UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES

ACATLAN

"APLICACION DEL METODO HILL-CLIMBING EN LA OPTIMIZACION NO LINEAL"

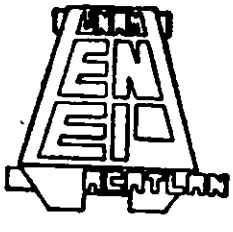
T E S I S

QUE PARA OBTENER EL TITULO DE:

LICENCIADA EN MATEMATICAS APLICADAS Y COMPUTACION

P R E S E N T A :

ANGELICA MENDOZA ANZALDO



Acatlán, Edo. México.

Mayo de 1998.

FALTAN PAGINAS

De la: I

A la: IV

Contenido

Dedicatoria

Quiero aprovechar para dedicar este primer trabajo impreso a:

Mis padres por habernos dado a mi hermano y a mi la mejor familia que pudimos tener.
Mi abuelito Joaquín que siempre ha sido tan ocurrente,
Mi hermano por que con su dedicación al estudio me animó a seguir hasta el final,
Mi tía Rosita Anzaldo por ser un ejemplo de superación,
Mi querida prima Leticia con la que ha compartido tantos momentos. Gracias por ser tan buena amiga,
Mi prima Sandra que siempre tuvo consideraciones conmigo,
Mis primos Arianna, Horacio, Ilia, Marifer,
Mis tíos Alicia, Manuel Rodríguez, Bertha, Manuel Aranda, Mayela,
No puede faltar en esta lista mi erudito primo Armando que ha sido un gran ejemplo para todos sus primos. El ha sido una gran motivación para todos nosotros,
Mis amigas que para mi han sido las mejores: Massiel, Alina, Marisela,
Mi querido amigo Mariano (q.d.e.p.), tú siempre estás ahí para alentarme y hacer que siempre quiera salir adelante,
No puedo dejar de mencionar a algunos de mis compañeros que convivieron conmigo mi vida escolar:

Eduardo Padilla, Patricia Valencia, Gerardo Lozano, Araceli Valerio, Alejandra Quirino, Perla Mejía, Patricia Pierres, Edgardo, Hebert del Castillo, Cecilia Montiel.
Gabriela Tapia, Diana García,
Teresa Bárcenas, María Elena, Luis Fernando Castillo, Rosaura Heredia, Sergio Martínez, Jorge Luis Cervantes, Josefina Camacho, Felipe, Ana Isela Bautista,
Ana Teresa, Sandra Rojas Cruz, Luis Jaime,
Patricia Gachuz, Silvia Ibarra, Manuel Estrada, Octavio Cruz, Laura Mónica, Eduardo Schmidkonz (q.d.e.p.), Mónica Violeta, Laura del Angel, Marsela Mijares, Mayte Van Scoit, Fernando Gómez, Rubí González,
Aquellos amigos que aunque no estuvieron en mi vida escolar si compariteron mi vida personal:
Omar Bracho, Carlos Padilla, Maribel González, Ignacio Padilla, Ricardo Ramírez, Norma Baltazar, Charlie Boccacio,
Y mis heroínas de la vida cotidiana Hilda Hernández, Lidia Barrasa, Irma Niemeyer.

Mis maestros que con tanto esmero dieron lo mejor de sí para que nosotros aprendieramos:
Profa. Bertha Leticia, Profa. Luz Hernández Padilla, Profa. Blanca, Prof. Lucio Pérez (q.d.e.p.), Act. Mari Carmen González Videgaray, Mat. Jorge Luis Suárez Madariaga, Ing. Reyes García Moncada, Mtro. Víctor Palencia, Ing. Erick Maissner, Ing. Miguel Cisneros, Ing. Fernando Matínez, Ing. Beatriz Clavel, Lic.Rosa Pérez, Mat. Acosta,

Por último al honorable jurado que tan amablemente revisó este trabajo.

A todos y cada uno les agredezco que me hayan apoyado en diferentes momentos de mi vida, siempre estarán en mi corazón.

Angélica Mendoza Anzaldo
Mayo 1998

Introducción

"La imaginación es más
importante que el conocimiento"
Albet Einstein

En el ámbito empresarial, a veces no basta la experiencia ni la intuición para resolver un problema, sino que, en la búsqueda de encontrar su mejor solución, es necesario ayudarse con técnicas matemáticas. Existe un área de estudio llamada Investigación de Operaciones que reúne algunas de las técnicas que resuelven las dificultades del ejecutivo de hoy. En la realidad, frecuentemente se encuentran problemas que sólo pueden ser expresados mediante funciones no lineales. Dentro de las áreas de la Investigación de Operaciones se encuentra una llamada: Programación no Lineal, que como su nombre lo dice, resuelve problemas no lineales. Pero existe cierto rechazo cuando aparece algún problema que requiera ser resulto por las técnicas de esta área antes mencionada. Esto es debido a que, dependiendo del tipo de problema con el que se encuentre (restringidos, no restringidos, de una variable, de varias variables, diferenciables, o no diferenciables, etc.), para cada uno de los casos se requiere un algoritmo diferente. Y por lo general, estos algoritmos utilizan derivadas u operaciones matriciales que hacen complicado su entendimiento, aplicación y en algunos casos su

programación en computadora.

Tomando en cuenta lo anterior elaboré un método muy sencillo (ya que es muy fácil de programar y no tiene complicaciones), que resuelve problemas no lineales para funciones de una o varias variables, así como también funciones restringidas y no restringidas llamándolo **Método Hill_Climbing**. Su nombre deriva por una comparación con una subida de montaña a la cual se quiere llegar partiendo de varios puntos, al cabo de un tiempo determinado se escogerá la mejor posición y desecharán las demás; para después partir de este nuevo punto otra vez con un conjunto de direcciones diferentes. Este proceso se repetirá hasta alcanzar la cima.

Este algoritmo esta basado en estudios e investigaciones de Rechenber (Rechenbarg [26]), solo que aquí se aplica a la Optimización No Lineal. La idea básica de este método es una semejanza con en la evolución biológica, utilizando algunas de las reglas de la evolución mas representativas, como son, la selección, la mutación, la duplicación, etc. Simplificando el método Hill-Climbing se resume de la siguiente forma: Se tienen n valores iniciales, estos se duplican y se generan **m** valores nuevos pero ahora con una variación aleatoria. Los nuevos valores se evalúan en la función Objetivo. Se eligen los n valores que den los mejores resultados. Estos pasan a ser los n valores iniciales de la siguiente iteración. De esta manera la solución se mejora en cada iteración. Así como según la teoría de Darwin sobre las especie que dice que si un individuo no se adapta al ambiente muere y que en cada generación se mejora la especie, así el método mejora de generación en generación.

Este método tiene mucha semejanza con otro llamado Algoritmos Genéticos, este también utiliza reglas de la evolución. La diferencia entre Algoritmos Genéticos y el método Hill_Climbing, radica en que, en el primero, antes de iniciar su procedimiento, transforma los datos en arreglos binarios de ceros y unos y al finalizar el procedimiento los regresa a su estado original. En cambio en el método Hill-Climbing no sucede esto.

Los objetivos a cumplir dentro de este trabajo son:

1) Presentar cual es la situación actual de la Optimización no lineal dentro de la Optimización en general.

2) Elaborar un algoritmo sencillo que sea comprensible para cualquier persona que resuelva problemas no lineales.

3) El algoritmo elaborado debe resolver problemas de una o varias variables, restringidos y no restringidos.

4) Comparar el nuevo algoritmo con métodos que resuelvan problemas semejantes en tiempo, número de iteraciones y exactitud.

5) Aplicar el algoritmo a un problema real que ya haya sido resuelto anteriormente para comparar soluciones.

La secuencia del trabajo es la siguiente: en el capítulo 1 se presenta un panorama de lo que es la investigación de Operaciones y la Programación no Lineal, además se exponen algunos de los problemas que presenta esta última. En el capítulo 2 se detallan los antecedentes del método Hill-Climbing. Se describe también el algoritmo que se programó en computadora. En el capítulo 3 se muestran algunas pruebas experimentales que se realizaron para funciones de una y varias variables, restringidas y no restringidas. En el capítulo 4 se describen 3 métodos que comúnmente se utilizan para resolver problemas no lineales. En el capítulo 5 se presenta una comparación con los métodos descritos en el capítulo anterior y el Método Hill_Climbing. Para evaluar las técnicas, se comparan: número de iteraciones, tiempo de ejecución y resultados. Por último en el capítulo 6 se expone un problema real, el problema de asignación de personal de un hospital. Se muestra la deducción del modelo matemático no lineal elaborado por Prawda[1] y su solución comparándola con los resultados del Método Hill_Climbing.

[1] Prawda [25]

1

Optimización

Lo que pretende este capítulo es dar una visión general de todo lo que engloba la Investigación de Operaciones (IO)[1], ya que la Optimización se encuentra dentro de ella. Se hablará del origen de IO, como se formula un proyecto, como se enuncia un problema que se desea resolver, como se construye un modelo y que técnicas matemáticas mas comunes existen para la solución de este y las limitaciones que hay para usar estas técnicas. En particular se hablará de la Optimización No Lineal: que tipos de problemas no lineales pueden hallarse y los diferentes métodos existen para su resolución, entre ellos el Método Hill-Climbing.

1.1 Generalidades de la Optimización en Investigación de Operaciones

¿Qué es Optimización?

Optimización es buscar la mejor solución a un problema. Esta palabra comúnmente se asocia con **Investigación de Operaciones** (también llamada **Ciencia de la Administración**)."La investigación de operaciones aspira a determinar el mejor curso de acción (óptimo) de un problema de decisión con la restricción de recursos limitados" (Taha [30]). Puede verse que la definición es aplicable a problemas de organizaciones (industrias, hospitales, instituciones públicas, etc.).

IO es una herramienta que resuelve, de manera conveniente, la diversidad de conflictos a los que se enfrenta un ejecutivo en la actualidad. **IO** le da una base sólida de carácter cuantitativo que le ayuda a tomar decisiones. En ocasiones no es suficiente la experiencia y la intuición, a ello, debe aunarse el conocimiento científico (Espejo [8]).

La tarea de **IO** consiste en hallar la solución a problemas, previamente transformados en **modelos matemáticos**, por medio de **técnicas matemáticas**. Pero existen situaciones que encierran importantes factores intangibles y no es posible representarlos de forma directa en

[1] De aquí en adelante, en lugar de utilizar el término Investigación de Operaciones, se usará IO.

un modelo. El elemento humano es el principal. Hay casos en donde el efecto del comportamiento humano tiene tal influencia en el problema o sistema, que una solución obtenida a partir del modelo matemático es considerada impráctica.

Un ejemplo que ilustra lo anterior es el problema del ascensor. Los usuarios de un edificio grande de oficinas, se quejaban de la demora del elevador cuando lo llamaban. Empleando las **líneas de espera**, una técnica matemática, se descubrió que las protestas de los inquilinos no tenían razón de ser. Al profundizar en el sistema, del que formaba parte el problema, se encontró que las quejas eran un caso de fastidio. El tiempo de espera era en verdad reducido. La solución propuesta fue colocar espejos a todo lo largo de las paredes de las entradas del ascensor en cada piso. Cesaron las quejas porque empleados aguardaban al ascensor mirándose y viendo a otras personas (Taha [30]).

Por lo anterior, es claro que para resolver un problema **IO** debe contarse con ciertos conocimientos de diferentes disciplinas. Generalmente: matemáticas, probabilidad, economía, administración, computación, ingeniería y ciencias del comportamiento. Claro está, que una sola persona no puede tener todos estos conocimientos, por lo general se necesita de un grupo interdisciplinario (Hiller y Lieberman [13]).

IO puede verse como una ciencia y como un arte. Es una ciencia porque ofrece técnicas y algoritmos matemáticos adecuados para solucionar problemas. Y es un arte porque, para lograr un buen resultado; la obtención de los datos para la construcción del modelo, la validación de este y la implantación de la solución, dependerá de la creatividad y habilidad del personal encargado (Taha [30]).

Historia de IO

A partir de la II Guerra Mundial **IO** tuvo una gran impulso. Durante este periódo la armada inglesa y norteamericana les pidió a un grupo de científicos, la tarea de asignar de la mejor manera (la óptima) los recursos limitados del material de guerra en las diferentes operaciones militares. En pocas palabras, les solicitaron que hicieran investigación sobre operaciones militares; de ahí el nombre **IO**.

Viendo los éxitos que se obtuvieron, los científicos siguieron investigando, lo que dio por resultado el surgimiento de los principales métodos que componen **IO**. Algo que ayudo en mucho al desarrollo de **IO**, fue el avance de las computadoras digitales que realizan cálculos aritméticos muchísimo más rápido que los seres humanos. La figura 1.1 muestra una breve historia de **IO**.

Siglos XV-XVI los cimientos de IO fueron los estudios de: Lagrange, Leibnitz, Newton,Reimman, con el Cálculo Diferencial e Integral; Bernoulli, Poisson, Gauss, Bayes, con Probabilidad y Estadística.
1759 Quesnay (economista) comienza a utiilizar modelos primitivos de Programación Matemática.
1873 Jordan usa Modelos Lineales.
1874 Walras (economista) emplea modelos similares a los de Quesnay.
1896 Minkowsky y 1903 Farkas aplicaron Modelos Lineales.
1890-1900 (aprox.) Markov investiga sobre los Modelos Dinámicos Probabilísticos.
1900-1910 (aprox.) Erlang utiliza Modelos de Líneas de Espera.
1920-1930 (aprox.) Aparecen los Modelos de Inventarios.
1920-1940 (aprox.) Köinig y Egervary (húngaros) estudian Problemas de Asignación.
1937 Von Neuman asienta los fundamentos de lo que después sería la Teoría de Juegos y con Morgenstern estudia la Teoría de Preferencias.
1939 Kantorovich (ruso) investiga Problemas de Distribución.
1939 Durante la II Guerra Mundial aparece ya de manera formal IO.
1947 El Dr. George Dantzing inventa el Método Simplex, lo que da inicio a la Programación Lineal.
1950-1960 (aprox.) En esta década numerosos científicos hacen diferentes investigaciones como:
Bellman en Programación Dinámica,
Kuhn y Tucker en Programación No Lineal,
Gomory en Programación Entera,
Ford y Fulkerson en Redes de Optimización,
Markowitzen en Simulación,
Arrow, Karlin, Scarf, Whitin en Inventarios,
Raiffa en Análisis de Decisiones.
Howard en Procesos Markovianos de Decisión, etc.

Prawda [25]

Figura 1.1 Breve historia de la Investigación de Operaciones.

Fases de un Proyecto IO

Las fases por las que atraviesa un proyecto de IO son las siguientes (Taha [30]) (Figura 1.2):

1. **Formulación del problema.** Hay que precisar el objetivo del proyecto y determinar las alternativas de decisión.

2. **Construcción del modelo.** Este deberá ser el más adecuado para expresar el sistema. Contendrá términos cuantitativos del objetivo y las restricciones en función de sus variables de decisión.

3. **Solución del modelo**. Aquí se elegirá la técnica matemática apropiada para la resolución.

4. **Prueba y control del modelo.** La manera en que se verifica la validez del modelo, es comparando los resultados con datos anteriores del sistema actual.

5. **Implantación de los resultados finales.** Estos son traducidos en instrucciones de acción para el personal encargado del sistema.

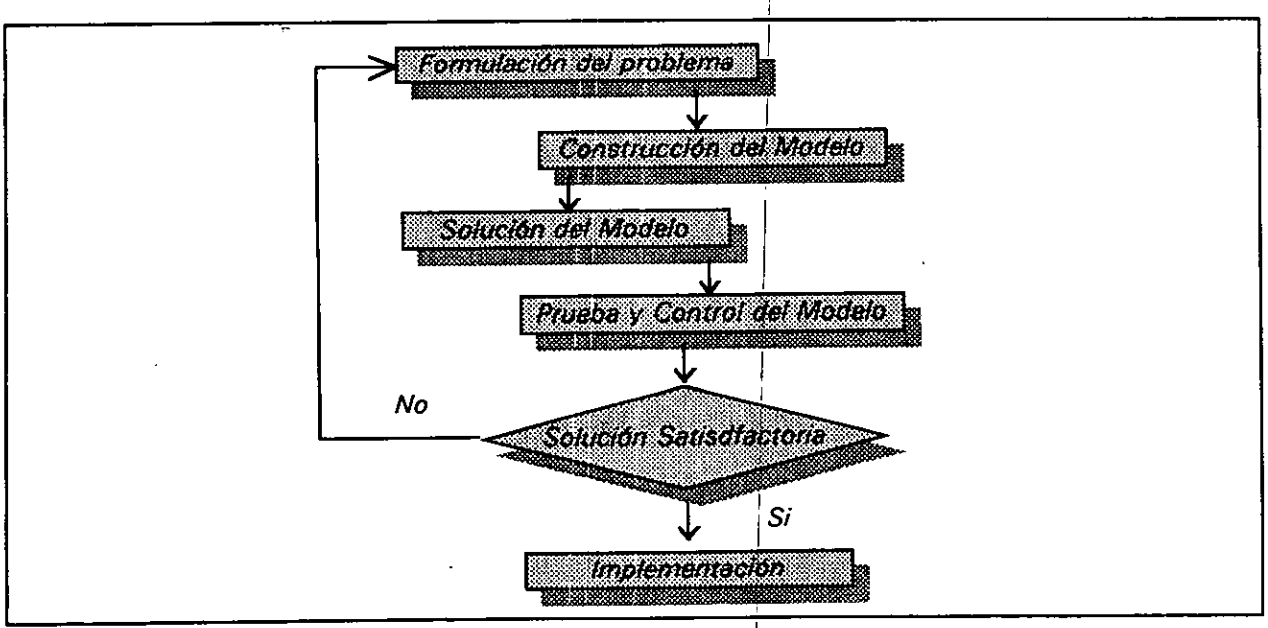


Figura 1.2 Diagrama de las fases de un proyecto de IO (Espejo[8]).

Las siguientes secciones explican de modo más detalladas las primeras tres etapas.

Formulación del Problema

Antes de expresar formalmente un problema debe fijarse los límites perfectamente. Para hacerlo, hay que responder, por lo menos, las siguientes preguntas (Prawda [25]):

a) ¿Cuál es el problema existente?

b) ¿De quién o quienes es el problema?

c) ¿Qué características tiene el sistema donde se encuentra inmerso el problema?

d) ¿Quiénes o quién toman las decisiones?

e) ¿Cuáles son los objetivos del sistema dentro del cuál está el problema?

f) ¿Cuáles son las componentes controlables y no controlables del sistema?

g) ¿Cuáles son las interacciones más importantes del sistema?

h) ¿Cómo se emplearán los resultados del proyecto **IO**? ¿Quién los ocupará? ¿Qué consecuencias tendrá?

i) ¿En cuánto tiempo se espera tengan efecto los resultados?

j) ¿Podrán las consecuencias de las soluciones, corregirse de manera simple?

k) ¿Cuántos elementos del sistema serán afectados por las soluciones conseguidas del proyecto y en qué medida?

Principalmente, dentro de las respuestas obtenidas es preciso identificar:

a) Las componentes controlables y no controlables de un sistema.

b) Las rutas factibles de acción.

c) El marco de referencia, dado por las componentes no controlables.

d) Los objetivos que se buscan y cuál es su orden jerárquico.

e) Las interacciones esenciales y las restricciones que existen.

Una vez identificados los aspectos anteriores, se enuncian en forma matemática.

Tipos de Modelos

Hasta aquí se ha mencionado el término modelo, pero no se ha definido. En esta sección, se expondrá: su significado, tipos de modelos que existen, porque algunos modelos son considerados de decisión y sus características de estos.

Un **modelo** es una representación del mundo real. Un sistema, cuyo problema se desea resolver, puede expresarse en tres tipos de modelos, principalmente: icónicos, analógicos y simbólicos.

Los **modelos icónicos** son imágenes a escala del sistema. Por ejemplo: las maquetas, dibujos, fotografías, etc.

Modelos analógicos son aquellos, en que para imitar de las propiedades de un sistema, utilizan otro de rasgos similares.

Letras, números, variables y ecuaciones, son empleados en los **modelos simbólicos** para enunciar ideas abstractas de un sistema. Estos son los más económicos, más sencillos de construir y usar; además son empleados en IO.

Los modelos se utilizan porque es imposible experimentar con la realidad. Entre más parecido sea un modelo al sistema existente, más complejo será. Para la elaboración de este, es importante separar los aspectos fundamentales de los superfluos. Los sistemas complicados frecuentemente son fraccionados en submodelos, que unidos, conforman al modelo del sistema original.

El sistema real supuesto es una concepción de la situación que existe. La forma de conseguir esto, es identificando los factores dominantes que controlan el comportamiento del sistema real. En un modelo se reconoce las relaciones primordiales del sistema en la forma de una sola función (función objetivo) y un conjunto de restricciones. Como los modelos expresan problemas de decisión, cabe decir, que son modelos de decisión. Estos esencialmente deben contener tres elementos (Taha [30]):

1. Alternativas de decisión.
2. Restricciones.
3. Criterios de evaluación.

Formulación de un Modelo

Para explicar como pude construirse un modelo, se presenta el siguiente ejemplo.

Una compañía dulcera fabrica los caramelos A y B. El precio de venta por kilo de A y B es $ 8.00 y $ 6.00 respectivamente. Producir un kilo del caramelo A cuesta a la empresa $ 0.40, mientras que el costo de producir B es de $ 0.60. La empresa asigna para la producción, $ 2,500.00 a la semana.

Se necesita, para la fabricación de un kilo de A, 570 gr. de azúcar y 430 gr. de chocolate. En cambio, para la elaboración de un kilo de B, hay que utilizar 740 gr. de azúcar y 260 gr. de chocolate. Los proveedores entregan a la compañía, semanalmente, 1,200 kg de azúcar y 1,900 kg de chocolate. Pero en el almacén solo hay capacidad para un total de 3,000 kg.

Con todo lo anterior, ¿cuál es la producción semanal de los dos tipos de caramelos que optimicen las ventas sin rebasar las limitaciones existentes?

Para diseñar el modelo, primero deben identificarse las variables decisión. Estas son:
X_1: kilos de caramelo A que se fabricará en una semana.
X_2: kilos de caramelo B que se fabricará en una semana.

Como lo que se desea es encontrar el máximo valor de ingresos por semana, la función objetivo resulta:

$$Max\ Z = 8\ X_1 + 6\ X_2 \qquad (1)$$

El costo de producir A y B es $ 0.40 y $ 0.60 respectivamente, pero se destina $ 2,500.00 para ello. Por lo que los costos a la semana para producir los caramelos no deben sobrepasar la inversión. Esta restricción queda:

$$0.40\ X_1 + 0.60\ X_2 \leq 2500$$

La fábrica se abastece semanalmente con 1,200 kg de azúcar. El dulce A requiere 570 gr. y b 740 gr. La restricción que se obtiene es:

$$0.57\ X_1 + 0.74\ X_2 \leq 1200$$

A y B necesitan para su elaboración 430 gr. y 260 gr. de chocolate por kilo, sin embargo reciben 1,900 kg semanalmente. Esta restricción se expresa así:

$$0.43\ X_1 + 0.26\ X_2 \leq 1900$$

La bodega solo tiene espacio para 3000 kg. Por lo que la última restricción es:

$$X_1 + X_2 \leq 3000$$

El modelo planteado queda de la siguiente manera:

$$Max\ Z = 8\ X_1 + 6\ X_2$$

sujeto a:

$$0.40\ X_1 + 0.60\ X_2 \leq 2{,}500$$

$$0.57\ X_1 + 0.74\ X_2 \leq 1{,}200$$

$$0.43\ X_1 + 0.26\ X_2 \leq 1{,}900$$

$$X_1 + X_2 \leq 3{,}000$$

$$X_1 \geq 0,\ X_2 \geq 0$$

La última restricción es definida como condición de no negatividad. Es importante que sea incluido este requisito por que en la realidad los valores negativos pierden sentido.

Este es un ejemplo simple, por que un problema real puede llegar a tener hasta 2,000 restricciones.

Solución del Modelo

Resolver un modelo, consiste en hallar los valores de las variables dependientes relacionadas a las componentes controlables del sistema. Esto se realiza con el fin de optimizar o por lo menos mejorar la eficiencia del sistema, dentro un marco de referencia previamente fijado. Entre las diferentes técnicas matemáticas que conforman IO, debe escogerse la más adecuada a las características del modelo planteado para encontrar su solución.

Técnicas de IO

La forma clásica de presentación de problemas de optimización es:

$$Optimzar\ Z = C_1X_1 + C_2\ X_2 + \ ... \ + C_nXn$$

sujeta a las restricciones:

$$a_{11}X_1 + a_{12}X_2 + \ ... \ + a_{1n}X_n \leq b_1$$

$$a_{12}X_1 + a_{22}X_2 + \ ... \ + a_{2n}X_n \leq b_2$$

.

.

.

$$a_{m1}X_1 + a_{m2}X_2 + \ ... \ + a_{mn}X_n \leq b_n$$

$$X_1 \geq 0,\ X_2 \geq 0,\ ... \ ,X_n \geq 0$$

Aquí, el término optimizar significa encontrar ya sea el máximo o el mínimo valor de la función Z, según sea el caso. Las restricciones pueden ser igualdades o desigualdades. Estas son tantas como se tengan o requieran.

Si tanto la función objetivo como las restricciones son lineales, es conveniente resolver el modelo por **Programación Lineal** (u Optimización Lineal). A veces, en un planteamiento, hay la limitante de que algunas variables deben tomar sólo números enteros. En este caso, se hallará la solución del modelo por **Programación Entera**. Si la función objetivo y/o alguna de las restricciones son no lineales, entonces, se tratará de encontrar la solución por **Optimización No Lineal**.

Cuando exista un problema demasiado complicado, y es posible descomponerlo en una serie de etapas, se recomienda utilizar **Programación Dinámica**. La solución secuencial de cada etapa es equivalente a la solución del problema de decisión del sistema original. Esto es según el **Principio de Optimalidad** (Bellman [3]).

Un modelo puede representar situaciones competitivas en las cuales cada oponente (dos o más) trata de ganar a expensas del otro. En casos como este se aplica **Teoría de Juegos**.

Si en una situación se quiere obtener: la ruta más corta, la ruta entre dos puntos cualesquiera de manera que se minimice la longitud total de estas conexiones, el flujo máximo de una red, o se desea planear y controlar proyectos. Entonces, es un problema de redes y lo adecuado es atacarlo por medio de **Teoría de Redes.** Esta, comprende, entre otras, las técnicas: **PERT** (Técnica de Evaluación y Revisión de Proyectos) y **CPM** (Método de la Ruta Crítica).

Otra manera de resolver un problema, es emplear la **simulación.** Este método se define como un experimento estadístico realizado en una computadora para dar una idea del funcionamiento del sistema.

Todas las técnicas anteriormente descritas, sólo son algunas de las que comúnmente se utilizan en IO. Se subraya, que para elegir una técnica que sea acorde al problema, hay que tomar en cuenta las particularidades de este.

Una encuesta (Weston) reflejó la importancia de las técnicas de **IO** en la planeación empresarial. La averiguación fue entre las 1000 compañías más grandes, según la revista Fortune, de Estados Unidos. Los resultados se muestran en la Figura 1.3

Tópico	Procentaje
Estudios de simulación	29
Programación lineal	21
Análisis de redes (incluyendo PERT)	14
Teroría de inventario	12
Programación no lineal	8
Programación dinámica	4
Programación entera	3
Teoría de colas	3
Otras	6
	100

Figura 1.3 Técnicas de IO que se utilizan con más frecuencias en la planeación empresarial.

Procedimiento de los Métodos de IO

La mayoría de los métodos de IO, buscan un vector inicial X_0, que resuelva la función en la primera iteración. Para la segunda, se produce otro vector mejorado X_1. El proceso se repite hasta hallar el vector X^*, que es la mejor solución del problema. De esta manera, se

encuentra una serie de puntos X_0, X_1,...,X^*. Pero en el caso de la programación no lineal, esta sucesión de puntos por lo común no consigue el punto óptimo, sino que más bien converge hacia él. El proceso de los algoritmos finaliza cuando se logra un punto cercano al punto solución[2] (Luengerg [22]).

1.2 Aspectos Importantes de la Optimización No Lineal

De la figura mencionada anteriormente, se parte para afirmar que la Programación No Lineal es de las técnicas menos usadas. Pero, en la realidad, los modelos que se crean pocas veces son lineales. A veces, cuando los investigadores se topan con un modelo que no es lineal: lo "linealizan", o idean otro modelo. Esto se debe a que, en general, las personas tienen cierto rechazo a los modelos no lineales[3], porque los métodos comúnmente usados para resolver este tipo de problemas son complicados.

El modelo general no lineal se plantea de acuerdo a la forma clásica de problemas IO, visto anteriormente, y debe contar con que, por lo menos, la función objetivo es no lineal.

Tipos de Problemas No Lineales

Los problemas pueden dividirse en (Prawda [25]):

1) Restringidos: cuando en estos problemas hay limitaciones.
2) No restringidos: si sólo existe la función objetivo.
3) Continuos: son aquellos que tienen todas sus funciones y variables continuas.
4) Discretos: cuando alguna de las variables y/o funciones es discreta.
5) Diferenciables: todas las funciones del problema son doblemente diferenciables.
6) Con una o varias variables independientes.

[2] Se deduce que se esta convergiendo al punto solución cuando la diferencia del punto encontrado y el de la iteración anterior es muy pequeña.

[3] Una función o modelo no lineal se puede identificar como tal, si existen términos elevados a una potencia diferente de 1, y/o si hay multiplicación entre dos variables.

Un Ejemplo

Aquí se muestra un ejemplo de Programación No Lineal restringido, continuo y diferenciable (Figura 1.4).

$$Min\ Z = X_1^2 + X_2^2 + 2X_2$$

sujeto a:

$$X_1 + 2X_2 - \frac{1}{2} = 0$$

$$X_1^2 + X_2^2 - 1 \leq 0$$

$$X_1 \geq 0,\ X_2 \geq 0$$

La línea sombreada es la región de factibilidad. Todos los puntos que quedan delimitados por las restricciones componen la región factible. La función objetivo esta representada por el círculo de línea no continua para diferentes valores de X_1 y X_2. La mejor solución a este problema, es el primer punto de la recta sombreada que toca al círculo de la función objetivo ($g_1(x)...g_4(x)$, representan cada una de las restricciones).

Si se deseara encontrar el máximo valor de la función objetivo, con las mismas restricciones, la solución óptima sería el punto más alto de la región factible.

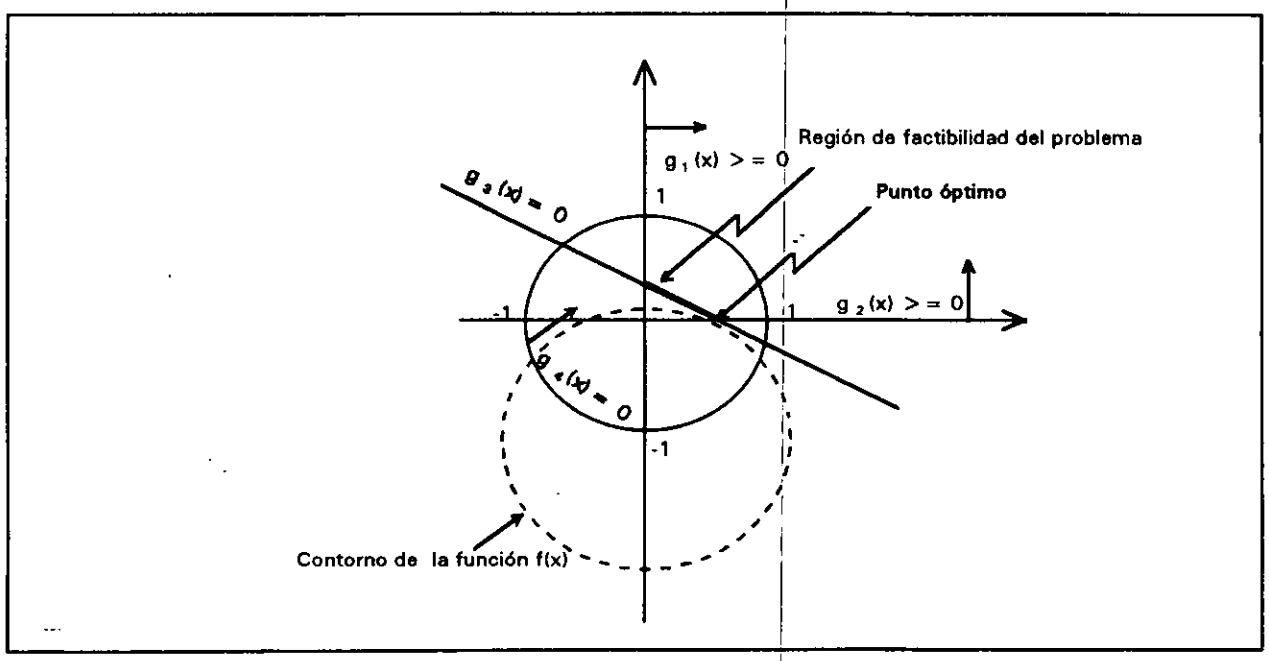


Figura 1.4 Un ejemplo de Programación no Lineal (Prawda [25]).

