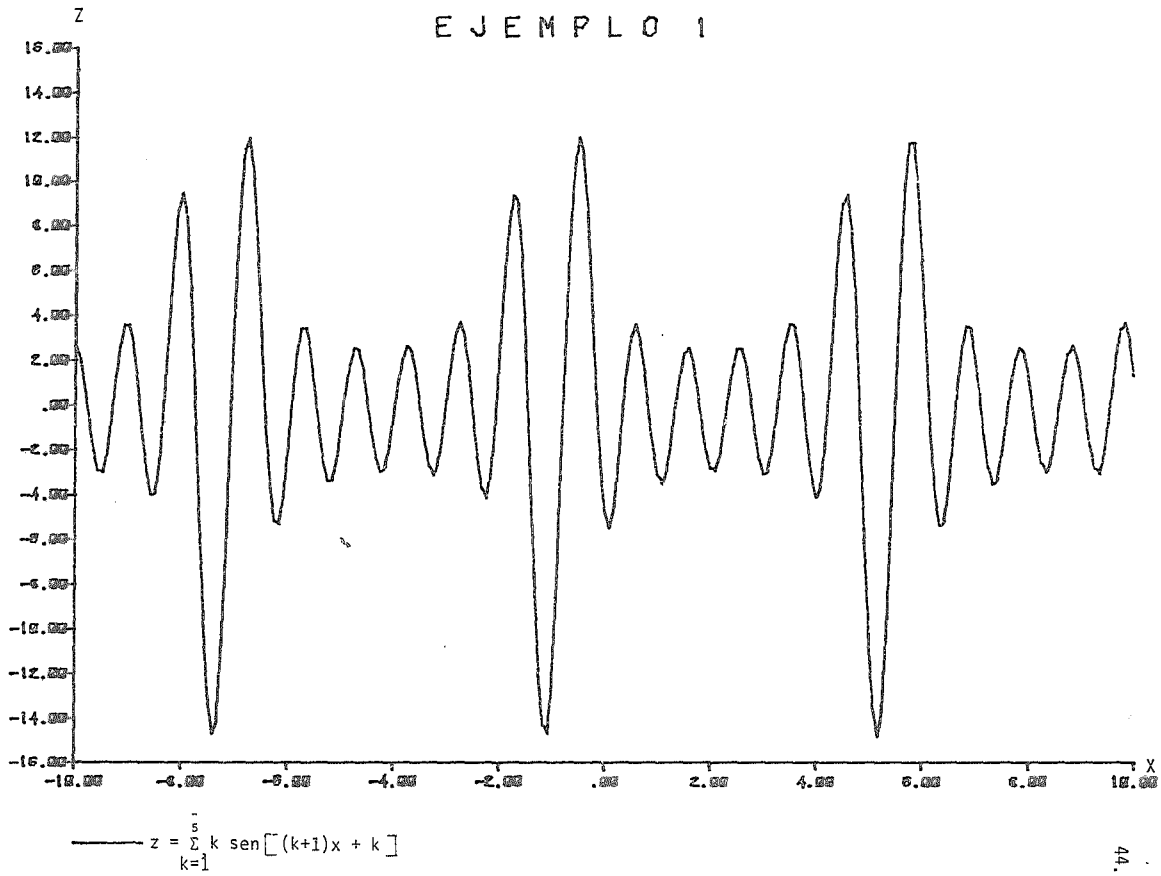

EJEMPLO 1
Z
X
16.00
14.00
12.00
10.00
8.00
6.00
4.00
2.00
.00
-2.00
-4.00
-6.00
-8.00
-10.00
-12.00
-14.00
-16.00
-10.00
-8.00
-6.00
-4.00
-2.00
.00
2.00
4.00
6.00
8.00
10.00
z = Σ k sen[(k+1)x + k]
k=1

Fig. IV.1

siendo este valor la constante de Lipschitz adoptada.

En este ejemplo se fijó una tolerancia al error permitido muy pequeña con el objeto de que el método finalizara al alcanzar 222 iteraciones. Esto se hizo para comparar los resultados con los que obtuvo Shubert con el mismo número de evaluaciones de la función.

Los resultados obtenidos son los mismos que menciona Shubert, a excepción que él considera dos máximos absolutos, siendo tres como corrige Hansen en [15].

Los resultados se muestran en la tabla IV.1

```
          ***  ALGORITMO DE SHUBERT PARA OPTIMIZACION GLOBAL  ***


INTERVALO CONSIDERADO          (     -10.000000,      10.000000)
PUNTO INICIAL                                .000000
CONSTANTE DE LIPSCHITZ                     70.000000
ERROR MAXIMO PERMITIDO EPSFUN                .000001
ERROR MAXIMO PERMITIDO EPSINT                .050000
NUMERO MAXIMO DE ITERACIONES                  222
TIPO DE IMPRESION                               2
PARAMETRO IOPSC                                 1
PARAMETRO IESTP                                 0




TERMINA POR ALCANZAR EL NUMERO MAXIMO DE ITERACIONES  222


INTERVALO(S) DONDE SE LOCALIZA(N) EL(LOS) OPTIMO(S)

     (      -6.790736,      -6.759548)
     (       -.512871,       -.426113)
     (       5.774923,       5.806149)


 PUNTO DE LA MAXIMA EVALUACION DE LA FUNCION OBJETIVO
EN  222  ITERACIONES           (      -6.791764 ,      12.422483)
```

TABLA IV.1

IV.1.2 Ejemplo 2

El problema se describe de la siguiente manera:

Dado un subconjunto $A = \{a_1, a_2, \ldots, a_m\}$ de R^n y un segmento de recta ℓ en R^n, encontrar un punto x° en ℓ tal que maximice la distancia al punto más cercano de A. La Fig. IV.2.1 muestra una instancia de este problema cuando n=2.

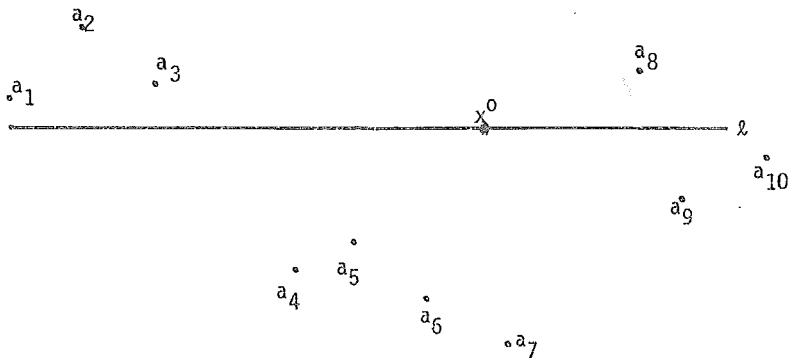


Fig. IV.2.1

El problema es entonces max z, donde

$$z = \min \{ ||x - a_i|| : x \in \ell , a_i \in A \}$$

El caso cuando n = 2 puede representar al problema de hallar la localización, en una calle o carretera, más lejana del punto más cercano de A. Esto puede tener importancia si A es un conjunto de puntos indeseables (con exceso de ruido, contaminación, etc.).

En el caso tratado se considero

i) $\ell = \{(t,0): 0 \leq t \leq 50\}$

ii) Al conjunto A como se muestra en la columna (2) de la tabla IV.2.1; la cardinalidad de A es 10.

La consideración i) siempre es posible obtenerla transladando y/o rotando el espacio original.

(1)	(2)	(3)	(4)
i	$a_i = (a_{i1}, a_{i2})$	t = 0	t = 50
1	(0,2)	0.00	2.00
2	(5,7)	-1.16	1.98
3	(10,3)	-1.92	1.99
4	(20,-10)	-1.79	1.90
5	(24,-8)	-1.90	1.91
6	(29,-12)	-1.85	1.74
7	(35,-15)	-1.84	1.41
8	(44,4)	-1.99	1.66
9	(47,-5)	-1.99	1.03
10	(53,-2)	-2.00	--

TABLA IV.2.1

Así entonces, con las características i), ii) el problema se define como max z, donde

A-0037497

$$z = \min_{i=1,..,10} z_i$$

$$\text{con} \quad z_i = \sqrt{(a_{i1}-t)^2 + a_{i2}^2} \qquad (2.1)$$

La relación anterior define, para cada i, una hipérbola con eje focal perpendicular a las abscisas y vértices en (a_{i1},a_{i2}), $(a_{i1},-a_{i2})$. Para nuestro caso nos interesa sólo la raíz positiva de (2.1) pues es la única que tiene significado real.

En los puntos de la curva descrita, el valor absoluto de la pendiente se incrementa en función directa de la distancia al vértice. Por esta razón, para encontrar la constante de Lipschitz, se tomó el máximo valor absoluto de las evaluaciones de las derivadas

$$\frac{dz_i}{dt} = \frac{2\,(t - a_{i1})}{\sqrt{(t-a_{i1})^2 + (a_{i2})^2}} \qquad i = 1,\ldots,10$$

en los puntos t = 0, t = 50. (Ver columnas (3) y (4) de la tabla IV.2.1 Previniendo inestabilidades numéricas debido a errores de redondeo, se tomó 2.0 como constante de Lipschitz.

La Fig. IV.2.2 muestra el comportamiento de la función objetivo.

En este caso el máximo se encuentra rapidamente, pues se localiza en una región dónde la pendiente de la curva es cercana a la constante de Limpschitz indicada.

En la tabla IV.2.2 se presentan los resultados obtenidos.

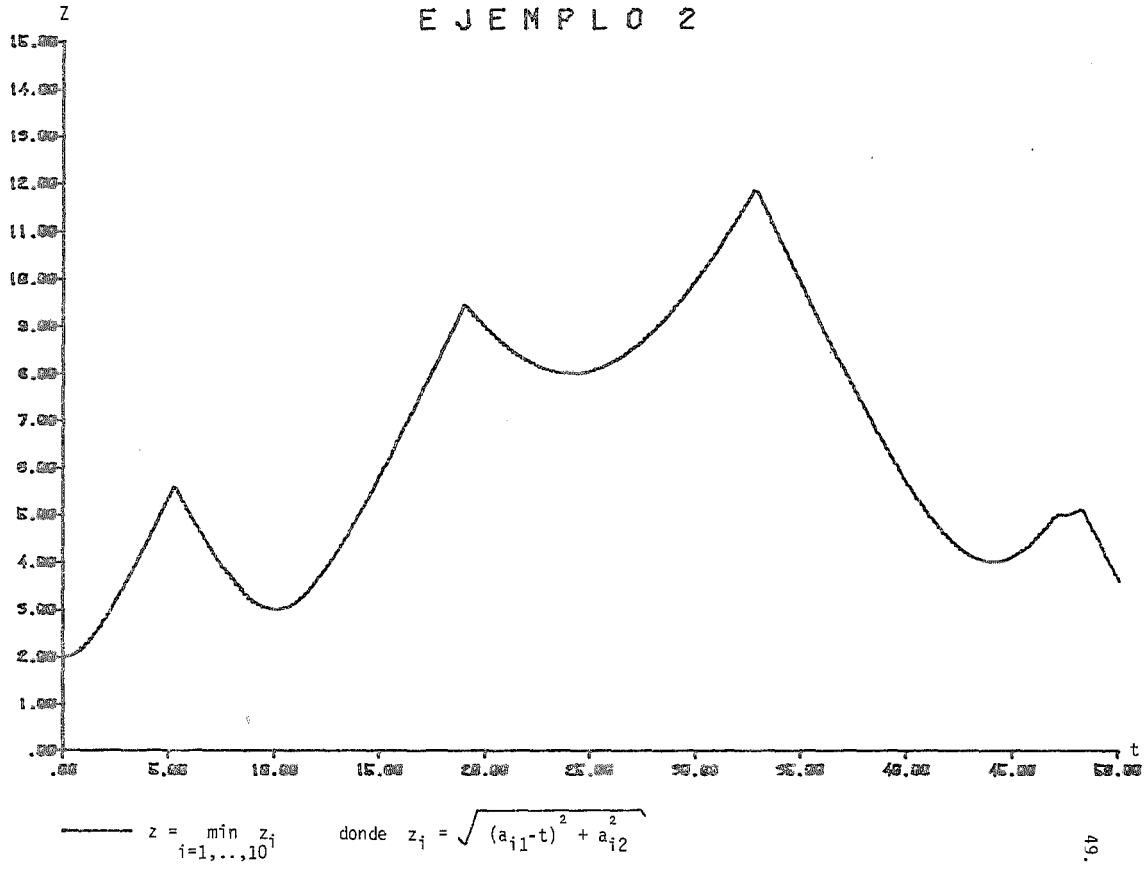

EJEMPLO 2
Z
t
15.00
14.00
13.00
12.00
11.00
10.00
9.00
8.00
7.00
6.00
5.00
4.00
3.00
2.00
1.00
.00
.00
5.00
10.00
15.00
20.00
25.00
30.00
35.00
40.00
45.00
50.00
z = min z_i donde z_i = √((a_i1-t)^2 + a_i2^2)
i=1,..,10

Fig. IV.2.2

```
*** ALGORITMO DE SHUBERT PARA OPTIMIZACION GLOBAL ***

INTERVALO CONSIDERADO            (    [illegible],    [illegible])
PUNTO INICIAL                           [illegible]
CONSTANTE DE LIPSCHITZ                  [illegible]
ERROR MAXIMO PERMITIDO EPSFUN           [illegible]
ERROR MAXIMO PERMITIDO EPSINT           [illegible]
NUMERO MAXIMO DE ITERACIONES            [illegible]
TIPO DE IMPRESION                       [illegible]
PARAMETRO IDESC                         1
PARAMETRO IESTP                         [illegible]


INTERVALO(S) DONDE SE LOCALIZA(N) EL(LOS) OPTIMO(S)

    (      32.791732,      32.806236)

PUNTO DE LA MAXIMA EVALUACION DE LA FUNCION OBJETIVO
EN   17  ITERACIONES          (      32.800396 ,       11.892462)
```

TABLA IV.2.2

IV.1.3 Ejemplo 3

Este caso es parecido al anterior. La formulación puede provenir del problema de localizar un centro de distribución en una calle o carretera, de tal manera que la suma de las distancias a los puntos consumidores (conjunto A) sea mínima.