Algunas Dificultades en los Problemas No Lineales

Tres características principales que dificultan la resolución de ciertos problemas no lineales son:

1) El mejor valor de la función objetivo, no siempre está en los puntos extremos de la región factible (Figura 1.5) como en la Programación Lineal.

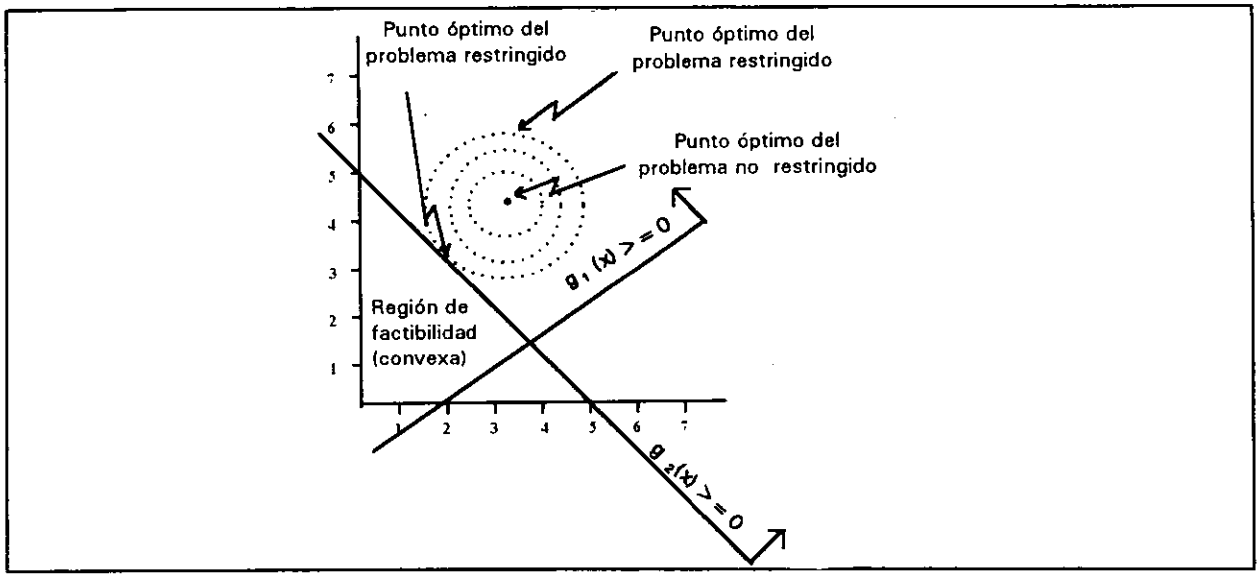


Figura 1.5 En ocasiones el óptimo no se encuentra en puntos extremos de la región factible.

2) Generalmente, los actuales métodos encuentran un punto óptimo local y no el punto óptimo global. Una función en ocasiones, tiene muchos puntos máximos y mínimos, como en la Figura 1.6. En esta circunstancia hallan el mejor valor más cercano al punto inicial de partida.

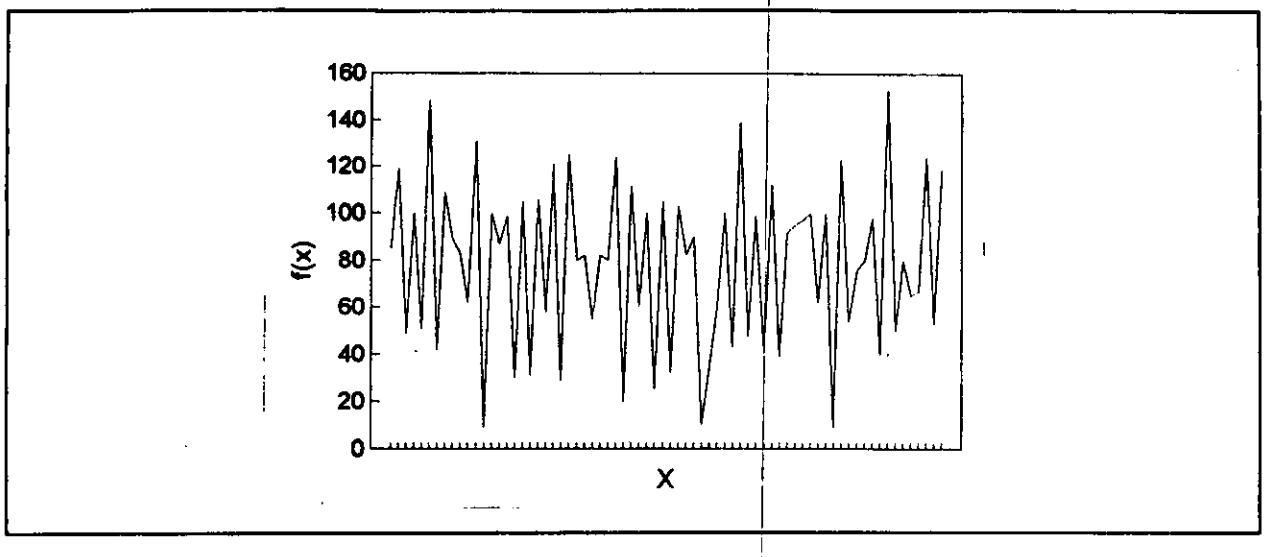
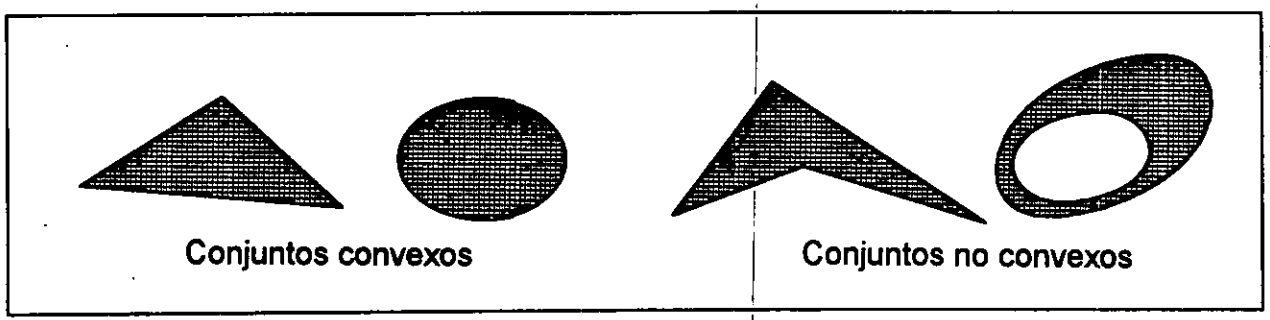


Figura 1.6 No siempre las funciones son unimodales (Golberg [10]).

3) Una condición de determinados algoritmos que resuelven problemas no lineales, es que la región factible sea convexa[4] pero no siempres es así. En la Figura 1.7 aparecen ejemplos de regiones convexas y no convexas.



1.7 Ejemplos de regiones convexas y no convexas.

[4] Una región convexa, es tal, que dos puntos cualesquiera pueden ser representados por una combinación *lineal* y tanto los puntos como la recta deben estar dentro de la región factible.

Métodos de Optimización No Lineal

Existen diferentes técnicas para distintos tipos de funciones en la programación no lineal. Entre algunos de los más usados se destacan los siguientes (Prawda [25]).

a) **Fibonacci, Sección Dorada, Interpolación Cúbica, Interpolación Cuadrada, Newton Raphson**, para funciones con una variable (univariable).

b) Métodos de Gradiente, como el **Método de Davidon-Fletcher-Powell, Métodos de Direcciones Conjugadas, Método de Variable Métrica**. Estos métodos son para funciones derivables de varias variables (multivariables).

c) **Método de Powell**, algoritmo de búsqueda para funciones multivariables no necesariamente diferenciables.

Para problemas con restricciones están:

a) **Programación Separable, Métodos de Aproximación Lineal** que emplean multiplicadores de Lagrange.

b) **Métodos Penales,**

c) **Métodos de Tolerancia Flexible.**

El cálculo diferencial es la base para la mayoría de las técnicas mencionadas arriba. Estas utilizan gradientes y Hessiannos, para lo cual es necesario que las funciones del problema sean continuas y diferenciables. Evaluar analíticamente las operaciones del gradiente y del Hessiano en cada iteración puede convertirse en una tarea muy tediosa. Por medio de métodos numéricos es posible calcular derivadas, pero el error de redondeo que se introduce, a veces produce resultados incorrectos (Prawda [25]).

Otros métodos, menos comunes, que no necesitan derivadas, por tanto evitan el problema antes mencionado, son los **algoritmos de búsqueda aleatoria**. Uno de los más populares es el **Método de Montecarlo**. Este es un método numérico que resuelve problemas matemáticos mediante la simulación de variables aleatorias (Sobol [28]). El algoritmo se emplea a casi cualquier tipo de problema no lineal.

La segunda técnica que aquí se mencionará, y es la propuesta del presente trabajo, es el **Método Hill-Climbing**. Es un método que está inspirado en la evolución biológica. Es una alternativa para la solución de problemas no lineales. Este algoritmo difiere de los métodos tradicionales en tres aspectos:

1. Busca a partir de un grupo de puntos, no de un único punto.

2. No emplea derivadas, o herramientas parecidas de cálculo diferencial.
3. Usa reglas probabilísticas de transición, no determinísticas.

En el siguiente capítulo se detallará más sobre su origen y algoritmo. Las funciones en las que puede utilizarse este método son: de una y varias variables, restringidas y no restringidas, continuas y discontinuas.

2

El Método Hill-Climbing

Muchas veces al relacionar ideas muy diferentes se llega a encontrar soluciones originales e incluso sencillas a problemas que de otra forma no hebrían surgido. En este capítulo se explica como la teoría de la Evolución Biológica influye de una manera importante en la creación del Método Hill-Climbing el cual resuelve problemas de la Programación No Lineal.

Si se enfoca a como es la vida en la tierra actualmente, se verá que esta es el resultado de una serie de modificaciones en los individuos, en donde los que han podido adaptarse al ambiente son los que sobreviven. Esta misma idea es la que toma Rechenberg y realiza algunos experimentos en donde teniendo en cuenta las reglas de la evolución halla la solución a problemas físicos. Rechenberg extrae las reglas mas importantes de la Evolución Biológica y las esquematiza con símbolos con los que crea algoritmos.

El Método Hill-Climbing es un algoritmo que recoge las leyes de la evolución biológica y se aplica a funciones no lineales. Más adelante se expondrá paso a paso el algoritmo del método. Además también se explica la convergencia del método y algunas observaciones importantes para la utilización del mismo. Por último se destacan las diferencias principales entre el Método Hill-Climbing y otro método que tiene un origen muy parecido, Algoritmos Genéticos.

2.1 Antecedentes del Método Hill-Climbing

Introducción

La vida en la tierra, tuvo que tener una serie de transformaciones progresivas para llegar a ser como actualmente se conoce. Es decir, lo que se llama evolución biológica. Está inició hace cerca de 4 mil millones de años en un primitivo océano con compuestos orgánicos, tales como aminoácidos, carbohidratos,etc. (Lazcano-Araujo [20]). De aquí surgieron los primeros organismos vivos. A través del tiempo, la estructura de estos fue siendo cada vez más compleja hasta que aparecieron plantas y animales acúaticos. El habitat de algunos de ellos dejo de ser el agua para instalarse en la tierra. Finalmente con el paso del tiempo

surgieron las diferentes especies de fauna y flora que actualmente se conoce; pero el proceso continúa.

Los seres más desarrollados procedieron de los más simples. A partir de un número reducido de organismos se crearon una gran variedad de especies. Esto sucedió, porque las constituciones físicas de los seres, a través del tiempo, fueron cada vez más complejas. (Casado [5]).

Esto puede ser el reflejo de la eficacia de una estrategia de la evolución biológica. Para ejemplificar esto, imagínese un grupo de organismos con diferencias en su naturaleza, respecto a los demás de la población. Si esos cambios ayudan a este conjunto a adaptarse a su ambiente particular, ellos tendrán una mayor oportunidad de sobrevivir. Por esta razón hay más probabilidad de que se propagaguen en un futuro, que los demás individuos de la población. Esto es conocido como la supervivencia del más apto. Si estas variaciones se heredan a la generación subsecuente, puede llegar el momento, en que una especie sea completamente diferente a la original. Todo lo anterior da una noción sobre el modo de operar de la evolución a través de muchos años.

Imitando a la Evolución

En 1964, tomando la idea de la estrategia de la evolución biológica, Ingo Rechenberg realizó un primer experimento (Rechenberg [26]). En un laboratorio de mecánica de fluidos, frente a un túnel de viento, situó una plancha de aluminio articulada (Figura 2.1). Esta tenía cinco bisagra y cada una con un número determinado de posiciones. Por lo que, la plancha podía colocarse hasta en 345,025,251 maneras diferentes. El objetivo era encontrar la forma con un mínimo de resistencia al aire. De antemano se sabe, que la solución es cuando la plancha esta totalmente en paralelo con la corriente. Pero supóngase que esto se ignora. La postura al comenzar el experimento era al azar. Para producir mutaciones o cambios en las cinco bisagras se uso para este primer experimento, un aparato mecánico (Figura 2.2). Cinco bolas representaban las cinco bisagras de la plancha. Estas pasaban a través de una pirámide de alfileres y llegaban hasta unas cajas. Cada una de ellas indicaba la variación del ángulo de cada bisagra.

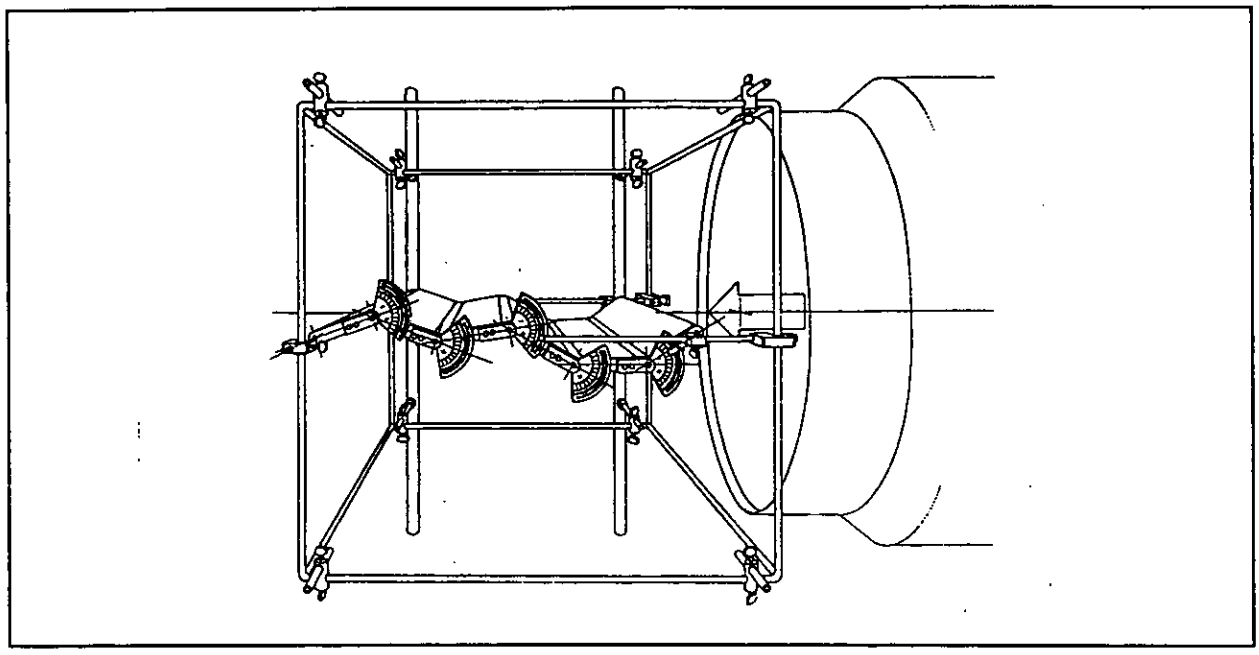


Figura 2.1. Primer experimento de Rechenberg. La plancha de aluminio frente a un túnel de viento.

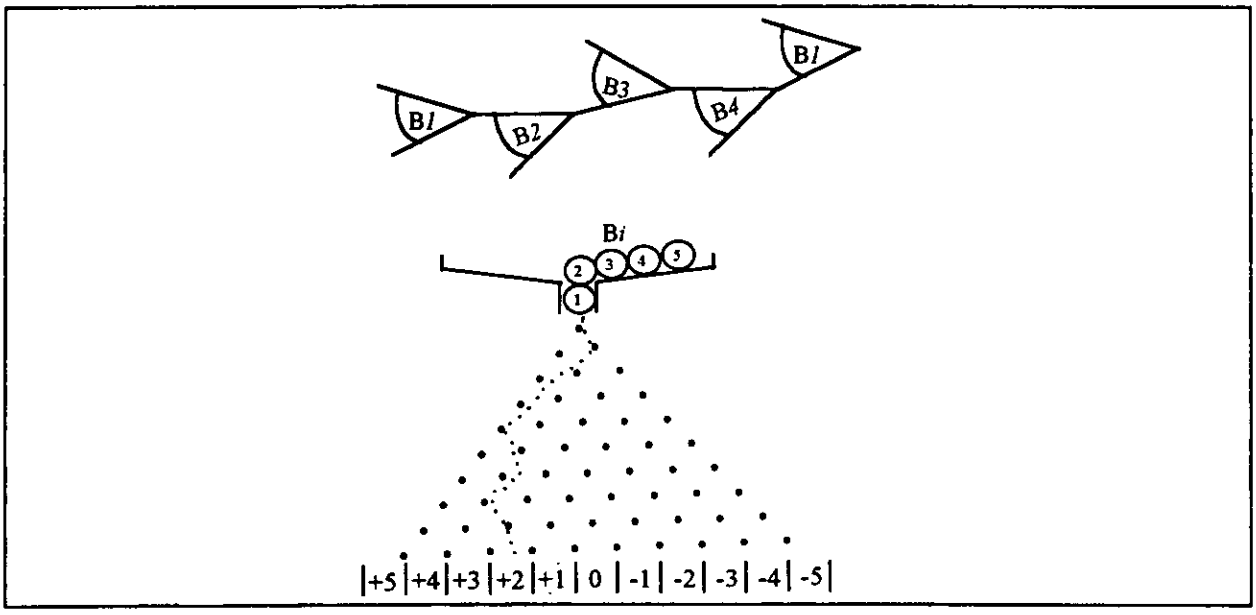


Figura 2.2. Aparato mecánico que determinaba las variaciones de cada bisagra de la plancha de aluminio.

Para iniciar el experimento del túnel, la plancha fue plegada en una posición zig zag, es decir, con un alto grado de resistencia. Para cada posición de la plancha se midió su oposición al viento. La idea principal era rechazar todos los estados de la plancha que incrementaran la resistenc-ia. Pero si una forma dada tenía baja resistencia, entonces, esta pasaba a ser la forma inicial del siguiente ensayo. La Figura 2.3 muestra el resultado de todo el experimento. Cada silueta representa la forma de la plancha después de cada 10 pruebas. Después de 300, se aprecia que, la postura de la plancha se ha acercado mucho al estado ideal. La oposición al aire se disminuyó casi por completo.

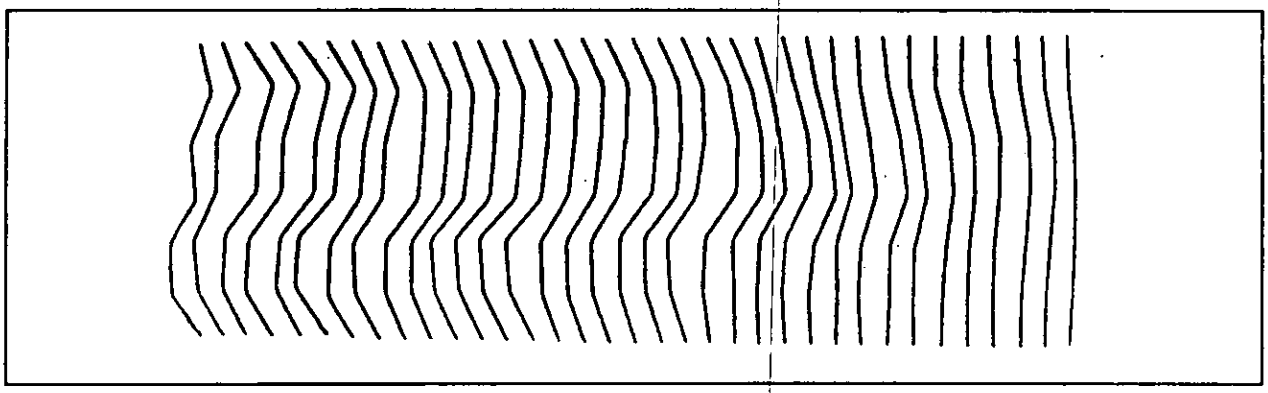


Figura 2.3. Forma de la plancha a través de 300 generaciones.

Comparando este experimento con la evolución biológica, cuando la forma de la plancha se modifica (o se muta), la forma que resulta es la descendencia. Por lo que cada ensayo es una nueva generación, para este caso, de un solo individuo. En cada generación existe un antecesor y un descendiente. La forma de la plancha es la característica que se hereda de una generación a otra o fenotipo en biología[1]. Por último, la resistencia al aire vendría siendo la aptitud de cada individuo para adaptarse al medio. En la Figura 2.4 aparece un diagrama de flujo del proceso del experimento de Rechenberg.

[1] Fenotipo es el conjunto de características hereditarias que constituyen una especie.

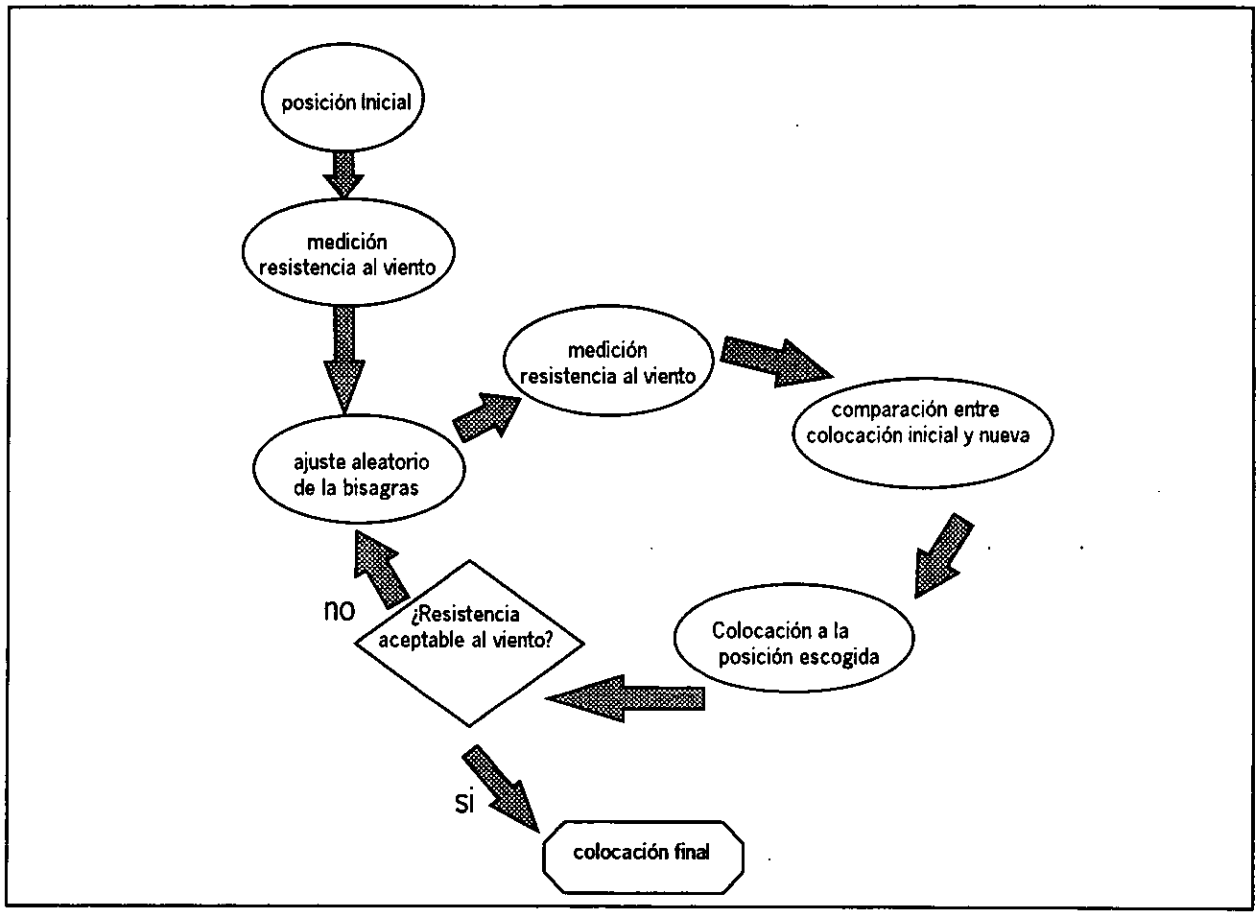


Figura 2.4. Diagrama que representa el proceso del experimento de Recheberg.

En su segundo experimento, Rechenberg deseaba encontrar la forma que debía tener una manguera con la mínima oposición a la corriente de un fluido (Rechenberg [26]). Para lograr esto, fueron colocadas dos mangueras flexibles de plástico en forma de un cuarto de círculo (Figura 2.5). Las mangueras se sujetaron con 6 barras ajustables en la zona de la curvatura. La posición de las barras son las variables a optimizar (en el caso del experimento anterior, las bisagras eran las variables).

Durante el experimento, mientras un tubo fue continuamente variado de acuerdo al procedimiento de la estrategia de la evolución, expuesto arriba. El segundo tubo permaneció *sin cambios* como una referencia del sistema. La Figura 2.6 muestra la forma inicial (a) y la forma óptima encontrada (b).

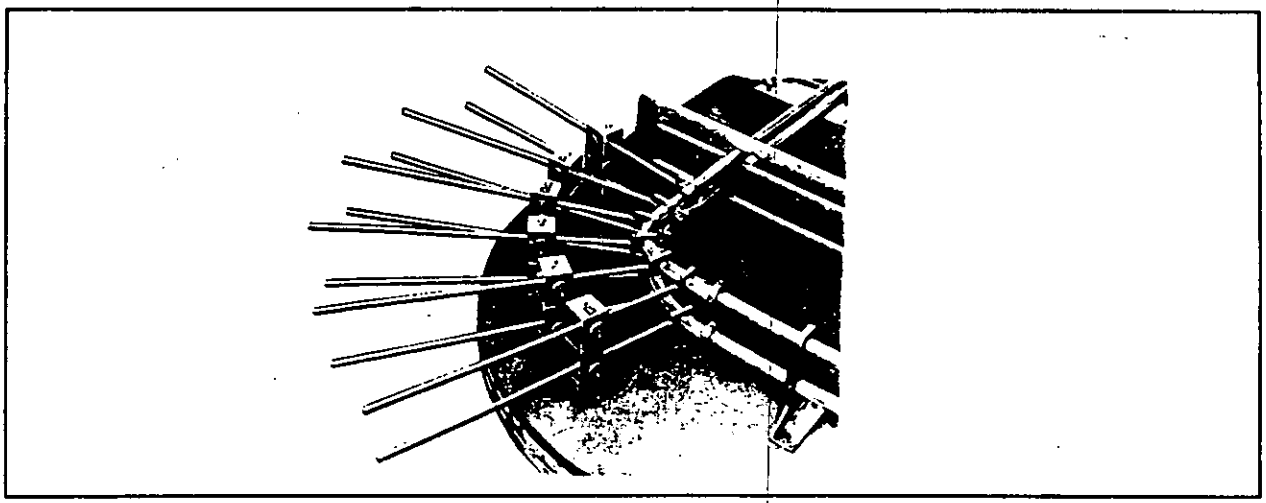


Figura 2.5. Experimento para encontrar la menor resistencia al flujo de corriente.

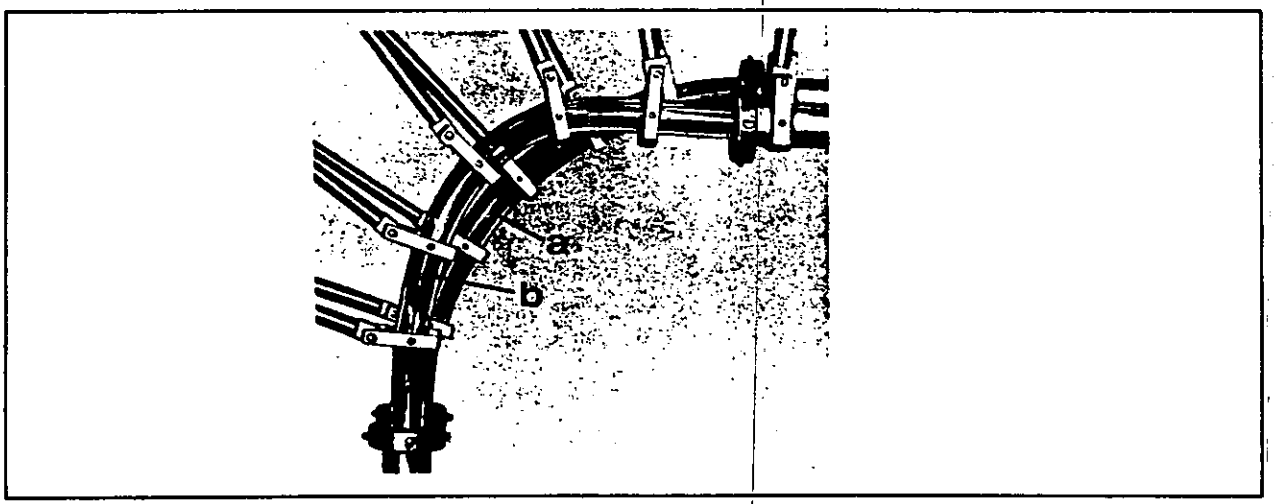


Figura 2.6. (a) representa la forma inicial. (b) representa la óptima encontrada.

Secuencia de Operaciones-EE[2]

Como se vió, la técnica que empleó Rechenberg imita la estrategia de la evolución bilógica. En base a esto, Rechenberg, destacó las característica esenciales de la evolución y las representó por medio de símbolos con cartas, para un mejor manejo gráfico (Figura 2.7). Más adelante se verán ejemplos con estos símbolos. La descripción de cada un de ellos es la siguiente:

1. **Variable**: Una carta contiene la información de las características de un solo individuo[3].
2. **Duplicación**: La doble flecha indica que la información de la carta es transferida a otra.
3. **Mutación**: La flecha quebrada denota que hay un cambio aleatorio (mutación) de la información de la carta.
4. **Población**: Una hilera de cartas representa la información de un conjunto de individuos.
5. **Selección**: La flecha de 2 direcciones con una Q abajo de una fila de cartas significa que, estas se dividirán en las mejores y las peores.
6. **Rendimiento**: La información se prueba con el problema a analizar (aptitud).
7. **Elección Aleatoria**: La flecha con un símbolo w al final quiere decir que una carta es elegida al azar.
8. **Recombinación**: Dos o más datos de la información de una carta son intercambiados por la misma cantidad de datos de otra carta.
9. **Evaluación**: La Q y la flecha, indican que se califica la información, de acuerdo al rendimiento que se tenga con respecto al problema real.

[2] Estrategia de la Evolución

[3] Haciendo una semejanza con biología, esta información sería el genotipo. Esto es, el conjunto de factores que se heredan y constituyen la naturaleza de un individuo.

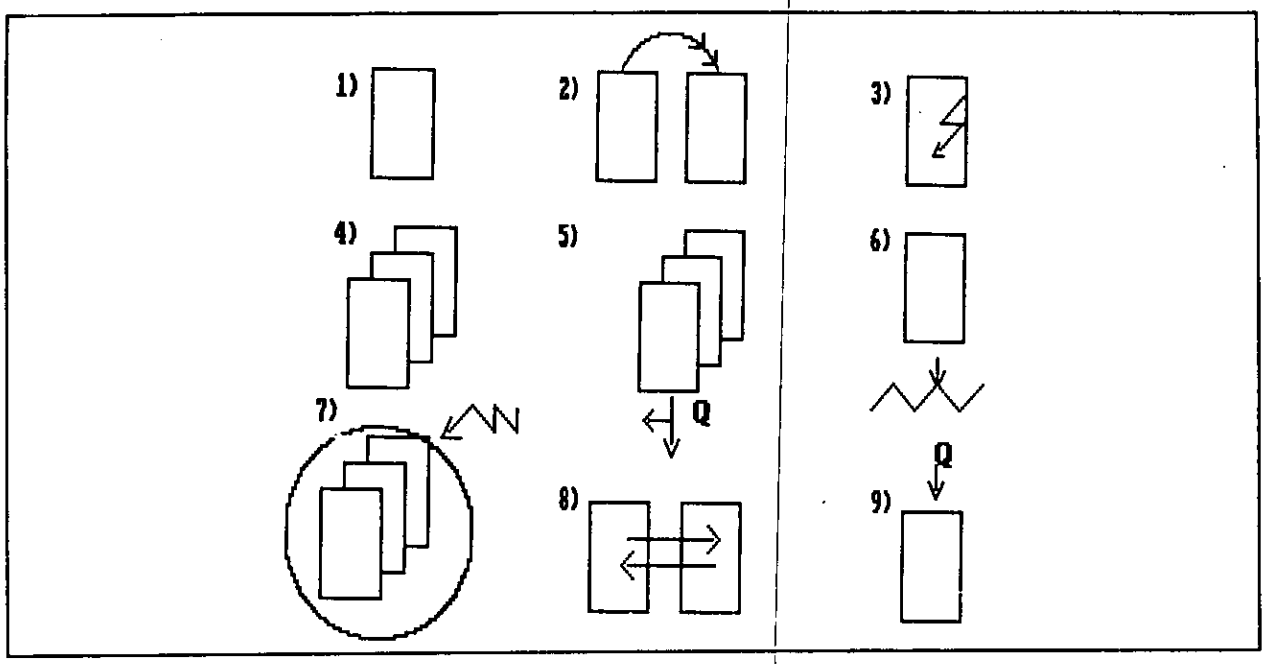


Figura 2.7. Representación gráfica de las reglas de la Estrategia de la Evolución.

Rechenberg utiliza para las operaciones de la estrategia de la evolución una nomenclatura especial (Rechenberg [26]). De esta manera resume el algoritmo a seguir.

$$(\mu/\rho+\lambda)\text{-}EE \quad o \quad (\mu/\rho,\lambda)\text{-}EE$$

El símbolo "μ" denota el número de padres de la siguiente generación. El término "λ" indica el número de descendientes. La letra griega "ρ" se refiere al número de padres que transmitirán su información a cada nuevo individuo. El símbolo "/" indica que habrá recombinación de la información entre los padres. Si tres padres combinan sus datos, el número de mezcla es 3 (esto es imposible en biología, pero si lo es en la teoría de la estrategia de la evolución).

El símbolo " + " significa que entre los padres y los hijos se escogerán a los mejores. Con el símbolo "," únicamente de entre los hijos se escogerá a los mejores. La cantidad de algoritmos que pueden crearse utilizando la estrategia de la evolución, es muy grande.

Para mostrar como trabaja un algoritmo de estos, utilizando los símbolos y la nomenclatura antes descritos. Se presentan varios ejemplos.

Ejemplo 1. Si se tiene (1 + 1)-EE (Figura 2.8), de acuerdo a lo anterior hay un padre y un hijo en cada generación. Entre ellos dos habrá que elegirse al que de una solución más adecuada. Explicándolo paso a paso, esto es: Se parte de datos iniciales y lo que se desea es perfeccionarlos de tal manera que sean la mejor solución al problema real. La información esta representada por una carta encerrada en un círculo. Como se recordará la doble flecha advierte que la información se duplicará en otra carta. La flecha quebrada avisa que en esta nueva carta los datos sufrirán una mutación. Después,la información ya transformada se aplica a la situación real y se califica. Luego, con los datos originales y los recientes (como lo indica el signo +), se aparta al óptimo. Este pasará a ser el padre de la siguiente generación o iteración. El procedimiento repite, hasta encontrar una solución que se considere muy buena para el problema. Este es el algoritmo que utilizó Rechenberg para sus primeros experimentos.

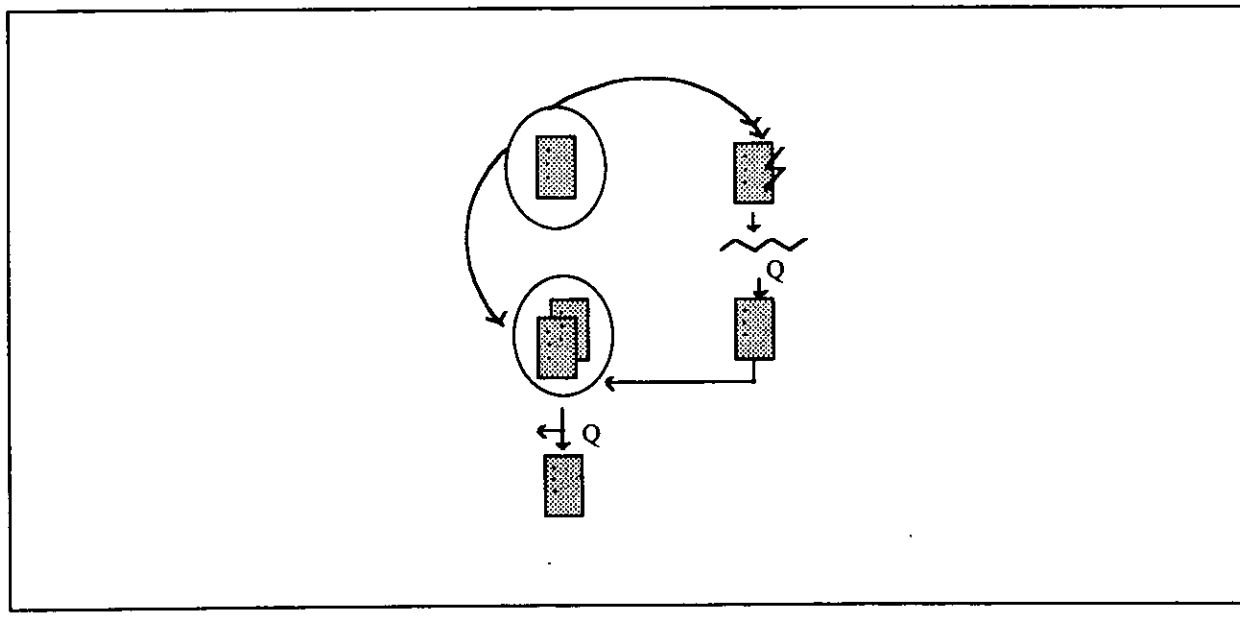


Figura 2.8. Ejemplo 1. Estrategia de la Evolución.

Ejemplo 2. (1 + 6)-ES (Figura 2.9), esto significa que: Se tiene una información o carta inicial la cual se duplica en seis descendientes. En cada una de las cartas se efectúa una mutación. Estas se emplean en la situación real y se califican de acuerdo a su rendimiento en el problema a analizar. De las siete cartas (seis cartas nuevas más la carta original) se toma al más conveniente para la situación que se esta estudiando. Si, en cambio, se tiene (1,6)-ES, la información inicial no intervendrá en la selección (Figura 2.9).

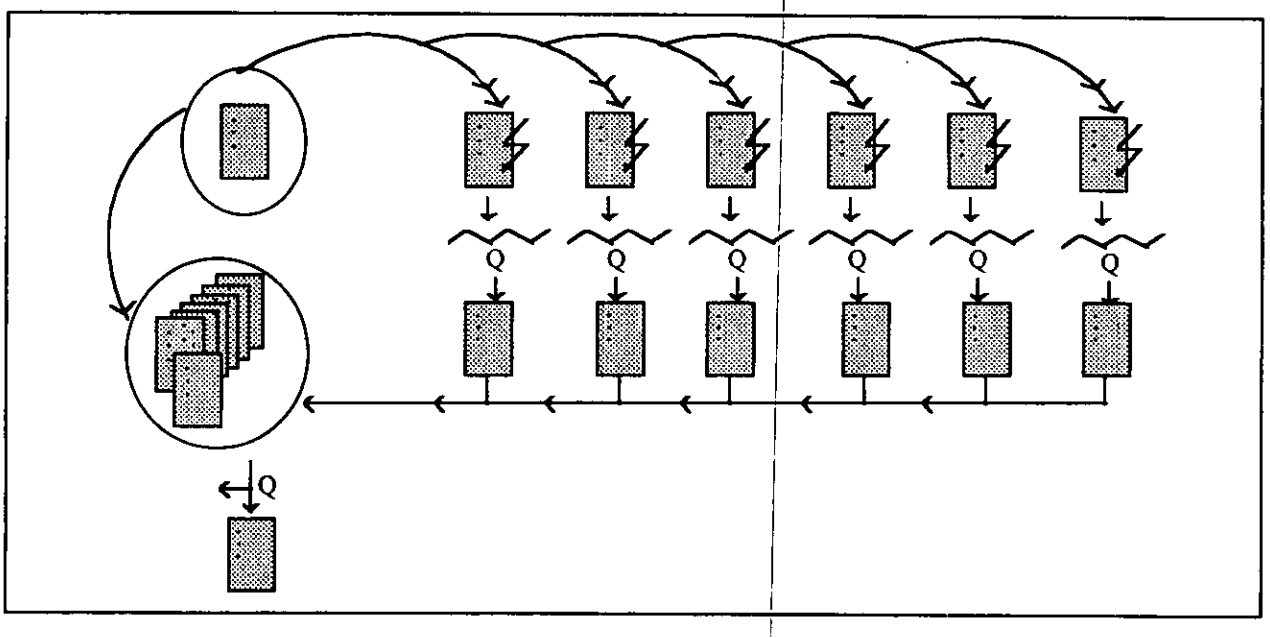


Figura 2.9. Ejemplo 2.

Ejemplo 3. (6/2,10)-ES. El algoritmo a seguir para este caso, es: El símbolo / indica que habrá recombinación, entonces, de las 6 cartas que hay, se aparta una aleatoriamente. Esta duplica solamente la mitad de su información a un descendiente. Enseguida se elige, otra también al azar, que transfiere la mitad de su información al mismo descendiente. De esta manera, se siguen eligiendo cartas hasta producir una nueva generación de 10 individuos. Aquí la información de los nuevos individuos se muta. Después, se prueban con el problema y se les asigna una puntuación a cada uno de ellos. De los 10 se escogen los 6 mejores.

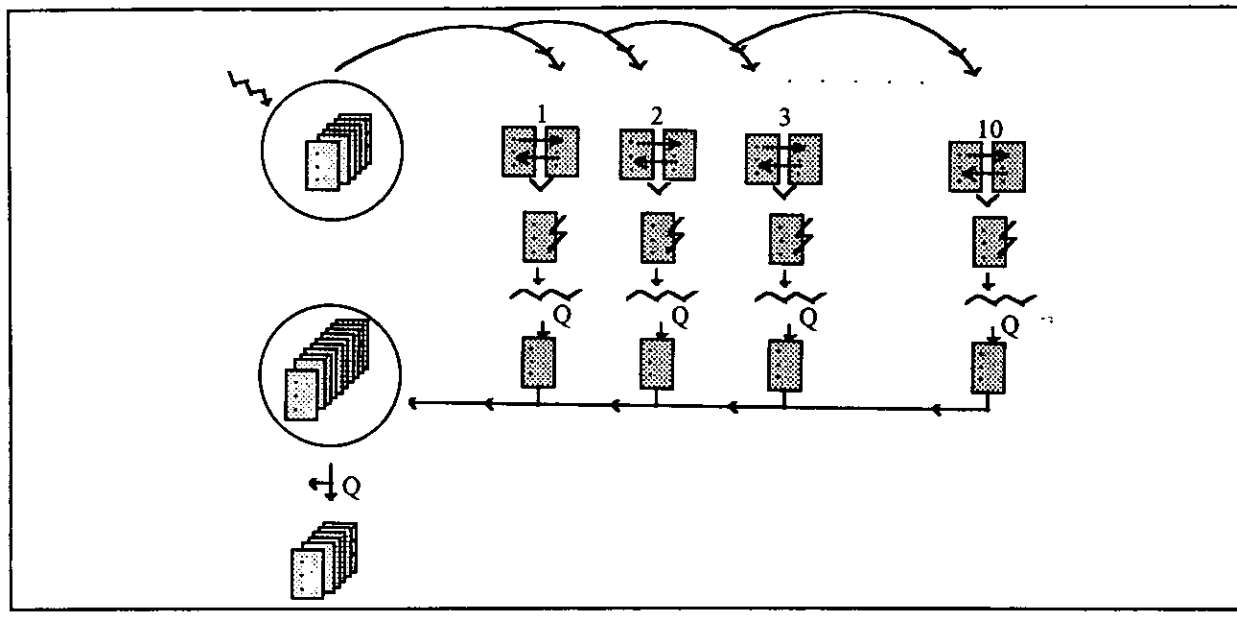


Figura 2.10. Ejemplo 3.

Si cada uno de los ejemplos anteriores se repite varias veces, entonces la generación siguiente es superior o igual a la anterior. Comparándolo con la evolución biológica, de generación en generación, la especie esta cada vez más adaptada a su medio. Estos algoritmos como cualquiera de sus variantes pueden aplicarse tanto a problemas físicos como a problemas matemáticos y este caso es el tema central del presente trabajo.

La idea de relacionar la biología con otros campos muy diferentes, como la computación y las matemáticas, no es nueva. A partir de los años 40's han surgido, entre otras disciplinas, la Inteligencia Artificial y las redes neuronales que toman ideas de la biología. De hecho, científicos computacionales y biólogos moleculares han comenzado a investigar conjuntamente (Lander [18]). Douglas Hofstadter de la Universidad de Indiana ha demostrado claramente que la acción del DNA y RNA durante la reproducción de una célula puede ser interpretada como un ejemplo de una máquina de Turing que se reproduce así misma (Hosftadter [14]).

2.2 Descripción del Método Hill-Climbing

Como pudo verse, los algoritmos de la sección anterior son realmente sencillos, y se penso emplear este tipo de procedimientos a problemas complicados como son la funciones no lineales. El Método Hill-Climbing es un algoritmo específico de las Operaciones de la Estrategia de la Evolución aplicado a funciones no lineales.

El Método Hill-Climbing puede verse como una analogía con el asenso de un grupo de alpinistas a una montaña (Figura 2.11). El procedimiento que ellos siguen es: Como su objetivo es llegar a la cúspide de una montaña. Ellos parten de una posición inicial (a), cada uno de ellos se apartan de ella aleatoriamente (b). Después, se averigua que alpinista está más cerca de la cima, su posición es tomada como nuevo punto de partida (c). Las demás posiciones desaparecen (d).

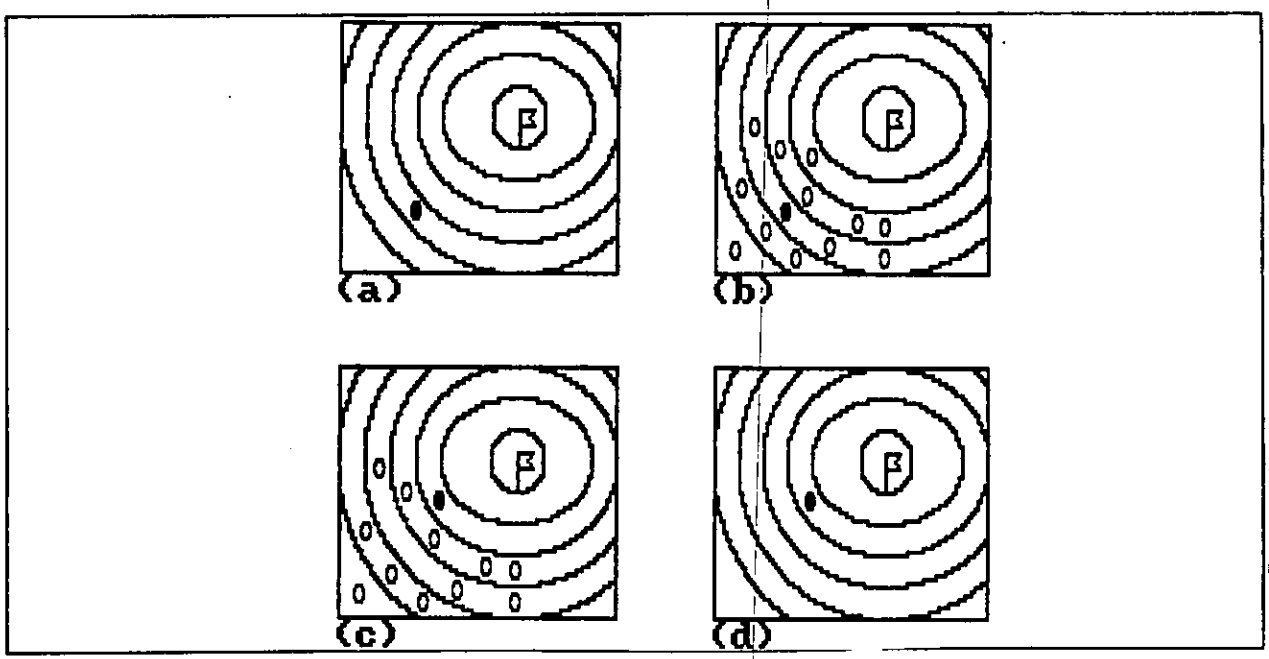


Figura 2.11. Representación gráfica del Método Hill-Climbing (Mendoza-Anzaldo [23]).

Comparando lo anterior con la optimización no lineal, la montaña es la función objetivo, la cima representa su punto óptimo y los alpinistas, las soluciones producidas en cada iteración.

En cada una de ellas, el punto óptimo se va cercando más y más. Este es el procedimiento mas común que utilizan muchos métodos de optimización.

Aplicando el Método Hill-Climbing a la programación no lineal

En este apartado se describirá el algoritmo del método Hill-Climbing, que fue empleado para encontrar el óptimo en funciones no lineales. El algoritmo sirve para optimizar funciones de univariables y multivariables, restringidas o no restringidas.

Usando la nomenclatura que se vió anteriormente, el algoritmo es: (3+24)-EE. Describiéndolo mas explícitamente es: se toman tres variables, los valores iniciales de las variables pueden ser proporcionados o generados aleatoriamente en la computadora. Los valores se sustituyen en la función objetivo. Se toma los tres mejores valores. Para la primera iteración, estos valores son los mismo que los valores iniciales.

Para crear los valores de la nueva generación, cada valor inicial se duplica un número determinado; en este caso, ocho. No se utiliza elección aleatoria porque así se asegura que el mejor valor se reproducirá. No solamente se duplica al mejor sino que la búsqueda del óptimo es a través de un conjunto de puntos, de lo contrario únicamente se exploraría alrededor de un solo punto.

Los valores resultantes se mutan de tres maneras diferentes de acuerdo a una probabilidad.

1. Con una variación pequeña (entre 0 y 1).
2. Con una variación que fluctúa entre 1 y el mejor valor obtenido hasta el momento.
3. Con una variación que esta entre 1 y un valor fijo para este algoritmo un valor de 10.

Estas modificaciones para los sucesores son importantes, porque de esta forma se abarca gran parte del espacio de soluciones. La primera mutación es para cuando los valores iniciales estén cerca del óptimo. Con variaciones pequeñas hay más exactitud en el resultado.

La segunda modificación es conveniente porque la variación cambia de acuerdo al tamaño del mejor valor obtenido. Si este valor es pequeño las variaciones serán reducidas. Si el valor es grande las variaciones serán considerables. En este segundo caso, si los valores iniciales están lejos del óptimo la convergencia del algoritmo es más rápida.

Y la última modificación es importante por los siguiente: cuando el mejor valor obtenido es pequeño las variaciones serán de la misma magnitud, y los valores nuevos serán muy cercanos. Pero si el óptimo global está más allá del alcance que puedan tener las variaciones, una modificación independiente del mejor valor obtenido ayuda a alcanzar ese óptimo.

Hay que tener en cuenta que las mutaciones son tomadas al azar en cada iteración porque el algoritmo trabaja "solo" y no se puede saber cuál de las tres mutaciones conviene más para cada iteración.

Por último los valores que fueron generados y mutados son evaluados en la función objetivo. De los valores nuevos más los valores iniciales, se eligen a los tres mejores (Lo conveniente es que estos tres valores sean diferentes entre sí, por que de otra manera el espacio solución se reduce). Estos pasan a ser los valores iniciales de la siguiente generación. El algoritmo se repite hasta encontrar un punto muy próximo al óptimo.

Algoritmo detallado del Método Hill-Climbing

En este apartado se describirá paso a paso el algoritmo que se aplicó a las funciones no lineales. Para emplearlo en una computadora sólo queda traducirlo a cualquier lenguaje de programación por computadora[4].

Paso 1. Se da un valor **P**, que establece el número de valores iniciales (o número de padres) para cada generación o iteración. Este valor se mantiene fijo para durante todo el procedimiento. $P = 3^5$.

La variable **R** indica las veces que se duplicarán cada uno de los valores iniciales. $R = 8$, para todos los valores iniciales, de esta forma tienen la misma posibilidad de duplicarse.

[4] Las letras y palabras que aparecen en mayúsculas y negritas, representan variables para un programa para computadora.

[5] Con pruebas experimentales se vio que con $N = 3$ y $M = 8$ los resultados eran bastante buenos y de esta manera no se trabaja con muchas variables.

Proporcionese una tolerancia arbitrariamente pequeña (**EPSILON**) y un número límite de iteraciones (**ITERACIONES**). Estos dos valores son necesarios para detener el algoritmo, así no hay la posibilidad que se entre en un ciclo infinito. Se recomineda para cada uno de ellos los valores de 0.005 y 100 repectivamente.

K señala el número de la iteración. **K** = 1.

Los valores iniciales son generados aleatoriamente. Para identificarlos a lo largo del algoritmo: **X_INICIAL**$_1$, **X_INICIAL**$_2$ y **X_INICIAL**$_3$.

Paso 2. Los valores son sustituidos en la función objetivo y los resultados se guardan en **RESULTADO**$_N$.

Paso 3. Se ordenan de mejor a peor. Si lo que se quiere es un mínimo estos se ordenarán de menor a mayor.

Paso 4. Seleccionese los primeros **P** valores que tuvieron los mejores resultados entre los iniciales y los de la nueva generación. Debe asegurarse que los valores sean diferentes entre sí. En el caso de la primera iteración solamente existen los **P** valores iniciales.

Paso 5. MEJOR guarda el mejor punto encontrado hasta el momento.

Paso 6. RESTA = ABS[6](**RESULTADO**$_1$-**RESULTADO**$_3$),

Es decir, se obtiene el valor absoluto del resultado del primer valor menos el resultado del tercer valor, obtenidos hasta esta iteración.

Si (**RESTA** < **EPSILON**) o (**K** > **ITERACIONES**) entonces

EL ALGORITMO TERMINA

en caso contrario

EL ALGORITMO CONTINUA

En palabras, el algoritmo termina si la diferencia entre los tres valores iniciales es muy pequeña, esto es en las subsecuentes iteraciones el método seguirá convergiendo hacia un punto, con lo que se ha encontrado un muy buen resultado. O, si se sobrepasa el número de iteraciones fijado antes, entonces el método esta divergiendo. Con la primera condición que se cumpla el algoritmo concluye.

Paso 7. **K** = **K** + 1.

Paso 8. Se toma un número aleatorio **RND** entre 0 y 1.

RND = RANDOM[7]. Este valor servirá para escoger al azar la mutación que tendrá la nueva generación.

Si **RND** < 0.33 entonces

MUTACION = RANDOM

(1a. mutación)

---

[6] Valor absoluto.

[7] RANDOM es un proceso que genera números aleatorios en el intervalo de 0 y 1, sin tomar nunca estos dos números.

Si **RND** ≥ 0.33 Y **RND** < 0.66 entonces

MUTACION = RANDOM[8](TRUNC[9](ABS(**MEJOR**)) + RANDOM[10]

(2a. mutación)

Con esta mutación se consigue un número aleatorio real entre 0 y el valor óptimo logrado hasta esta iteración.

Si **RND** ≥ 0.66 entonces

MUTACION = RANDOM(10) + RANDOM

(3a. mutación)

Aquí se obtendrá un número aleatorio real entre 0 y 10.

De ninguna manera **MUTACION** debe tomar valor de 0, porque los descendientes serían iguales a los progenitores (lo que no sucede en biología). Hay que notar que tal y como se escogen las mutaciones existe la misma probabilidad para obtener mutaciones grandes y pequeñas.

Paso 9.

BOX = SQRT[11](-2*LN[12](RANDOM)) * SEN[13](6.2832*RANDOM)/SQRT(**N**)

Esta es una fórmula que genera números aleatorios con una distribución normal con media 0 y desviación $1/\sqrt{n}$ (BOX y MULLER [4]). Así, los nuevos valores estarán distribuidos simétricamente alrededor de los valores iniciales.

$$\mathbf{X\_NUEVA}_L = \mathbf{X\_INICIAL}_N + \mathbf{MUTACION} * \mathbf{BOX}$$

Cada $\mathbf{X\_INICIAL}_N$ tendrá **R** descendientes. Es decir, debe repetirse esta fórmula **R** veces para cada valor inicial. La mutación es la misma para todos los nuevos valores en cada generación, pero en cambio el valor de **BOX** es diferente para en uno de ellos (Cabe aclarar que L = {1..24} y N = {1..3}). Regresar al paso 2.

En el apéndice A se incluyen algunas de pruebas experimentales con diferentes parámetros para el método Hill-Climbing. El apéndice B contiene el programa en lenguaje Pascal del método Hill-Climbing.

---

[8] En este caso el procedimiento RANDOM generará un número entre 0 y el valor de la variable **MEJOR**. Esta función solo acepta números enteros y positivos.

[9] Función que trunca un número real en entero.

[10] Esta función al final de la fórmula indica que se agregarán decimales al valor que se obtenga, esto es para dar una mayor excatitud en el resultado. Cada vez que se llama a la función RANDOM se tiene un número diferente.

[11] Raíz cuadrada.

[12] Logaritmo natural.

[13] Función Seno

Observaciones sobre del Método.

Para utilizar este método con una mayor eficacia hay que tener en cuenta que:

1. Conviene tener una idea del tamaño de los valores que deben tomar las variables del problema. De esta manera el algoritmo convergirá más rápido. Es decir, si los resultados que se esperan están en miles procurar que los valores iniciales estén en ese rango.

2. Es muy necesario saber que valores están permitidos en el problema, para así agregar las restricciones necesarias al algoritmo Hill-Climbing. Por ejemplo, si en la función objetivo hay un cociente entre dos variables, entonces hay que especificar en el algoritmo que en caso de que la variable del denominador tome el valor de 0, no se evalúe la función objetivo y que el algoritmo busque otro valor.

3. Si lo que se desea es encontrar un mínimo, por ejemplo, y al algoritmo parece divergir, tal vez el problema planteado no tenga un mínimo sino más bien un máximo, o no exista un mínimo global en el que se detenga el algoritmo. Para este caso hay que volver a probar el algoritmo del método Hill-Climbing ahora buscando un máximo.

4. De los tres tipos de mutaciones que tiene el algoritmo para la nueva generación de valores, la tercera es la más importante; en caso de tener un problema con un espacio solución no convexo y los valores iniciales son muy pequeños, esta última mutación ayuda a "saltar" las áreas en donde no hay continuidad.

5. Si el problema es multivariable, entonces cada variable se maneja de forma independiente, y de esta manera el mejor valor obtenido es diferente en cada variable.

Convergencia del Método.

Por su estructura el método convergirá a un punto óptimo, cuando éste exista en el problema y además esté especificado correctamente el tipo de óptimo (máximo o mínimo) que se quiere encontrar. El algoritmo halla al óptimo porque en cada iteración se escoge al mejor punto.

"La condición de que una sucesión sea acotada y monótona es suficiente para la

convergencia" (Courant).

Reuniendo el primero de los mejores valores encontrados en cada iteración, se obtiene una sucesión. Una sucesión es monótona creciente si $X_m \geq X_n$ para $m > n$. O $X_m \leq X_n$ para $m > n$, si la sucesión es decreciente. Cuando el problema a estudiar incluye restricciones el método no diverge de ninguna manera porque el espacio de soluciones esta limitado.

Para que una sucesión sea convergente es preciso que cumpla con el Criterio de Convergencia de Cauchy. Este establece que:

"La sucesión $X_1, X_2, X_3, \ldots$ converge si y sólo si para todo ϵ positivo existe un N tal que $|X_n - X_m| < \epsilon$ para todos n y m mayores que N". (Courant)

Es decir, una sucesión es convergente si dos cualesquiera de sus elementos difieren en menos de ϵ uno del otro. Y como este es el criterio de término del algoritmo, entonces se deduce que el método converge siempre y cuando halla un óptimo global y se tomen en cuenta las observaciones mencionadas en el apartado anterior. Además el algoritmo también es detenido cuando llega a un número de iteraciones determinado, por lo que el método no se vuelve un ciclo infinito.

2.3 El Método Hill-Climbing vs. Algoritmos Genéticos.

Los **Algoritmos Genéticos** (GA's, por sus siglas en inglés, Genetic Algorihtms) son muy semejantes a al método Hill-Climbing. Los Algoritmos Genéticos también toman la idea de la evolución biológica para resolver diferentes tipos de problemas. Utilizan la reproducción (o duplicación), recombinación y mutación (Karr).

Los Algoritmos Genéticos fueron creados por John Holland en la Universidad de Michigan alrededor de 1960. Holland, sus colegas y estudiantes, buscaban la forma de construir una máquina que fuera capaz de aprender. Holland notó que el aprendizaje puede ocurrir no solo por acoplamiento de un simple organismo a su medio, sino que por adaptación a través de muchas generaciones de una especie. El se basó en la noción de Darwin en la cuál solamente el más apto sobrevive. Lo que Holland proponía era que una máquina adquiriría conocimientos buscando a través de un grupo de estrategias de un conjunto de candidatas,

en vez de la construcción y refinamiento de una sola estrategia (Denning [7]).

El nombre de Algoritmos Genéticos comienza a popularizarse después de que Holland publica un libro en 1975 (Holland [16]). En 1989, David Golberg de la Universidad de Alabama, publica un libro en donde demuestra las bases sólidas y científicas para este campo y cita 73 aplicaciones interesantes (Goldberg [9]). En 1991, Lawrence Davis de Tica Associates publica un libro de bolsillo de Algoritmos Genéticos (Davis).

Este método al igual que el método Hill-Climbing utiliza lo siguiente: una población de puntos y no un punto único, no utiliza derivadas, emplea reglas de probabilidad de transición y no reglas determinísticas. Pero la principal diferencia entre estos dos métodos, radica en que Algoritmos Genéticos convierte los valores de las variables a optimizar a código binario (ceros y unos). Por lo general usa 8 dígitos para representar los números. Por ejemplo, 19 en binario es: 00010011. Cuando el número a representar es real, se manejan por dos números en binario. Esto no sucede en el método Hill-Climbing. Este sólo utiliza una variable para cada valor.

Debe hacerse notar que las operaciones de combinación, mutación, etc. se realizan con directamente los números en binario. Y otra diferencia entre estos dos procedimientos, es que en Algoritmos Genéticos es posible que haya combinación con problemas univariables, ya que los valores, pueden seccionarse en dos o más partes. Por último, si se están empleando para Algoritmos Genéticos 8 dígitos, entonces el número de individuos diferentes que pueden tenerse es: $2^8 = 256$. En Hill-Climbing, la cantidad de individuos distintos puede ser muchísimo mayor.

3

Pruebas Experimentales

En este capítulo se verá como trabaja el Método Hill-Climbing. En el primer ejemplo se explica el algoritmo paso a paso con una función de una sola variable y un solo óptimo. En seguida hay otro ejemplo de una función multimodal de una variable. En la segunda parte de este capítulo se muestran dos ejemplos de funciones de varias variables. En la tercera y última parte aparecen dos funciones restringidas, de una y varias variables respectivamente.

En todos los casos presentados el Método Hill-Climbing resuelve los problemas satrisfactoriamente. Esto demuestra lo versátil que puede llegar a ser este metodo.

3.1 FUNCIONES DE UNA SOLA VARIABLE

A continuación se describirá paso a paso el algoritmo del Método Hill-Climbing aplicado a un problema de una variable.

EJEMPLO A

Encontrar el mínimo de la siguiente función unimodal:

$$Min\ Z = X^2-6X+2$$

Utilizando el algoritmo se tiene:

Paso 1. P = 3; Número de valores iniciales.

EPSILON = 0.005; Número arbitrariamente pequeño.

ITERACIONES = 100;

R = 8; Número de duplicación para cada valor.

K = 0; Contador de las iteraciones.

$X\_INICIAL_1$ = 1091.405

$X\_INICIAL_2$ = -2205.4845

$X\_INICIAL_3$ = 263.9043

(Como se recordará estos valores son generados aleatoriamente.)

Paso 2. Los valores se sustituyen en la función objetivo.

$f(X\_INICIAL_1) = f(1091.4075) = 1184623.9784 = RESULTADO_1$

$f(X\_INICIAL_2) = f(-2205.4845) = 4877396.9865 = RESULTADO_2$

$f(X\_INICIAL_3) = f(2673.9043) = 7133722.6151 = RESULTADO_3$

Paso 3. Se ordenan los valores, en este caso de menor a mayor.