Considerando los conjuntos ℓ y A definidos en el ejemplo 2, el problema se describe como

$$\min \quad z = \min \sum_{i=1}^{10} \sqrt{(a_{i1}-t)^2 + (a_{i2})^2}$$

Como se mencionó en la sección I.4, la constante de Lipschitz de una suma de funciones de Lipschitz es la suma de sus respectivas constantes. Considerando esto y los datos de la tabla IV.2.1, se fijó 17.5 como constante de Lipschitz para este problema.

La Fig. IV.3.1 muestra esta función. Sin embargo, como la implantación del algoritmo de Shubert fue hecha para obtener puntos máximos, se cambió el signo de esta función dando lugar a la que se muestra en la Fig. IV.3.2. A esta nueva función se le aplicó el algoritmo, obteniéndose los resultados presentados en la tabla IV.3.

Como aquí el óptimo esta en una región con una pendiente cercana a cero, a pesar de que el resultado obtenido es bueno (error menor a 5×10^{-4}), los intervalos de incertidumbre son 6 en 160 iteraciones permitiendo no menos de 10^{-1} unidades de separación entre ellos.

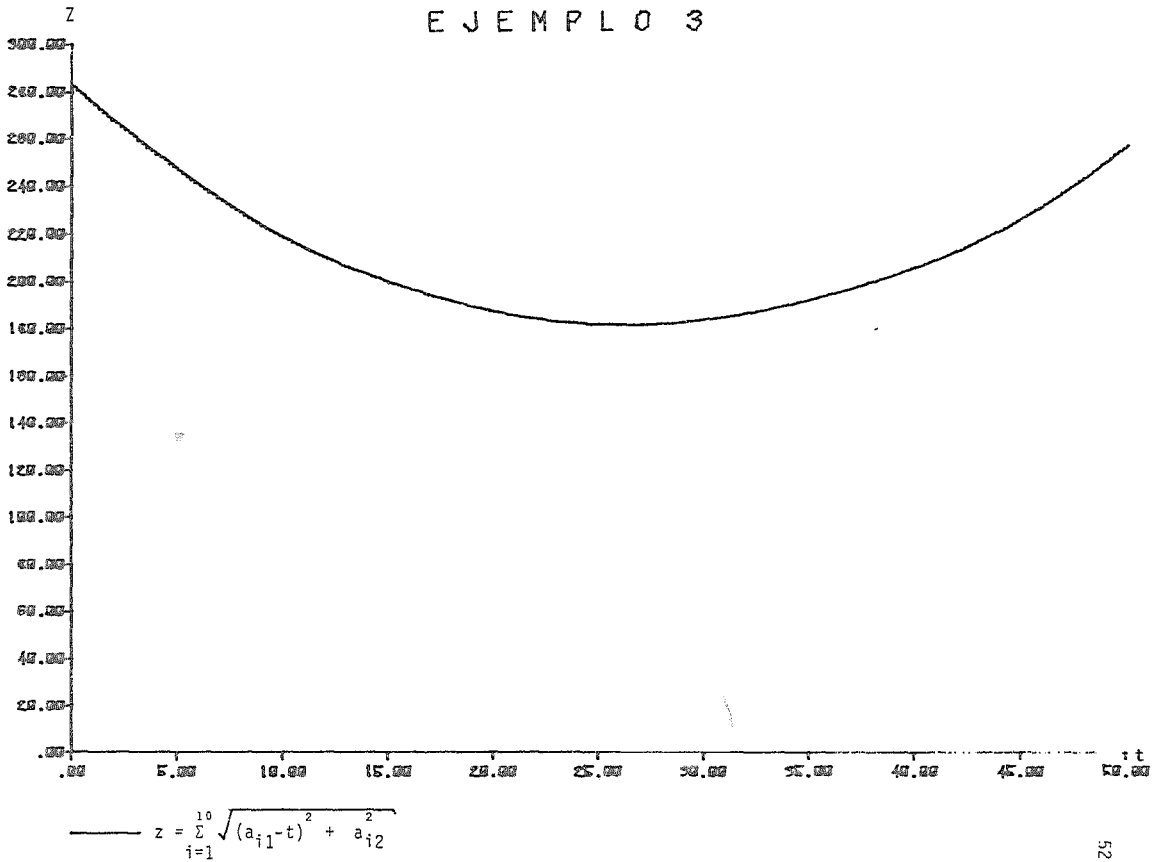


Fig. IV.3.1

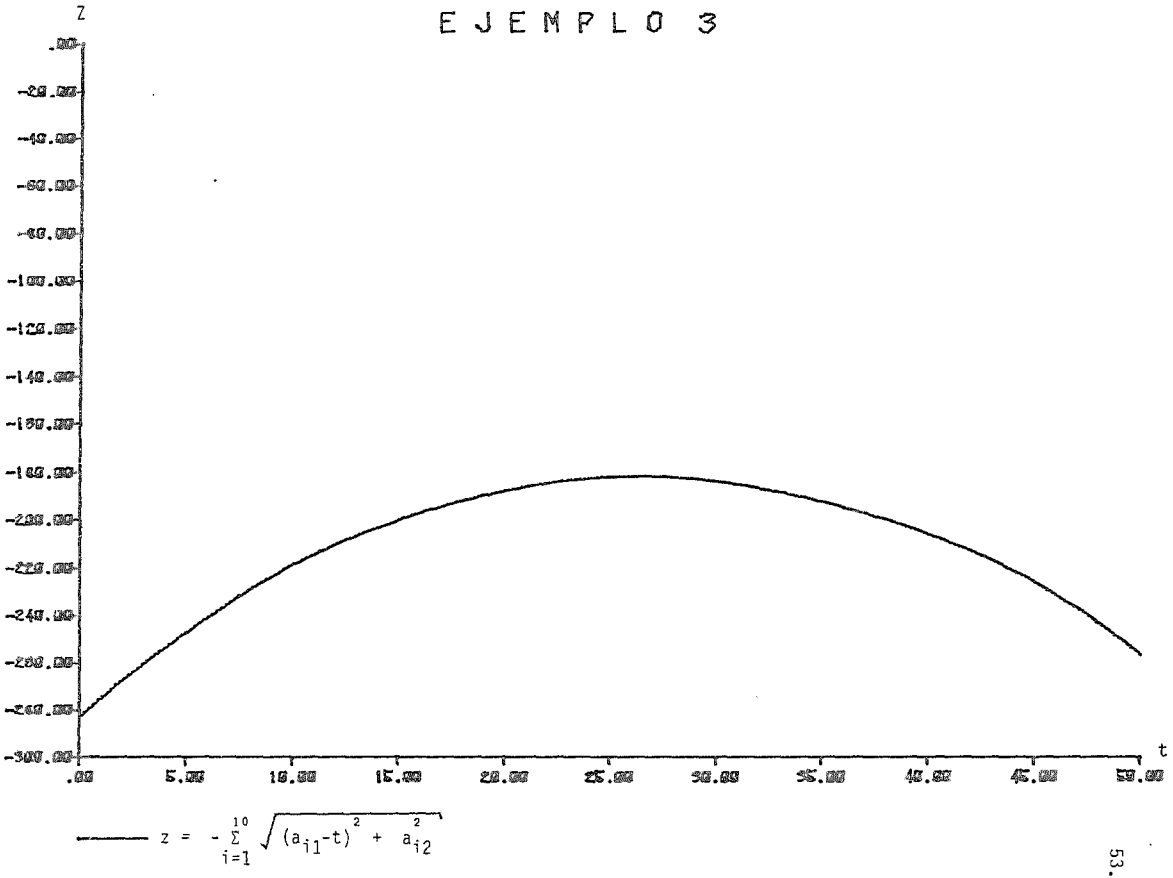

EJEMPLO 3
Z
t
.00
-20.00
-40.00
-60.00
-80.00
-100.00
-120.00
-140.00
-160.00
-180.00
-200.00
-220.00
-240.00
-260.00
-280.00
-300.00
.00
5.00
10.00
15.00
20.00
25.00
30.00
35.00
40.00
45.00
50.00
$z = -\sum_{i=1}^{10} \sqrt{(a_{i1}-t)^2 + a_{i2}^2}$

Fig. IV.3.2

```
          ***  ALGORITMO DE SHUBERT PARA OPTIMIZACION GLOBAL  ***


INTERVALO CONSIDERADO          (        .000000,      50.000000)
PUNTO INICIAL                              25.000000
CONSTANTE DE LIPSCHITZ                     17.500000
ERROR MAXIMO PERMITIDO EPSFUN                .000500
ERROR MAXIMO PERMITIDO EPSINT                .100000
NUMERO MAXIMO DE ITERACIONES                     500
TIPO DE IMPRESION                                  2
PARAMETRO IDESC                                    1
PARAMETRO IESTO                                    0



INTERVALO(S) DONDE SE LOCALIZA(N) EL(LOS) OPTIMO(S)

     (       23.323621,       23.326880)
     (       23.454331,       23.457590)
     (       24.186164,       24.251210)
     (       24.781936,       27.057565)
     (       27.387398,       27.490814)
     (       27.978313,       28.202565)


 PUNTO DE LA MAXIMA EVALUACION DE LA FUNCION OBJETIVO
EN  160  ITERACIONES            (      26.055746 ,     -181.361446)
```

TABLA IV.3

IV.1.4 Ejemplo 4

Aquí se consideró la función original del ejemplo anterior (Fig. IV.3.1), pero se buscó el punto máximo.

Los resultados obtenidos se muestran en la tabla IV.4

```
            ***  ALGORITMO DE SHUBERT PARA OPTIMIZACION GLOBAL  ***


 INTERVALO CONSIDERADO          (        .000000,       50.000000)
 PUNTO INICIAL                          25.000000
 CONSTANTE DE LIPSCHITZ                 17.500000
 ERROR MAXIMO PERMITIDO EPSFUN            .000001
 ERROR MAXIMO PERMITIDO EPSINT            .010000
 NUMERO MAXIMO DE ITERACIONES                 500
 TIPO DE IMPRESION                              2
 PARAMETRO IDESC                                1
 PARAMETRO IESTO                                0



 INTERVALO(S) DONDE SE LOCALIZA(N) EL(LOS) OPTIMO(S)

     (        -.000002,         .000047)

 PUNTO DE LA MAXIMA EVALUACION DE LA FUNCION OBJETIVO
EN   12  ITERACIONES            (        .000006 ,     282.649433)
```

TABLA IV.4

En este ejemplo el algoritmo alcanza muy rápido la solución (12 iteraciones con un error menor a 10^{-6}) debido a que el óptimo se localiza en el punto con mayor pendiente, siendo ésta muy cercana a la constante de Lipschitz especificada.

IV.1.5 Ejemplo 5

El problema que se describe a continuación se obtuvo tratando de encontrar una función cuya constante de Lipschitz fuese fácil de determinar y que su valor máximo no pudiera ser encontrado analíticamente en forma sencilla.

La función de este ejemplo fue inventada, sin embargo problemas de este tipo son usuales en Teoría de Juegos donde se desea maximizar la pérdida del adversario, suponiendo que éste va a elegir la mínima pérdida entre varias alternativas.

Se desea max z, donde

$$z = \min \{f_1(x) , f_2(x), f_3(x)\} \qquad \text{en } 10 \leq x \leq 20$$

con

$$f_1(x) = \text{sen } (1.5\ x)$$

$$f_2(x) = ((x-9)/5) \text{ sen } ((x-9)^{1.3})$$

$$f_3(x) = \text{sen}(\ 2 \text{ sen}(x))$$

En la Fig. IV.5.1 se presentan las funciones $f_i : i=1,2,3$. Y en la Fig. IV.5.2 la función objetivo.