Paso 4. Se toman los tres mejores valores obtenidos, que en esta primera iteración son los mismos valores iniciales.

Paso 5. Se identifica al mejor valor obtenido hasta el momento.

MEJOR = 1091.4075 y f(MEJOR) = 1184623.9784

Paso 6. Se verifica si: los tres mejores resultados son muy cercanos o se ha llegado al límite de iteraciones, ya que en cualquiera de los dos casos el algoritmo termina.

$RESTA = ABS(RESULTADO_1 - RESULTADO_3)$

$= ABS(1184623.9784 - 7133722.6151) = 5949098.6367$

Como RESTA > EPSILON y K < ITERACIONES entonces el algoritmo continúa.

Paso 7. K = K+1 = 1.

Paso 8. RND = 0.0960 (valor generado aleatoriamente). Como RND < 0.33 se aplica la 1a. mutación.

MUTACION = RANDOM = 0.9048

Paso 9. La nueva generación de valores se genera a partir de:

BOX = SQRT(-2*LN(RANDOM))*SEN(PI*RANDOM)

$X\_NUEVA_L = X\_INICIAL_N + MUTACION * BOX$

Recuerdese que cada valor inicial tendrá R descendientes. En este caso 8 por cada uno; lo que da en total, para los 3 valores iniciales, un conjunto de 24 puntos diferentes.

Debe regresarse al **Paso 2**, en donde se evalúa esta nueva generación en la función objetivo.

Paso 3. Ordenar los valores de menor a mayor de acuerdo a su resultado. Aquí están también incluidos los valores iniciales.

Paso 4. Escoger los 3 primeros lugares que fueron los 3 mejores valores obtenidos en esta iteración. Hay para elegir a los mejores se incluyen los valores iniciales. En este caso resultó que hubo dos nuevos valores más un valor inicial. Entonces los nuevos valoes iniciales para la segunda iteración son:

$X\_INICIAL_1 = 1090.8565$ y f(1090.5865) = 1182837.4752

$X\_INICIAL_2 = 1090.7640$ y f(1090.7640) = 1183223.4531

$X\_INICIAL_3 = 1091.4075$ y f(1091.4075) = 1184623.9784

Paso 5. El mejor valor obtenido es igual al 1° lugar.

Existe una mejoría en el resultado de: 1786.5032 puntos.

Paso 6. Como ninguna de las condiciones se cumple (RESTA = 1786.5032 y K = 2) el algoritmo continúa.

En la Figura 3.1 pueden verse los resultados de la primera iteración.

n	X_INICIAL	+ MUTACION * BOX =	X_NUEVA	Z
1	1,091.4075	-0.8210	1,090.5865	1,182,837.4752
2	1,091.4075	-0.6436	1,090.7639	1,183,223.4531
3	1,091.4075	0.1114	1,091.5189	1,184,866.4201
4	1,091.4075	0.5361	1,091.9436	1,185,791.2501
5	1,091.4075	0.5814	1,091.9889	1,185,889.8574
6	1,091.4075	0.5983	1,092.0058	1,185,926.7376
7	1,091.4075	0.6505	1,092.0580	1,186,040.3928
8	1,091.4075	0.6873	1,092.0948	1,186,120.5976
9	-2,205.4845	2.4443	-2,203.0402	4,866,606.4216
10	-2,205.4845	1.2269	-2,204.2576	4,871,979.3486
11	-2,205.4845	1.1905	-2,204.2940	4,872,140.0838
12	-2,205.4845	0.5153	-2,204.9692	4,875,121.3775
13	-2,205.4845	0.2078	-2,205.2767	4,876,479.1510
14	-2,205.4845	-0.1535	-2,205.6380	4,878,075.0797
15	-2,205.4845	-0.1919	-2,205.6764	4,878,244.5380
16	-2,205.4845	-0.7779	-2,206.2624	4,880,833.4188
17	2,673.9043	-1.9034	2,672.0009	7,123,558.3924
18	2,673.9043	0.0099	2,673.9142	7,133,775.3388
19	2,673.9043	0.2972	2,674.2015	7,135,310.0568
20	2,673.9043	0.4879	2,674.3922	7,136,329.1177
21	2,673.9043	0.5558	2,674.4601	7,136,691.8464
22	2,673.9043	0.7757	2,674.6800	7,137,866.9615
23	2,673.9043	0.8172	2,674.7215	7,138,088.8382
24	2,673.9043	0.9942	2,674.8985	7,139,034.4474

Figura 3.1 Primera iteración ya ordenada.

Paso 7. K = 2.

Paso 8. RND = 0.6358, con este valor hay que aplicar la 2a. mutación ya que el valor esta entre 0.33 y 0.66. Entonces se tiene:

MUTACION = RANDOM(10) + RANDOM = 9.8388

Paso 9. Se generan los nuevos valores con la fómula anterior (Figura 3.2).

Regresan al **Paso 2.** Se evalúan los valores en la función objetivo, se ordenan de menor a mayor (**Paso 3**), se escoger los tres mejores valores (**Paso 4**), y se remplazan los tres valores iniciales.

$X\_INICIAL_1$ = 1073.9398 y $f(X\_INICIAL_1)$ = 1146905.0723

$X\_INICIAL_2$ = 1083.0734 y $f(X\_INICIAL_2)$ = 1166551.5332

$X\_INICIAL_3$ = 1085.9294 y $f(X\_INICIAL_3)$ = 1172729.1018

Paso 5. El mejor valor hasta el momento es: 1073.9398. Se avanzó 35932.4029 puntos en el resultado.

Paso 6. Ya que RESTA = 25824.0295 y K = 2, entonces el algoritmo sigue.

La Figura 3.2 contiene los resultados obtenidos en la segunda iteración.

n	X_INICIAL	+ MUTACION * BOX =	X_NUEVA	Z
1	1,091.4075	-17.4677	1,073.9398	1,146,905.0723
2	1,090.5865	-7.5131	1,083.0734	1,166,551.5332
3	1,091.4075	-5.4781	1,085.9294	1,172,729.1018
4	1,090.5865	-3.7856	1,086.8009	1,174,617.4591
5	1,090.7640	-3.7838	1,086.9802	1,175,006.0461
6	1,090.7640	-2.7068	1,088.0572	1,177,342.0071
7	1,090.7640	-2.4893	1,088.2747	1,177,814.1763
8	1,091.4075	-2.3860	1,089.0215	1,179,435.6829
9	1,091.4075	-1.2808	1,090.1267	1,181,837.4688
10	1,090.5865	0.6917	1,091.4782	1,184,777.8543
11	1,091.4075	0.5209	1,091.9284	1,185,758.0978
12	1,090.7640	1.3032	1,092.0672	1,186,060.3776
13	1,090.5865	1.6553	1,092.2418	1,186,440.8546
14	1,091.4075	1.0681	1,092.4756	1,186,950.0866
15	1,091.4075	1.4660	1,092.8735	1,187,817.2916
16	1,090.7640	3.0758	1,093.8398	1,189,924.4128
17	1,090.7640	5.7361	1,096.5001	1,195,735.4666
18	1,090.5865	6.0160	1,096.6025	1,195,959.4449
19	1,090.7640	10.3353	1,101.0993	1,205,815.0680
20	1,090.5865	11.6416	1,102.2281	1,208,295.5030
21	1,090.5865	12.9839	1,103.5704	1,211,248.3152
22	1,091.4075	13.2601	1,104.6676	1,213,664.5589
23	1,090.7640	14.7336	1,105.4976	1,215,493.9933
24	1,090.5865	18.6505	1,109.2370	1,223,753.4287

Figura 3.2 Segunda iteración ordenada.

Paso 7. K = 3.

Paso 8. RND = 0.5928 entonces con la segunda mutación tenemos MUTACION = 627.6239.

Paso 9. Se generan los nuevos valores. Regresar al paso 2.

Paso 2. La nueva generación de valores se sustituye en la función objetivo.

Paso 3. Se ordenan de maner a mayor.

Paso 4. Se remplazan a los valores iniciales con los mejores tres valores obtenidos hasta el momento.

Paso 5. MEJOR = 491.4102 Y f(MEJOR) = 238537.5319. En esta iteración hay una mejora de 908367.5404 puntos en el resultado.

Después de 14 iteraciones el mejor resultado que se obtiene es:

$$X = 3.0026 \quad Z = -7.0000$$

Las siguientes dos iteraciones no alteran los resultados. En la Figura 3.3 están los resultados de cada iteración que se consiguieron siguiendo el algoritmo hasta su término. En la primera columna aprarecen el mejor valor de cada iteración, en la segunda, el valor evaluado en la función a estudiar, en la tercera la diferencia entre los resultado del 1° y 3° de cada iteración y por último en la cuarta columna esta la diferencia en los resultado

actual contra el anterior.

IT.	X_NUEVA		Z	DIFERENCIA 1° VS. 3°	DIFERENCIA 1° VS. IT ANTERIOR
1	1°	1,090.5865	1,182,837.4752	1,786.5032	1,786.5032
	2°	1,090.7840	1,183,223.4531		
	3°	1,091.4075	1,184,623.9784		
2	1°	1,073.8388	1,146,905.0723	25,824.0295	35,932.4029
	2°	1,083.0734	1,166,551.5332		
	3°	1,085.9294	1,172,729.1018		
3	1°	491.4102	238,537.5319	21,654.1400	908,367.5404
	2°	500.5303	247,529.4017		
	3°	513.0987	260,191.6719		
4	1°	449.9789	199,783.1110	25,338.4105	38,754.4209
	2°	462.3312	210,978.1948		
	3°	477.4771	225,121.5215		
5	1°	449.7706	199,596.9776	75.4646	186.1334
	2°	449.8129	199,634.7869		
	3°	449.8551	199,672.4422		
6	1°	448.2186	196,212.6263	282.1365	1,384.3513
	2°	448.2825	196,269.4955		
	3°	448.5129	196,474.7628		
7	1°	436.2501	169,435.6338	7,855.8434	8,778.9825
	2°	440.7040	191,577.8099		
	3°	447.1829	197,291.4772		
8	1°	437.0611	188,402.0407	955.3904	1,033.5931
	2°	437.5126	188,794.2012		
	3°	438.1602	189,357.4311		
9	1°	425.8864	176,825.8861	398.6348	9,576.1548
	2°	426.3328	179,203.8957		
	3°	426.3574	179,224.5209		

IT.	X_NUEVA		Z	DIFERENCIA 1° VS. 3°	DIFERENCIA 1° VS. IT ANTERIOR
10	1°	420.6430	174,418.6378	2,456.7643	4,407.2485
	2°	423.3479	176,685.3985		
	3°	423.5739	176,875.4019		
11	1°	413.3513	168,381.1894	4,937.6311	8,037.4482
	2°	416.1951	172,380.0114		
	3°	419.3242	173,318.6205		
12	1°	108.4133	10,687.3027	54,749.6898	157,893.8867
	2°	188.8858	27,444.7857		
	3°	256.8202	65,436.9923		
13	1°	8.0120	18.1204	5,007.0148	10,989.1823
	2°	-60.7414	4,055.9888		
	3°	73.9375	5,025.1350		
14	1°	3.5136	-6.7382	2.8404	24.8586
	2°	2.4260	-6.6706		
	3°	1.2958	-4.0956		
15	1°	3.0026	-7.0000	0.0495	0.2838
	2°	2.9140	-6.9926		
	3°	2.7774	-6.9505		
16	1°	3.0026	-7.0000	0.0495	0.0000
	2°	2.9140	-6.9926		
	3°	2.7774	-6.9505		
17	1°	3.0026	-7.0000	0.0039	0.0000
	2°	3.0080	-6.9989		
	3°	3.0627	-6.9961		

Figura 3.3 Resultados obtenidos por el Método Hill-Climbing.

Nota: Hay que tomar en cuenta que la solución exacta (Prawda [25]) es: X = 3 y Z = -7.

EJEMPLO B

Hallar el valor mínimo de la función:

$$Min\ Z = X^4 - 10X^3 + 35X^2 - 50X$$

Empleando el algoritmo, se tiene que:

P = 3;

EPSILON = 0.005;

ITERACIONES = 100;

R = 8;

K = 0;

$X\_INICIAL_1$ = -1,582.8789
$X\_INICIAL_2$ = 2,987.1072
$X\_INICIAL_3$ = 3,080.3436

$f(X\_INICIAL_1)$ = 6,277,558,372,090.2625 = $RESULTADO_1$
$f(X\_INICIAL_2)$ = 79,350,306,288,640.2784 = $RESULTADO_2$
$f(X\_INICIAL_3)$ = 89,740,002,080,896.5597 = $RESULTADO_3$

En la Figura 3.4 son presentados los resultados que se obtuvieron con el método Hill-Climbing. Esta función es bimodal por lo que tiene dos óptimos globales X = 1.39 y X = 3.62 con Z = 25 (Prawda [25]). A lo largo del experimento se vio que el algoritmo encontró puntos cercanos al primer óptimo (1.39) pero también había hallado puntos mucho más cercanos al segundo óptimo por lo que se "decidió" por este. Esto da una idea que el método puede "saltar" de un óptimo pero siempre eligiendo el mejor.

IT.	X_NUEVA		Z	DIFERENCIA 1° VS. 3°	DIFERENCIA 1° VS. IT ANTERIOR
1	1°	-100.9675	114,581,438.4900	1,499,857.4100	3,847,032.1300
	2°	-101.1178	115,248,868.4800		
	3°	-101.3044	116,081,295.9000		
2	1°	-87.8073	66,490,187.2250	11,800,922.3930	48,091,251.2650
	2°	-91.0290	76,498,898.2960		
	3°	-91.5717	78,291,109.6180		
3	1°	-84.0338	58,052,994.3240	8,310,513.1340	10,437,192.9010
	2°	-86.7582	63,456,527.4040		
	3°	-87.0764	64,363,507.4580		
4	1°	-83.0499	53,546,192.2390	586,969.6560	2,506,802.0850
	2°	-83.0868	53,638,691.7180		
	3°	-83.2833	54,133,161.8950		
5	1°	-78.8207	43,715,870.5960	4,283,022.3050	9,630,321.6430
	2°	-80.2195	46,802,959.5110		
	3°	-80.7429	47,998,892.9010		
6	1°	-55.3902	11,222,671.4410	11,856,868.9010	32,493,199.1550
	2°	-66.3531	22,462,779.9000		
	3°	-66.8206	23,079,340.3420		
7	1°	-54.6384	10,650,727.4620	392,426.7310	571,943.9790
	2°	-54.7333	10,721,854.4880		
	3°	-55.1574	11,043,154.1930		

IT.	X_NUEVA		Z	DIFERENCIA 1° VS. 3°	DIFERENCIA 1° VS. IT ANTERIOR
8	1°	18.0682	58,112.9044	27,838.1986	10,592,614.5576
	2°	-14.1314	75,794.3766		
	3°	-14.6600	85,951.1030		
9	1°	-10.7373	30,243.1107	7,493.7096	27,869.7937
	2°	15.9001	31,770.1489		
	3°	16.4848	37,736.8203		
10	1°	4.9481	-2.5019	4,022.4902	30,245.6126
	2°	-4.8656	2,784.5416		
	3°	10.5530	4,019.9883		
11	1°	3.9350	-24.3454	11.7388	21.8435
	2°	3.9974	-24.0152		
	3°	4.6841	-12.6066		
12	1°	3.5977	-24.9980	0.1175	0.6528
	2°	3.5274	-24.9622		
	3°	3.4510	-24.8805		
13	1°	3.5977	-24.9980	0.0238	0.0000
	2°	3.6428	-24.9969		
	3°	3.6877	-24.9742		
14	1°	3.6101	-24.9997	0.0028	0.0017
	2°	3.5977	-24.9980		
	3°	3.6428	-24.9969		

Figura 3.4 Resultados del ejemplo B.

3.2 FUNCIONES DE VARIAS VARIABLES

En esta ocasión se aplicara el método para en contrar una el máximo valor de una función con tres variables.

EJEMPLO C

$$MAX\ Z = X_1+2X_3+X_2X_3-X_1^2-X_2^2-X_3^2$$

P = 3;

EPSILON = 0.005;

ITERACIONES = 100;

R = 8;

K = 0;

$X\_INICIAL_{1,1}$ = -47.1150

$X\_INICIAL_{1,2}$ = -5.6394

$X\_INICIAL_{1,3}$ = -58.8644

$X\_INICIAL_{2,1}$ = -54.2841

$X\_INICIAL_{2,2}$ = -79.7273

$X\_INICIAL_{2,3}$ = -37.2056

$X\_INICIAL_{3,1}$ = 86.4945

$X\_INICIAL_{3,2}$ = -12.6070

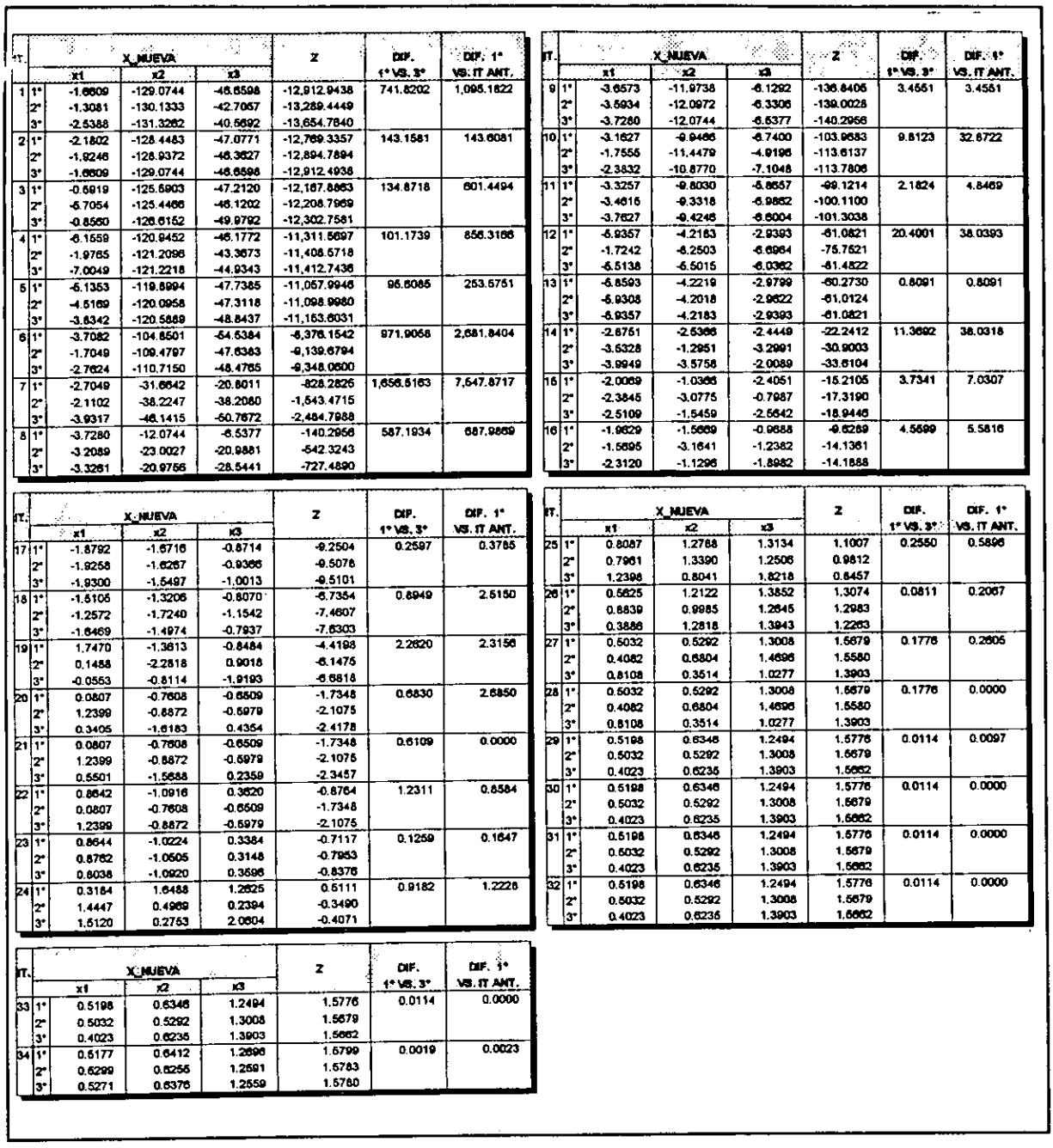
$X\_INICIAL_{3,3}$ = 45.4572

$f(X\_INICIAL_1)$ = -5,549.5275 = $RESULTADO_1$

$f(X\_INICIAL_2)$ = -7,849.8558 = $RESULTADO_2$

$f(X\_INICIAL_3)$ = -10,102.2620 = $RESULTADO_3$

La Figura 3.5 presenta los resultados obtenidos por el método. Los valores óptimos para esta función son: X_1 = 1/2, X_2 = 2/3, X_3 = 4/3 con Z = 1.58 (Prawda [25]).



IT.		X_NUEVA x1	x2	x3	Z	DIF. 1° VS. 3°	DIF. 1° VS. IT ANT.
1	1°	-1.6809	-129.0744	-46.8598	-12,912.9438	741.8202	1,095.1822
	2°	-1.3081	-130.1333	-42.7057	-13,289.4449		
	3°	-2.5388	-131.3282	-40.5692	-13,654.7840		
2	1°	-2.1802	-128.4483	-47.0771	-12,769.3357	143.1581	143.6081
	2°	-1.9246	-128.9372	-46.3627	-12,894.7894		
	3°	-1.6809	-129.0744	-46.8598	-12,912.4938		
3	1°	-0.6919	-125.6903	-47.2120	-12,187.8863	134.8718	601.4494
	2°	-5.7054	-125.4466	-46.1202	-12,208.7969		
	3°	-0.8560	-126.6152	-49.9792	-12,302.7581		
4	1°	-6.1559	-120.9452	-46.1772	-11,311.5697	101.1739	856.3186
	2°	-1.9765	-121.2096	-43.3673	-11,408.5718		
	3°	-7.0049	-121.2218	-44.9343	-11,412.7436		
5	1°	-5.1353	-119.8994	-47.7385	-11,057.9946	96.6085	253.5751
	2°	-4.5169	-120.0958	-47.3118	-11,098.9980		
	3°	-3.8342	-120.5889	-48.8437	-11,163.6031		
6	1°	-3.7082	-104.8501	-54.5384	-9,376.1542	971.9058	2,681.8404
	2°	-1.7049	-109.4797	-47.6383	-9,139.6794		
	3°	-2.7624	-110.7150	-48.4765	-9,348.0600		
7	1°	-2.7049	-31.6642	-20.8011	-828.2826	1,656.5163	7,547.8717
	2°	-2.1102	-38.2247	-38.2080	-1,543.4715		
	3°	-3.9317	-46.1415	-50.7672	-2,484.7988		
8	1°	-3.7280	-12.0744	-6.5377	-140.2956	587.1934	687.9869
	2°	-3.2089	-23.0027	-20.9881	-542.3243		
	3°	-3.3261	-20.9756	-28.5441	-727.4890		

IT.		X_NUEVA x1	x2	x3	Z	DIF. 1° VS. 3°	DIF. 1° VS. IT ANT.
9	1°	-3.6573	-11.9738	-6.1292	-136.8405	3.4551	3.4551
	2°	-3.5934	-12.0972	-6.3306	-139.0028		
	3°	-3.7280	-12.0744	-6.5377	-140.2956		
10	1°	-3.1627	-9.9486	-6.7400	-103.9583	9.8123	32.6722
	2°	-1.7555	-11.4479	-4.9196	-113.6137		
	3°	-2.3832	-10.8770	-7.1048	-113.7806		
11	1°	-3.3257	-9.8030	-5.8657	-99.1214	2.1824	4.8469
	2°	-3.4616	-9.3318	-5.9852	-100.1100		
	3°	-3.7627	-9.4246	-6.6004	-101.3038		
12	1°	-5.9357	-4.2183	-2.9393	-61.0821	20.4001	38.0393
	2°	-1.7242	-6.2503	-6.6964	-75.7521		
	3°	-5.5138	-5.5015	-6.0362	-81.4822		
13	1°	-5.8593	-4.2219	-2.9799	-60.2730	0.8091	0.8091
	2°	-5.9308	-4.2018	-2.9822	-61.0124		
	3°	-5.9357	-4.2183	-2.9393	-61.0821		
14	1°	-2.8751	-2.5366	-2.4449	-22.2412	11.3692	38.0318
	2°	-3.5328	-1.2951	-3.2991	-30.9003		
	3°	-3.9949	-3.5758	-2.0089	-33.6104		
16	1°	-2.0069	-1.0366	-2.4051	-15.2105	3.7341	7.0307
	2°	-2.3845	-3.0775	-0.7987	-17.3190		
	3°	-2.5109	-1.5459	-2.5642	-18.9446		
16	1°	-1.9829	-1.5669	-0.9688	-9.6289	4.5699	5.5816
	2°	-1.5695	-3.1641	-1.2382	-14.1361		
	3°	-2.3120	-1.1296	-1.8982	-14.1888		

IT.		X_NUEVA x1	x2	x3	Z	DIF. 1° VS. 3°	DIF. 1° VS. IT ANT.
17	1°	-1.8792	-1.6716	-0.8714	-9.2504	0.2597	0.3785
	2°	-1.9258	-1.6267	-0.9366	-9.5076		
	3°	-1.9300	-1.5497	-1.0013	-9.5101		
18	1°	-1.5105	-1.3206	-0.8070	-6.7354	0.8949	2.5150
	2°	-1.2572	-1.7240	-1.1542	-7.4607		
	3°	-1.8469	-1.4974	-0.7937	-7.6303		
19	1°	1.7470	-1.3813	-0.8484	-4.4198	2.2820	2.3156
	2°	0.1488	-2.2818	0.9018	-6.1475		
	3°	-0.0553	-0.8114	-1.9193	-6.6818		
20	1°	0.0807	-0.7608	-0.6509	-1.7348	0.6830	2.6850
	2°	1.2399	-0.8872	-0.5979	-2.1075		
	3°	0.3405	-1.6183	0.4354	-2.4178		
21	1°	0.0807	-0.7608	-0.6509	-1.7348	0.6109	0.0000
	2°	1.2399	-0.8872	-0.5979	-2.1075		
	3°	0.5501	-1.5688	0.2359	-2.3457		
22	1°	0.8642	-1.0916	0.3520	-0.8764	1.2311	0.8584
	2°	0.0807	-0.7608	-0.6509	-1.7348		
	3°	1.2399	-0.8872	-0.5979	-2.1075		
23	1°	0.8644	-1.0224	0.3384	-0.7117	0.1259	0.1647
	2°	0.8782	-1.0505	0.3148	-0.7953		
	3°	0.8038	-1.0920	0.3696	-0.8376		
24	1°	0.3184	1.6488	1.2825	0.5111	0.9182	1.2226
	2°	1.4447	0.4969	0.2394	-0.3490		
	3°	1.5120	0.2763	2.0804	-0.4071		

IT.		X_NUEVA x1	x2	x3	Z	DIF. 1° VS. 3°	DIF. 1° VS. IT ANT.
25	1°	0.8087	1.2788	1.3134	1.1007	0.2550	0.5896
	2°	0.7961	1.3390	1.2506	0.9812		
	3°	1.2396	0.8041	1.8218	0.8457		
26	1°	0.5625	1.2122	1.3852	1.3074	0.0811	0.2067
	2°	0.8839	0.9985	1.2645	1.2983		
	3°	0.3886	1.2818	1.3943	1.2263		
27	1°	0.5032	0.5292	1.3008	1.5679	0.1776	0.2605
	2°	0.4082	0.6804	1.4696	1.5580		
	3°	0.8108	0.3514	1.0277	1.3903		
28	1°	0.5032	0.5292	1.3008	1.5679	0.1776	0.0000
	2°	0.4082	0.6804	1.4696	1.5580		
	3°	0.8108	0.3514	1.0277	1.3903		
29	1°	0.5198	0.6346	1.2494	1.5776	0.0114	0.0097
	2°	0.5032	0.5292	1.3008	1.5679		
	3°	0.4023	0.6235	1.3903	1.5662		
30	1°	0.5198	0.6346	1.2494	1.5776	0.0114	0.0000
	2°	0.5032	0.5292	1.3008	1.5679		
	3°	0.4023	0.6235	1.3903	1.5662		
31	1°	0.5198	0.6346	1.2494	1.5776	0.0114	0.0000
	2°	0.5032	0.5292	1.3008	1.5679		
	3°	0.4023	0.6235	1.3903	1.5662		
32	1°	0.5198	0.6346	1.2494	1.5776	0.0114	0.0000
	2°	0.5032	0.5292	1.3008	1.5679		
	3°	0.4023	0.6236	1.3903	1.5662		

IT.		X_NUEVA x1	x2	x3	Z	DIF. 1° VS. 3°	DIF. 1° VS. IT ANT.
33	1°	0.5198	0.6346	1.2494	1.5776	0.0114	0.0000
	2°	0.5032	0.5292	1.3008	1.5679		
	3°	0.4023	0.6236	1.3903	1.5662		
34	1°	0.5177	0.6412	1.2896	1.5799	0.0019	0.0023
	2°	0.5299	0.5255	1.2591	1.5783		
	3°	0.5271	0.5376	1.2559	1.5780		

Figura 3.5 Resultados del ejemplo C.

EJEMPLO D

Para este caso debe encontrarse el mínimo de una función llamada patológica[1], creada por White y Holst [33]:

$$Min\ Z = (X_2^2 - X_1^2)^2 + (1 - X_1)^2$$

Las condiciones iniciales son:

P = 3;

EPSILON = 0.005;

ITERACIONES = 100;

R = 8;

K = 0;

$X\_INICIAL_{1,1}$ = 34.2799

$X\_INICIAL_{1,2}$ = 0.6708

$X\_INICIAL_{2,1}$ = -92.4733

$X\_INICIAL_{2,2}$ = 73.0867

$X\_INICIAL_{3,1}$ = -95.2077

$X\_INICIAL_{3,2}$ = -3.7907

$f(X\_INICIAL_1)$ = 1,380,937.3590 = $RESULTADO_1$

$f(X\_INICIAL_2)$ = 10,310,561.4678 = $RESULTADO_2$

$f(X\_INICIAL_3)$ = 81,914,230.8039 = $RESULTADO_3$

Los resultados finales son mostrados en la Figura 3.6. Para este caso el óptimo ha encontrar es: $X_1 = 1$ y $X_2 = 1$ con $Z = 0.0$.

[1] Este tipo de funciones son creadas especialmente para probar métodos de funciones no lineales. Si el método logra encontrar el óptimo, entonces, este se considera como bueno.

IT.		X_NUEVA x1	X_NUEVA x2	Z	DIF. 1° VS. 3°	DIF. 1° VS. IT ANT.
1	1°	-27.2805	-18.1544	234,340.4423	16,157.0560	35,200.6512
	2°	-27.0659	-15.2619	250,424.8427		
	3°	-27.4433	-15.9200	250,497.4983		
2	1°	-28.4880	-18.5430	126,753.9713	36,025.5816	105,586.4710
	2°	-25.8886	-16.7861	151,536.1460		
	3°	-28.3550	-17.0170	164,779.5529		
3	1°	-22.0914	-18.0260	27,132.7120	12,122.9541	101,621.2593
	2°	-23.8798	-19.6086	35,121.6931		
	3°	-26.3516	-22.3198	39,255.6661		
4	1°	-21.5668	-21.2218	754.3781	5,272.7723	26,378.3339
	2°	-22.7518	-22.3939	825.3415		
	3°	-22.0207	-20.2874	6,027.1504		
5	1°	-21.4891	-21.6124	534.0065	58.8756	220.3716
	2°	-21.4648	-21.6289	554.8720		
	3°	-23.0445	-22.9811	592.8821		
6	1°	-21.4513	-21.4248	505.3614	17.4535	28.6451
	2°	-21.4833	-21.4214	512.5556		
	3°	-21.7725	-21.7252	522.8149		
7	1°	-21.4209	-21.4118	502.8506	0.9510	2.5108
	2°	-21.4241	-21.4398	503.2822		
	3°	-21.4277	-21.4068	503.8016		
8	1°	-11.1925	-11.4177	174.6045	328.6777	328.2481
	2°	-21.4209	-21.4118	502.8506		
	3°	-21.4241	-21.4398	503.2822		

IT.		X_NUEVA x1	X_NUEVA x2	Z	DIF. 1° VS. 3°	DIF. 1° VS. IT ANT.
9	1°	-11.1925	-11.4177	174.6045	328.2481	0.0000
	2°	-15.4361	-15.6678	470.1811		
	3°	-21.4209	-21.4118	502.8506		
10	1°	-11.1690	-11.1209	149.2325	18.7547	25.3720
	2°	-11.1928	-11.3516	161.4818		
	3°	-11.4868	-11.6371	167.9872		
11	1°	-11.0738	-11.1001	146.1172	0.7769	3.1153
	2°	-11.1057	-11.1219	146.8774		
	3°	-11.1114	-11.1319	146.8941		
12	1°	-11.0738	-11.1001	146.1172	0.7769	0.0000
	2°	-11.1057	-11.1219	146.8774		
	3°	-11.1114	-11.1319	146.8941		
13	1°	-11.0740	-11.0980	146.0638	0.0348	0.0534
	2°	-11.0713	-11.0984	146.0775		
	3°	-11.0757	-11.0994	146.0986		
14	1°	-11.0421	-11.0523	145.0839	0.3493	0.9899
	2°	-11.0413	-11.0638	145.2403		
	3°	-11.0583	-11.0628	145.4132		
15	1°	2.2651	0.2998	27.0064	118.0555	118.0555
	2°	-3.6078	-4.8942	140.8519		
	3°	-11.0421	-11.0523	145.0839		
16	1°	0.6061	0.6655	0.1589	15.7889	26.8495
	2°	1.7516	0.5574	6.1679		
	3°	2.0291	0.5088	15.9478		

IT.		X_NUEVA x1	X_NUEVA x2	Z	DIF. 1° VS. 3°	DIF. 1° VS. IT ANT.
17	1°	0.7354	0.6420	0.0868	0.0300	0.0723
	2°	0.6841	0.6795	0.1132		
	3°	0.6811	0.6285	0.1166		
18	1°	0.7354	0.6420	0.0868	0.0300	0.0000
	2°	0.6841	0.6795	0.1132		
	3°	0.6811	0.6285	0.1166		
19	1°	0.7354	0.6420	0.0868	0.0300	0.0000
	2°	0.6841	0.6795	0.1132		
	3°	0.6811	0.6285	0.1166		
20	1°	0.7354	0.6420	0.0868	0.0300	0.0000
	2°	0.6841	0.6795	0.1132		
	3°	0.6811	0.6285	0.1166		
21	1°	0.7354	0.6420	0.0868	0.0300	0.0000
	2°	0.6841	0.6795	0.1132		
	3°	0.6811	0.6285	0.1166		

IT.		X_NUEVA x1	X_NUEVA x2	Z	DIF. 1° VS. 3°	DIF. 1° VS. IT ANT.
22	1°	0.9690	0.8567	0.0430	0.0438	0.0438
	2°	0.7438	0.8038	0.0742		
	3°	0.7354	0.6420	0.0868		
23	1°	1.0873	1.0876	0.0064	0.0366	0.0366
	2°	0.9178	0.8705	0.0139		
	3°	0.9890	0.8567	0.0430		
24	1°	1.0090	1.0293	0.0018	0.0057	0.0048
	2°	1.0873	1.0876	0.0064		
	3°	1.0152	0.9724	0.0075		
25	1°	1.0090	1.0293	0.0018	0.0057	0.0000
	2°	1.0873	1.0876	0.0064		
	3°	1.0152	0.9724	0.0075		
26	1°	1.0090	1.0293	0.0018	0.0048	0.0000
	2°	0.9848	1.0203	0.0053		
	3°	1.0873	1.0876	0.0064		

Figura 3.6 Resultados del ejemplo D.

3.3 FUNCIONES RESTRINGIDAS

Para las funciones restringidas al método Hill-Climbing se le agrega una condición: los valores iniciales deben cumplir con un número adecuado de restricciones (por lo menos 50%).

EJEMPLO E

$$Max\ Z = X_1^2 - 6X_1 + 2$$

$$(X_2-6)^2 - X_1 \geq -4$$

$$10X_2 + X_1 \leq 150$$

$$X_1 \geq 0, X_2 \geq 0$$

Es muy importante aclarar que aunque las restricciones son de dos variables se despejó X_2 quedando solamente X_1 como la variable a resolver (o variable de desición), esto con el fin de hacerlo más sencillo. En la Figura 3.7 aparece la grafica de esta función. El conjunto solución es un área no convexa.

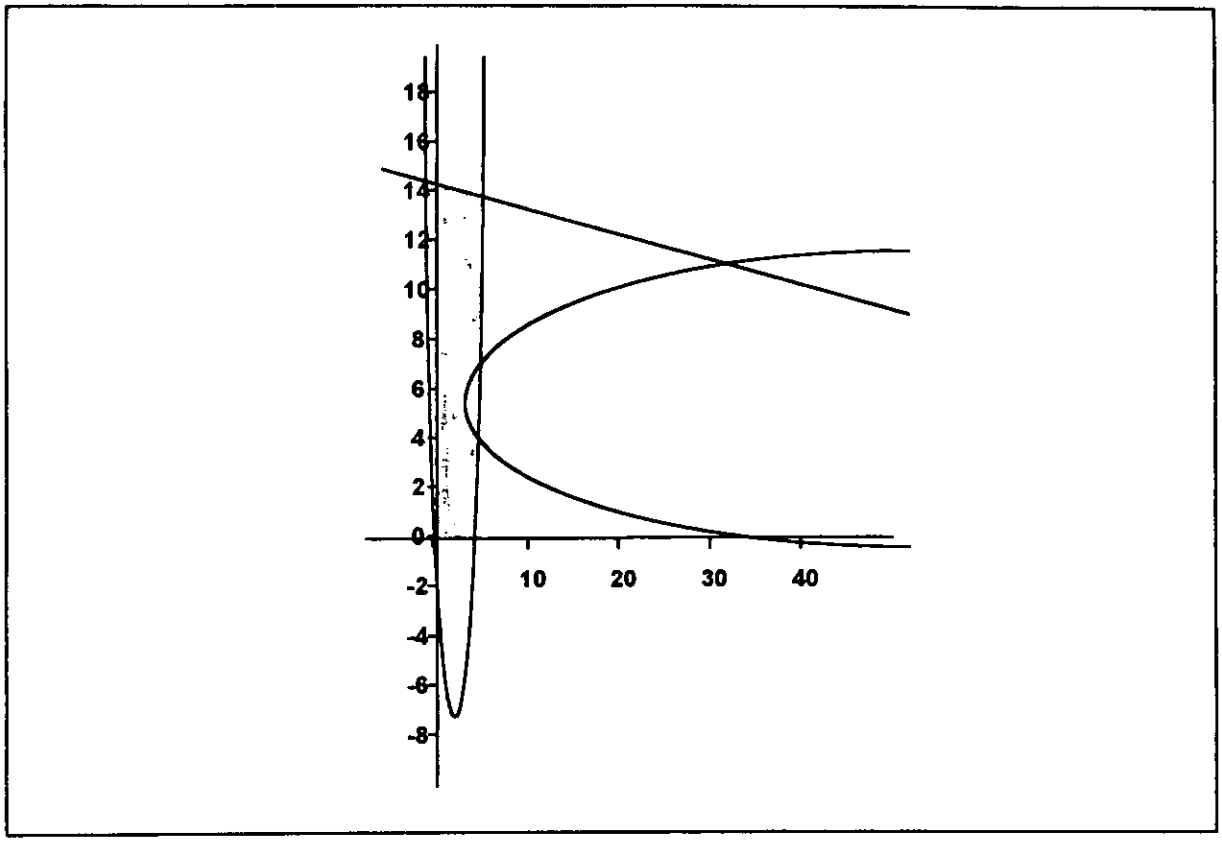


Figura 3.7 Gráfica del ejemplo E.

P = 3;

EPSILON = 0.005;

ITERACIONES = 100;

R = 8;

K = 0;

$X\_INICIAL_1$ = 7.2770

$f(X\_INICIAL_1)$ = 11.2923 = $RESULTADO_1$

RESTRICIONES = 4

$X\_INICIAL_2$ = 79.9747

$f(X\_INICIAL_2)$ = 5918.0968 = $RESULTADO_2$

RESTRICCIONES = 3

$X\_INICIAL_3 = 41.51$

$f(X\_INICIAL_3) = 1476.1769 = RESULTADO_3$

RESTRICCIONES = 3

Los resultados de este ejemplo están en la Figura 3.8. Es muy importante aclarar que las iteraciones que no aparecen en esta tabla es porque el resultado es exactamente el mismo que en la iteración anteiror.

IT.	X_NUEVA		Z	RES	DIFERENCIA 1° VS. 3°	DIFERENCIA 1° VS. IT ANTERIOR
1	1°	7.4885	13.1468	4	1.5581	1.8545
	2°	7.4656	12.9415	4		
	3°	7.3115	11.5887	4		
3	1°	7.4885	13.1468	4	0.8625	0.0000
	2°	7.4656	12.9415	4		
	3°	7.3914	12.2843	4		
4	1°	7.5202	13.4325	4	0.2857	0.2857
	2°	7.5034	13.2809	4		
	3°	7.4885	13.1468	4		
5	1°	7.5991	14.1516	4	0.7191	0.7191
	2°	7.5372	13.5866	4		
	3°	7.5202	13.4325	4		
6	1°	7.5991	14.1516	4	0.5650	0.0000
	2°	7.5715	13.8984	4		
	3°	7.5372	13.5866	4		
10	1°	7.5991	14.1516	4	0.1298	0.0000
	2°	7.5944	14.1088	4		
	3°	7.5650	14.0218	4		

IT.	X_NUEVA		Z	RES	DIFERENCIA 1° VS. 3°	DIFERENCIA 1° VS. IT ANTERIOR
11	1°	7.6084	14.2193	4	0.1105	0.0877
	2°	7.5991	14.1516	4		
	3°	7.5944	14.1088	4		
22	1°	7.6071	14.2256	4	0.0418	0.0063
	2°	7.6064	14.2193	4		
	3°	7.6026	14.1838	4		
39	1°	7.6072	14.2265	4	0.0072	0.0009
	2°	7.6071	14.2256	4		
	3°	7.6064	14.2193	4		
86	1°	7.6072	14.2265	4	0.0071	0.0000
	2°	7.6071	14.2256	4		
	3°	7.6065	14.2194	4		
100	1°	7.6072	14.2265	4	0.0071	0.0000
	2°	7.6071	14.2256	4		
	3°	7.6065	14.2194	4		

Figura 3.8 Resultados ejemplo E.

EJEMPLO F

$$Min\ Z = X_1^2 + X_2^2 - 16X_1 - 10X_2$$

$$11 - X_1^2 + 6X_1 - 4X_2 \geq 0$$

$$X_1X_2 - 3X_2 - e^{X_1 - 3} + 1 \geq 0$$

$$X_1 \geq 0,\ X_2 \geq 0$$

P = 3;

EPSILON = 0.005;

ITERACIONES = 100;

R = 8;

K = 0;

$X\_INICIAL_{1,1} = 3.3760$

$X\_INICIAL_{1,2} = 2.4636$

$f(X\_INICIAL_1) = -61.1849 = RESULTADO_1$

RESTRICIONES = 4

$X\_INICIAL_{2,1} = 28.5870$

$X\_INICIAL_{2,2} = 27.8936$

$f(X\_INICIAL_2) = 858.9394 = RESULTADO_2$

RESTRICCIONES = 2

$X\_INICIAL_{3,1} = 9.6024$

$X\_INICIAL_{3,2} = 48.8215$

$f(X\_INICIAL_3) = 1833.8898 = RESULTADO_3$

RESTRICCIONES = 2

Los resultados de este ejemplo están en la Figura 3.9.

IT.		X_NUEVA x1	X_NUEVA x2	Z	RES	DIFERENCIA 1° VS. 3°	DIFERENCIA 1° VS. IT ANTERIOR
1	1°	4.2834	3.9050	-73.9879	4	12.8030	12.8030
	2°	3.0129	4.4954	-63.8746	4		
	3°	3.3760	2.4636	-61.1849	4		
2	1°	4.6879	4.0550	-77.0039	4	2.2765	3.0180
	2°	4.5027	4.0550	-76.0765	4		
	3°	4.4045	4.1662	-74.7274	4		
3	1°	5.1022	3.5801	-78.5868	4	1.1031	1.5829
	2°	5.0274	3.4109	-77.6362	4		
	3°	4.7614	3.9244	-77.4837	4		
5	1°	5.2089	3.7313	-79.6000	4	0.5768	1.0132
	2°	5.1735	3.6981	-79.3162	4		
	3°	5.1590	3.6197	-79.0234	4		
8	1°	5.2041	3.7518	-79.8251	4	0.1613	0.0251
	2°	5.2089	3.7313	-79.6000	4		
	3°	5.1918	3.7154	-79.4638	4		
19	1°	5.2179	3.7606	-79.7237	4	0.0640	0.0986
	2°	5.2270	3.7254	-79.6857	4		
	3°	5.2233	3.7220	-79.6597	4		

IT.		X_NUEVA x1	X_NUEVA x2	Z	RES	DIFERENCIA 1° VS. 3°	DIFERENCIA 1° VS. IT ANTERIOR
45	1°	5.2195	3.7661	-79.7463	4	0.0606	0.0226
	2°	5.2179	3.7606	-79.7237	4		
	3°	5.2270	6.7254	-79.6857	4		
47	1°	5.2195	3.7661	-79.7463	4	0.0543	0.0000
	2°	5.2179	3.7606	-79.7237	4		
	3°	5.2243	3.7338	-79.6920	4		
60	1°	5.2372	3.7402	-79.7798	4	0.0298	0.0335
	2°	5.2230	3.7613	-79.7537	4		
	3°	5.2198	3.7669	-79.7500	4		
71	1°	5.2387	3.7458	-79.8020	4	0.0483	0.0222
	2°	5.2372	3.7402	-79.7798	4		
	3°	5.2230	3.7613	-79.7537	4		
100	1°	5.2387	3.7458	-79.8020	4	0.0483	0.0000
	2°	5.2372	3.7402	-79.7798	4		
	3°	5.2230	3.7613	-79.7537	4		

Figura 3.9 Ejemplo F.

$X_1 = 5.27$ y $X_2 = 3.68$ con $Z = -79.9$ es el óptimo ha encontrar (Prawda [25]).

4

Descripción de Métodos Alternos Mas Comunes

Para medir la eficiencia de el Método de Hill-Climbign es necesario compararlo con otros métodos conocidos que resuelvan funciones no lineales. Los métodos que se describen en este capítulo son: El Método de Fibonacci para funciones unimodales de una sola variable, el Método de Newton-Rapshon para funciones multimodales de una sola variable y el Método de Davidon-Fletcher-Powell para funciones multimodales de varias variables. Estos métodos serán comparados en el siguiente capítulo contra el Método de Hill-Climbing.

4.1 Método de Fibonacci

Este método encuentra con cierta tolerancia de error el valor óptimo de una función unimodal de una sola variable, sea este máximo o mínimo. Este método no utiliza derivadas, por lo que puede calcular los puntos óptimos de funciones tanto diferenciables como no diferenciables. El procedimiento del método puede resumirse de la siguiente manera: Se da intervalo inicial. En cada iteración vuelve a definirse la cota inferior y la cota superior, por esta razón el intervalo va reduciendose cada vez. El algortimo finaliza cuando se llega a un número de iteraciones preestablecido de acuerdo al límite de error que se desea (Bazaraa [2]).

La parte fundamental en la que se basa el método es la serie de Fibonacci, que tiene la siguiente secuencia: 0,1,1,2,3,5,... Es decir el siguiente número es el resultado de la suma de los dos anteriores. Para utilizar el método se asigna: $F_0 = 1$, $F_1 = 1$, $F_2 = 2$, etc. Por lo que el número siguiente en la serie es: $F_{v+1} = F_v + F_{v-1}$.

En este método se deben calcular 2 puntos λ_k y μ_k

donde:

$$\lambda_k = a_k + \frac{F_{n-k-1}}{F_{n-k+1}}(b_k - a_k)$$

$$\mu_k = a_k + \frac{F_{n-k}}{F_{n-k+1}}(b_k - a_k)$$

a_k = límite izquierdo del intervalo.

b_k = límite derecho del intervalo.

k = número de iteración.

El algoritmo del método para encontrar un mínimo, se define como sigue:

Paso 1.

Se indica el intervalo de incertidumbre. $[a_k, b_k]$. $K = 1$. Se determina un $\xi > 0$.

Paso 2.

Se indica el número de iteraciones que se desea (n). El error que se obtendrá con el método se calcula con la fórmula: $1/F_n$, donde F_n es el término n de la serie de Fibonacci. Por lo general se calcula un máximo de 20 iteraciones, ya que $1/F_{20} = 1/10946 = 0.0000914$.

Paso 3.

Se calcula λ_k y μ_k.

Paso 4.

Se evaluan $f(\lambda_k)$ y $f(\mu_k)$.

Paso 5.

Mientras $k + 1 < n$ hacer lo siguiente:

Si $f(\lambda_k) \leq f(\mu_k)$ entonces

Se realiza $a_{k+1} = a_k$, $b_{k+1} = \mu_k$ y $\mu_{k+1} = \lambda_k$. Se realiza $k = k + 1$.

Se calcula λ_k. Se evalúa $f(\lambda_k)$.

En caso contrario

Se realiza $a_{k+1} = \lambda_k$, $b_{k+1} = b_k$ y $\lambda_{k+1} = \mu_k$. Se realiza $k = k + 1$.

Se calcula μ_k.

Se evalúa $f(\mu_k)$.

Regresar al inicio de este ciclo, para comprobar si la condición sigue siendo verdadera.

Paso 6.

Se realiza: $\lambda_n = \lambda_{n-1}$ y $\mu_n = \mu_{n-1} + \xi$.

Si $f(\lambda_n) \leq f(\mu_n)$ entonces

$a_n = a_{n-1}$ y $b_n = \lambda_n$

En otro caso

$a_n = \lambda_n$ y $b_n = b_{n-1}$.

El algoritmo termina.

En cada iteración μ_k contiene el valor mínimo. La solución óptima se encuentra en el intervalo $[a_n , b_n]$. Si lo que se desea es encontrar un punto máximo en el Paso 5 y Paso 7 se sustituye $\leq$ por $\geq$.

Si la función tiene más de un óptimo el método solo localizará alguno de ellos. Este método se puede utilizar como subrutina de búsqueda para otros algoritmos de problemas no restringidos multivariables, sean estos diferenciables o no.

4.2 Método de Newton-Raphson

Lo que realiza este método es: localizar los puntos que hacen que la primera derivada de la función sea igual a cero; porque un punto X^* es óptimo si $f'(X^*) = 0$ (Prawda [25]). El método se utiliza para funciones de una variable, continuas y obviamente diferenciables.

El algoritmo de este método se puede definir como sigue:

Paso 1.

Se da un valor inicial X^k.

Paso 2.

Se calcula:

$$X^{k+1} = X^k - \frac{f'(X^k)}{f''(X^k)}$$

$$donde\ f''(X^k) \neq 0$$

es decir que la primera y segunda derivada se evaluarán en un mismo punto X^k.

Paso 3.

El algoritmo termina cuando se realizaron un número de iteraciones anteriormente fijado o se ha llegado a un margen de error también preestablecido. Este se obtiene: $\xi = |X^{k+1} - X^k|$. En caso de que ninguna de estas dos condiciones se haya cumplido volver al Paso 2.

En el caso de que en la función exista más un óptimo debe de utilizarse varias veces el método, ya que este hallará uno a la vez; el óptimo que se encuentre dependerá del valor que inicialmente se haya dado. No siempre este método encuentra los valores óptimos de una función, ya que en ocasiones puede divergir si no se da el valor inicial adecuado.

4.3 Método de Davidon-Fletcher-Powell

Este método es considerado como uno de los algoritmos más eficientes para problemas de funciones diferenciables y multivariables.

El algoritmo de este método se puede definir como:

Paso 1.

Se da una matriz D_k arbitraria. Esta debe ser simétrica[1], positiva definida[2], de n por n. Donde n es el número de variables que intervienen. Por lo general se utiliza la matriz identidad[3]. $K = 1$.

---

[1] Una matriz A es simétrica si $A_{ij} = A_{ji}$ (Grossman [11]).

[2] Una matriz A es positiva definida si cumple (Lang [19]):
(a) Todos sus eigen valores son > 0.
(b) Una forma cuadrática $F(X) = F(x_1, x_2, \ldots, x_n)$ asociada A debe ser ≥ 0 para todo $X \in N^n$ y $F(X) = 0$ si y sólo si $X = 0$.

[3] La matriz identidad es aquella en la en que el elemento a_{ij} es igual a 1 si $i = j$ y es igual a 0 si $i \neq j$.

Paso 2.

Se realiza el gradiente:

$$\nabla f(X) = \begin{pmatrix} \frac{\partial f(X)}{\partial x_1} \\ \cdot \\ \cdot \\ \cdot \\ \frac{\partial f(X)}{\partial x_n} \end{pmatrix}$$

Paso 3.

Se da un punto (vector) $\mathbf{X}^k$ inicial arbitrario.

Paso 4.

Se obtiene la dirección $\mathbf{s}_k$, definido como:

$$s_k = -D_k \nabla f(X^k)$$

$\mathbf{S}_k$ es un vector.

Paso 5.

Se obtiene un vector $\boldsymbol{\beta}_k$ minimizando la función de una sola variable:

$$g(\beta_k) = f(X^k + \beta_k s_k)$$

Para encontrar el mínimo, la función que resulta se deriva y se iguala a cero.

Paso 6.

Se obtiene un parámetro $\boldsymbol{\sigma}_k$, definido como:

$$\sigma_k = \beta_k s_k$$

Paso 7.

Se obtiene el siguiente punto X^{k+1}:

$$X^{k+1} = X^k + \sigma_k$$

Paso 8.

Se calcula la nueva matriz D^{k+1}:

$$D_{k+1} = D_k + A_k + B_k$$

donde

$$A_k = \frac{\sigma_k \, \sigma_k^t}{\sigma_k^t \, \Delta_k}$$

$$\Delta_k = \nabla f(X^{k+1}) - f(X^k)$$

y

$$B_k = \frac{-(D_k \, \Delta_k \, \Delta_k^t \, D_k)}{\Delta_k^t \, D^k \, \Delta_k}$$

Es conveniente hacer notar que en los numeradores el resultado son matrices. En los denominadores son escalares.

Ir al Paso 3.

El algoritmo termina en un número determinado de iteraciones.

La desventaja de este método es que, para ser implementado en una computadora, se deben hacer previamente algunas operaciones, como por ejemplo el gradiente.

5

Comparación con Cada uno de los Métodos

Los métodos que se explican en el capítulo anterior (Fibonnacci, Newton_Raphson, y Davidon_Fletcher_Powell) fueron programados en computadora para comparar cada uno de ellos con el Método Hill-Climbing. Hay que tener encuenta que la técnica de este último método para resolver problemas no lineales es muy diferente con las técnicas de los métodos con que se compara. Por esta razón lo parámetros a comparar únicamente fueron: exactitud, número de iteraciones y tiempo de ejecución de cada metodo. Además esta metodología para el análisis técnicas ha sido empleado por muchos autores entre ellos Powell (Powell [34]), con lo que puede concluirse que estos son parámetros muy importante a considerar.

Es importante hacer notar que la máquina que se utilizó tenía un procesador 386 a una velocidad de 20HMz, ya que era el equipo con el que se contaba.

En cada sección se compararán cada uno de los métodos del capítulo anterior con el método Hill-Climbing. Esto es para que sea más fácil ver los resultados.

5.1 Fibonacci VS. Hill-Climbing

EJEMPLO G

La función que se utilizó para los dos métodos fue:

$$Min\ Z = 3X^2 - 6X + 4$$

Para el método de Fibonacci, se usó un intervalo inicial de [0,100], un epsilon de 0.01 y 20 iteraciones. Las 20 iteraciones, dan un límite de error, contra el resultado exacto, de 0.0000 ($1/F_{20} = 1/10946 = 0.00009$). La figura 5.1 muestra los resultados obtenidos a lo largo del proceso. Para el valor de X se tomó λ_k ó μ_k que diera el menor resultado en para cada iteración.

ITERACION	X	Z	DIFERENCIA VS. ANTERIOR
1	38.1966	4,151.7615	———
2	23.6068	1,534.2018	2,617.5597
3	14.5898	555.0484	979.1534
4	9.0170	193.8165	361.2319
5	5.5728	63.7318	130.0847
6	3.4442	18.9221	44.8097
7	2.1286	4.8214	14.1007
8	1.3155	1.2987	3.5227
9	0.8131	1.1048	0.1939
10	0.8131	1.1048	0.0000
11	1.0049	1.0001	0.1047
12	1.0049	1.0001	0.0000
13	1.0049	1.0001	0.0000
14	1.0049	1.0001	0.0000
15	1.0049	1.0001	0.0000
16	1.0049	1.0001	0.0000
17	0.9958	1.0001	0.0000
18	0.9958	1.0001	0.0000
19	0.9958	1.0001	0.0000
20	1.0008	1.0000	0.0001

Figura 5.1 Resultados obtenidos por el método de Fibonacci con el ejemplo G.

El tiempo en que se realizó el algoritmo fue: 00:00:00:00 (hora:min:seg:seg100). Es decir, el tiempo de ejecución fue de menos de una centésima de segundo. Y el resultado fue de X = 1.0000 y Z = 1.0000

Para el método de Hill-Climbing, las condiciones fueron las siguientes: n = 3 (número de valores iniciales en cada iteración), m = 8 (número de duplicación para cada valor inicial). Las condiciones para detener el algoritmo fueron: epsilon = 0.005 y 100 iteraciones.

Los valores iniciales fueron:

1) X = 100 con Z = 29,404

2) X = 101 con Z = 30,001

3) X = 102 con Z = 30,604

En la Figura 5.2 aparecen los resultados (sólo se presenta el mejor resultado en cada

iteración).

ITERACION	X	Z	DIFERENCIA VS. ANTERIOR
1	19.2667	1,002.0206	———
2	16.4554	717.6061	284.4145
3	15.5372	634.9867	82.6194
4	14.9564	585.3439	49.6428
5	14.7121	565.0645	20.2794
6	-3.0625	50.5109	514.5536
7	-1.8364	25.1352	25.3757
8	-0.0008	4.0050	21.1302
9	1.0934	1.0262	2.9788
10	0.9907	1.0003	0.0259

Figura 5.2 Resultados obtenidos por el método de Hill-Climbing con el ejemplo G.

El tiempo de ejecución del algoritmo fue: 00:00:00:73 con X = 0.9907 y Z = 1.0003

Puede observarse en la tabla de comparación (Figura 5.3) el número de iteraciones para el método de Fibonacci fue mayor pero en cambio el resulado de Z con Hill-Climbing fue un poco menos exacto en un 0.0003 aunque este es un resultado bastante aceptable. La desventaja de Fibonacci con respecto a Hill-Climbing es que hay que dar el rango, además de que es solamente para funciones continuas. Pero la ventaja de Fibonacci es que puede determinarse el límite de error. El tiempo requerido para Hill-Climbing fue mayor (casi un segundo).

	Fibonacci	Hill-Climbing
Condiciones iniciales	rango [0,100] epsilon: 0.001 iteraciones: 20	valores iniciales de X: 100,101,102 epsilon: 0.005 iteraciones: 100
Iteraciones	20	10
Tiempo*	0:00:00:00	0:00:00:99
Resultado	X = 1.0008 Z = 1.0000	X = 0.9907 Z = 1.0003

* (horas:minutos:segundos:centésimas de segundo)

Figura 5.3 Tabla comparativa entre Fibonacci y Hill-Climbing. G.

5.2 Newton-Raphson V.S. Hill-Climbing

EJEMPLO H

Para los dos métodos el problema que se utilizó fue:

$$Max\ Z = 12X - 3X^4 - 2X^6$$

Con el método de Newton-Rapson, se utilizó un valor inicial de 100 y epsilon de 0.005. En este método epsilon es un valor que detiene el algoritmo. Si $|X^n - X^{n-1}| <=$ epsilon el algoritmo termina.

Puede verse el resultado de todo el proceso en la Figura 5.4.

ITERACION	X	Z	DIFERENCIA VS. ANTERIOR
1	100.0000	-2,000,299,998,800.0000	———
2	63.9984	-524,379,419,600.0000	-1,475,920,579,200.0000
3	51.1974	-137,468,150,970.0000	-386,911,268,630.0000
4	40.9564	-36,038,593,514.0000	-101,429,557,456.0000
5	32.7632	-9,448,178,206.8000	-26,590,415,307.2000
6	26.2081	-2,477,142,204.6000	-6,971,036,002.2000
7	20.9634	-649,514,790.7600	-1,827,627,413.8400
8	16.7669	-170,326,394.4800	-479,188,396.2800
9	13.4088	-44,674,506.2680	-125,651,888.2120
10	10.7211	-11,721,096.1190	-32,953,410.1490
11	8.5695	-3,076,630.1309	-8,644,465.9881
12	6.8463	-808,122.2513	-2,268,507.8796
13	5.4656	-212,468.8450	-595,653.4063
14	4.3584	-55,929.2309	-156,539.6141
15	3.4694	-14,738.2655	-41,190.9654
16	2.7549	-3,881.1339	-10,857.1316
17	2.1802	-1,014.0626	-2,867.0713
18	1.7195	-256.4270	-757.6356
19	1.3559	-57.2778	-199.1492
20	1.0849	-6.2979	-50.9799
21	0.9147	5.6025	-11.9004
22	0.8472	7.7050	-2.1025
23	0.8378	7.8814	-0.1764
24	0.8376	7.8839	-0.0025

Figura 5.4 Resultados obtenidos por el método de Newton-Raphson con el ejemplo H.

El tiempo de ejecución fue: 00:00:00:27, con una solución de X = 0.8376 y Z = 7.8839.

Para el método Hill-Climbing se usaron las siguientes condiciones: n=3, m=8, epsilon=0.005 y 100 iteraciones.

Los valores iniciales:

1) X = 100 con Z = -2,000,299,998,800

2) X = 100 con Z = -2,132,352,481,193

3) X = 100 con Z = -2,252,649,566,952

En la siguiente Figura 5.5 se muestran los resultados para cada iteración.

ITERACION	X	Z	DIFERENCIA VS. ANTERIOR
1	99.4779	-1,938,448,915,172.0000	———
2	0.0979	1.1750	-1,938,448,915,173.1750
3	0.9947	7.0622	-5.8872
4	0.8241	7.8790	-0.8168
5	0.8241	7.8790	0.0000
6	0.8474	7.8813	-0.0023
7	0.8349	7.8837	-0.0024

Figura 5.5 Resultados obtenidos por el método de Hill-Climbing con el ejemplo H.

El tiempo que se obtuvo de ejecución fue: 00:00:01:05, con una solución de X = 0.8349 con Z = 7.8837.

Con la tabla de comparación de estos dos últimos métodos (Figura 5.6) se muestra que la diferencia contra el resultado de Hill-Climbing es de 0.0002. Pero hay que tomar en cuenta que el número de iteraciones es menor. La desventaja del método Newton-Raphson es que utiliza derivadas, lo que lo hace más complejo para programarse en computadora, además de no ser muy fácil de entender para personas de otras áreas donde las matemáticas se utilizan menos. Otra desventaja del método de Newton-Rapshon, es cuando el punto de parida esta cerca de un punto de inflexión, los resultados convergen a ese punto y no logran evadirlo para llegar al óptimo. En cuanto a el tiempo, el método Hill-Climbing utilizó tres veces más tiempo.

	Newton-Raphson	Hill-Climbing
Condiciones iniciales	valor inicial de X: 100 epsilon: 0.005	valores iniciales de X: 100,101,102 epsilon: 0.005
Iteraciones	24	7
Tiempo*	0:00:00:27	0:00:01:05
Resultado	X = 0.8376 Z = 7.8839	X = 0.8349 Z = 7.8837

* (horas:minutos:segundos:centésimas de segundo)

Figura 5.6 Comparación entre los dos métodos.

5.3 Davidon-Fletcher-Powell V.S. Hill-Climbing

EJEMPLO I

El problema para los dos métodos fue:

$$Min\ Z = 2\ X_1^2 + X_2^2 + 3\ X_3^2$$

Con el método de D-F-P y valores iniciales de: $X_1 = 36$, $X_2 = 46$ y $X_3 = 20$. En la Figura 5.7 puede verse los resultados.

ITERACION	X_1	X_2	X_3	Z	DIFERENCIA VS. ANTERIOR
1	2.2945	24.4659	(8.0880)	805.3544	———
2	(4.8682)	7.6199	1.9473	116.8379	688.5165
3	(0.0000)	0.0000	(0.0000)	0.0000	116.8379
4	0.0000	0.0000	0.0000	0.0000	0.0000

Figura 5.7 Resultados obtenidos por el método de D-F-P con el ejemplo I.

El tiempo de proceso fue: 00:00:00:00 con $X_1 = 0.0000$, $X_2 = 0.0000$, $X_3 = 0.0000$ y $Z = 0.0000$

Para el método Hill-Climbing se utilizaron las siguientes condiciones: $n=3, m=8$, epsilon = 0.005 y 100 iteraciones.

Valores iniciales:

1) $X_1 = 36$, $X_2 = 46$ y $X_3 = 20$ con $Z = 5{,}908$

2) $X_1 = 37$, $X_2 = 47$ y $X_3 = 21$ con $Z = 6{,}270$

3) $X_1 = 38$, $X_2 = 48$ y $X_3 = 22$ con $Z = 6{,}644$

En la siguiente tabla (Figura 5.8) se muestran los mejores valores obtenidos en cada iteración.