Como constante de Lipschitz se tomó la mayor constante de las tres funciones (ver sección I.4), siguiendo el procedimiento ya señalado anteriormente

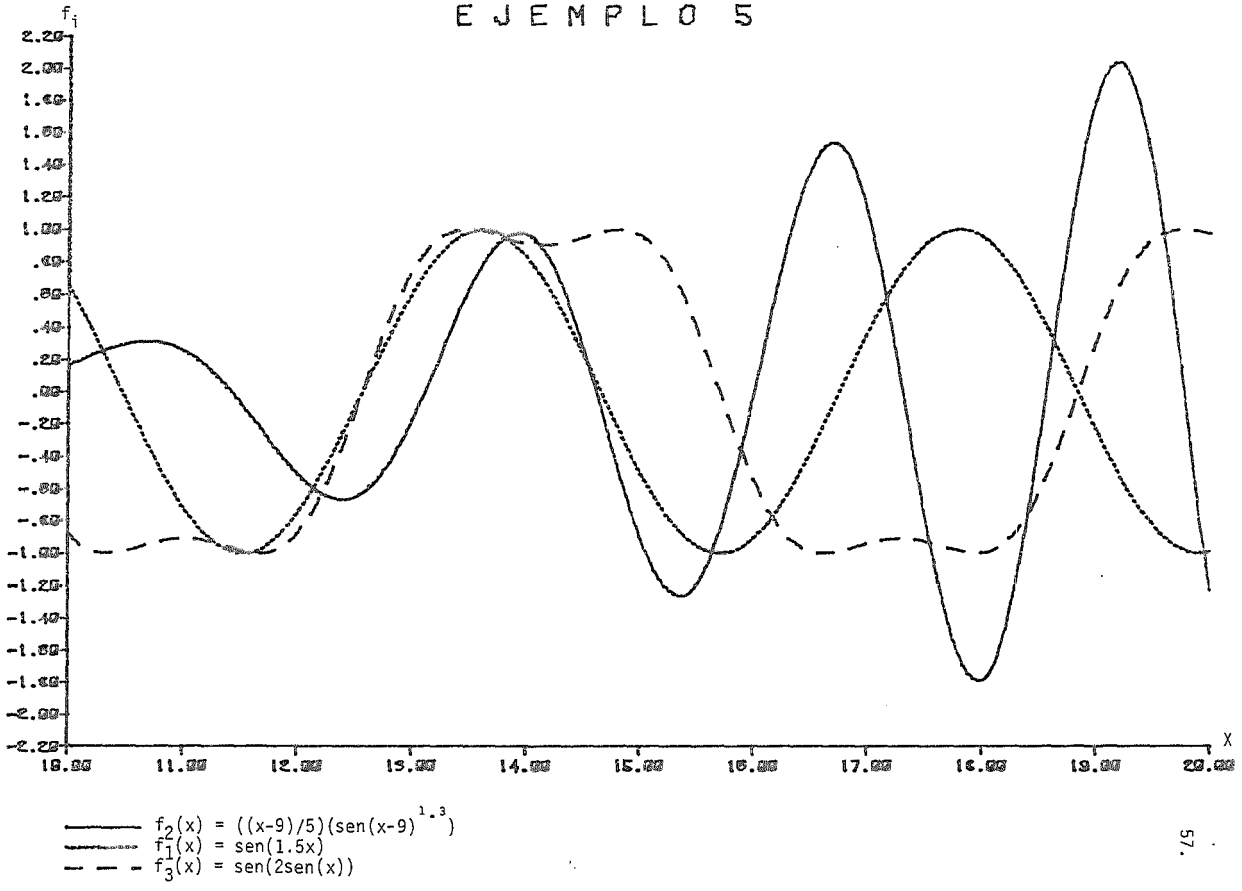

EJEMPLO 5
f_i
X
$f_2(x) = ((x-9)/5)(\text{sen}(x-9)^{1.3})$
$f_1(x) = \text{sen}(1.5x)$
$f_3(x) = \text{sen}(2\text{sen}(x))$

Fig. IV.5.1

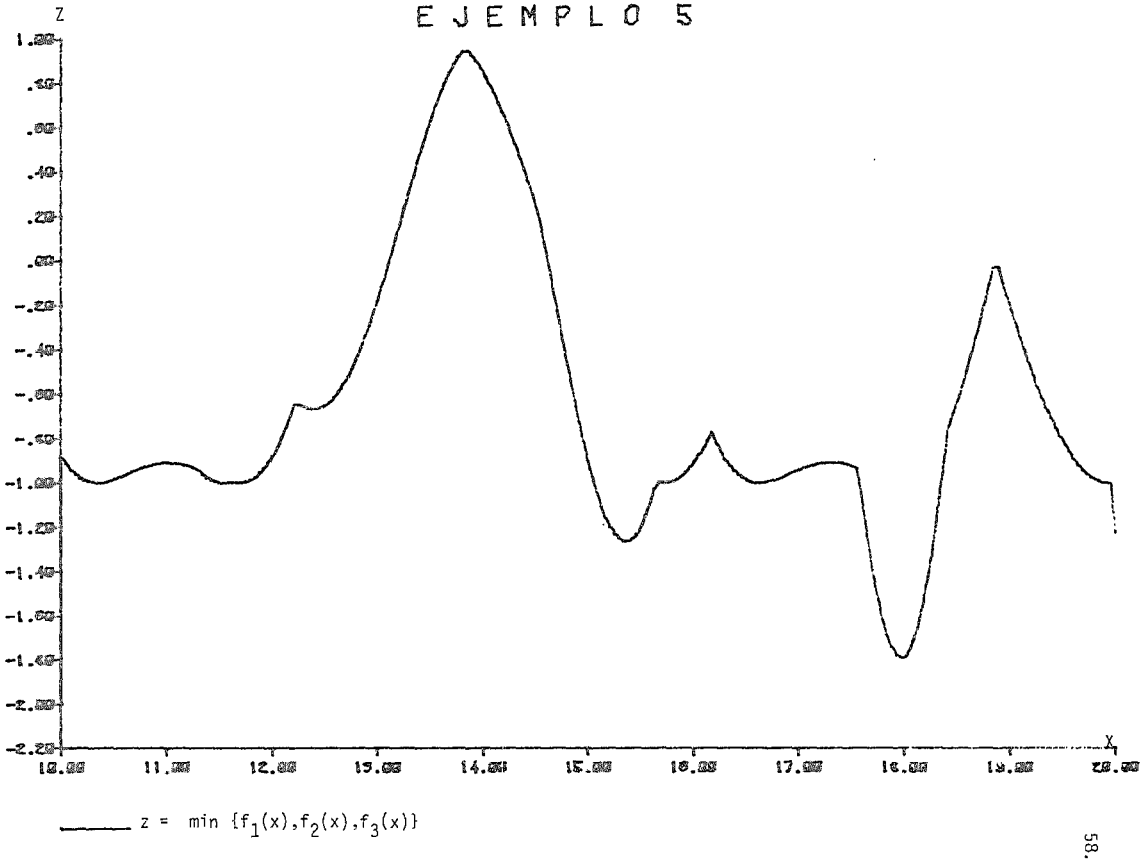

EJEMPLO 5
Z
X
1.20
.80
.60
.40
.20
.00
-.20
-.40
-.60
-.80
-1.00
-1.20
-1.40
-1.60
-1.80
-2.00
-2.20
10.00
11.00
12.00
13.00
14.00
15.00
16.00
17.00
18.00
19.00
20.00
z = min {f1(x),f2(x),f3(x)}

Fig. IV.5.2

$$\frac{df_1}{dx} = 1.5 \cos(1.5\ x) \qquad \Longrightarrow \qquad c_1 = 1.5$$

$$\frac{df_2}{dx} = \frac{1}{5}\{(x-9)\cos((x-9)^{1.3})(1.3)(x-9)^{0.3} + \operatorname{sen}((x-9)^{1.3})\}$$

$$\Longrightarrow \qquad c_2 = 4.8$$

$$\frac{df_3}{dx} = \cos(2 \operatorname{sen} x)\ 2 \cos x \qquad \Longrightarrow \qquad c_3 = 2.0$$

$$c = \max\ \{c_1, c_2, c_3\} = 4.8$$

Esta función es complicada para ser resuelta analíticamente y por su estructura no es posible obtener su solución con métodos convencionales de optimización en una linea. Los resultados obtenidos se presentan en la tabla IV.5.

IV.1.6 Ejemplo 6

En la Teoría del Análisis Numérico, el problema de encontrar los ceros de una función ha sido estudiado por diversos autores. Una aplicación importante del algoritmo de Shubert es la resolución de este problema, maximizando el negativo del valor absoluto de la función original. Los resultados que se obtienen son muy buenos pues se localizan todos los ceros y la función unicamente debe satisfacer la propiedad de Lipschitz.

Aquí se utilizó la función del ejemplo anterior. La grá-

```
            ***  ALGORITMO DE SHUBERT PARA OPTIMIZACION GLOBAL  ***


INTERVALO CONSIDERADO          (      10.000000,      20.000000)
PUNTO INICIAL                          15.000000
CONSTANTE DE LIPSCHITZ                  4.800000
ERROR MAXIMO PERMITIDO EPSFUN            .000050
ERROR MAXIMO PERMITIDO EPSINT            .010000
NUMERO MAXIMO DE ITERACIONES                  500
TIPO DE IMPRESION                               2
PARAMETRO IDESC                                 1
PARAMETRO IESTO                                 0




 INTERVALO(S) DONDE SE LOCALIZA(N) EL(LOS) OPTIMO(S)

     (      13.903398,      13.906402)

 PUNTO DE LA MAXIMA EVALUACION DE LA FUNCION OBJETIVO
EN   85  ITERACIONES           (      13.905579 ,      -.949155)
```

TABLA IV.5

fica del negativo del valor absoluto de dicha función se muestra en la Fig. IV.6.

Cuando se trató este problema con la implantación realizada hubo dificultades originadas por inestabilidades numéricas, debido a que el algoritmo termina cuando la diferencia relativa entre los valores máximos de la función lineal por pedazos (M_n) y la función objetivo (ϕ_n) es menor que un valor pequeño predeterminado. Esto es, termina si

$$\left| \frac{M_n - \phi_n}{\phi_n} \right| < \varepsilon \qquad (6.1)$$

En este ejemplo el numerador y denominador de (6.1) tienden ambos a cero conforme el método converge a la solución. Para evitar las irregularidades que esto causa, la función se trasladó una unidad positivamente sobre el eje ordenado. De esta forma el denominador tiende a uno y ya no hay dificultades.

Los resultados con esta modificación se presentan en la tabla IV.6

IV.2 CONCLUSIONES

La implantación del algoritmo de Shubert mediante un árbol binario con estructura heap es el punto fundamental de esta tesis. Es una idea original que permite explotar las características de la metodología a través de la estructura mencionada. También es importante señalar el uso de un árbol binario y su etiquetado en inorder para obtener eficientemente los intervalos de incertidumbre donde se localiza la solución final.