El tiempo de proceso fue: 00:00:08:38 con $X_1 = -0.0110$, $X_2 = -0.003$, $X_3 = -0.0073$ y $Z = 0.0004$

ITERACION	X_1	X_2	X_3	Z	DIFERENCIA VS. ANTERIOR
1	34.8986	46.0623	19.8901	5,744.4116	———
2	39.2359	22.3941	12.9959	4,087.0920	1,657.3196
3	25.6387	8.5642	18.5858	2,424.3297	1,662.7623
4	25.6252	8.2133	17.653	2,315.6526	108.6771
5	24.6173	7.1259	16.7594	2,105.4358	210.2168
6	9.9912	6.859	13.0323	756.2125	1,349.2233
7	9.3533	-2.2649	12.9757	685.2005	71.0120
8	9.3192	-2.2052	12.9347	680.4758	4.7247
9	7.3455	-1.1732	10.2819	426.4418	254.0340
10	-0.2530	1.3257	5.376	88.5900	337.8518
11	-0.1378	1.1419	-0.7955	3.2403	85.3497
12	0.0277	0.1093	0.3217	0.3240	2.9163
13	0.0277	0.1093	0.3217	0.3240	0.0000
14	0.0277	0.1093	0.3217	0.3240	0.0000
15	0.0247	0.1125	0.3050	0.2929	0.0311
16	0.0029	-0.2316	0.1131	0.0920	0.2009
17	0.0029	-0.2316	0.1131	0.0920	0.0000
18	0.1561	0.1046	-0.0743	0.0762	0.0158
19	0.1561	0.1046	-0.0743	0.0762	0.0000
20	0.1561	0.1046	-0.0743	0.0762	0.0000
21	0.0233	-0.1592	-0.0746	0.0431	0.0331
22	-0.0317	0.1046	0.0022	0.0130	0.0301
23	-0.0317	0.1046	0.0022	0.0130	0.0000
24	-0.0317	0.1046	0.0022	0.0130	0.0000
25	-0.0317	0.1046	0.0022	0.0130	0.0000
26	-0.0317	0.1046	0.0022	0.0130	0.0000
27	-0.0317	0.1046	0.0022	0.0130	0.0000
28	-0.0447	0.0234	0.0158	0.0053	0.0077
29	-0.0213	0.0342	0.0113	0.0025	0.0028
30	-0.0213	0.0342	0.0113	0.0025	0.0000
31	-0.0213	0.0342	0.0113	0.0025	0.0000
32	-0.0213	0.0342	0.0113	0.0025	0.0000
33	-0.0110	-0.0033	-0.0073	0.0004	0.0021

Figura 5.8 Resultados obtenidos por el método Hill-Climbing con el ejemplo I.

Puede verse en la tabla de la Figura 5.9 para Hill-Climbing la diferencia contra el resultado fue de 0.004, el tiempo de ejecución fue mucho mayor, también el número de iteracionespero el método de D-F-P tiene un grado de complejidad mayor ya que utiliza operaciones con vectores, matrices y derivadas.

	D-F-P			Hill-Climbing		
Condiciones iniciales	valores iniciales:			valores iniciales:		
	X 1	X 2	X 3	X 1	X 2	X 3
	36	46	20	36	46	20
				37	47	21
				38	48	22
	epsilon: 0.000			epsilon: 0.005		
Iteraciones	4			33		
Tiempo*	0:00:00:00			0:00:08:38		
Resultado	X 1	X 2	X 3	X 1	X 2	X 3
	0.0000	0.0000	0.0000	(0.0110)	(0.0033)	(0.0073)
	Z = 0.0000			Z = 0.0004		

* (horas:minutos:segundos:centésimas de segundo)

Figura 5.9 Tabla de comparación.

Observaciones

Comparando los resultados del Método Hill-Climbing con los otros métodos sus resultados son menos exactos, el tiempo de proceso es mayor, tuvo más iteraciones; pero este método es muy simple y lo más importante es que se utilizó el mismo algoritmo para resolver diferentes tipos de funciones.

6

Aplicación del Método Hill-Climbing a un Problema Real

Uno de los principales problemas a los que se enfrenta la administración de un hospital es la asignación de personal. En algunas ocasiones el destinar un número equivocado de personal puede representar un alto costo. El problema que se presenta en este capítulo es: definir un patrón de asignación de enfermeras en los cuatro diferentes turnos de un hospital. En primera instancia se describe el problema detalladamente y se definen las variables de decisión. Después, la función objetivo (función no lineal) y las restricciones se exponen. Por último este problema se resuelve por el método Hill-Climbing. Además se comparan los resultados con otros obtenidos por otro método. El ejemplo que aquí se presenta es solo uno de los muchos que hay en donde se presentan funciones no lineales y donde puede aplicarse el método Hill-Climbing. Es importante considerar que como uno de los objetivos de la presente tesis es aplicar el método Hill-Climbing a funciones no lineales era necesario presentar un problema que ya hubiera sido resuelto con anterioridad y así poder comparar los resultados.

6.1 Descripción del Problema

El costo del personal representa un gasto importante para cualquier empresa. Y si esta consta principalmente de servicios, mucha mayor relevancia tendrá el ser eficientes con la menor cantidad de personal. En un hospital, el contratar y asignar personal demás generara costos adicionales y si al contrario hay escasez se presentan problemas con los enfermos y en ocasiones estos pueden ser muy graves. En el caso que se presenta en seguida, la cantidad de enfermeras esta en función de una mejor atención a los pacientes.

En este capítulo se plantea y resuelve un modelo matemático que indica el patrón de asignación de personal (enfermeras solamente) por turno y por área para un periódo de tiempo. El modelo (Warner y Prawda [32], desarrollado para el hospital Touro de Nueva Orleans, E.U) debe tener las siguientes condiciones:

(a) especificar el número de enfermeras de cada categoría (Enfermeras Tituladas, Pasantes de Enfermería, Auxiliares de Enfermería y Afanadoras (es)), que se deben destinarse por turno a cada area,

(b) cumplir con las restricciones de recursos humanos,

(c) permitir la substitución de un tipo de personal por otro,

(d) satisfacer la demanda de servicios y

(e) minimizar el costo del excedente de personal para un período de tiempo.

Este tipo de problemas ya ha sido resuelto anteriormente con enfoques diferentes. Análisis Markoviano (Balintfy [1], Kolesar [17], Thomas [31]), o como proceso de líneas de espera (Singer [27]).

6.2 Descripción de la Función Objetivo

Sea $\mathbf{T}=\{1,2,..,T\}$ el conjunto que denota los turnos de trabajo (de 8 horas cada uno de ellos) establecidos para el personal de enfermería, en un hospital común este número no excede a 4. Sea $\mathbf{I}=\{1,2,...,I\}$ el conjunto que indica el número de pabellones del hospital a los cuales serán asignadas las enfermeras. Y $\mathbf{N}=\{1,2,...,N\}$ el conjunto de las diferentes categorías de enfermeras[1], en este caso solamente se están considerando 4.

Sea R_{tin} la demanda de servicio de enfermeras de categoría n, requeridas para el pabellón i, durante el turno t. En general, esta variable no es entera. Esto surge debido a que cada paciente requiere una parte del tiempo de una o varias enfermeras, por turno. Por lo que grupos de pacientes, requerirán parte del tiempo total de las enfermeras.

Sea X_{tin} el número de enfermeras de la categoría n a ser asignado al pabellón i, durante el turno t. Esta variable es entera porque representa el número de enfermeras en cada turno.

Para la construcción de la función objetivo

La función objetivo resulta:

[1] Cada categoría tiene especificado debidamente sus labores.

$$Min \quad Z = \sum_{t=1}^{T} \sum_{i=1}^{I} \sum_{n=1}^{N} (R_{tin} - X_{tin})^2 \qquad (1)$$

Esta función presenta dos desventajas:

1) No toma en cuenta los costos relativos entre las diferentes categorías de enfermeras, en los diferentes pabellones y los diferentes turnos. Para compensar esto se agrega el siguiente parámetro: sea W_{tin} un parámetro de ponderación que cuantifica la gravedad de la diferencia entre la demanda R_{tin} y la existencia X_{tin}. Se considera igual de grave la falta o el exceso de personal, esto se ve reflejando en que hay un coeficiente par en la función objetivo.

2) Ya que no se indica el límite de substitución entre las tareas del personal, se introducen los siguientes parámetros: sea Q_{timn} el parámetro que denota el coeficiente de substitución de un personal tipo m por otro n, en el área i, en el turno t. Este parámetro cuantifica que tanto puede substituir un tipo de personal a otro. Y por último, sea U_{tin} la cantidad de personal de tipo m que esta funcionando como personal tipo n, en el pabellón i durante el turno t. Esta variable no es entera. Por lo que se establece esta igualdad.

$$X_{tin} = \sum_{m=1}^{N} Q_{timn} \; U_{timn} \qquad (2)$$

Reuniendo los parámetros con la igualdad **(2)** e incorporandola a la ecuación **(1)** queda:

$$Min \; Z = \sum_{t=1}^{T} \sum_{i=1}^{I} \sum_{n=1}^{N} W_{tin}(R_{tin} - \sum_{m=1}^{N} Q_{timn} U_{timn})^2 \qquad (3)$$

6.3 Descripción de las Restricciones

El tiempo de enfermera categoría n funcionando como alguna de categoría p, en el pabellón i, durante el turno t ($p \in \mathbf{N}$, incluyendo $p = n$), no puede exceder la cantidad de enferemeras categaría n asignado a este pabellón en ese turno. Lo que da esta restricción:

$$\sum_{p=1}^{P} U_{tinp} \leq X_{tin} \quad \forall\, t \in T,\ i \in I,\ n \in N \tag{4}$$

El total de enfermeras de categoría n de todos los turnos de todos los pabellones no puede exceder la capacidad de recursos B_n del hospital (variable conocida).

$$\sum_{t=1}^{T} \sum_{i=1}^{I} X_{tin} \leq B_n \quad \forall\, n \in N \tag{5}$$

A_{tin} es porcentaje mínimo de servicio R_{tin} permitido por ley que debe proporcionarse para no generar sanciones.

$$X_{tin} \geq A_{tin}\ R_{tin} \quad \forall\, t \in T,\ i \in I,\ n \in N \tag{6}$$

Sabiendo cual es el requerimiento de personal en un pabellón y tuno de terminado, es deseable establecer un límite superior.

$$X_{tin} \leq R_{tin} + e_{tin} \quad \forall\, t \in T,\ i \in I,\ n \in N \tag{7}$$

donde e_{tin} (≥ 0) es un escalar.

Además se necesita que:

$$X_{tin} \quad entero \quad \forall\, t \in T,\ i \in I,\ n \in N \tag{8}$$

y

$$U_{timn} \geq 0 \quad \forall\, t \in T,\ i \in I,\ m, n \in N \tag{9}$$

6.4 Modelo Resultante

Reuniendo la función objetivo (3), con las restricciones (4), (5), (6), (7), (8) y (9) obtenemos el siguiente modelo que la forma de un problema de programación cuadrática entera-mixta.

$$Min\ Z = \sum_{t=1}^{T} \sum_{i=1}^{I} \sum_{n=1}^{N} W_{tin}(R_{tin} - \sum_{m=1}^{N} Q_{timn} U_{timn})^2 \tag{10}$$

sujeto a:

$$\sum_{p=1}^{N} U_{tinp} - X_{tin} \leq 0 \quad \forall\, t \in T,\ i \in I,\ n \in N$$

$$X_{tin} \geq A_{tin}\ R_{tin} \quad \forall\, t \in T,\ i \in I,\ n \in N$$

$$\sum_{t=1}^{T} \sum_{i=1}^{I} X_{tin} \leq B_n \quad \forall\, n \in N$$

$$X_{tin} \leq R_{tin} + e_{tin} \quad \forall\, t \in T,\ i \in I,\ n \in N$$

$$X_{tin} \quad entero \quad \forall\, t \in T,\ i \in I,\ n \in N$$

$$U_{tin} \geq 0 \quad \forall\, t \in T,\ i \in I,\ m, n \in N$$

El problema tiene (T·I·N) variables enteras, (T·I·N^2) variables continuas y (3·T·I·N + N) restricciones. Teniendo un período de 4 días (T = 12), 10 pabellones (I = 10) y 4 categorías de enfermeras (N = 4), da un total de 2400 variables y 1444 restricciones.

Pero este problema es posible descomponerlo en (T) subproblemas mas pequeños (fijando t, t$\forall \in$T);es decir para un solo turno. Por lo que el problema se puede escribir de la forma:

$$Min \quad Z = Min \quad \sum_{t=1}^{T} z_t \tag{11}$$

sujeto a:

$$\sum_{t=1}^{T} X_{tin} \quad \leq B_n \quad \forall \; i \in I, \quad n \in N$$

donde

$$Min \; z_t = \sum_{i=1}^{I} \sum_{n=1}^{N} W_{tin}(R_{tin} - \sum_{m=1}^{N} Q_{timn} U_{tinm})^2 \tag{12}$$

sujeto a:

$$\sum_{p=1}^{N} U_{tinp} \quad - X_{tin} \quad \leq 0 \quad \forall \; i \in I, \quad n \in N$$

$$A_{tin} \; R_{tin} \quad \leq X_{tin} \quad \leq R_{tin} \quad + e_{tin} \quad \forall \; i \in I, \quad n \in N$$

$$X_{tin} \quad entero \quad , \; U_{tin} \quad \geq 0 \quad \forall \; i \in I, \quad m, n \in N$$

6.5 Análisis

Para resolver el modelo (12) sólo se tomó en cuenta: 1 turno, 6 pabellón (A, B, C, D, E, F) y las cuatro categorías de enfermeras: Enfermeras Tituladas (ET), Enfermeras Pasantes (EP), Auxiliares de Enfermería (AE) y AFanadoras (AF). Con lo que el conjunto de turnos queda: T = {1}; el conjunto de pabellones queda: I = {1..6} y el conjunto de categorías de enfermera queda: N = {1..4}. Por lo que solamente hay 76 restricciones resultantes.

Los parámetros para Q_{timn} aparecen en la Figura 6.1, con lo que podemos ver que una Enfermera Titulada puede sustituir al 100% (1.0) a cualquier otra categoria, en cambio una Auxiliar de enfermería solamente puede cubrir a una Enferemera Titulada en un 20% (0.2).

Qm1mn	ET	EP	AE	AF
ET	1.0	1.0	1.0	1.0
EP	0.7	1.0	1.0	1.0
AE	0.2	0.4	1.0	0.5
AF	0.2	0.4	0.5	1.0

Figura 6.1 Parámetros de la variable Q.

Los demás parámetros (W_{tin}, A_{tin} y e_{tin}) están en la Figura 6.2. En esta tabla puede verse que la deficiencia en atención (W_{tin}) de una Enfermera Titulada afecta en unidades de 1000, más sin embargo la ausencia de una AFanadora afecta en unidades de 100. También debe fijarse que los mínimos requerimientos de atención (A_{tin}) para las diferentes categorías de enfermeras son del 70% (0.7) y 60% (0.6). Por último el escalar que da el límite superior para establecer la cantidad de personal no debe rebasar una unidad con respecto a la demanda (ver ecuación (7)).

La demanda de atención de enfermeras (R_{tin}) para cada pabellón aparece en la tabla de la Figura 6.3. Hay que recordar que todos los datos anteriores han sido tomados de Prawda [25].

	ET	EP	AE	AF
W tin	1000	7000	100	100
A tin	0.7	07	0.7	0.6
e tin	1	1	1	1

Figura 6.2 Parámetros de las variables W, A y e.

Pabellón	ET	EP	AE	AF
A	1.5	3.5	6.0	2.3
B	3.8	1.5	6.4	2.5
C	0.9	2.2	2.3	1.3
D	2.0	0.5	1.7	1.4
E	2.3	0.3	1.7	1.4
F	2.1	0.2	1.5	1.2

Figura 6.3 Parámetros para la variable R.

Pabellón	ET	EP	AE	AF
A	2	3	6	2
B	3	2	7	2
C	1	2	3	1
D	2	0	2	2
E	2	1	2	1
F	2	0	2	1

Figura 6.4 Resultados obtenidos en Prawda [25].

Ahora, resolviendo el modelo por medio del método de programación cuadrática con los datos de las tablas anteriores, se obtienen los resultados de la Figura 6.4. Estos a su vez se sustituyen en la función objetivo del modelo **(12)** lo que da por resultado **Min z_1 = 2,151** (obviamente aplicando la igualdad **(2)**). Estos resultados se presentan para poderlos comparar con los que se obtendrán por el Método Hill-Climbing.

Para la aplicación del método Hill-Climbing se usaron 3 matrices de valores iniciales (n = 3), 8 duplicaciones para cada matriz (m = 8) y como parámetros para detener el algoritmo: 100 iteraciones y epsilon = 0.005. Además de los mismos datos de las tablas de las Figuras 6.1, 6.2 y 6.3.

En la Figura 6.5 puede verse la matriz de los datos iniciales que da el menor resultado en la función objetivo (**Min z_1 = 2,511**). La Figura 6.6 presenta los resultados de z_1 de todo el proceso, hay que reiterar que las iteraciones que no aparecen es porque el resultado no cambió. Estos resultados cumplen con todas las restricciones del modelo. Para simplificar sólo se muestra el mejor resultado en cada iteración (hay que recordar que el método hill-Climbing da tres resultados en cada iteración).

Pabellón	ET	EP	AE	AF
A	2	3	6	2
B	4	2	5	3
C	1	2	2	1
D	2	3	2	2
E	2	1	2	1
F	2	1	2	1

Figura 6.5 Datos iniciales.

Puede observarse en la Figura 6.6 la matriz de datos finales, lo cual sustituyendolos en la función objetivo del modelo **(12)** nos da **Min z_1 = 1,891** que es un resultado menor que el obtenido por Prawda [25].

ITERACION	Z_i
0	2511
1	2171
2	2171
3	2151
4	2091
6	2031
7	2011
12	1991
13	1951
15	1931
28	1891
100	1891

Figura 6.6 Resultados obtenidos por Hill-Climbing durante 100 iteraciones.

Pabellón	ET	EP	EA	AF
A	2	3	6	2
B	4	2	6	3
C	1	2	6	1
D	2	1	2	1
E	2	1	2	1
F	2	1	2	1

Figura 6.7 Resultado final.

Análisis de sensibilidad.

Con este análisis puede comprobarse que aumentado o restando una enfermera para cada categoría en los diferentes pabellones, el resultado es mayor o no cumple con todas las restricciones. La columna derecha abajo de cada categoría, representa el número de restricciones que cumple, siendo 76 el número máximo. Lo que se concluye con esto es que el resultado fue el mejor posible encontrado.

ANALISIS DE SENSIBILIDAD +1

Pabellón	ET		EP		AE		AF	
A	3891	75	1891	76	1991	76	1931	76
B	3291	75	3291	75	1911	76	2091	75
C	3091	75	2311	76	1931	76	1931	76
D	2891	76	3291	75	2051	75	1911	76
E	2291	76	3571	75	2051	75	1911	76
F	2691	76	3711	75	2091	75	1951	76

ANALISIS DE SENSIBILIDAD -1

Pabellón	ET		EP		AE		AF	
A	1891	74	3291	74	1991	76	2051	74
B	2491	75	1891	74	2071	76	1891	75
C	2691	74	2871	74	2051	74	2051	74
D	2891	74	1891	74	1931	74	2071	74
E	3491	74	1611	74	1931	75	2071	74
F	3091	74	1471	74	1891	75	2031	74

Figura 6.8

Conclusiones

En la Investigación de Operaciones existen múltiples técnicas para resolver problemas con funciones matemáticas, no siempre estas técnicas son simples, y en especial las que pertenecen a la Programación No Lineal. Para utilizar estas técnicas, siempre es necesario tener conocimientos matemáticos. En este trabajo se presentó un algoritmo realmente sencillo que puede aplicarse a funciones no lineales de una o varias variables, restringidas y no restringidas: el **Método Hill_Climbing.** Este método llama la atención ya que la idea básica para desarrollarlo se tomó de la evolución biológica. El algoritmo es muy sencillo y lógico. Esta no es la primera vez que se reúnen la biología, la computación y la matemáticas algunos otros autores ya lo han realizado con anterioridad dando resultados sorprendentes, contribuyendo unas ideas con otras para un mutuo avance.

Los resultados que se obtuvieron de este trabajo fueron:

1) A pesar de que en la realidad los problemas no lineales se presentan con mucha frecuencia el uso de la Optimización no Lineal no es tanto, esto debido a que para utilizar los algoritmos de esta área se necesitan buenos conocimientos de matemáticas (derivadas, gradianes, hessianos, matrices, etc.) Lo que hace que esto sea extraño y demasiado abstracto para las personas en general, lo que provoca un menor uso de estas técnicas aunque con ellas se obtengan resultados muy buenos. Lo importante no es tener muchos algorimtos que resuelvan estos tipos de problemas sino que algoritmos que sean sencillos de aplicar y que puedan ser entendidos por cualquier persona.

2) El algoritmo del método Hill-Climbing, por su estructura, es fácil de comprender y aplicar; puede ser entendido por un número mayor de personas aún sin conocimientos de matemáticas avanzadas. Es más este algoritmo puede aplicarse solamente con calculadora

pudiéndose prescindir de una computadora.

3) La convergencia hacia el punto óptimo es muy rápida y aunque se den datos muy alejados en muy pocas iteraciones se obtiene un muy buen resultado.

4) En muy pocas iteraciones puede verse si el método esta convergiendo o divergiendo.

5) En caso de que haya más de un óptimo y se realiza todo el proceso más de una vez es muy probable que se identifiquen los otros puntos óptimos.

6) Por su estructura no se detiene en los puntos de inflexión como ocurre con algunos métodos.

7) En conjuntos solución no convexos encuentra también la solución.

8) El método es eficiente en un 99% para funciones de una variable, de varias variables, además para problemas restringidos y no restringidos.

9) En comparación con otros métodos puede tener un mayor número de iteraciones, mayor tiempo, menor exactitud y mayor memoria en una computadora, pero la gran ventaja es que solamente se utiliza un solo algoritmo para los diferentes tipos de problemas.

10) Otra característica del método es que busca la solución en un conjunto de puntos y uno a partir de uno solo. Esto puede ayudar a encontrar en algunos caso mejores soluciones que con los métodos determinísticos.

11) El algoritmo del método Hill-Climbing también puede utilizarse para funciones enteras como en el caso de asignación de recursos humanos, inclusive puede obtenerse resultados menores.

12) Por su estructura sencilla puede ser muy bien utilizado por personas del área de las ciencias sociales. Inclusive el método puede usarse con conocimientos muy básicos aún con estudios de enseñanza media.

13) Hay que recalcar que es obvio que puede usarse con muy buenos resultados en la Optimización no Lineal, ya que estos problemas son mucho más sencilos que los que aquí se presentaron.

Por último, las áreas en las que se hallan problemas no lineales como los aquí presentados son múltiples. Estas áreas pueden ser el cálculo de estructuras, transmisión de señales, óptica, acústica, etc. Por lo que se puede deducir que el método Hill-Climbing puede ser motivo de futuras investigaciones dentro de cualquier área.

apéndice A

En este apéndice se presentan dos pruebas adicionales del Método Hill-Climbing, para aquellos que quisieran saber un poco más sobre las características del algoritmo.

Prueba 1

Como se recordará, cada valor inicial se duplica ocho veces para crear los nuevos valores, quedando un total de 24 nuevos valores. De estos se escogen tres que serán los valores iniciales de la siguiente iteración. Con el problema del ejemplo A se desarrollo esta prueba:

$$Min\ Z = X^2-6X+2$$

En este ejercicio se varió el número de duplicación con 5, 8 y 10. Se repitió el ejercicio diez veces para cada duplicación. En la figura B.1 puede verse la comparación con respecto al número de iteraciones que se necesitaron para llegar al término del algoritmo.

Las condiciones iniciales fueron: Epsilon = 0.005 y Iteraciones = 100.

	DUPLICACION: 5			DUPLICACION: 8			DUPLICACION: 10		
n	X_NUEVA	Z	It	X_NUEVA	Z	It	X_NUEVA	Z	It
1	2.9905	-6.9999	15	2.9972	-7.0000	9	2.9797	-6.9996	13
2	3.0071	-6.9999	12	2.9963	-7.0000	15	3.0048	-7.0000	8
3	2.9990	-7.0000	19	3.0110	-6.9999	12	2.9701	-6.9991	6
4	3.0308	-6.9990	20	3.0247	-6.9994	16	3.0136	-6.9998	12
5	2.9382	-6.9962	9	3.0244	-6.9994	11	3.0123	-6.9998	14
6	3.0001	-7.0000	17	3.0116	-6.9999	9	2.9993	-7.0000	8
7	2.9992	-7.0000	12	3.0161	-6.9997	6	3.0031	-7.0000	12
8	3.0236	-6.9994	21	2.9506	-6.9976	11	2.9968	-7.0000	13
9	3.0512	-6.9974	14	3.0237	-6.9994	15	3.0187	-6.9996	23
10	2.9717	-6.9992	9	3.0220	-6.9995	16	2.9988	-7.0000	7

Figura B.1 Prueba 1

Para el ejemplo con duplicación 5 la media en iteraciones para llegar al término del resultado fue de 13.8, con duplicación 8 la media fue de 12 y con duplicación 10 la media fue de 11.6. Esto nos da una idea de que si incrementamos el número de veces que se replican los valores iniciales el número de iteraciones disminuirá, pero hay que tomar en cuenta que para tres valores iniciales se obtiene 30 valores nuevos. Por esta razón el espacio de memoria que se necesita cada vez es mayor.

Prueba 2

Para comprobar la eficacia del método se realizaron 100 ensayos con el problema del ejemplo E:

Se escogió este problema por que el espacio solución no es una área convexa

(véase la figura 3.7 del capítulo 3). Solamente se esta comprando el resultado final en cada ensayo. Para cada ejercicio se tomó como condiciones iniciales epsilon:0.005 y Iteraciones:50. Es decir cuando las iteraciones lleguen a más de 50 o cuando la diferencia entre el primer y tercer resultado de los valores iniciales sea menor a 0.005, entonces el método termina.

n	X_NUEVA	Z	n	X_NUEVA	Z	n	X_NUEVA	Z	n	X_NUEVA	Z
1	7.6063	14.2179	26	7.6072	14.2264	51	7.6066	14.2208	76	7.6072	14.2265
2	7.6037	14.1938	27	7.5947	14.1114	52	7.6075	14.2286	77	7.6081	14.2349
3	7.6046	14.2021	28	7.6071	14.2249	53	7.6071	14.2258	78	7.6075	14.2294
4	7.6070	14.2245	29	7.6084	14.2370	54	7.6081	14.2350	79	7.6078	14.2316
5	7.6062	14.2175	30	7.6061	14.2164	55	7.6084	14.2377	80	7.6086	14.2391
6	7.6080	14.2339	31	7.6083	14.2366	56	7.6084	14.2374	81	7.6079	14.2323
7	7.6044	14.2006	32	7.6080	14.2338	57	7.6082	14.2355	82	7.6081	14.2347
8	7.6068	14.2226	33	7.6083	14.2362	58	7.6072	14.2261	83	7.6078	14.2318
9	7.6068	14.2389	34	7.6074	14.2279	59	7.6071	14.2258	84	7.6079	14.2324
10	7.6080	14.2336	35	7.6078	14.2321	60	7.6063	14.2181	85	7.6083	14.2367
11	7.6073	14.2276	36	7.6076	14.2297	**61**	**0.0001**	**1.9992**	86	7.6069	14.2233
12	7.6081	14.2348	37	7.6052	14.2079	62	7.6081	14.2345	87	7.6085	14.2384
13	7.6070	14.2247	38	7.6073	14.2275	63	7.6080	14.2334	88	7.6066	14.2210
14	7.6046	14.2024	39	7.6070	14.2246	64	7.6084	14.2372	89	7.6085	14.2385
15	7.6060	14.2152	40	7.6081	14.2345	65	7.6081	14.2348	90	7.6085	14.2385
16	7.6007	14.1665	41	7.6061	14.2158	66	7.6078	14.2317	91	7.6075	14.2294
17	7.6083	14.2363	42	7.6085	14.2379	67	7.6067	14.2217	92	7.6080	14.2333
18	7.6078	14.2320	43	7.6086	14.2388	68	7.6064	14.2186	93	7.6074	14.2282
19	7.6079	14.2325	44	7.6078	14.2316	69	7.6084	14.2371	94	7.6085	14.2384
20	7.6072	14.2260	45	7.6070	14.2247	70	7.6066	14.2208	95	7.6081	14.2343
21	7.6073	14.2271	46	7.5992	14.1526	71	7.6072	14.2259	96	7.6069	14.2236
22	7.6081	14.2350	47	7.6084	14.2377	72	7.6063	14.2181	97	7.6083	14.2363
23	7.6074	14.2280	48	7.6060	14.2151	73	7.6085	14.2387	98	7.6078	14.2318
24	7.6065	14.2197	49	7.6085	14.2378	74	7.6064	14.2190	99	7.6028	14.1856
25	7.6015	14.1738	50	7.6058	14.2135	75	7.6081	14.2342	100	7.6084	14.2373

Figura B.2 Prueba 2

Como se ve en la iteración 61, el algoritmo no pudo resolver el problema, pero siendo este caso ùnico entre 100 ensayos, entonces el error de este algoritmo es del 1%, lo que da un grado bastante bueno de confiabilidad.

apéndice B

Aquí se encuentran los programas, desarrollados en Pascal 5, de los capítulos 3, 5 y 6. Los programas del Método Hill_Climbing están transcritos en su totalidad. El lector encontrará que son muy similares entre sí; pero se realizó de esta manera para que fuera sencilla su comprensión, al mismo tiempo que facilitar su codificación a otro lenguaje.