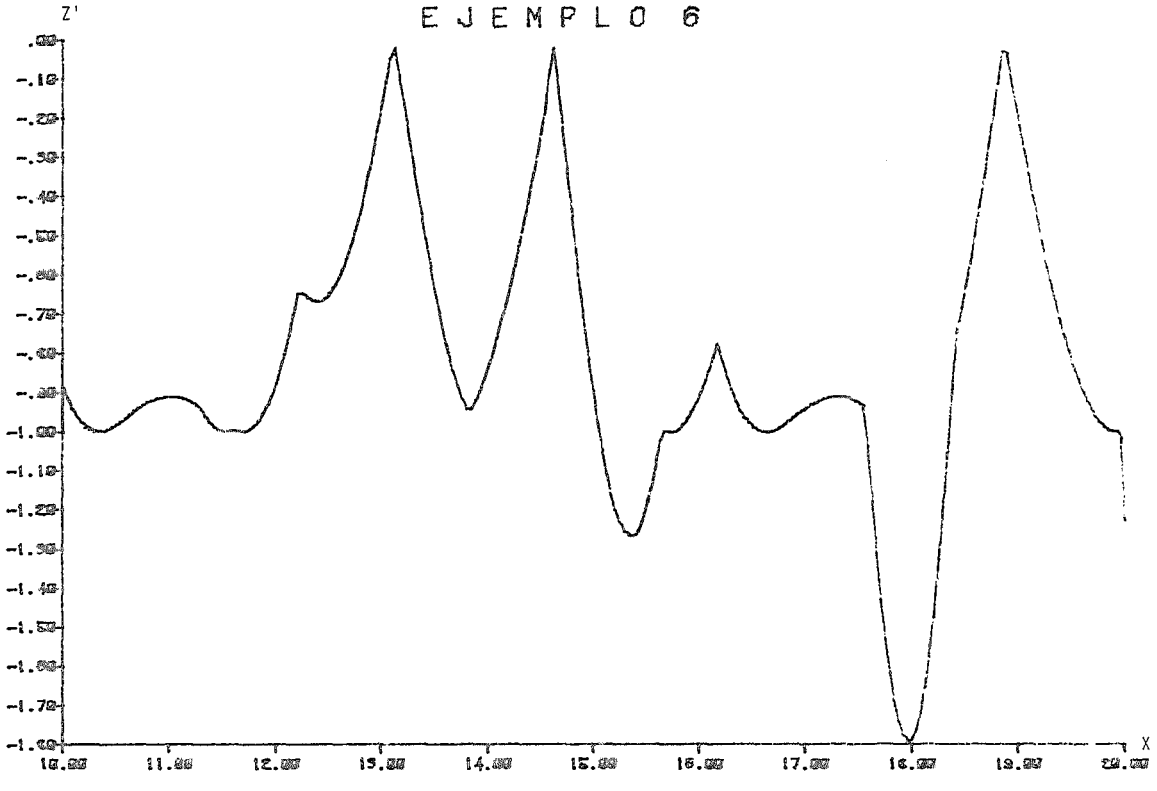


Fig. IV.6

```
            ***  ALGORITMO DE SHUBERT PARA OPTIMIZACION GLOBAL  ***


INTERVALO CONSIDERADO          (       10.000000,       30.000000)
PUNTO INICIAL                                15.000000
CONSTANTE DE LIPSCHITZ                        4.500000
ERROR MAXIMO PERMITIDO EPSFUN                  .000100
ERROR MAXIMO PERMITIDO EPSINT                  .010000
NUMERO MAXIMO DE ITERACIONES                       500
TIPO DE IMPRESION                                    2
PARAMETRO IOPSC                                      1
PARAMETRO IESTO                                      0




 INTERVALO(S) DONDE SE LOCALIZA(N) EL(LOS) OPTIMO(S)


     (       17.111269,       18.111391)
     (       14.616032,       14.616274)
     (       18.949375,       18.950176)


 PUNTO DE LA MAXIMA EVALUACION DE LA FUNCION OBJETIVO
EN   62  ITERACIONES           (       13.111359 ,          .999984)
```

TABLA IV.6

La implantación requirió 1.1215 K de memoria para las instrucciones y 6.425 K para las variables utilizadas. Estas cantidades incluyen los programas ejemplo del usuario presentados en el apéndice B, cuya área total es de 0.195 y 0.153 K de instrucciones y variables respectivamente.

El algoritmo de Shubert ha sido criticado por dos razones. Primera, que su convergencia es lenta comparada con la de otros métodos; segunda, que la determinación de la constante de Lipschitz de funciones utilizadas en la práctica muchas veces es compleja.

Ante estas dos razones es posible argüir lo siguiente. Primero, que existen pocos algoritmos de optimización global; y de estos, los que obtienen un resultado determinístico son menores aún. Además si la función tiene varios óptimos globales, el algoritmo de Shubert los determina todos mediante el uso de los intervalos de incertidumbre. Segundo, el método funciona con cotas superiores a la mínima constante de Lipschitz (tal vez no eficientemente, pero funciona), siendo éstas usualmente más fáciles de obtener.

Observando los ejemplos que se presentaron en la sección anterior hay varios puntos que conviene remarcar. El método tiende a ser menos eficiente cuando la región de la curva que contiene al óptimo tiene una pendiente pequeña o cuando se incrementa el valor de la diferencia entre la constante de Lipschitz elegida y la real.

En los ejemplos considerados los resultados obtenidos fueron buenos y el tiempo de CPU fue menor a 0.6 seg. en el problema más difícil.

La aplicación de obtener los ceros de una función de Lipschitz, transformando el problema a uno de optimización da excelentes resultados pues se obtienen todos los ceros del intervalo considerado. Es un problema de interés para el Análisis Numérico y que se ha estudiado bastante debido a sus diversas aplicaciones.

APENDICE A.

GUIA DE OPERACION PARA EL ALGORITMO DE SHUBERT

A.I PROCEDIMIENTO

En esta sección se presentan las indicaciones necesarias para utilizar las subrutinas que constituyen la implantación del algoritmo de Shubert.

El usuario debe de elaborar un programa principal desde el cual se llamará al algoritmo de Shubert, y otro para evaluar la función objetivo(*). Estos programas deben cumplir las especificaciones detalladas en las siguientes secciones.

A.I.1 Programa Principal

Este programa deberá tener la estructura mostrada en el esquema A.1.

Y de acuerdo a este esquema los pasos que deberán seguirse son los siguientes

(*) En el Apéndice B se listan, a guiza de ejemplo, el programa principal y la función objetivo utilizados para generar los ejemplos que se presentan en el capítulo III.

```
     Cols.  5    6   7
(1)              COMMON/USERIN/XA,XINI,XB,CL,EPSFUN,EPSINT,NITMAX,
                 $  IIMP,IDESC,IESTD,IUSR,INIC,XSOL,FSOL,IFIN,IERR
(2)              EXTERNAL EJMPLO

(3)                 (SE DEBEN ASIGNAR VALORES A LAS VARIABLES
                     QUE LO REQUIERAN DEL AREA COMUN 'USERIN')

(4)              CALL SHUBRT(EJMPLO)


                 END
```

ESQUEMA A.1

(1) Declarar el área común de que dispone el usuario(*). Con ella se proporcionan los valores de las variables requeridas por el algoritmo y se obtienen los resultados.

(2) Definir como función externa el nombre de la subrutina elaborada para evaluar la función objetivo.

(3) A lo largo del programa se realizan los cálculos que requiera el usuario. Además se deben asignar valores a las variables que lo necesiten del área común mencionada en (1). Los valores deben ser asignados según las convenciones de Fortran; esto es, las variables cuyo nombre comience con I,J,K,L,M,N son enteras, y las otras, reales de precisión sencilla.

(*) En la sección A.II de este Apéndice se describe el significado de las variables que constituyen esta área común.

(4) Llamar al algoritmo de Shubert. Cabe hacer notar que el nombre del parámetro usado al llamar la subrutina SHUBRT debe ser el mismo que se definió en (2).

A.I.2 Subrutina para Evaluar la Función Objetivo

La estructura que debe tener es la mostrada en el esquema A.2

	Cols. 5	6	7
(1)			FUNCTION EJMPLO (X)
(2)			COMMON/USERIN/XA,XINI,XB,CL,EPSFUN,EPSINT,NITMAX,
		$	IIMP,IDESC,IESTD,IUSR,INIC,XSOL,FSOL,IFIN,IERR
(3)			EJMPLO =
			RETURN
			END

ESQUEMA A.2

Dónde las indicaciones son las siguientes

(1) Definirla como función, cuyo único parámetro es el punto donde se desea evaluar la función objetivo. El nombre de la función debe ser el mismo al declarado en el programa principal (ver sección A.I.1)

(2) Declarar al área común 'USERIN' (mencionada en A.I.1). Aquí esta declaración es opcional y sólo se requiere cuando es necesario utilizar alguna de sus variables.

(3) Calcular el valor que toma la función objetivo en el punto mencionado en (1).

A.II PARAMETROS DEL ALGORITMO

En esta sección se explica el significado de las variables que forman al área común 'USERIN'. Esta, fue mencionada en la sección anterior y se refiere a las variables que interesan al usuario(*).

Como se mencionó anteriormente, el área común se declara

```
COMMON/USERIN/XA,XINI,XB,CL,EPSFUN,EPSINT,NITMAX,
$ IIMP,IDESC,IESTD,IUSR,INIC,XSOL,FSOL,IFIN,IERR
```

En la descripción de las variables que se da a continuación se sigue la siguiente convención

Tipo I significa variable de entrada o INPUT.
Tipo O significa variable de salida o OUTPUT.
Tipo D significa que no es obligatorio asignarle valor, esto es, tiene un valor de DEFAULT.

(*) Si se desea tener acceso a alguna otra variable que utilicen los programas, será necesario consultar el listado de la subrutina SHUBRT (Apéndice B), en el cual se explica el significado de las variables más importantes incluídas en áreas comunes.

Parámetro	Tipo	Valor	Significado
XA	I		Cota izquierda del intervalo de análisis de la función objetivo.
XINI	I		Punto inicial o semilla donde comienza el algoritmo. Si no se tiene preferencia por un punto determinado, es recomendable fijarlo como el punto medio del intervalo por analizar. Restricción : XA < XINI < XB.
XB	I		Cota derecha del intervalo de análisis de la función objetivo.
CL	I		Cota mínima superior calculada para la constante de Lipschitz de la función objetivo.
EPSFUN	I,D		Valor máximo aceptado para la tolerancia al error relativo en el valor máximo de la función objetivo(*). Valor de Default = 0.001.
EPSINT	I,D		Valor máximo aceptado para agrupar intervalos al calcular los intervalos de incertidumbre(**). Valor de Default = 0.01.
NITMAX	I,D		Número máximo de iteraciones permitidas al algoritmo(*). Valor de Default = 500
IIMP	I		Indicador del tipo de impresión deseada.
		0	Sin impresión
		1	Impresión de encabezado señalando los parámetros de entrada, abscisa y ordenada de la mayor evaluación de la función objetivo.
		2	Impresión (1) más el conjunto de intervalos de incertidumbre(**).
		3	Impresión (2) más valores de los nodos del Heap final con estadísticas(***).
		4	Impresión (2) más resultados intermedios en cada iteración del algoritmo.

(*) Ver "Criterios de Terminación", Cap. III, sección 2.7

(**) Ver "Intervalos de Incertidumbre", Cap. III, sección 2.6

(***) Ver "Estadísticas del Heap", Cap. III, sección 2.5

Parámetro	Tipo	Valor	Significado
		5	Impresiones (3) y (4).
IDESC	I,D	1	Descarta el nodo que se introduce al final del Heap en cada iteración y cuyo valor sea menor que la máxima evaluación de la función objetivo en ese momento. Valor de Default = 0.
IESTD	I,D	1	Calcula valor medio, varianza y desviación de los nodos del Heap. Si 'IIMP' toma valores 3 ó 5, IESTD toma valor forzoso = 1
IUSR	I,O,D		Es una variable disponible para el usuario. NOTA : Si se desea utilizar en la subrutina que evalúa la función objetivo, deberá incluir en esta subrutina el área común 'USERIN'.
INIC	O	0	Valor de la variable al iniciarse el algoritmo de Shubert.
		1	Valor cuando ya ha sido evaluada la función objetivo. Este indicador es de gran utilidad cuando se deben leer coeficientes que intervienen en el cálculo de la función objetivo, pues éstos sólo es necesario leerlos en la primera evaluación. NOTA : Si se desea utilizar en la subrutina que evalúa la función objetivo, deberá incluir en esta subrutina el área común 'USERIN'.
XSOL	O		Abscisa donde se encontró la mayor evaluación de la función objetivo.
FSOL	O		Valor de la función objetivo en XSOL.
IFIN	O		Indicador de terminación del algoritmo de Shubert.
		1	Terminó porque la diferencia relativa del error es menor que EPSFUN.
		2	Terminó por alcanzar el número máximo de iteraciones permitido.
		3	Terminó porque se ocuparon todos los nodos disponibles en el Heap (600). Si ocurre esto y se desea continuar el proceso, será necesario agrandar el parámetro NMAX en todas las subrutinas que componen el algoritmo.

Parámetro	Tipo	Valor	Significado
		4	Termina si en alguna iteración se registra que el nodo generado es menor que el valor de la función objetivo en la misma abscisa.
IERR	0		Indicador de error en el algoritmo.
		1	Es el mismo caso de IFIN = 4.
		2	Se ocupó totalmente el arreglo donde se guardan los intervalos de incertidumbre. (Dimensión 300). Si esto ocurre será necesario agrandar el parámetro NINTMX definido en las subrutinas INTSOL e IMPRES; o bien, procurando que el algoritmo continue para descartar intervalos de incertidumbre(*).

(*) Ver "Intervalos de Incertidumbre", Cap. III, sección 2.6

APENDICE B.

PROGRAMAS DE LA IMPLANTACION DEL ALGORITMO DE SHUBERT

B.I DIAGRAMA DE LA IMPLANTACION

El algoritmo de Shubert se implantó mediante 8 subrutinas (sin contar las 2 proporcionadas por el usuario), interaccionando como se muestra en el siguiente diagrama

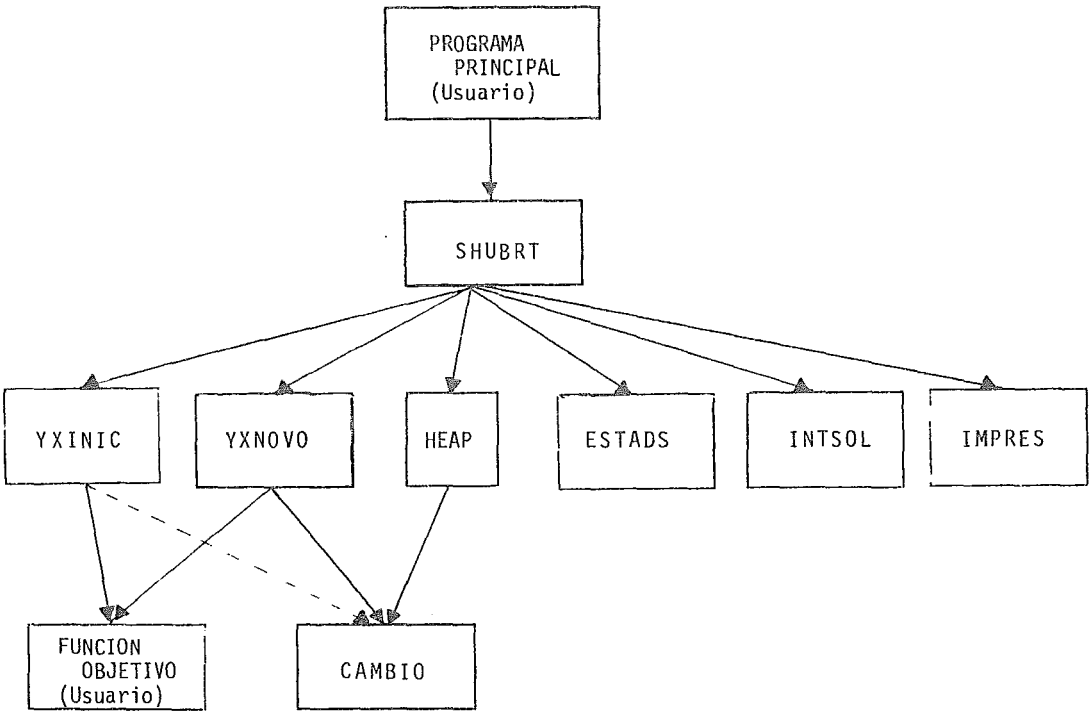


DIAGRAMA B.1

Las funciones que desempeña cada subrutina en la implantación del algoritmo de Shubert, se enumeran a continuación(*) :

SHUBRT
- Subrutina principal del algoritmo de Shubert.
- En ella se dirige la secuencia de ejecución de las otras subrutinas.
- En ella se explican, mediante comentarios, el significado de todas las variables importantes del algoritmo localizadas en áreas comunes.

YXINIC
- Realiza la etapa que se le llamó "Inicialización del Algoritmo" descrita en el Cap. III.

YXNOVO
- Prende un switch para que termine la ejecución si
  1) El error relativo obtenido es menor que la tolerancia permitida,
  2) Alcanzó el número máximo de iteraciones permitido o
  3) El Heap está totalmente ocupado y no es posible guardar más nodos en memoria.
- Realiza la etapa que se llamó "Generación de nuevos nodos" descrita en el Cap. III.

HEAP
Verifica y hace los cambios necesarios en el árbol para que se mantenga la estructura Heap, tal y como se describe en "Conservación del Heap" en el capítulo III.

CAMBIO
Intercambia el valor del i-ésimo nodo con sus abscisas izquierda y derecha por los valores que están en el j-ésimo nodo.

(*) Para comprender esta descripción es necesario conocer el algoritmo de Shubert (Cap. II), y la implantación realizada (Cap. III).

ESTADS — Actualiza para cada iteración el valor medio, varianza y desviación de los nodos del Heap. Ver "Estadísticas del Heap" en el Cap. III.

INTSOL — Realiza la parte denominada "Selección de intervalos de incertidumbre", descrita en el Cap. III.

IMPRES — Subrutina para imprimir encabezados y valores de variables determinadas que intervienen en el algoritmo de Shubert.

B.II LISTADOS DE LOS ELEMENTOS SIMBOLICOS DEL PROGRAMA PRINCIPAL Y SUBRUTINA DE EVALUACION DE LA FUNCION OBJETIVO UTILIZADOS EN LA GENERACION DE LOS EJEMPLOS DESCRITOS EN EL CAPITULO III.

```
C*************************************************************************
C
C     P R O G R A M A    P R I N C I P A L    D E    P R U E B A S
C
C*************************************************************************
C
C
C  AREA COMUN CUYOS VALORES SON PROPORCIONADOS POR EL JSJARIO
C
      COMMON/USERIN/XA,XINI,XB,CL,EPSFUN,EPSINT,NITMAX,IIMP,
     * IDESC,IESTD,IUSR,INIC,XSOL,FSOL,IFIN,IERR
C
C  FUNCION EXTERNA PROPORCIONADA POR EL USUARIO
C
      EXTERNAL FUNCIO
C
C  LECTURA DE DATOS
C
      READ(5,5) XA,XINI,XB,CL,EPSFUN,EPSINT,NITMAX,IIMP,
     $  IDESC,IESTD,IUSR
    5 FORMAT()
C
C  LLAMA A SUBRUTINA SHUBERT
C
      CALL SHUBRT(FUNCIO)
C
      STOP
      END
```

```
C*******************************************************************************
C
C      F U N C T O N E S    O B J E T I V O    E J E M P L O
C
C*******************************************************************************
C
C
C  VER LA SUBRUTINA PRINCIPAL PARA
C    - SIGNIFICADO DE LAS VARIABLES EN AREAS COMUNES
C    - FUNCIONES DE LOS SUBPROGRAMAS
C
C
      FUNCTION FUNCIO(X)
C
      DIMENSION V(10,2)
      COMMON/USERIN/XA,XINI,XB,CL,EPSFUN,EPSINT,NITMAX,IINP,
     * IDESC,IESTD,IUSR,INIC,XSOL,FSOL,IFIN,IERR
C
      GO TO (100,200,300,300,400,400),IUSR
C
    5 FORMAT()
C
C  E J E M P L O  1
C
  100 T = 0.0
      DO 150 K = 1,5
  150 T = T + K * SIN((K+1)*X + K)
      FUNCIO = T
      RETURN
C
C  E J E M P L O  2
C
  200 IF(INIC .NE. 0) GO TO 210
      READ(9,5) (V(I,1),V(I,2),I = 1,10)
  210 T = 0.0
      T2 = 0.0
C
      DO 250 I = 1,10
      T1 = 0.0
      T1 = (V(I,1)-X)**2.0 + V(I,2)**2.0
      T2 = SQRT(T1)
      IF(I .EQ. 1) THEN
      T = T2
      ELSE
      T = AMIN1(T2,T)
      END IF
  250 CONTINUE
      FUNCIO = T
      RETURN
C
C  E J E M P L O S  3  Y  4
C
  300 IF(INIC .NE. 0) GO TO 310
      READ(9,5) (V(I,1),V(I,2),I=1,10)
  310 T = 0.0
      DO 350 I = 1,10
      T1 = 0.0
      T2 = 0.0
      T1 = (V(I,1)-X)**2.0 + V(I,2) ** 2.0
      T2 = SQRT(T1)
      T = T + T2
```

```
  350 CONTINUE
      IF(IUSR .EQ. 4) T = -T                          @ EJEMPLO 4
      FUNCIO = T
      RETURN
C
C  E J E M P L O S  5  Y  6
C
  400 IF(INIC .NE. 0) GO TO 410
      READ(9,5) SK2
  410 F1 = SIN(1.5*X)
      F2 = ((X-SK2)/5.0) * SIN((X-SK2)**1.3)
      F3 = SIN(X)
      F3 = SIN(2.0 * F3)
      T = AMIN1(F1,F2,F3)
      IF(IUSR .EQ. 5) GO TO 420
      IF(T .GE. 3.0) T = -T                           @ EJEMPLO 6
      T = 1.0 + T                                     @ EJEMPLO 6
  420 FUNCIO = T
      RETURN
C
      END
```