PROGRAMA EJEMPLOA.PAS

$$Min\ Z = X^2-6X+2$$

```
program HILL_CLIMBING_UNIMODAL_UNA_VARIABLE;

{ ESTE PROGRAMA MINIMIZA LA FUNCION Z = X² - 6X + 2 POR EL METODO HILL_CLIMBING.
  ESTE METODO ES UNA ALTERNATIVA PARA RESOLVER FUNCIONES NO LINEALES. FUE
CREADO
  A PARTIR DE LA IDEA DE LA EVOLUCION BIOLOGICA.

  ANGELICA MENDOZA ANZALDO
  ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES "A C A T L A N"
  UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO}

{_____TIPOS, CONSTANTES Y VARIABLES DEL PROGRAMA PRINCIPAL_____}
```

PROGRAMA EJEMPLOA.PAS... continuación

```
uses
  {Libreria de Turbo Pascal}
  crt;

type
  reales = array[1..27] of real; {Este tipo es para los arreglos reales}
  enteros = array[1..10] of integer; {Este tipo es para los arreglos enteros}

var
  X_padre,          {Padres iniciales}
  X_hijo,           {Hijos de la siguiente generacion}
  Z_aux,            {Resultados de la nueva generacion}
  Z:reales;         {Resultados de la generacion inicial}
  duplica:enteros;  {Numero de duplicacion de cada padre}
  indice,i,j,k,     {indices}
  papas,            {Numero de padres iniciales para cada generacion}
  tipo,             {Tipo de mutacion}
  generacion,       {Numero total de la generacion}
  iteraciones:integer; {Numero limite de iteraciones}
  resta,            {Diferencia entre el primero y ultimo de los mejores
                     resultado de la nueva generacion}
  box,              {Valor aleatorio de una distribucion normal}
  epsilon,          {Limite de error}
  mutacion,         {mutacion para cada valor}
  rnd:real;         {valor aleatorio entre 0 y 1}
  respuesta:char;

{_____PROCEDIMIENTOS Y FUNCIONES_____}

procedure Presenta;
{Este es un procedimiento de presentacion. Crea un marco para la pantalla}
 var
  i:integer;
begin
  gotoxy(1,1);write(' ╔'); {Esquina superior izquerda}
  for i:=2 to 79 do
  begin
    gotoxy(i,1);write('═'); {Linea superior}
  end;
  gotoxy(80,1);write('╗'); {Esquina superior derecha}
  for i:=2 to 23 do
  begin
    gotoxy(80,i);write(' ‖ '); {Linea derecha}
  end;
  gotoxy(80,24);write('╝'); {Esquina inferior derecha}
  for i:=79 downto 2 do
  begin
    gotoxy(i,24);write('═'); {Linea inferior}
  end;
  gotoxy(1,24);write(' ╚'); {Esquina inferior izquierda}
  for i:=23 downto 2 do
  begin
    gotoxy(1,i);write(' ‖ '); {Linea izquierda}
  end;
  window(2,2,79,23); {Deja fijo el marco durante todo el programa}

  gotoxy(32,3);write('METODO HILL-CLIMBING');
  gotoxy(32,9);write('MIN Z = X² - 6X + 2');
```

PROGRAMA EJEMPLOA.PAS... continuación

```
  gotoxy(14,11);write('La solución óptima de la función es: X = 3 y Z = -7');
  gotoxy(28,19); write('Oprima RETURN para continuar');
  readln;
  clrscr;
end; {Presenta}

procedure Impresion(var arreglo1,arreglo2:reales;limite:integer);
{Este procedimiento imprime dos arreglos en forma de columna al mismo tiempo}
begin
  for i:=1 to limite do
  begin
     gotoxy(18,9+i);write(i,'   X =  ',arreglo1[i]:3:4);
     gotoxy(43,9+i);write('Z = ',arreglo2[i]:3:4);
  end;
  gotoxy(28,19); writeln('Oprima RETURN para continuar');
end {Impresion};

function Eleva(numero:real;potencia:integer):real;
{Esta funcion eleva un numero a una potencia dada}
var
  mult:real;
  k:integer;
begin
  mult:=1;
  for k:=1 to potencia do
    mult:=mult*numero;
  eleva:=mult;
end {Eleva};

function Objetivo(x:real): real;
{Evalua la funcion objetivo para un solo valor}
begin
  objetivo:=sqr(x)-6*x+2;
end {Objetivo};

procedure Ordena(var elemento,Z:reales;cuenta:integer);
{Se ordenan los valores de menor a mayor con respecto a su resultado}
var
  i,j,k:integer;
  aux,aux1:real;

begin
  for j:=1 to 3 do
   for i:=j+1 to cuenta do
    if (Z[i]<Z[j]) then
    begin
       aux:=Z[j];
       Z[j]:=Z[i];
         Z[i]:=aux;

       aux1:=elemento[j];
       elemento[j]:=elemento[i];
       elemento[i]:=aux1;
    end;{for's}
end {Ordena};

{_____PROGRAMA PRINCIPAL_____}
```

PROGRAMA EJEMPLOA.PAS... continuación

```
begin
  clrscr;
  Presenta;

  {Variables principales}
  papas: = 3; {Numero de valores iniciales}

  gotoxy(32,2);writeln(' VALORES INICIALES');
  gotoxy(20,4);write('¿Desea dar los valores iniciales ? (S/N) ');readln(respuesta);
  if upcase(respuesta) = 'N' then
  begin
    {Valores iniciales aleatorios}
    randomize;
    for i: = 1 to papas do
    begin
      delay(5);
      X_padre[i]: = (random(3000) + random) * eleva(-1,random(2));
    end
  end
  else
    {Valores iniciales por teclado}
    for i: = 1 to papas do
    begin
      gotoxy(29,5 + i);write('X ',i,' '); readln(X_padre[i]);
    end;

  {Cada valor es empleado en la funcion objetivo}
  for i: = 1 to papas do
  Z[i]: = objetivo(X_padre[i]);

  {Se ordenan los valores resultados de menor a mayor, por su resultado}
  Ordena(X_padre,Z,papas);

  {Se dan los valores de duplicacion para los valores iniciales}
  for i: = 1 to papas do
     duplica[i]: = 8;

  {Total de individuos por cada generacion}
  generacion: = 0;
  for i: = 1 to papas do
     generacion: = generacion + duplica[i];
  Impresion(X_padre,Z,papas);

  gotoxy(33,15);write('EPSILON ? '); readln(epsilon);{Se recomienda epsilon = 0.005}
  gotoxy(33,17);write('ITERACIONES ? '); readln(iteraciones);{Se recomienda iteraciones = 100}

  k: = 0; {Indica el numero de iteracion}

  {Se obtiene la diferencia entre los resultados de los valores iniciales (1ro. y 3er.)}
  {Ya que los valores se ordenaron anteriormente Z[1] tiene al mejor
  resultado y X_padre[1] contienen al mejor punto encotrado hasta el momento}
  resta: = abs(Z[papas]-Z[1]);

  {Si la diferencia es mayor a epsilon o el numero de iteraciones es menor a
  las indicadas, entonces el algoritmo continua}

while (resta > epsilon) and (k < iteraciones) do
begin
```

PROGRAMA EJEMPLOA.PAS... continuación

```
{Se incrementa el numero de la iteracion}
inc(k);
{Esta variable indica que tipo de mutacion rige para esta generacion}
tipo:=0;
randomize;
{Rnd es para elegir el tipo de mutacion}
repeat
   rnd:=random;
until rnd>0;

{Se eligen el tipo de mutacion}
if (rnd <= 0.33) then
  {Primera mutacion: entre 0 y 1}
  repeat
    mutacion:=random;
    tipo:=1;
  until mutacion>0
else
if (rnd > 0.33) and (rnd <= 0.66) then
  {Segunda mutacion: entre el 0 y el mejor encontrado}
  repeat
    mutacion:=random(trunc(abs(X_padre[1]))) + random;
    tipo:=2;
  until mutacion>0
else
  if (rnd > 0.66) then
  {Tercera mutacion: entre 0 y 10}
   repeat
    mutacion:=random(10) + random;
    tipo:=3;
  until mutacion>0;

clrscr;
{Impresion de parametros}
gotoxy(25,5);write('K        ',K);
gotoxy(25,7);write('RESTA    ',resta:3:4);
gotoxy(25,9);write('MUTACION ',mutacion:3:4);
gotoxy(25,11);write('TIPO     ',tipo);
gotoxy(25,13);write('RND      ',rnd:3:4);
gotoxy(28,19); write('Oprima RETURN para continuar');
readln;

{Se generan los hijos}
{A la vez que se generan se imprimen}
indice:=0;
for i:=1 to papas do
begin
    clrscr;
    gotoxy(9,5);writeln('No.  X_INICIAL  +  MUT*BOX  =  X_NUEVA          Z');
    gotoxy(9,6);writeln('-------------------------------------------------------------');
  for j:=1 to duplica[i] do
  begin
   indice:=indice+1;

   gotoxy(9,7+j);write(indice);
   {Impresion del padre que da origen al nuevo individuo}
   gotoxy(15,7+j);write(abs(X_padre[i]):3:4);
   if X_padre[i] < 0 then
```

PROGRAMA EJEMPLOA.PAS... continuación

```
    begin
      gotoxy(14,7+j);write('-');
    end;

    {Esta formula de Box y Muller [ ] que genera numeros aleatorios con una
     distribucion normal. Notese que cada valor es diferente para cada individuo}
    box:=sqrt(-2*ln(random))*sin(2*pi*random);

    gotoxy(29,7+j);write(abs(mutacion*box):3:4);
      if mutacion*box < 0 then
      begin
       gotoxy(28,7+j);write('-');
      end;

    {Nuevo individuo}
    X_hijo[indice]:=X_padre[i]+mutacion*box;

    gotoxy(43,7+j);write(abs(X_hijo[indice]):3:4);
    if X_hijo[indice] < 0 then
    begin
      {En caso de que los valores sean negativos}
      gotoxy(42,7+j);write('-');
    end;

    {Sustitucion del nuevo valor en la funcion objetivo}
    Z_aux[indice]:=objetivo(X_hijo[indice]);

    gotoxy(62,7+j);write(abs(Z_aux[indice]):3:4);
      if Z_aux[indice] < 0 then
      begin
       gotoxy(61,7+j);write('-');
      end;
  end;{j}
  gotoxy(28,19); write('Oprima RETURN para continuar');
  readln;
end;{i}

  {Para elegir los tres mejores valores de esta iteracion, se agregan los
  padres}
  X_hijo[25]:=X_padre[1];
  Z_aux[25]:=Z[1];
  X_hijo[26]:=X_padre[2];
  Z_aux[26]:=Z[2];
  X_hijo[27]:=X_padre[3];
  Z_aux[27]:=Z[3];

  {Se ordenan y los tres primeros lugares son los mejores tres valores
  encontrados hasta el momento}
  Ordena(X_hijo,Z_aux,generacion+3);

  {Los tres mejores valores son los padres de la siguiente generacion}
   for i:=1 to papas do
   begin
     X_padre[i]:=X_hijo[i];
     Z[i]:=Z_aux[i];
   end;

   {Impresion en pantalla de los tres mejores resultados encontrados}
```

PROGRAMA EJEMPLOA.PAS... continuación

```
    clrscr;
    gotoxy(18,6);writeln('Los mejores valores de la I T E R A C I O N ',k);
    Impresion(X_padre,Z,papas);
    readln;

    {Condicion para continuar el algoritmo}
    resta: =abs(Z[3]-Z[1]);
    {Si resta es menor a epsilon el algoritmo termina}

end; {while principal}
gotoxy(37,21);write('F I N ');
writeln;
repeat until keypressed;
window(0,0,25,80);
end

{_____}
{Si tiene algun comentario o duda puede escribir a:
Angelica Mendoza Anzaldo
Av. Tamaulipas 109-12 Col. Hipodromo Condesa
Mexico, D.F. C.P. 06170
Tel. 5-53-12-37}
{FIN}.
```

PROGRAMA EJEMPLOC.PAS

$$MAX\ Z = X_1+2X_3+X_2X_3-X_1^2-X_2^2-X_3^2$$

```
program HILL_CLIMBING_MULTIVARIABLE;

{_________TIPOS, CONSTANTES Y VARIABLES DEL PROGRAMA PRINCIPAL_________}
uses
  {Libreria de Turbo Pascal}
  crt;

type
  realmat= array[1..27,1..3] of real; {Este tipo es para las matrices reales}
  reales= array[1..27] of real; {Este tipo es para los arreglos reales}
  enteros=array[1..10] of integer; {Este tipo es para los arreglos enteros}

var
  X_padre,          {Padres iniciales}
  X_hijo: realmat;   {Hijos de la siguiente generacion}
  Z_aux,            {Resultados de la nueva generacion}
  Z,                {Resultados de la generacion anterior}
  mutacion:reales;   {Mutacion para cada variable de la funcion}
  duplica:enteros;   {Numero de duplicacion de cada padre}
  indice,i,j,k,k1,   {indices}
  papas,            {Numero de padres iniciales para cada generacion}
  generacion,        {Numero total de la generacion}
  iteraciones:integer;  {Numero limite de iteraciones}
  resta,            {Diferencia entre el primero y ultimo de los mejores
                    resultado de la nueva generacion}
  box,              {Valor aleatorio de una distribucion normal}
  epsilon,          {Limite de error}
  rnd:real;         {valor aleatorio entre 0 y 1}
  respuesta:char;

{_____________________PROCEDIMIENTOS Y FUNCIONES_____________________}

procedure Presenta;
{Este es un procedimiento de presentacion. Crea un marco para la pantalla}
 var
  i:integer;
begin
  gotoxy(1,1);write('╔'); {Esquina superior izquerda}
  for i:=2 to 79 do
  begin
    gotoxy(i,1);write('═'); {Linea superior}
  end;
  gotoxy(80,1);write('╗'); {Esquina superior derecha}
  for i:=2 to 23 do
  begin
    gotoxy(80,i);write('║'); {Linea derecha}
  end;
  gotoxy(80,24);write('╝'); {Esquina inferior derecha}
  for i:=79 downto 2 do
  begin
    gotoxy(i,24);write('═'); {Linea inferior}
  end;
```

PROGRAMA EJEMPLOC.PAS... continuación

```
  gotoxy(1,24);write(' ╚'); {Esquina inferior izquierda}
  for i:=23 downto 2 do
  begin
    gotoxy(1,i);write(' ‖ '); {Linea izquierda}
  end;
  window(2,2,79,23); {Deja fijo el marco durante todo el programa}

  gotoxy(32,3);write('METODO HILL-CLIMBING');
  gotoxy(20,9);write('MIN Z = X1 + 2X3 + X2X3 - X1*X1 - X2*X2 - X3*X3');
  gotoxy(5,11);write('La solución óptima de la función es: X1 = , X2 = , X3 = y Z = 0.0000');
  gotoxy(28,19); write('Oprima RETURN para continuar');
  readln;
  clrscr;
end; {Presenta}

procedure Impresion(var valor: realmat; var Z:reales;limite:integer);
{Este procedimiento imprime dos arreglos en forma de columna al mismo tiempo}
begin
  for i:=1 to limite do
  begin
     gotoxy(2,9+i);write('X',i,1,' = ');gotoxy(9,9+i);write(valor[i,1]:3:4);
     gotoxy(22,9+i);write('X',i,2,' = ');gotoxy(29,9+i);write(valor[i,2]:3:4);
     gotoxy(42,9+i);write('X',i,3,' = ');gotoxy(49,9+i);write(valor[i,3]:3:4);
     gotoxy(60,9+i);write('   Z = ',Z[i]:3:4);
  end;
  gotoxy(28,19); writeln('Oprima RETURN para continuar');
end {Impresion};

function Eleva(numero:real;potencia:integer):real;
{Esta funcion eleva un numero a una potencia dada}
var
  mult:real;
  k:integer;
begin
  mult:=1;
  for k:=1 to potencia do
    mult:=mult*numero;
  eleva:=mult;
end {Eleva};

function Objetivo(x1,x2,x3:real): real;
{Evalua la funcion objetivo para un solo valor}
begin
  objetivo:=x1+2*x3+x2*x3-sqr(x1)-sqr(x2)-sqr(x3);
end {Objetivo};

procedure Ordena(var elemento: realmat; var Z:reales;cuenta:integer);
{Se ordenan los valores de menor a mayor con respecto a su resultado}
var
  i,j,k:integer;
  aux:real;

begin
  for j:=1 to 3 do
   for i:=j+1 to cuenta do
    if (Z[i]>Z[j]) then
    begin
       aux:=Z[j];
```

PROGRAMA EJEMPLOC.PAS... continuación

```
      Z[j]:=Z[i];
        Z[i]:=aux;

      for k:=1 to 3 do
      begin
        aux:=elemento[j,k];
        elemento[j,k]:=elemento[i,k];
        elemento[i,k]:=aux;
      end
   end{for's}
end {Ordena};

{_____PROGRAMA PRINCIPAL_____}

begin
  clrscr;
  Presenta;

  {Variables principales}
  papas:=3; {Numero de valores iniciales}

  gotoxy(32,2);writeln(' VALORES INICIALES');
  gotoxy(20,4);write('¿Desea dar los valores iniciales ? (S/N) ');readln(respuesta);
  if upcase(respuesta)='N' then
  begin
    {Valores iniciales aleatorios}
    randomize;
    for i:=1 to papas do
    for j:=1 to 3 do
    begin
      delay(5);
      X_padre[i,j]:=(random(200)+random)*eleva(-1,random(2));
    end
  end
  else
    {Valores iniciales por teclado}
    for i:=1 to papas do
    begin
      gotoxy(25,9+i);
      for j:=1 to 3 do
      begin
        write('  X',i,' '); readln(X_padre[i,j]);
      end
    end;

  {Cada valor es empleado en la funcion objetivo}
  for i:=1 to papas do
  Z[i]:=objetivo(X_padre[i,1],X_padre[i,2],X_padre[i,3]);

  {Se ordenan los valores resultados de menor a mayor, por su resultado}
  Ordena(X_padre,Z,papas);

  {Se dan los valores de duplicacion para los valores iniciales}
  for i:=1 to papas do
     duplica[i]:=8;

  {Total de individuos por cada generacion}
  generacion:=0;
```

PROGRAMA EJEMPLOC.PAS... continuación

```
for i:=1 to papas do
   generacion:=generacion+duplica[i];
Impresion(X_padre,Z,papas);

gotoxy(33,15);write('¿EPSILON ? '); readln(epsilon);{Se recomienda epsilon=0.005}
gotoxy(33,17);write('¿ITERACIONES ? '); readln(iteraciones);{Se recomienda iteraciones=100}

k:=0; {Indica el numero de iteracion}

{Se obtiene la diferencia entre los resultados de los valores iniciales (1ro. y 3er.)}
{Ya que los valores se ordenaron anteriormente Z[1] tiene al mejor
resultado y X_padre[1] contienen al mejor punto encotrado hasta el momento}
resta:=abs(Z[papas]-Z[1]);

{Si la diferencia es mayor a epsilon o el numero de iteraciones es menor a
las indicadas, entonces el algoritmo continua}

while (resta > epsilon) and (k < iteraciones) do
begin
 {Se incrementa el numero de la iteracion}
 inc(k);
 for j:=1 to 3 do
 begin
  randomize;

  {Rnd es para elegir el tipo de mutacion}
  repeat
    rnd:=random;
  until rnd>0;

  {Se eligen el tipo de mutacion}
  if (rnd <= 0.33) then
  {Primera mutacion: entre 0 y 1}
    repeat
      mutacion[j]:=random;
    until mutacion[j]>0
  else
  if (rnd > 0.33) and (rnd <= 0.66) then
  {Segunda mutacion: entre el 0 y el mejor encontrado}
    repeat
      mutacion[j]:=random(trunc(abs(X_padre[1,j])))+random;
    until mutacion[j]>0
  else
  if (rnd > 0.66) then
  {Tercera mutacion: entre 0 y 10}
   repeat
     mutacion[j]:=random(10)+random;
    until mutacion[j]>0;
 end;{j}
 {Se generan los hijos}
 indice:=0;
 for i:=1 to papas do
 for k1:=1 to duplica[i] do
 begin
   indice:=indice+1;
   for j:=1 to 3 do
   begin
     {Esta formula de Box y Muller [4] que genera numeros aleatorios con una
```

PROGRAMA EJEMPLOC.PAS... continuación

```
    distribucion normal. Notese que cada valor es diferente para cada individuo}
    box: = sqrt(-2*ln(random))*sin(2*pi*random);

    {Nuevo individuo}
    X_hijo[indice,j]: = X_padre[i,j] + mutacion[j]*box;
  end;{j}

  Z_aux[indice]: = objetivo(X_hijo[indice,1],X_hijo[indice,2],X_hijo[indice,3]);
end;{k1,i}

{Para elegir los tres mejores valores de esta iteracion, se agregan los padres}
for i: = 1 to papas do
begin
  for j: = 1 to 3 do
  X_hijo[24 + i,j]: = X_padre[i,j];
  Z_aux[24 + i]: = Z[i];
end;{i}

{Se ordenan y los tres primeros lugares son los mejores tres valores
encontrados hasta el momento}
Ordena(X_hijo,Z_aux,generacion + 3);

{Los tres mejores valores son los padres de la siguiente generacion}
for i: = 1 to papas do
begin
  for j: = 1 to 3 do
    X_padre[i,j]: = X_hijo[i,j];
    Z[i]: = Z_aux[i];
end;{i}

{Impresion en pantalla de los tres mejores resultados encontrados}
clrscr;
gotoxy(18,6);writeln('Los mejores valores de la I T E R A C I O N  ',k);
Impresion(X_padre,Z,papas);
readln;

{Condicion para continuar el algoritmo}
resta: = abs(Z[3]-Z[1]);
{Si resta es menor a epsilon el algoritmo termina}

end; {while principal}
gotoxy(37,21);write('F I N ');
writeln;
repeat until keypressed;
window(0,0,25,80);
end.
```

PROGRAMA EJEMPLOF.PAS

$$Min\ Z = X_1^2 + X_2^2 - 16X_1 - 10X_2$$

$$11 - X_1^2 + 6X_1 - 4X_2 \geq 0$$

$$X_1X_2 - 3X_2 - e^{X_1 - 3} + 1 \geq 0$$

$$X_1 \geq 0,\ X_2 \geq 0$$

```
Program HILL_CLIMBING_UNIMODAL_UNA_VARIABLE;

{_____TIPOS, CONSTANTES Y VARIABLES DEL PROGRAMA PRINCIPAL_____}
uses
  {Libreria de Turbo Pascal}
  crt;

type
  realmat = array[1..27,1..2] of real; {Este tipo es para las matrices reales}
  reales = array[1..27] of real; {Este tipo es para los arreglos reales}
  enteros = array[1..10] of integer; {Este tipo es para los arreglos enteros}

var
  X_padre,          {Padres iniciales}
  X_hijo: realmat;  {Hijos de la siguiente generacion}
  Z_aux,            {Resultados de la nueva generacion}
  Z,                {Resultados de la generacion inicial}
  restric,          {Numero de restricciones que cumple cada par de valores iniciales}
  restric1,         {Numero de restricciones que cumple cada par nuevo de valores}
  mutacion:reales;  {Mutacion para cada variable de la funcion objetivo}
  duplica:enteros;  {Numero de duplicacion de cada padre}
  indice,i,j,k,k1,  {indices}
  papas,            {Numero de padres iniciales para cada generacion}
  tipo,             {Tipo de mutacion}
  generacion,       {Numero total de la generacion}
  iteraciones       :integer;{Numero limite de iteraciones}
  resta,            {Diferencia entre el primero y ultimo de los mejores
                    resultado de la nueva generacion}
  box,              {Valor aleatorio de una distribucion normal}
  epsilon,          {Limite de error}
  rnd:real;         {valor aleatorio entre 0 y 1}
  respuesta:char;
  ver:boolean;

{_____PROCEDIMIENTOS Y FUNCIONES_____}

procedure Presenta;
{Este es un procedimiento de presentacion. Crea un marco para la pantalla}
 var
  i:integer;
begin
  gotoxy(1,1);write(' ╔'); {Esquina superior izquerda}
  for i:=2 to 79 do
  begin
    gotoxy(i,1);write('═'); {Linea superior}
  end;
  gotoxy(80,1);write('╗'); {Esquina superior derecha}
  for i:=2 to 23 do
```

PROGRAMA EJEMPLOF.PAS... continuación

```
 begin
   gotoxy(80,i);write(' ‖ '); {Linea derecha}
 end;
 gotoxy(80,24);write('╝ '); {Esquina inferior derecha}
 for i:=79 downto 2 do
 begin
   gotoxy(i,24);write('═'); {Linea inferior}
 end;
 gotoxy(1,24);write(' ╚'); {Esquina inferior izquierda}
 for i:=23 downto 2 do
 begin
   gotoxy(1,i);write(' ‖ '); {Linea izquierda}
 end;
 window(2,2,79,23); {Deja fijo el marco durante todo el programa}

 gotoxy(32,3);write('METODO HILL-CLIMBING');
 gotoxy(28,19); write('Oprima RETURN para continuar');
 readln;
end; {Presenta}

function verifica(x:realmat;var res:reales; limite:integer):boolean;
{Esta funcion verifica que al menos un par de los valores cumpla con las
 restricciones}
var
  i,suma:integer;
  cuenta:boolean;
begin
 cuenta:=false;
 for i:=1 to limite do
 begin
    suma:=0;
    if (11-sqr(x[i,1])+6*x[i,1]-4*x[i,2] >= 0) then inc(suma);

    if (x[i,1]*x[i,2]-3*x[i,2]-exp(x[i,1]-3)+1 >= 0) then inc(suma);

    if (x[i,1] >= 0.0) then inc(suma);

    if (x[i,2] >= 0.0) then inc(suma);

    if (suma = 4) then cuenta:=true;
    res[i]:=suma;
  end;

  verifica:=cuenta;
end{verifica};

procedure Impresion(x: realmat; Z,res:reales;limite:integer);
{Este procedimiento imprime dos arreglos en forma de columna al mismo tiempo}
begin
 for i:=1 to limite do
 begin
    gotoxy(8,9+i); write('  X',i,1,' = ',x[i,1]:3:4);
    gotoxy(25,9+i); write('  X',i,2,' = ',x[i,2]:3:4);
    gotoxy(42,9+i);write('  Z = ',Z[i]:3:4);
    gotoxy(59,9+i);write('  restricción ',res[i]:3:0);
 end;
 gotoxy(28,19); writeln('Oprima RETURN para continuar');
end {Impresion};
```

PROGRAMA EJEMPLOF.PAS... continuación

```
function Objetivo(x1,x2:real): real;
{Evalua la funcion objetivo para un solo valor}
begin
  objetivo:=sqr(x1)+sqr(x2)-16*x1-10*x2;
end {Objetivo};

procedure Ordena(var elemento:realmat;var Z,res:reales;limite:integer);
{Se ordenan los valores de menor a mayor con respecto a su resultado}
var
  i,j,k:integer;
  aux:real;

begin
  for i:=1 to 3 do
   for j:=i+1 to limite do
    if (res[j]>res[i]) or ((res[j]=res[i]) and (Z[j]<Z[i])) then
    begin

        aux:=res[i];
        res[i]:=res[j];
          res[j]:=aux;

        aux:=Z[i];
        Z[i]:=Z[j];
          Z[j]:=aux;

        for k:=1 to 2 do
        begin
          aux:=elemento[i,k];
          elemento[i,k]:=elemento[j,k];
          elemento[j,k]:=aux;
        end;
    end;{for's}
end {Ordena};

{_____PROGRAMA  PRINCIPAL_____}

begin
  clrscr;
  Presenta;

  {Variables principales}
  papas:=3; {Numero de valores iniciales}

  {Valores iniciales aleatorios}
  repeat
    randomize;
    for i:=1 to papas do
    for j:=1 to 2 do
    begin
      delay(5);
      X_padre[i,j]:=random(50)+random;
    end;

    {Cada valor es empleado en la funcion objetivo}
    for i:=1 to papas do
    Z[i]:=objetivo(X_padre[i,1],X_padre[i,2]);
  until verifica(X_padre,restric,papas);
```

PROGRAMA EJEMPLOF.PAS... continuación

```
{Se ordenan los valores resultados de menor a mayor, por su resultado}
Ordena(X_padre,Z,restric,papas);

{Se dan los valores de duplicacion para los valores iniciales}
for i:=1 to papas do
   duplica[i]:=8;

{Total de individuos por cada generacion}
generacion:=0;
for i:=1 to papas do
   generacion:=generacion+duplica[i];
Impresion(X_padre,Z,restric,papas);

gotoxy(33,15);write('¿EPSILON ? '); readln(epsilon);{Se recomienda epsilon=0.005}
gotoxy(33,17);write('¿ITERACIONES ? '); readln(iteraciones);{Se recomienda iteraciones=100}

k:=0; {Indica el numero de iteracion}

{Se obtiene la diferencia entre los resultados de los valores iniciales (1ro. y 3er.)}
{Ya que los valores se ordenaron anteriormente Z[1] tiene al mejor
resultado y X_padre[1] contienen al mejor punto encotrado hasta el momento}
resta:=abs(Z[papas]-Z[1]);

{Si la diferencia es mayor a epsilon o el numero de iteraciones es menor a
las indicadas, entonces el algoritmo continua}

while (resta > epsilon) and (k < iteraciones) do
begin
 {Se incrementa el numero de la iteracion}
 inc(k);
 for j:=1 to 2 do
 begin
 randomize;

 {Rnd es para elegir el tipo de mutacion}
 repeat
    rnd:=random;
 until rnd>0;

 {Se eligen el tipo de mutacion}
 if (rnd <= 0.33) then
   {Primera mutacion: entre 0 y 1}
   repeat
     mutacion[j]:=random;
     tipo:=1;
   until mutacion[j]>0
 else
 if (rnd > 0.33) and (rnd <= 0.66) then
   {Segunda mutacion: entre el 0 y el mejor encontrado}
   repeat
     mutacion[j]:=random(trunc(abs(X_padre[1,j]))) + random;
   until mutacion[j]>0
 else
  if (rnd > 0.66) then
  {Tercera mutacion: entre 0 y 10}
   repeat
     mutacion[j]:=random(10) + random;
     tipo:=3;
```

PROGRAMA EJEMPLOF.PAS... continuación

```
    until mutacion[j]>0;
  end {j};

{Se generan los hijos}
 indice:=0;
 for i:=1 to papas do
 for k1:=1 to duplica[i] do
 begin
    indice:=indice+1;
    for j:=1 to 2 do
    begin
      {Esta formula de Box y Muller [4] que genera numeros aleatorios con una
      distribucion normal. Notese que cada valor es diferente para cada individuo}
      box:=sqrt(-2*ln(random))*sin(2*pi*random);

      {Nuevo individuo}
      X_hijo[indice,j]:=X_padre[i,j]+mutacion[j]*box;
     end{j};

    {Sustitucion del nuevo valor en la funcion objetivo}
    Z_aux[indice]:=objetivo(X_hijo[indice,1],X_hijo[indice,2]);

  end{k1,i};

  ver:=verifica (X_hijo,restric1,generacion);

  {Para elegir los tres mejores valores de esta iteracion, se agregan los
  padres}

  for i:=1 to papas do
  begin
    for j:=1 to 2 do
    X_hijo[24+i,j]:=X_padre[i,j];
    Z_aux[24+i]:=Z[i];
    restric1[24+i]:=restric[i];
  end{i};

  {Se ordenan y los tres primeros lugares son los mejores tres valores
  encontrados hasta el momento}
  Ordena(X_hijo,Z_aux,restric1,generacion+3);

  {Los tres mejores valores son los padres de la siguiente generacion}
   for i:=1 to papas do
   begin
    for j:=1 to 2 do
     X_padre[i,j]:=X_hijo[i,j];
     Z[i]:=Z_aux[i];
     restric[i]:=restric1[i]
   end;

   {Impresion en pantalla de los tres mejores resultados encontrados}
   clrscr;
   gotoxy(18,6);writeln('Los mejores valores de la I T E R A C I O N ',k);
   Impresion(X_padre,Z,restric,papas);
   readln;

   {Condicion para continuar el algoritmo}
   resta:=abs(Z[papas]-Z[1]);
```

PROGRAMA EJEMPLOF.PAS... continuación

```
   {Si resta es menor a epsilon el algoritmo termina}

end; {while principal}
gotoxy(37,21);write('F I N ');
writeln;
repeat until keypressed;
window(0,0,25,80);
end.
```

PROGRAMA EJEMPLOJ.PAS

$$Min\ z_t = \sum_{i=1}^{I} \sum_{n=1}^{N} W_{tin}(R_{tin} - \sum_{m=1}^{N} Q_{timn} U_{timn})^2$$

sujeta a:

$$\sum_{p=1}^{N} U_{tinp} - X_{tin} \leq 0 \quad \forall\ i \in I,\ n \in N$$

$$A_{tin} R_{tin} \leq X_{tin} \leq R_{tin} + e_{tin} \quad \forall\ i \in I,\ n \in N$$

$$X_{tin}\ entero,\ U_{tin} \geq 0 \quad \forall\ i \in I,\ m,n \in N$$

```
Program HILL_CLIMBING_MULTIVARIABLE_CON_RESTRICCIONES;

{Este programa esta hecho para 3 turnos de 8 horas cada uno (matutino,
 vespertino, nocturno), 6 pabellones y 4 categorias de enfermeras.}

{_____TIPOS, CONSTANTES Y VARIABLES DEL PROGRAMA PRINCIPAL_____}

uses
  {Libreria de Turbo Pascal}
  crt;

type
  realmat5 = array[1..7,1..3,1..6,1..4,1..4] of real; {Matriz valores U iniciales}
  realmat4 = array[1..7,1..6,1..4,1..4] of real; {Matriz para valores optimos
  por pabellon, personal y turno. Considera la sustitucion de una categoria
  de personal por otra}
  realmat3 = array[1..3,1..6,1..4] of real; {Matriz para el requerimiento de
  personal por pabellon, personal y turno}
  enteromat4 = array[1..7,1..3,1..6,1..4] of integer; {Matriz para los valores
  optimo de enfermeras de cada categoria por pabellon y turno}
  reales = array[1..7] of real; {Arreglo para los resultados}
  enteros = array[1..7] of integer;

var
  U:realmat5;       {Padres e hijos (fraccion real de personal que esta
                   cubriendo el pabellon p en el turno t)}
  Q:realmat4;       {Factor de reemplazo}
  Muta,             {Esta matriz guarada los valores para la mutacion tipo 2}
  Req,              {Requerimiento del personal}
  W,                {Factor de gravedad de la falta o sobra de personal}
  A,                {Factor minimo de asistencia}
  E:realmat3;       {Factor límite de personal}
  B:enteros;        {Límite de personal por categoria}
  X:enteromat4;     {Variable de decision: numero optimo de enfermeras}
  ResultaX,ResultaU {Resultados de la generacion anterior y la nueva}
  :reales;
  Restric,          {Numero de restricciones que se cumplen en cada individuo}
  Lugar,            {Posicion acuerdo a su eficiencia}
```