B.III LISTADOS DE LOS ELEMENTOS SIMBOLICOS DE LAS SUBRUTINAS QUE CONSTITUYEN LA IMPLANTACION DEL ALGORITMO DE SHUBERT.

```
C*************************************************************************
C
C   S U B R U T I N A    P R I N C I P A L    D E L    A L G O R I T M O
C
C                        D E    S H U B E R T
C
C*************************************************************************
C
C
C   EN ESTE PROGRAMA  SE EXPLICAN A TRAVES DE COMENTARIOS
C       - LAS VARIABLES EN AREAS COMUNES Y
C       - LAS FUNCIONES DE CADA SUBPROGRAMA
C
      SUBROUTINE SHUBRT(FUNCOB)
C
C     NMAX   = DIMENSION DEL ARBOL, ESTO ES, NUMERO MAX DE NODOS
C
      PARAMETER NMAX = 615
C
C   INFORMACION PROPORCIONADA POR EL USUARIO
C     XA     = LIMITE IZQUIERDO DEL INTERVALO CONSIDERADO
C     XINI   = PUNTO INICIAL (SEMILLA) PARA INICIAR EL ALGORITMO
C     XB     = LIMITE DERECHO DEL INTERVALO CONSIDERADO
C     CL     = MINIMA COTA SUPERIOR A LA CONSTANTE DE LIPSCHITZ
C     EPSFUN = COTA SUPERIOR AL ERROR PERMITIDO AL CALCULAR EL OPTIMO
C              (VALOR DE DEFAULT  0.001)
C     EPSINT = COTA SUPERIOR AL ERROR PERMITIDO EN LA SELECCION DE
C              INTERVALOS EN LAS ABSCISAS QUE CONTENGAN EL OPTIMO
C              (VALOR DE DEFAULT  0.01)
C     NITMAX = NUMERO MAXIMO DE ITERACIONES PERMITIDAS
C              (VALOR DE DEFAULT  500)
C     IIMP   = INDICADOR DEL TIPO DE IMPRESION REQUERIDA
C     IDESC  = INDICADOR  0 = DESCARTA EL NODO GENERADO QUE SEA MENOR
C                             QUE EL VALOR MAXIMO EN LAS EVALUACIONES
C                             DE LA FUNCION OBJETIVO
C     IESTD  = INDICADOR  1 = GENERA MEDIA,VARIANZA Y DESVIACION DE
C                           LOS NODOS DEL HEAP
C     IUSR   = INDICADOR DISPONIBLE PARA EL USUARIO
C
C   INFORMACION PROPORCIONADA AL USUARIO
C     INIC   = INDICADOR  0 = NO HA SIDO EVALUADA LA FUNCION OBJETIVO
C                         1 = YA HA SIDO EVALUADA LA FUNCION OBJETIVO
C     XSOL   = ABCSISA MAS APROXIMADA A LA SOLUCION
C     FSOL   = VALOR DE LA FUNCION OBJETIVO MAS APROXIMADA A LA SOLUCION
C     IFIN   = INDICADOR DE TERMINACION
C     IERR   = INDICADOR DE ERROR
C
      COMMON/USERIN/XA,XINI,XB,CL,EPSFUN,EPSINT,NITMAX,IIMP,
     * IDESC,IESTD,IUSR,INIC,XSOL,FSOL,IFIN,IERR
C
C   INFORMACION SOBRE LOS NODOS DEL ARBOL
C     N      = NUMERO DE NODOS OCUPADOS
C     Y(I)   = VALOR DEL I-ESIMO NODO
C     XI(I)  = VALOR DE LA ABSCISA IZQUIERDA DEL I-ESIMO NODO
C     XD(I)  = VALOR DE LA ABSCISA DERECHA DEL I-ESIMO NODO
C     YMIN   = VALOR DEL NODO MINIMO
C
      COMMON/SNODOS/N,Y(NMAX),XI(NMAX),XD(NMAX),YMIN
C
C   PARAMETROS
C     NUMIT  = CONTADOR DEL NUMERO DE ITERACIONES
```

```
C     DIF     = DIFERENCIA ENTRE EL NODO MAYOR Y LA MAXIMA EVALUACION
C               DE LA FUNCION OBJETIVO
C     DIFREL  = DIFERENCIA RELATIVA = DIF / MAXIMA EVALUACION DE LA FUNC.
C
      COMMON/PARAMS/NUMIT,DIF,DIFREL
C
C  VALORES ESPECIALES EN LAS EVALUACIONES DE LA FUNCION OBJETIVO
C     VFXA    = VALOR DE LA FUNCION EVALUADA EN XA
C     VFXINI  = VALOR DE LA FUNCION EVALUADA EN XINI
C     VFXB    = VALOR DE LA FUNCION EVALUADA EN XB
C     VFM     = VALOR MAXIMO EN LAS EVALUACIONES DE LA FUNCION OBJETIVO
C     XVFM    = ABSCISA DE LA MAYOR EVALUACION DE LA FUNCION OBJETIVO
C
C
      COMMON/VALFUN/VFXA,VFXINI,VFXB,VFI,VFD,VFM,XVFM
C
C  AUXILIARES EN EL CALCULO DE MEDIA Y VARIANZA
C     SUMY    = SUMA DE LOS VALORES DE LOS NODOS
C     SUMY2   = SUMA DE LOS CUADRADOS DE LOS VALORES DE LOS NODOS
C     VALANT  = VALOR DEL NODO Y(I) QUE SE ELIMINARA
C     VALAN2  = CUADRADO DE VALANT
C     YMED    = MEDIA DE LOS NODOS Y(I)
C     YVAR    = VARIANZA DE LOS NODOS Y(I)
C     ABSINT  = ARREGLO CON LOS INTERVALOS-SOLUCION
C     NINT    = NUMERO DE INTERVALOS-SOLUCION
C     YDES    = DESVIACION DE LOS NODOS Y(I)
C
      COMMON/MEDVAR/SUMY,SUMY2,VALANT,VALAN2,YMED,YVAR,YDES
C
C  SELECCION DE INTERVALOS-SOLUCION
C
      COMMON/SELINT/ABSINT(300,2),NINT
C
C  ASIGNA VALORES A EPSFUN,EPSINT,NITMAX SI ESTOS NO FUERON DADOS
C  POR EL USUARIO
C
      IF(EPSFUN .EQ. 0.0) EPSFUN = 1.E-01
      IF(EPSINT .EQ. 0.0) EPSINT = 1.E-1
      IF(NITMAX .EQ. 0) NITMAX = 500
      IF(IIMP .EQ. 3 .OR. IIMP .EQ. 5) IESTD = 1
C
C  LA SUBRUTINA 'IMPRES' IMPRIME VALORES DE VARIABLES O INDICACIONES
C  DEPENDIENDO DE SUS PARAMETROS DE ENTRADA
C  SI IIMP ES MAYOR O IGUAL A UNO IMPRIME ENCABEZADO INICIAL
C
      IF(IIMP .GE. 1) CALL IMPRES(1,INDINT)
C
C  LA SUBRUTINA 'YXINIC' EFECTUA EL ALGORITMO DE SHUBERT Y OBTIENE
C  LOS PRIMEROS NODOS CON ABSCISAS IZQUIERDA Y DERECHA.
C
      CALL YXINIC(FUNCOB)
      INDINT = 2
C
C  PRINCIPIA EL PROCESO ITERATIVO DEL ALGORITMO DE SHUBERT
C  CON UN LIMITE SUPERIOR DE NITMAX ITERACIONES
C
      DO 100 NUMIT = 1, NITMAX
C
C  SI IIMP MAYOR O IGUAL A CUATRO, ESCRIBE LOS NODOS GENERADOS EN
C  CADA ITERACION CON SUS RESPECTIVAS ABSCISAS IZQUIERDA Y DERECHA
C  ADEMAS LA EVALUACION DE LA FUNCION OBJETIVO EN ESTAS ABSCISAS Y
C  EL VALOR MAXIMO EN CADA ITERACION
```

```
      IF(IIMP .LT. 4) GO TO 9
      CALL IMPRES(5,INDINT)
      CALL IMPRES(6,INDINT)
C
C  LA SUBRUTINA 'HEAP' EFECTUA LOS CAMBIOS, SI SE REQUIEREN, PARA CON-
C  SERVAR LA ESTRUCTURA 'HEAP' DE LOS NODOS Y(IND),XI(IND),XD(IND)
C  DONDE 'IND' PUEDE TOMAR VALORES INICIALES '1' O 'N'.
C  INTERCAMBIA AL PADRE CON EL MAYOR DE SUS HIJOS.
C
   90 IND = 1
      CALL HEAP(IND)
C
      IF(INDINT .EQ. 2) THEN
      IND = N
      CALL HEAP(IND)              & SOLO SI SE GENERO EL NODO(N)
      END IF
C
C
C  LA SUBRUTINA 'YXNOVO' OBTIENE LA DIFERENCIA RELATIVA ENTRE EL NODO
C  MAYOR GENERADO POR EL ALGORITMO DE SHUBERT Y EL MAXIMO VALOR
C  EVALUADO DE LA FUNCION OBJETIVO.
C  SI ESTA DIFERENCIA ES MENOR QUE EPSFUN LA EJECUCION TERMINA. EN CASO
C  CONTRARIO GENERA DOS NUEVOS NODOS, INTRODUCIENDO  AL MAYOR DE ELLOS
C  POR LA RAIZ Y AL MENOR POR LA ULTIMA HOJA. SI ESTE ULTIMO ES MENOR
C  QUE LA MAXIMA EVALUACION DE LA FUNCION, LO DESCARTA.
C  'INDINT' ES UN INDICADOR DE CUANTOS NODOS NUEVOS SE GENERARON.
C  'IFIN' ES UN INDICADOR DE TERMINACION
C
      CALL YXNOVO(FUNCOB,INDINT)
C
      IF(IIMP .GE. 4) CALL IMPRES(7,INDINT)
C
      IF(IFIN .NE. 0) GO TO 200              @ TERMINA EL ALGORITMO
C
C  LA SUBRUTINA 'ESTADS' OBTIENE LA MEDIA,VARIANZA Y DESVIACION
C  DE LOS NODOS DEL HEAP.
C
      IF(IESTD .EQ. 1) CALL ESTADS(INDINT)
C
  100 CONTINUE
C
C
  200 IF(IFIN .EQ. 2) CALL IMPRES(8,INDINT) @ ALCANZO NITMAX ITERACIONES
      IF(IFIN .EQ. 3) CALL IMPRES(10,INDINT) @ SE OCUPO TODO EL HEAP
      IF(IERR .EQ. 1) CALL IMPRES(9,INDINT)  @ ERROR EN LA CTE DE LIP.
C
C  SI IIMP IGUAL A TRES O A CINCO IMPRIME AL ARBOL ENTERO Y
C  LA MEDIA, VARIANZA Y DESVIACION DE SUS NODOS
C
      IF(IIMP .EQ. 3 .OR. IIMP .EQ. 5) CALL IMPRES(4,INDINT)
C
C  LA SUBRUTINA 'INTSOL' GENERA INTERVALOS A PARTIR DE LOS
C  NODOS MAYORES QUE EL VALOR MAXIMO EVALUADO DE LA FUNCION
C  OBJETIVO.  EN ALGUNO(S) DE ESTE(OS) INTERVALO(S) SE LOCA-
C  LIZA EL MAXIMO
C
      CALL INTSOL
C
C  SI IIMP MAYOR O IGUAL A DOS IMPRIME LOS INTERVALOS-SOLUCION
C
      IF(IIMP .GE. 2) CALL IMPRES(3,INDINT)
      IF(IIMP .GE. 1) CALL IMPRES(2,INDINT) @ SOLUCION APROXIMADA
C
      RETURN
      END
```

```
C*************************************************************************
C
C          I N I C I A L I Z A C I O N   D E L   A R B O L          B.12
C
C*************************************************************************
C
C
C  VER LA SUBRUTINA PRINCIPAL PARA
C    - SIGNIFICADO DE LAS VARIABLES EN AREAS COMUNES
C    - FUNCIONES DE LOS SUBPROGRAMAS
C
C
      SUBROUTINE YXINIC(FUNCOB)
C
      PARAMETER NMAX = 600
      COMMON/USERIN/XA,XINI,XB,CL,EPSFUN,EPSINT,NITMAX,IIMP,
     * IDESC,IESTO,IUSR,INIC,XSOL,FSOL,IFIN,IERR
      COMMON/SNODOS/XN,YI(NMAX),XI(NMAX),XD(NMAX),YMIN
      COMMON/PARAMS/NUMIT,DIF,DIFREL
      COMMON/VALFUN/VFXA,VFXINI,VFXB,VFI,VFD,VFM,XVFM
      COMMON/MEDVAR/SUMY,SUMY2,VALANT,VALAN2,YMED,YVAR,VDES
      DIMENSION XT(2),YT(2),VFT(2)          ! VARIABLES AUXILIARES
C
      INIC = 0          ! ANTES DE LLAMAR A LA FUNCION OBJETIVO
      IFIN = 0          ! MIENTRAS DIFREL MAYOR QUE EPSFUN
      IERR = 0          ! EJECUCION SIN ERRORES
      YMIN = 1.0E35     ! VALOR INICIAL MUY GRANDE
      VFM = -1.0E35     ! VALOR INICIAL MUY CHICO
C
C  EVALUA LA FUNCION OBJETIVO EN LOS EXTREMOS DEL INTERVALO CONSIDERADO
C  Y EN EL PUNTO INICIAL (SEMILLA)
C
      VFXINI = FUNCOB(XINI)
      INIC = 1          ! CUANDO YA EVALUO LA FUNCION OBJETIVO
      VFXA = FUNCOB(XA)
      VFXB = FUNCOB(XB)
C
C  OBTIENE LOS DOS PRIMEROS NODOS. CADA UNO CON UNA ABSCISA.
C
      XT(1) = (CL * XINI + VFXINI + CL * XA - VFXA) / (2.0 * CL)
      YT(1) = (CL * XINI + VFXINI - CL * XA + VFXA) / 2.0
      VFT(1) = FUNCOB(XT(1))
C
      XT(2) = (CL * XB + VFXB + CL * XINI - VFXINI) / (2.0 * CL)
      YT(2) = (CL * XB + VFXB - CL * XINI + VFXINI) / 2.0
      VFT(2) = FUNCOB(XT(2))
C
C  OBTIENE LOS DOS PRIMEROS NODOS CON DOS ABSCISAS CADA UNO.
C  ADEMAS INICIALIZA SUMY  Y  SUMY2
C
      DO 10 J = 1,2
      XI(J) = (2.0  *  CL * XT(J) - YT(J) + VFT(J)) / (2.0 * CL)
      XD(J) = (2.0  *  CL * XT(J) + YT(J) - VFT(J)) / (2.0 * CL)
      Y(J) = VFT(J) + (YT(J) - VFT(J)) / 2.0
C
      SUMY = SUMY + Y(J)
      SUMY2 = SUMY2 + Y(J)**2.0
   10 CONTINUE
C
C  ASIGNA AL NODO MAYOR A LA RAIZ  Y AL MENOR A LA ULTIMA HOJA
C
      IF(Y(2) .GT. Y (1)) CALL CAMBIO(1,2)
      N = 2
      RETURN
C
      END
```

```
C*************************************************************************
C
C          G E N E R A C I O N    D E    N U E V O S    N O D O S
C
C*************************************************************************
C
C
C  VER LA SUBRUTINA PRINCIPAL PARA
C     - SIGNIFICADO DE LAS VARIABLES EN AREAS COMUNES
C     - FUNCIONES DE LOS SUBPROGRAMAS
C
C
      SUBROUTINE YXNOVO(FUNCOB,INDINT)
C
      PARAMETER NMAX = 600
      COMMON/USERIN/XA,XINI,XB,CL,EPSFUN,EPSINT,NITMAX,IIMP,
     * IDESC,IESTD,IUSR,INIC,XSOL,FSOL,IFIN,IERR
      COMMON/NODOS/N,Y(NMAX),XI(NMAX),XD(NMAX),YMIN
      COMMON/PARAMS/NUMIT,DIF,DIFREL
      COMMON/VALFUN/VFXA,VFXINI,VFXB,VFI,VFD,VFM,XVFM
      COMMON/MEDVAR/SUMY,SUMY2,VALANT,VALAN2,YMED,YVAR,YDES
C
C  EVALUA LA FUNCION OBJETIVO EN LAS ABSCISAS IZQUIERDA Y DERECHA
C  DEL NODO(1). ESTE SERA POSTERIORMENTE ELIMINADO, DANDO LUGAR A
C  DOS NUEVOS NODOS.
C
      VFI = FUNCOB(XI(1))
      VFD = FUNCOB(XD(1))
C
C  GUARDA LA MAXIMA EVALUACION DE LA FUNCION OBJETIVO
C
      VFM = AMAX1(VFM,VFI,VFD)
C
C  GUARDA LA ABSCISA DE LA MAYOR EVALUACION DE LA FUNCION OBJETIVO
C
C
      IF(VFM .EQ. VFI) XVFM = XI(1)
      IF(VFM .EQ. VFD) XVFM = XD(1)
C
C
C  OBTIENE LA DIFERENCIA RELATIVA Y LA COMPARA CON EPS
C
      DIF = Y(1) - VFM
C
C  SI DIF ES MENOR QUE CERO IMPLICA LA CONSTANTE DE LIPSCHITZ
C  ELEGIDA ES MENOR QUE LA REAL
C
      IF(Y(1) .GE. VFM) GO TO 8
      IFIN = 4
      IERR = 1
      RETURN
C
    8 DIFREL = ABS(DIF/VFM)
      IF(DIFREL .GT. EPSFUN) GO TO 10
      IFIN = 1                                  @ DIFERENCIA RELATIVA MENOR
      RETURN                                    @ QUE EPSFUN, TERMINA.
C
C  VERIFICA QUE EL NUMERO DE ITERACIONES SEA MENOR QUE EL NUMERO
C  MAXIMO DE ITERACIONES PERMITIDO
C
   10 IF(NUMIT .LT. NITMAX) GO TO 12
```

```
      IFIN = 2
      RETURN
C
C  VERIFICA QUE EXISTE LUGAR EN EL ARBOL PARA GENERAR NUEVOS NODOS
C
   12 IF(N .LT. NMAX) GO TO 14
      IFIN = 2
      RETURN
C
C  INDINT ES EL NUMERO DE NODOS QUE SE GENERARON
C
   14 INDINT = 2                                  @ INICIALIZA EN 2
      N = N + 1                                   @ AUMENTA TAMANIO DEL ARBOL
C
C  GUARDA EL VALOR Y EL CUADRADO DE LA RAIZ, PUES SE VA A ELIMINAR,
C  Y SE DEBE QUITAR SU PESO DE LA MEDIA Y VARIANZA DE LOS NODOS
C
      VALANT = Y(1)
      VALAN2 = Y(1)**2.0
C
C  CALCULA NUEVOS VALORES PARA NODOS '1', 'N'
C
      DIFI = Y(1) - VFI
      Y(N) = VFI + (DIFI / 2.0)
      XI(N) = (2.0 * CL * XI(1) - DIFI) / (2.0 * CL)
      XD(N) = (2.0 * CL * XI(1) + DIFI) / (2.0 * CL)
C
      DIFD = Y(1) - VFD
      Y(1) = VFD + (DIFD / 2.0)
      XI(1) = (2.0 * CL * XD(1) - DIFD) / (2.0 * CL)
      XD(1) = (2.0 * CL * XD(1) + DIFD) / (2.0 * CL)
C
C
C  COLOCA AL MAYOR DE LOS NODOS EN LA RAIZ Y AL MENOR EN LA ULTIMA HOJA
C
      IF(Y(N) .GT. Y(1)) THEN
      JC = N
      CALL CAMBIO(1,JC)
      END IF
C
C  OBTIENE EL NODO MINIMO
C
      YMIN = AMIN1(YMIN,Y(1),Y(N))
C
C  SI LA ULTIMA HOJA ES MENOR QUE EL VALOR MAXIMO DE LA FUNCION
C  LA DESCARTA. (SI 'IDESC' DISTINTO DE CERO.)
C
      IF(IDESC .NE. 0) GO TO 100
      IF(Y(N) .GE. VFM) RETURN
      N = N - 1
      INDINT = 1
  100 RETURN
C
      END
```

```
C*************************************************************************
C
C       C O N S E R V A    L A    E S T R U C T U R A    H E A P          B.15
C
C*************************************************************************
C  VER LA SUBRUTINA PRINCIPAL PARA
C    - SISNIFICADO DE LAS VARIABLES EN AREAS COMUNES
C    - FUNCIONES DE LOS SUBPROGRAMAS
C
C
      SUBROUTINE HEAP(IND)
      PARAMETER NMAX = 600
      COMMON/SNODOS/N,Y(NMAX),XI(NMAX),XD(NMAX),YMIN
C
C
C  'IND' AL ENTRAR TOMA LOS VALORES '1' O 'N', PERO COMO CAMBIA
C  A TRAVES DEL PROGRAMA, SE CREA UNA VARIABLE AUXILIAR IORD.  ESTA
C  ES CERO SI SE REVISAN ASCENDIENTES Y UNO SI DESCENDIENTES.
C
      IORD = 0
      IF(IND .EQ. 1) IORD = 1
      JC = IND                          @ INICIALIZA
C
C  INDDES  ES EL APUNTADOR DEL ULTIMO NODO CON DESCENDIENTES
C
      INDDES = N / 2
C
C  NUMDES  ES EL NUMERO DE HIJOS QUE TIENE EL NODO CONSIDERADO
C
   10 NUMDES = 2
      IF(IORD .EQ. 1) THEN
      IND = JC
C
C  SI 'IND' ES MAYOR QUE 'INDDES', IMPLICA ES UNA HOJA Y NO VERI-
C  FICA DESCENDIENTES.
C
      IF(IND .GT. INDDES) RETURN
      ELSE
      IND = IND / 2
      IF(IND .EQ. 0) RETURN                 @ ES LA RAIZ, YA NO PUEDE
      END IF                                @ ASCENDER MAS
C
      INDI = 2 * IND                        @ APUNTADOR HIJO-IZQUIERDO
      INDD = INDI + 1                       @ APUNTADOR HIJO-DERECHO
C
C  LA SIGUIENTE INSTRUCCION IMPLICA QUE EL NODO CONSIDERADO  SOLO
C  TIENE UN DESCENDIENTE.
C
      IF(INDD .GT. N) NUMDES = 1
C
C  SI UN NODO ES MAYOR QUE SUS HIJOS NO REQUIERE CAMBIOS
C
      IF(NUMDES .EQ. 1) THEN
      IF(Y(IND) .GE. Y(INDI)) RETURN
      ELSE
      IF(Y(IND) .GE. Y(INDI) .AND. Y(IND) .GE. Y(INDD)) RETURN
      END IF
C
C  CAMBIA AL PADRE CON EL HIJO MAYOR
C
      JC = INDI
      IF(Y(INDD) .GT. Y(INDI) .AND. INDD .LE. N) JC = INDD
      CALL CAMBIO(IND,JC)
C
      GO TO 10
      END
```

```
C*************************************************************************
C
C                    I N T E R C A M P I A    N O D O S
C
C*************************************************************************
C
C
C  VER LA SUBRUTINA PRINCIPAL PARA
C     - SIGNIFICADO DE LAS VARIABLES EN AREAS COMUNES
C
C
      SUBROUTINE CAMBIO(IC,JC)
      PARAMETER NMAX = 600
      COMMON/SNODOS/N,Y(NMAX),XI(NMAX),XD(NMAX),YMIN
C
C  ESTA SUBRUTINA INTERCAMBIA LOS NODOS DE LA POSICION 'IC' CON LA 'JC'
C
      TY = Y(JC)
      TXI = XI(JC)
      TXD = XD(JC)
C
      Y(JC) = Y(IC)
      XI(JC) = XI(IC)
      XD(JC) = XD(IC)
C
      Y(IC) = TY
      XI(IC) = TXI
      XD(IC) = TXD
C
      RETURN
      END
```

```
C*************************************************************************
C
C                          E S T A D I S T I C A S
C
C*************************************************************************
C
C
C  VER LA SUBRUTINA PRINCIPAL PARA
C     - SIGNIFICADO DE LAS VARIABLES EN AREAS COMUNES
C     - FUNCIONES DE LOS SUBPROGRAMAS
C
C
      SUBROUTINE ESTADS(INDINT)
      PARAMETER NMAX = 500
      COMMON/SNODOS/N,Y(NMAX),XI(NMAX),XD(NMAX),YMIN
      COMMON/MEDVAR/SUMY,SUMY2,VALANT,VALAN2,YMED,YVAR,YDES
C
C
C  GUARDA EL VALOR Y CUADRADO DEL (LOS) NUEVO(S) NODO(S) GENERADO(S)
C
      IF(INDINT .EQ. 1) THEN
      VAL = Y(1)
      VAL2 = Y(1)**2.0
      ELSE
      VAL = Y(1) + Y(N)
      VAL2 = Y(1)**2.0 + Y(N)**2.0
      END IF
C
C  OBTIENE LA NUEVA SUMA DE LOS NODOS Y DE SUS CUADRADOS
C
      SUMY = SUMY - VALANT + VAL
      SUMY2 = SUMY2 - VALAN2 + VAL2
C
C  OBTIENE LA MEDIA, VARIANZA Y DESVIACION DE LOS NODOS
C
      ACTN = FLOAT(N)
      YMED = SUMY / ACTN
      YVAR = (SUMY2/ACTN) - (YMED**2.0)
      YDES = SQRT(YVAR)
C
C
      RETURN
      END
```

```
C*********************************************************************
C
C  G E N E R A C I O N   D E   I N T E R V A L O S - S O L U C I O N
C
C*********************************************************************
C
C
C  VER LA SUBRUTINA PRINCIPAL PARA
C    - SIGNIFICADO DE LAS VARIABLES EN AREAS COMUNES
C    - FUNCIONES DE LOS SUBPROGRAMAS
C
C
      SUBROUTINE INTSOL
C
      PARAMETER NMAX = 600, NINTMX = 300
      COMMON/USERIN/XA,XINI,XB,CL,EPSFUN,EPSINT,NITMAX,IIMP,
     * IDESC,IESTD,IUSR,INIC,XSOL,FSOL,IFIN,IERR
      COMMON/SNODOS/N,Y(NMAX),XI(NMAX),XD(NMAX),YMIN
      COMMON/VALFUN/VFXA,VFXINI,VFXB,VFI,VFD,VFM,XVFM
      COMMON/SELINT/ABSINT(NINTMX,2),NINT
      DIMENSION IJIZQ(NINTMX),IJDER(NINTMX),ISTACK(NINTMX)
      DIMENSION ABSIZQ(NINTMX),ABSDER(NINTMX)
C
C  INICIALIZA INDICES
C
      IK = 0
      NINT = 1
      INIC = 0
C
C  EL SIGUIENTE LOOP FORMA EL ARBOL BINARIO EN INORDER MEDIANTE
C  APUNTADORES. ORDENANDO ASI A LAS ABSCISAS XI(J) PARA TODA 'J'
C  QUE SATISFAGA QUE Y(J) SEA MAYOR QUE EL VALOR MAXIMO EN LAS
C  EVALUACIONES DE LA FUNCION OBJETIVO.
C  IJIZQ = APUNTADOR DEL NODO-HIJO-IZQUIERDO
C  IJDER = APUNTADOR DEL NODO-HIJO-DERECHO
C
      DO 100 I=1,N
      IF(Y(I) .LE. VFM) GO TO 100
C
      IF(INIC .EQ. 0) THEN
      INIC = I
      GO TO 100
      END IF
C
      IPOS = INIC
   50 IF(XI(I) .LE. XI(IPOS)) THEN
C
      IF(IJIZQ(IPOS) .EQ. 0) THEN
      IJIZQ(IPOS) = I
      GO TO 100
      END IF
      IPOS = IJIZQ(IPOS)
C
      ELSE
      IF(IJDER(IPOS) .EQ. 0) THEN
      IJDER(IPOS) = I
      GO TO 100
      END IF
      IPOS = IJDER(IPOS)
C
      END IF
```

```
      GO TO 5
  1[ CONTINUE
C
C  EN LO SIGUIENTE LA SUBRUTINA RECORRE EL ARBOL EN INORDER
C
      IPOS = 1
      NSTACK = 1
      ISTACK(1) = 1
C
  4[ CONTINUE
C
      IF(IJIZQ(IPOS) .NE. 0) THEN
      IPOS = IJIZQ(IPOS)
      NSTACK = NSTACK + 1
      ISTACK(NSTACK) = IPOS
      GO TO 4[
      END IF
C
  5[ CONTINUE
C
C  GENERA LOS INTERVALOS DONDE SE ENCUENTRA LA SOLUCION. SI LA
C  DIFERENCIA INCREMENTO ENTRE DOS INTERVALOS ES MAYOR QUE EL
C  PARAMETRO EPSINT LOS UNE.
C
      T = VEM - Y(IPOS)
      ABSIZQ(1) = (T + CL * XI(IPOS)) / CL
      ABSDER(1) = (-T + CL * XI(IPOS)) / CL
      ABSIZQ(2) = (T + CL * XD(IPOS)) / CL
      ABSDER(2) = (-T + CL * XD(IPOS)) / CL
C
      DO 45[ J = 1,2
      IF(IK .EQ. 0 .AND. J .EQ. 1) THEN
      ABSINT(1,1) = ABSIZQ(1)
      IK = 1
      GO TO 45[
      END IF
C
      DIFINT = ABSIZQ(J) - ABSINT(NINT,2)
C
      IF(DIFINT .GT. EPSINT) THEN
      NINT = NINT + 1
      IF(NINT .LT. NINTMX) GO TO 44[
      IERR = 2
      RETURN
 44[  ABSINT(NINT,1) = ABSIZQ(J)
      END IF
C
  45[ ABSINT(NINT,2) = ABSDER(J)
C
C
      NSTACK = NSTACK - 1
C
      IF(IJDER(IPOS) .NE. 0) THEN
      IPOS = IJDER(IPOS)
      NSTACK = NSTACK + 1
      ISTACK(NSTACK) = IPOS
      GO TO 4[0
      END IF
C
      IF(NSTACK .EQ. 0) RETURN
      IPOS = ISTACK(NSTACK)
      GO TO 5[
C
      END
```

```
C*********************************************************************
C
C                          I M P R E S I O N                          B.20
C
C*********************************************************************
C
C
C  VER LA SUBRUTINA PRINCIPAL PARA
C    - SIGNIFICADO DE LAS VARIABLES EN AREAS COMUNES
C    - FUNCIONES DE LOS SUBPROGRAMAS
C
C
      SUBROUTINE IMPRES(KIND,INDINT)
      PARAMETER NMAX = 600
      COMMON/USERIN/XA,XINI,XB,CL,EPSFUN,EPSINT,NITMAX,IIMP,
     * IDESC,IESTD,IUSR,INIC,XSOL,FSOL,IFIN,IERR
      COMMON/SNODOS/N,Y(NMAX),XI(NMAX),XD(NMAX),YMIN
      COMMON/PARAMS/NUMIT,DIF,DIFREL
      COMMON/VALFUN/VFXA,VFXINI,VFXB,VFI,VFD,VFM,XVFM
      COMMON/MEDVAR/SUMY,SUMY2,VALANT,VALAN2,YMED,YVAR,YDES
      COMMON/SELINT/ABSINT(300,2),NINT
C
C
      GO TO (10,20,30,40,50,60,70,80,90,100),KIND
C
C  SI KIND = 1 IMPRIME LA INFORMACION CON QUE EL USUARIO ALIMENTO
C  AL ALGORITMO DE SHUBERT
C
   10 WRITE(6,11)
   11 FORMAT(1H1,///,15X,'***  ALGORITMO DE SHUBERT PARA OPTIMIZACION',
     * ' GLOBAL  ***',////)
   12 WRITE(6,13) XA,XB,XINI,CL,EPSFUN,EPSINT,NITMAX,IIMP,IDESC,IESTD
   13 FORMAT('  INTERVALO CONSIDERADO',T32,'(',F15.6,',',F15.6,')',/,
     *       '  PUNTO INICIAL',T40,F15.6,/,
     *       '  CONSTANTE DE LIPSCHITZ',T40,F15.6,/,
     *       '  ERROR MAXIMO PERMITIDO EPSFUN',T40,F15.6,/,
     *       '  ERROR MAXIMO PERMITIDO EPSINT',T40,F15.6,/,
     *       '  NUMERO MAXIMO DE ITERACIONES',T40,I15,/,
     *       '  TIPO DE IMPRESION',T40,I15,/,
     *       '  PARAMETRO IDESC',T40,I15,/,
     *       '  PARAMETRO IESTD',T40,I15,//////)
      RETURN
C
C  SI KIND = 2 IMPRIME LA ABSCISA DE LA MAYOR EVALUACION DE LA
C  FUNCION OBJETIVO Y DICHA EVALUACION.
C
   20 WRITE(6,21) NUMIT,XVFM,VFM
   21 FORMAT(//,'  PUNTO DE LA MAXIMA EVALUACION DE LA FUNCION OBJETIVO'
     $ ,/,' EN',I5,'  ITERACIONES ',10X,'(',F15.6,' ,',F15.6,')')
      RETURN
C
C  SI KIND = 3 IMPRIME LOS INTERVALOS DONDE (EN ALGUNO DE ELLOS)
C  SE LOCALIZA EL OPTIMO.
C
   30 WRITE(6,31)
   31 FORMAT(//,'  INTERVALO(S) DONDE SE LOCALIZA(N) EL(LOS) OPTIMO(S)'
     $  ,//)
      WRITE(6,32) (ABSINT(J,1),ABSINT(J,2),J=1,NINT)
   32 FORMAT(6X,'(',F15.6,',',F15.6,')')
      RETURN
C
```

```
C  SI KIND=4 IMPRIME LOS NODOS MAXIMO, MINIMO, MEDIA, VARIANZA
C  Y DESVIACION DE LOS NODOS QUE FORMAN EL HEAP.  ADEMAS IMPRIME
C  TODOS LOS NODOS DEL HEAP CON SUS ABSCISAS IZQUIERDAS Y DERECHAS
C  QUE LES CORRESPONDEN
C
   40 WRITE(6,41) Y(1),YMIN,YMED,YVAR,YDES
   41 FORMAT(/,'  NODO MAXIMO',F15.6,'  NODO MINIMO',F15.6,/,'  MEDIA',
     * F15.6,'   VARIANZA',F15.6,'   DESVIACION',F15.6)
C
      WRITE(6,42)
   42 FORMAT(/,'  NODO',5X,6X,'VALOR',4X,5X,2X,'ABSCISA IZQ  ',
     * 5X,2X,'ABSCISA DER',//)
      DO 44 I = 1,N
      WRITE(6,43) I,Y(I),XI(I),XD(I)
   43 FORMAT(I6,3(5X,F15.6))
   44 CONTINUE
      RETURN
C
C  SI KIND=5 IMPRIME EL NUMERO DE ITERACION Y LA DIMENSION DEL ARBOL
C  AL MOMENTO DE SER LLAMADA LA SUBRUTINA
C
   50 WRITE(6,51) NUMIT,N
   51 FORMAT(/,'  ITERACION NUMERO',I5,'   NUMERO DE NODOS DEL'
     * ' ARBOL',I5)
      RETURN
C
C  SI KIND=6 IMPRIME (EL) LOS NUEVO(S) NODO(S) GENERADO(S) CON SU(S)
C  ABSCISAS IZQUIERDA(S) Y DERECHA(S) QUE LE(S) CORRESPONDE(N)
C
   60 WRITE(6,61)
   61 FORMAT(/,'  NODO(S) GENERADO(S) ',/,19X,'VALOR',11X,'ABSCISA',
     $  ' IZQ  ',7X,'ABSCISA DER')
      WRITE(6,62) Y(1),XI(1),XD(1)
      IF(INDINT .EQ. 2) WRITE(6,62) Y(N),XI(N),XD(N)
   62 FORMAT(9X,3(5X,F15.6))
      RETURN
C
C  SI KIND = 7 IMPRIME LA ULTIMA EVALUACION DE LA FUNCION OBJETIVO
C  EN LAS ABSCISAS IZQUIERDA Y DERECHA DEL NODO(1).  ADEMAS LA MAXIMA
C  EVALUACION HASTA AHORA OBTENIDA, EL NODO MAXIMO Y LAS DIFERENCIAS
C  ABSOLUTA Y RELATIVA ENTRE ELLOS
C
   70 WRITE(6,71) VFI,VFD,XVFM,VFM,Y(1),DIF,DIFREL
   71 FORMAT(/,'  VALORES DE LA FUNCION EVALUADA ',/,
     *  '  IZQUIERDO=',F15.6,'   DERECHO=',F15.6,/,
     *  '  MAXIMO=(X,F(X))= (',F15.6,',',F15.6,')',T65,'...(1)',/,
     *  '  NODO MAXIMO=Y(1)=',F15.6,T65,'...(2)',/,
     *  '  DIFERENCIA=(2)-(1)=',F15.6,
     *  '  DIFERENCIA RELATIVA= DIFERENCIA/(1) =',F15.6)
      RETURN
C
C  SI KIND = 8 IMPRIME MENSAJE DE TERMINACION POR LLEGAR AL NUMERO
C  MAXIMO DE ITERACIONES PROPUESTO
C
   80 WRITE(6,81) NITMAX
   81 FORMAT('  TERMINA POR ALCANZAR EL NUMERO MAXIMO DE',
     * ' ITERACIONES',I5)
      RETURN
C
C  SI KIND = 9 IMPRIME INDICACION DE ERROR EN LA CONSTANTE DE LIPSCHITZ
C
   90 WRITE(6,91)
```

```
 91 FORMAT(//,'  *** ERROR  LA CONSTANTE DE LIPSCHITZ DEBE SER MAYOR',
     $    ' QUE LA INDICADA ***',/)
      RETURN
C
C  SI KIND = 10 IMPRIME MENSAJE DEBIDO A QUE TODOS LOS NODOS DEL ARBOL
C  SE OCUPARON, POR LO QUE ES NECESARIO AGRANDAR EL PARAMETRO NMAX EN
C  TODOS LOS PROGRAMAS
C
  100 WRITE(6,101) NMAX
  101 FORMAT(//,'  *** PROBLEMA  LOS',I5,' NODOS DEL ARBOL ESTAN '
     $  'OCUPADOS DEBERA AGRANDARSE EL PARAMETRO "NMAX" ***')
      RETURN
C
      END
```

M-0037497

BIBLIOGRAFIA

[1] Aho A. V., Hopcroft J. E., Ullman J. D., "The Design and Analysis of Computer Algorithms", Addison-Wesley Publishing Co., Inc., October 1975.

[2] Apostol Tom M., "Mathematical Analysis", Addison-Wesley Publishing Co., Inc., 1974.

[3] Beale E. M. L., Forrest J. J. H., "Global Optimization Using Special Ordered Sets". Mathematical Programming 10, North-Holland Publishing Company, february 1976.

[4] Boender C. G. E., Rinnooy K., Stougie L., Timmer G. T., "A - Stochastic Method for Global Optimization", Econometric Institute, Erasmus University, Rotterdam, 1980.

[5] Bondy J. A., Murty U.S.R., "Graph Theory with Applications", - The MacMillan Press Ltd., London and Basingstoke, 1976.

[6] Bradley, Stephen P., Hax, Arnoldo C., Magnanti, Thomas L. - - "Applied Mathematical Programming", Cap. 14, Addison-Wesley - Publishing Company, 1977.

[7] Branin, F. H. Jr., "Widely Convergent Method for Finding Multiple Solutions of Simultaneous Nonlinear Equations", IBM Journal of Research and Development, (504-522), Sept. 1972.

[8] Calvillo V. Gilberto, "A Direct Search Method for Multivariable Global Optimization", Master in Applied Science Thesis, Department of Management Sciences, University of Waterloo, February. 1975

[9] Conte, S. D., Boor Carl, "Análisis Numérico Elemental", McGraw Hill, Inc. U.S.A., 1972.

[10] Dixon L., Szegö G., "Towards Global Optimization", North-Holland Publishing Company, Amsterdam 1975.

[11] Escudero L. F., Váquez Muñiz A., "The Use of Mixed Integer Programming for the Evaluation of Some Alternative Air Pollution - Abatement Polices", Optimization Techniques, Proceedings 7th - International Federation for Information Processing, Nice, September, 1975.

[12] Fishburn Peter C., "Decision and Value Theory", Operations Research Society of America, John Wiley & Sons, Inc. 1964.

[13] Goldstein, A. A., y Price J. F., "On Descendent from Local Minima", Maths of Computation, 25, No. 115, 1971.

[14] Gottfried, B. S., Weisman, J., "Introduction to Optimization - Theory", Prentice-Hall, Inc. New Yersey, 1973.

[15] Hansen, E. R., "Global Optimization Using Interval Analysis: The One-Dimensional Case", Journal of Optimization Theory and Applications, Vol. 29, No. 3. November, 1979.

[16] Himmelblau David M., "Applied Nonlinear Programming" McGraw-Hill Inc. 1972.

[17] "Journal of Optimization Theory and Applications". Edited by Plenum Publishing Corporation, New York-London.

[18] Knuth, Donald E. "The Art of Computer Programming". Vol. 1, Fundamental Algorithms. Addison-Wesley Publishing Co., Inc. 1973.

[19] Lasdon, L. S., Fox, R., Ratner, M. "Nonlinear Optimization -- Using the Generalized Reduced Gradient Method" Tech. Memo. 325 - Dept. of Operations Research, Case Western Reserve U., Cleveland Ohio, March 1975.

[20] Linkens, D. A., "Optimization in the Modelling of Digestive Tract Electrical Signals" Optimization Techniques, Proceedings 7th. International Federation for Information Processing, Nice. September 1975.

[21] Litt, F. S., Smets, H. "Optimal Pollution Control of a Lake", - Optimization Techniques, Proceedings 7th. International Federation for Information Processing, Nice, September, 1975.

[22] Luenberger, David G., "Introduction to Linear and Nonlinear Programming", Addison-Wesley Publishing Co., Inc. 1973.

[23] "Mathematical Programming" Edited by the Mathematical Programming Society, North-Holland Publishing Company, Amsterdam.

[24] "Operations Research" Journal Edited by the Operations Research Society of America, Waverly Press, Inc. Baltimore.

[25] Shubert, Bruno O. "A Sequential Method Seeking the Global Maximum of a Function", SIAM Journal Numerical Analysis, Vol. 9, No. 3. September 1972.

[26] Trajtenbrot, B. A. "Introducción a la Teoría Matemática de las - Computadoras y de la Programación" Siglo XXI Editores, S. A., México, 1967.

[27] Velasco, Levy A., y Montalvo, A. "Algoritmo de Tunelización para Optimización Global de Funciones", Comunicaciones Técnicas, - IIMAS, UNAM, No. 204.

[28] Wetts, R. J., Solís, F. "Random Search Techniques" A la fecha no publicado.