PROGRAMA EJEMPLOJ.PAS... continuación

```
 duplica,        {Numero de duplicacion de cada padre}
 Pos:enteros;    {Guarda las posiciones de los tres mejores valores}
 i,j,k,k1,t,p,f,n, {indices}
 papas,          {Numero de padres iniciales por cada generacion}
 tipo,           {Tipo de mutacion}
 generacion,     {Numero total de la generacion}
 iteraciones,    {Numero limite de iteraciones}
 Turno,          {Numero de Turnos}
 Enf,            {Numero total de enfermeras}
 Pab,            {Numero de Pabellones}
 indice:integer; {Indica la posicion en donde debe ir el nuevo valor}
 mutacion:real;  {Mutacion para cada variable de la funcion objetivo}
 resta,          {Diferencia entre el primero y ultimo de los mejores
                 resultado de la nueva generacion}
 box,            {Valor aleatorio de una distribucion normal}
 epsilon,        {Limite de error}
 rnd:real;       {valor aleatorio entre 0 y 1}

{_____PROCEDIMIENTOS Y FUNCIONES_____}

procedure Presenta;
{Este es un procedimiento de presentacion. Crea un marco para la pantalla}
 var
  i:integer;
begin
  gotoxy(1,1);write(' ╔'); {Esquina superior izquerda}
  for i:=2 to 79 do
  begin
    gotoxy(i,1);write('═'); {Linea superior}
  end{i};
  gotoxy(80,1);write('╗'); {Esquina superior derecha}
  for i:=2 to 23 do
  begin
    gotoxy(80,i);write(' ‖ '); {Linea derecha}
  end{i};
  gotoxy(80,24);write('╝'); {Esquina inferior derecha}
  for i:=79 downto 2 do
  begin
    gotoxy(i,24);write('═'); {Linea inferior}
  end{i};
  gotoxy(1,24);write(' ╚'); {Esquina inferior izquierda}
  for i:=23 downto 2 do
  begin
    gotoxy(1,i);write(' ‖ '); {Linea izquierda}
  end{i};
  window(2,2,79,23); {Deja fijo el marco durante todo el programa}

  gotoxy(32,3);write('METODO HILL-CLIMBING');
  gotoxy(15,5);write('ASIGNACION DE ENFERMERAS POR CATEGORIAS EN UN HOSPITAL');
  gotoxy(28,19); write('Oprima RETURN para continuar');
  readln;
end; {Presenta}

Procedure Genera_X(i:integer; var X:enteromat4);
Var
  t,p,f,n:integer;
  suma:real;
Begin
```

PROGRAMA EJEMPLOJ.PAS... continuación

```
 for t: = 1 to Turno do
  for p: = 1 to Pab do
   for f: = 1 to Enf do
   repeat
     X[i,t,p,f]: = random(trunc(Req[t,p,f] + E[t,p,f] + 1));
   until (A[t,p,f]*Req[t,p,f] < = X[i,t,p,f]);
end {Genera_X};

Procedure Genera_U(i:integer; var U:realmat5; X:enteromat4);
{Este procedimiento generalos Valores iniciales aleatoriamente}
Var
  t,p,f,n:integer;
  suma:real;
Begin
  randomize;
   for t: = 1 to Turno do
    for p: = 1 to Pab do
     for f: = 1 to Enf do
     repeat
      suma: = 0;
      for n: = 1 to Enf do
      begin
        U[i,t,p,f,n]: = random(X[i,t,p,f] + 1) + random;
        suma: = U[i,t,p,f,n] + suma;
      end{n};
     until (suma-X[i,t,p,f] < = 0);
end {Genera_U};

Procedure Verifica1(i:integer;U:realmat5; X:enteromat4; var Res:integer);
{Se verifican las restricciones}
Var
   t,p,f,n:integer;
   suma:real;
Begin
 Res: = 0;

 for t: = 1 to Turno do
  for p: = 1 to Pab do
   for f: = 1 to Enf do
   Begin
    suma: = 0;
    for n: = 1 to Enf do
      suma: = U[i,t,p,f,n] + suma;

    {Restriccion A}
    if suma < = X[i,t,p,f] then inc(Res);

    {Restriccion B}
    if X[i,t,p,f] > = A[t,p,f]*Req[t,p,f] then inc(Res);

    {Restriccion C}
    if X[i,t,p,f] < = Req[t,p,f] + E[t,p,f] then inc(Res);

   end{f,p,t};

   for f: = 1 to Enf do
   Begin
    suma: = 0;
```

PROGRAMA EJEMPLOJ.PAS... continuación

```
    for p:=1 to Pab do
     for t:=1 to Turno do
      suma:=X[i,t,p,f] + suma;

    {Restriccion D}
    if suma <= B[f] then inc(Res);

   end{f};

   {Las restriciones de que X debe ser entero y U >= 0 quedan implicitas}
end{Verifica1};

Procedure Verifica(i:integer; X:enteromat4; var Res:integer);
{Se verifican las restricciones}
Var
   t,p,f,n:integer;
   suma:real;
Begin
 Res:=0;

   for f:=1 to Enf do
   Begin
    suma:=0;
    for p:=1 to Pab do
     for t:=1 to Turno do
      suma:=X[i,t,p,f] + suma;

      {Restriccion D}
      if suma <= B[f] then inc(Res);
   end{f};
   {Las restriciones de que X debe ser entero y U >= 0 quedan implicitas}
end{Verifica};

function Objetivo_X(i:integer; X:enteromat4):real;
{Evalua la funcion Objetivo para valores de X}
Var
  t,p,f:integer;
  Z,d:real;
begin
  Z:=0;
  for t:=1 to Turno do
   for p:=1 to Pab do
    for f:=1 to Enf do
    begin
      d:=Req[t,p,f] - X[i,t,p,f];
      Z:=W[t,p,f]*d*d+Z;
    end{f,p,t};
   objetivo_X:=Z;
end {Objetivo_X};

function Objetivo_U(i:integer;U:realmat5): real;
{Evalua la funcion objetivo para valores de U}
Var
  t,p,f,n:integer;
  Z,suma,d:real;
begin
  Z:=0;
  for t:=1 to Turno do
```

PROGRAMA EJEMPLOJ.PAS... continuación

```
  for p:=1 to Pab do
   for f:=1 to Enf do
   begin
    suma:=0;
    for n:=1 to Enf do
      suma:=Q[t,p,f,n]*U[i,t,p,f,n]+suma;
    d:=Req[t,p,f]-suma;
    Z:=W[t,p,f]*d*d+Z;
   end{f,p,t};
  objetivo_U:=Z;
end {Objetivo_U};

procedure Ordena(Resultado:reales; Res:enteros; var Lugar:enteros);
{Se observa su eficientencia y se catalogan el 1°,2° y 3° lugar}
var
 i,j,k,aux,lu:integer;
Const
 veces=2;
begin
 lu:=papas;
 for i:=1 to papas do
   begin
     Lugar[i]:=lu;
     dec(lu)
   end{i};

 for k:=1 to veces do
  for i:=1 to papas-1 do
    for j:=i+1 to papas do
      if Lugar[j]=Lugar[i]-1 then
         if (Res[i]>Res[j]) or ((Res[i]=Res[j]) and
           (Resultado[i]<Resultado[j])) then
         begin
           aux:=Lugar[i];
           Lugar[i]:=Lugar[j];
           Lugar[j]:=aux;
         end{j,i,k};

end {Ordena};

procedure ImpresionX(X:enteromat4;ResultadoX: reales;Pos:enteros;k:integer);
{Impresion de los valores obtenidos}
var
 t,p,f:integer;
 aux:real;

begin
 clrscr;
 gotoxy(30,3); write('X X X X X X X X X X  ITERACION ',k);
 for t:=1 to Turno do
  for p:=1 to Pab do
  begin
   gotoxy(31,6+p);
   for f:=1 to Enf do
    write(X[Pos[1],t,p,f],'    ');
  end;{p,t}
 gotoxy(28,20);write('Resultado X',1,'=');write(ResultadoX[Pos[1]]:3:2);
 readln;
```

PROGRAMA EJEMPLOJ.PAS... continuación

```
  clrscr;
end{ImpresionX};

procedure ImpresionU(U:realmat5; ResultadoU:reales;Res,Pos:enteros);
{Impresion de los valores obtenidos}
var
  i,j,k,t,p,f,n,l:integer;
  aux:real;

begin
 clrscr;
 gotoxy(30,3); write('U U U U U U U U U U');
  for t:=1 to Turno do
   for p:=1 to Pab do
    begin
      l:=p;
      if l>3 then l:=l-3;
      for f:=1 to Enf do
      begin
        gotoxy(28,5*l+f);
        for n:=1 to Enf do
          write(U[Pos[1],t,p,f,n]:3:2,'  ');
      end;{f}
      if p=3 then
         begin
           readln;
           clrscr;
         end;
    end{p,t};
  gotoxy(28,21);write('Resultado U',1,'=');write(ResultadoU[Pos[1]]:3:2);
  gotoxy(28,22);write('Restricciones ');write(Res[Pos[1]]);
  readln;
  clrscr;
end{ImpresionU};

Function Busca(Lugar:enteros;limite:integer;Pos:enteros):integer;
var
  i,j:integer;
  respuesta:boolean;
Begin
   j:=0;
   respuesta:=false;
   repeat
     inc(j);
     if (Lugar[j]=4) then
     begin
       Busca:=j;
       respuesta:=true;
       for i:=1 to papas do
           if (Pos[i]=j) then respuesta:=false;
     end{if}
   until respuesta
end {Busca};

Procedure Reordena(Var Lugar:enteros; Res:enteros; Resultado:reales;
                   limite, indice:integer);
Var
  i,aux:integer;
```

PROGRAMA EJEMPLOJ.PAS... continuación

```
 respuesta:boolean;
Begin
 respuesta: = true;
 for i: = 1 to limite do
  if i< > indice then
   if (Resultado[indice] = Resultado[i]) then
     respuesta: = false;

 if respuesta then
 for i: = 1 to limite do
   if i< >indice then
     if Lugar[i] = Lugar[indice]-1 then
       if (Res[indice] > Res[i]) or ((Res[indice] = Res[i]) and
         (Resultado[i] > Resultado[indice])) then
       Begin

         aux: = Lugar[i];
         Lugar[i]: = Lugar[indice];
         Lugar[indice]: = aux;
         i: = 0;
        end{i};

end{Reordena};

Procedure Situa(var Pos:enteros; limite:integer);
var
 i,j:integer;
Begin
 for i: = 1 to papas do
 begin
  j: = 0;
  repeat
    inc(j);
    Pos[i]: = j;
  until (Lugar[j] = i);
 end {i};
end{Situa};

Procedure Sensibilidad(X1:enteromat4;ResultaX:reales;Pos:integer);
var
 k,t,p,f,bin,Restric:integer;
const
 veces = 2;
Begin
 bin: = 1;
 for k: = 1 to veces do
 Begin
  gotoxy(30,3); write('ANALISIS DE SENSIBILIDAD');
  for t: = 1 to Turno do
   for p: = 1 to Pab do
   Begin
    gotoxy(18,5 + p*2);
    for f: = 1 to Enf do
    begin
       X1[Pos,t,p,f]: = X[Pos,t,p,f] + bin;
       write(Objetivo_X(Pos,X1):3:2,'  ');
       Verifica1(Pos,U,X1,Restric);
       write(Restric,'  ');
```

PROGRAMA EJEMPLOJ.PAS... continuación

```
        X1[Pos,t,p,f]:=X[Pos,t,p,f];
    end{f};
    writeln;
  end{p,t};
  bin:=bin*(-1);
  readln;
 end{k};
end{Sensibilidad};

{_____PROGRAMA PRINCIPAL_____}

begin
  clrscr;
  Presenta;
  {Ejemplo: 1 turno, 6 pabellones y 4 categorias de enfermeras}
  Turno:=1;
  Pab:=6;
  Enf:=4;

  Req[1,1,1]:=1.5; Req[1,1,2]:=3.5; Req[1,1,3]:=6.0; Req[1,1,4]:=2.3;
  Req[1,2,1]:=3.8; Req[1,2,2]:=1.5; Req[1,2,3]:=6.4; Req[1,2,4]:=2.5;
  Req[1,3,1]:=0.9; Req[1,3,2]:=2.2; Req[1,3,3]:=2.3; Req[1,3,4]:=1.3;
  Req[1,4,1]:=2.0; Req[1,4,2]:=0.5; Req[1,4,3]:=1.7; Req[1,4,4]:=1.4;
  Req[1,5,1]:=2.3; Req[1,5,2]:=0.3; Req[1,5,3]:=1.7; Req[1,5,4]:=1.4;
  Req[1,6,1]:=2.1; Req[1,6,2]:=0.2; Req[1,6,3]:=1.5; Req[1,6,4]:=1.2;

  Q[1,1,1,1]:=1.0; Q[1,1,1,2]:=1.0; Q[1,1,1,3]:=1.0; Q[1,1,1,4]:=1.0;
  Q[1,2,1,1]:=1.0; Q[1,2,1,2]:=1.0; Q[1,2,1,3]:=1.0; Q[1,2,1,4]:=1.0;
  Q[1,3,1,1]:=1.0; Q[1,3,1,2]:=1.0; Q[1,3,1,3]:=1.0; Q[1,3,1,4]:=1.0;
  Q[1,4,1,1]:=1.0; Q[1,4,1,2]:=1.0; Q[1,4,1,3]:=1.0; Q[1,4,1,4]:=1.0;
  Q[1,5,1,1]:=1.0; Q[1,5,1,2]:=1.0; Q[1,5,1,3]:=1.0; Q[1,5,1,4]:=1.0;
  Q[1,6,1,1]:=1.0; Q[1,6,1,2]:=1.0; Q[1,6,1,3]:=1.0; Q[1,6,1,4]:=1.0;

  Q[1,1,2,1]:=0.7; Q[1,1,2,2]:=1.0; Q[1,1,2,3]:=1.0; Q[1,1,2,4]:=1.0;
  Q[1,2,2,1]:=0.7; Q[1,2,2,2]:=1.0; Q[1,2,2,3]:=1.0; Q[1,2,2,4]:=1.0;
  Q[1,3,2,1]:=0.7; Q[1,3,2,2]:=1.0; Q[1,3,2,3]:=1.0; Q[1,3,2,4]:=1.0;
  Q[1,4,2,1]:=0.7; Q[1,4,2,2]:=1.0; Q[1,4,2,3]:=1.0; Q[1,4,2,4]:=1.0;
  Q[1,5,2,1]:=0.7; Q[1,5,2,2]:=1.0; Q[1,5,2,3]:=1.0; Q[1,5,2,4]:=1.0;
  Q[1,6,2,1]:=0.7; Q[1,6,2,2]:=1.0; Q[1,6,2,3]:=1.0; Q[1,6,2,4]:=1.0;

  Q[1,1,3,1]:=0.2; Q[1,1,3,2]:=0.4; Q[1,1,3,3]:=1.0; Q[1,1,3,4]:=0.5;
  Q[1,2,3,1]:=0.2; Q[1,2,3,2]:=0.4; Q[1,2,3,3]:=1.0; Q[1,2,3,4]:=0.5;
  Q[1,3,3,1]:=0.2; Q[1,3,3,2]:=0.4; Q[1,3,3,3]:=1.0; Q[1,3,3,4]:=0.5;
  Q[1,4,3,1]:=0.2; Q[1,4,3,2]:=0.4; Q[1,4,3,3]:=1.0; Q[1,4,3,4]:=0.5;
  Q[1,5,3,1]:=0.2; Q[1,5,3,2]:=0.4; Q[1,5,3,3]:=1.0; Q[1,5,3,4]:=0.5;
  Q[1,6,3,1]:=0.2; Q[1,6,3,2]:=0.4; Q[1,6,3,3]:=1.0; Q[1,6,3,4]:=0.5;

  Q[1,1,4,1]:=0.2; Q[1,1,4,2]:=0.4; Q[1,1,4,3]:=0.5; Q[1,1,4,4]:=1.0;
  Q[1,2,4,1]:=0.2; Q[1,2,4,2]:=0.4; Q[1,2,4,3]:=0.5; Q[1,2,4,4]:=1.0;
  Q[1,3,4,1]:=0.2; Q[1,3,4,2]:=0.4; Q[1,3,4,3]:=0.5; Q[1,3,4,4]:=1.0;
  Q[1,4,4,1]:=0.2; Q[1,4,4,2]:=0.4; Q[1,4,4,3]:=0.5; Q[1,4,4,4]:=1.0;
  Q[1,5,4,1]:=0.2; Q[1,5,4,2]:=0.4; Q[1,5,4,3]:=0.5; Q[1,5,4,4]:=1.0;
  Q[1,6,4,1]:=0.2; Q[1,6,4,2]:=0.4; Q[1,6,4,3]:=0.5; Q[1,6,4,4]:=1.0;

  W[1,1,1]:=1000; W[1,1,2]:=700; W[1,1,3]:=100; W[1,1,4]:=100;
  W[1,2,1]:=1000; W[1,2,2]:=700; W[1,2,3]:=100; W[1,2,4]:=100;
  W[1,3,1]:=1000; W[1,3,2]:=700; W[1,3,3]:=100; W[1,3,4]:=100;
  W[1,4,1]:=1000; W[1,4,2]:=700; W[1,4,3]:=100; W[1,4,4]:=100;
```

PROGRAMA EJEMPLOJ.PAS... continuación

```
W[1,5,1]:=1000; W[1,5,2]:=700; W[1,5,3]:=100; W[1,5,4]:=100;
W[1,6,1]:=1000; W[1,6,2]:=700; W[1,6,3]:=100; W[1,6,4]:=100;

A[1,1,1]:=0.7; A[1,1,2]:=0.7; A[1,1,3]:=0.7; A[1,1,4]:=0.6;
A[1,2,1]:=0.7; A[1,2,2]:=0.7; A[1,2,3]:=0.7; A[1,2,4]:=0.6;
A[1,3,1]:=0.7; A[1,3,2]:=0.7; A[1,3,3]:=0.7; A[1,3,4]:=0.6;
A[1,4,1]:=0.7; A[1,4,2]:=0.7; A[1,4,3]:=0.7; A[1,4,4]:=0.6;
A[1,5,1]:=0.7; A[1,5,2]:=0.7; A[1,5,3]:=0.7; A[1,5,4]:=0.6;
A[1,6,1]:=0.7; A[1,6,2]:=0.7; A[1,6,3]:=0.7; A[1,6,4]:=0.6;

E[1,1,1]:=1.0; E[1,1,2]:=1.0; E[1,1,3]:=1.0; E[1,1,4]:=1.0;
E[1,2,1]:=1.0; E[1,2,2]:=1.0; E[1,2,3]:=1.0; E[1,2,4]:=1.0;
E[1,3,1]:=1.0; E[1,3,2]:=1.0; E[1,3,3]:=1.0; E[1,3,4]:=1.0;
E[1,4,1]:=1.0; E[1,4,2]:=1.0; E[1,4,3]:=1.0; E[1,4,4]:=1.0;
E[1,5,1]:=1.0; E[1,5,2]:=1.0; E[1,5,3]:=1.0; E[1,5,4]:=1.0;
E[1,6,1]:=1.0; E[1,6,2]:=1.0; E[1,6,3]:=1.0; E[1,6,4]:=1.0;

B[1]:=19; B[2]:=14; B[3]:=27; B[4]:=16;
{Bn se tomo aleatoriamente}

papas:=3; {Numero de valores iniciales}

k:=0; {Indica el numero de iteracion}

for i:=1 to papas*2+1 do
begin
  ResultaX[i]:=0;
  Pos[i]:=0;
end{i};

for i:=1 to papas do
   Genera_X(i,X); {Se genera los valores de la matriz X}


{Se contabiliza el numero de restricciones para cada valor}
for i:=1 to papas do
  Verifica(i,X,Restric[i]);

{Cada matriz es evaluada en la funcion objetivo}
for i:=1 to Papas do
 ResultaX[i]:=Objetivo_X(i,X);


for i:=1 to papas*2+1 do
    Lugar[i]:=4;

{Se ordenan los valores resultados de menor a mayor, por su resultado}
Ordena(ResultaX,Restric,Lugar);

Situa(Pos,papas);

Genera_U(Pos[1],U,X); {Se generan los valores iniciales}
ResultaU[Pos[1]]:=Objetivo_U(pos[1],U);

ImpresionX(X,ResultaX,Pos,k);

ImpresionU(U,ResultaU,Restric,Pos);
```

PROGRAMA EJEMPLOJ.PAS... continuación

```
{Se obtiene la diferencia entre los Resultados de los valores iniciales (1ro. y 3er.)}
{Ya que los valores se ordenaron anteriormente resultado[1] tiene al mejor
resultado y padre[1] contienen al mejor punto encotrado hasta el momento}
resta:=abs(ResultaX[Pos[papas]]-ResultaX[Pos[1]]);

{Si la diferencia es mayor a epsilon o el numero de iteraciones es menor a
las indicadas, entonces el algoritmo continua}

gotoxy(33,15);write('EPSILON ? '); readln(epsilon);{Se recomienda epsilon=0.005}
gotoxy(33,17);write('ITERACIONES ? '); readln(iteraciones);{Se recomienda iteraciones=100}

{Se dan los valores de duplicacion para los valores iniciales}
for i:=1 to papas do
   duplica[i]:=8;

for i:=papas+1 to papas*2+1 do
   Restric[i]:=0;

while (resta > epsilon) and (k < iteraciones) do
begin
 {Se incrementa el numero de la iteracion}
 inc(k);

 {Se generan los hijos}
 for i:=1 to papas do
 begin
  randomize;

  {Rnd es para elegir el tipo de mutacion}
  repeat
    rnd:=random
  until rnd>0;

  {Se eligen el tipo de mutacion}
  if (rnd <= 0.33) then
  {Primera mutacion: entre 0 y 1}
  begin
   repeat
     mutacion:=random
   until mutacion>0;
   tipo:=1
  end
  else
  if (rnd > 0.33) and (rnd <= 0.66) then
  {Segunda mutacion: entre el 0 y el mejor encontrado}
  begin
   for t:=1 to Turno do
    for p:=1 to Pab do
     for f:=1 to Enf do
      repeat
        muta[t,p,f]:=random(X[Pos[1],t,p,f]+1)+random
      until muta[t,p,f]>0;
   tipo:=2
  end
  else
  if (rnd > 0.66) then
  {Tercera mutacion: entre 0 y 10}
  begin
```

PROGRAMA EJEMPLOJ.PAS... continuación

```
      repeat
        mutacion: = random(10) + random
      until mutacion > 0;
        tipo: = 3
    end;

    for j: = 1 to duplica[i] do
    begin
     indice: = Busca(Lugar,papas*2 + 1,Pos);
      for t: = 1 to Turno do
       for p: = 1 to Pab do
        for f: = 1 to Enf do
        begin
         repeat
          {Esta formula de Box y Muller [ ] que genera numeros aleatorios con
          una distribucion normal. Notese que cada valor es diferente para cada
          individuo}
          box: = sqrt(-2*ln(random))*sin(2*pi*random);

          {Nuevo individuo}
          if tipo < > 2 then
            X[indice,t,p,f]: = abs(round(X[Pos[i],t,p,f] + mutacion*box))
          else
            X[indice,t,p,f]: = abs(round(X[Pos[i],t,p,f] + muta[t,p,f]*box));

         until ((A[t,p,f]*Req[t,p,f]) < = X[indice,t,p,f]) and
              (X[indice,t,p,f] < = (Req[t,p,f] + E[t,p,f]));

        end{f,p,t};

        Verifica(indice,X,Restric[indice]);

        {Sustitucion del nuevo valor en la funcion objetivo}
        ResultaX[indice]: = Objetivo_X(indice,X);
        Reordena(Lugar,Restric,ResultaX,papas*2 + 1,indice);

      write(j);
     end{j};
    end{i};

    {Impresion en pantalla de los tres mejores resultados encontrados}

    Situa(Pos,papas*2 + 1);

    ImpresionX(X,ResultaX,Pos,K);

    {Condicion para continuar el algoritmo}
    resta: = abs(ResultaX[Pos[papas]]-ResultaX[Pos[1]]);
    {Si resta es menor a epsilon el algoritmo termina}
  end;
    Genera_U(Pos[1],U,X);
    ResultaU[Pos[1]]: = Objetivo_U(Pos[1],U);
    Verifica1 (Pos[1],U,X,Restric[Pos[1]]);
    ImpresionU(U,ResultaU,Restric,Pos);
{ _____ ANALISIS DE SENSIBILIDAD _____ }

Sensibilidad(X,ResultaX,Pos[1]);
end.
```

PROGRAMA FIBO.PAS

$$Min\ Z = 3X^2 - 6X + 4$$

```
program fibonacci;
{METODO DE FIBONACCI
Este porgrama encuentra el mínimo de un función continua de una sola variable,
por medio de la serie de Fibonacci.}

uses
  crt,dos;
type
  arreglo = array[0..20] of integer;

var
  fibo:arreglo; {Este arreglo gurada los números de la serie de Fibonacci}
  i,k,n:integer;  {Contadores}
  a,      {Guarada el límite izquierdo del intervalo}
  b,      {Guarada el límite derecho del intervalo}
  lambda,
  miu,
  flambda,
  fmiu,
  epsilon:real;
  hora,hora1,min,min1,seg,seg1,seg100,seg1001:word;

{_____SUBRUTINAS_____}

function fx(x:real):real;
{Aquí solo se evalua la función objetivo}
begin
  fx:=3*x*x-6*x+4;
end{FX};
function lk(ak,bk:real;n,k:integer;f:arreglo):real;
begin
   lk:=ak+F[n-k-1]/F[n-k+1]*(bk-ak);
end;
function mk(ak,bk:real;n,k:integer;f:arreglo):real;
begin
   mk:=ak+F[n-k]/F[n-k+1]*(bk-ak);
end;

{_____PROGRAMA PRINCIPAL_____}
begin
  clrscr;

  {Se genera la serie de fibonacci}
  fibo[0]:=1;
  fibo[1]:=1;
  for i:=2 to 20 do
    fibo[i]:=fibo[i-1]+fibo[i-2];
  gotoxy(18,5);write(' LIMITE IZQUIERDO:      ');readln(a);
  gotoxy(18,7);write(' LIMITE DERECHO:        ');readln(b);
  gotoxy(18,9);write(' EPSILON:               ');readln(epsilon);
  gotoxy(18,11);write(' ITERACIONES DESEADAS: ');readln(n);
  gotoxy(18,13);write(' EL ERROR OBTENIDO ES: ',1/Fibo[n]:2:3);
```

PROGRAMA FIBO.PAS... continuación

```
readln;
GetTime(hora,min,seg,seg100);
k:=1;
{Se generan los dos primero valores}
lambda:=lk(a,b,n,k,fibo);
miu:=mk(a,b,n,k,fibo);
{Se evaluan el la función objetivo}
flambda:=fx(lambda);
fmiu:=fx(miu);
clrscr;
gotoxy(6,5);writeln('K      IZQ      DER      LAMBDA      MUI    f(LAMBDA)   f(MUI)');
while (k<>n) do
 begin
  gotoxy(6,6+K);write(k); gotoxy(14,6+k);write(a:4:4);
  gotoxy(25,6+k);write(b:4:4); gotoxy(36,6+k);write(lambda:4:4);
  gotoxy(47,6+k);write(miu:4:4); gotoxy(57,6+k);write(flambda:4:4);
  gotoxy(68,6+k);write(fmiu:4:4);
  k:=k+1;
  if k<>n then
  if (flambda <= fmiu) then
  begin
     b:=miu;
     miu:=lambda;
     fmiu:=flambda;
     lambda:=lk(a,b,n,k,fibo);
     flambda:=fx(lambda);
  end
  else
  begin
     a:=lambda;
     lambda:=miu;
     flambda:=fmiu;
     miu:=mk(a,b,n,k,fibo);
     fmiu:=fx(miu);
  end;
end; {while}
k:=n;
miu:=lambda+epsilon;
fmiu:=fx(miu);
if flambda <= fmiu then
     b:=lambda
else
     a:=lambda;
gotoxy(6,6+K);write(k); gotoxy(14,6+k);write(a:4:4);
gotoxy(25,6+k);write(b:4:4); gotoxy(36,6+k);write(lambda:4:4);
gotoxy(47,6+k);write(miu:4:4); gotoxy(57,6+k);write(flambda:4:4);
gotoxy(68,6+k);write(fmiu:4:4);
gotoxy(10,8+k);write('LA SOLUCION SE ENCUENTRA EN EL INTERVALO [',a:3:4,' ',b:3:4,']');
GetTime(hora1,min1,seg1,seg1001);
{Fin del programa}
gotoxy(25,10+k);write('TIEMPO INICIAL: ',hora,': ',min,': ',seg,': ',seg100);
gotoxy(25,11+k);write('TIEMPO FINAL  : ',hora,': ',min,': ',seg,': ',seg100);
gotoxy(35,13+k);writeln('F I N');
readln;
{Ejemplo:
 Límite Izquierdo:  0   Límite derecho  : 10
 Epsilon        : 0.01    Iteraciones    : 12}
end.
```

PROGRAMA NEWTON.PAS

$$Max\ Z = 12X - 3X^4 - 2X^6$$

```
program newton_raphson;
{METODO DE NEWTON-RAPHSON
 Este programa encuentra el óptimo de una función continua}

uses
 crt,dos;

var
 i:integer;    {Contador}
 x,            {Guarda el valor anterior}
 x1,           {Guarda nuevo el valor generado}
 der1,         {Guarda la primera derivada}
 der2,         {Guarda la segunda derivada}
 epsilon,      {Guarda el epsilon dado}
 tol,          {Guarda la resta del valor nuevo menos el anterior}
 funcion:real; {Guarda el valor del punto evaluado en la función}
 hora,hora1,min,min1,seg,seg1,seg100,seg1001:word;
{_____________________SUBRUTINAS_____________________}

function eleva(numero:real;potencia:integer):real;
{Esta función eleva un numero a una potencia dada}
var
  mult:real;
  k:integer;
begin
  mult:=1;
  for k:=1 to potencia do
    mult:=mult*numero;
  eleva:=mult;
end;{ELEVA}

{_____________PROGRAMA PRINCIPAL_____________}
begin
  clrscr;
  gotoxy(10,5);write('DE EL VALOR INICIAL ');readln(x);
  gotoxy(10,7);write('DE EL VALOR DEL EPSILON ');readln(epsilon);
  clrscr;
  GetTime(hora,min,seg,seg100);
  i:=0;tol:=0;
  repeat

    inc(i);
    {Se obtiene la primera drivada}
    der1:=12*(1-eleva(x,3)-eleva(x,5));

    {Se obtiene la segunda derivada}
    der2:=-12*(3*sqr(x)+5*eleva(x,4));

    {Se evalua el valor en la función}
    funcion:=12*x-3*eleva(x,4)-2*eleva(x,6);

    {Se obtiene el nuevo valor}
```

PROGRAMA NEWTON.PAS... continuación

```
  x1:=x-der1/der2;

  {Se presenta en pantalla los resultados}
  gotoxy(15,2+i);write('ITERACION ',i);
  gotoxy(32,2+i);write('X = ',x1:4:4);
  gotoxy(45,2+i);writeln('Z =  ',funcion:4:4);
   {Se obtiene la diferencia de los valores}
   tol:=abs(x1-x);
   {El valor nuevo pasa ha ser el valor anterior de la siguiente iteración}
   x:=x1;
 until (tol<=epsilon);
 GetTime(hora1,min1,seg1,seg1001);
 gotoxy(20,4+i);write('TIEMPO INICIO: ',hora,': ',min,': ',seg,': ',seg100);
 gotoxy(20,5+i);write('TIEMPO FINAL : ',hora1,': ',min1,': ',seg1,': ',seg1001);
 writeln;
 readln;

 {Ejemplo:
    Valor inicial:  25
    Epsilon      :  0.005}
end.
```

PROGRAMA DFP.PAS

$$Min\ Z = 2\ X_1^2 + X_2^2 + 3\ X_3^2$$

```
program Davidon_Fletcher_Powell;
{METODO DAVIDON FLETER POWELL
 Este método encuentra el óptimo de funciones multivariables.}
uses
  crt,dos;
const
   variables = 3;
type
  matriz = array[1..variables,1..variables] of real;

var
  i,j,k,itera:integer;              {Contadores}
  arriba,abajo,bo,factor:real;      {Escalares}
  so,d,x,gfx,gfxn,a,b,sigma,sigmat, {Matrices}
  delta,deltat,nueva,nueva1:matriz;
  hora,hora1,min,min1,seg,seg1,seg100,seg1001:word;
{_____SUBRUTINAS_____}

procedure inicial(var d:matriz;ren,col:integer);
{Este procedimiento convierte la matriz D en una matriz identidad}
var
 i,j:integer;
begin
 for i:=1 to ren do
     for j:=1 to col do
     begin
       d[i,j]:=0;
       if i=j then d[i,j]:=1;
     end;
end{INICIAL};

procedure evalua(var gfx:matriz;x:matriz);
{En este procedimiento se obtiene el gradiente evaluado en un punto}
begin
   gfx[1,1]:=4*x[1,1];
   gfx[2,1]:=2*x[2,1];
   gfx[3,1]:=6*x[3,1];
end{EVALUA};

procedure multiplica(var c:matriz;a:matriz;rena,cola:integer;b:matriz;colb:integer);
{Este procedimiento multiplica matrices}
var
 i,j,k:integer;
begin
   for j:=1 to colb do
   for i:=1 to rena do
   begin
     c[i,j]:=0;
     for k:=1 to cola do
       c[i,j]:=c[i,j]+a[i,k]*b[k,j];
   end;
end{MULTIPLICA};
```

PROGRAMA DPF.PAS... continuación

```
procedure escalar(var a:matriz;factor:real;b:matriz;ren,col:integer);
{Este procedimiento multiplica una matriz por un escalar}
var
  i,j:integer;
begin
   for i:=1 to ren do
     for j:=1 to col do
       a[i,j]:=factor*b[i,j];
end{ESCALAR};

procedure despege;
{Este procedimiento obtiene el factor B}
begin
  arriba:=4*so[1,1]*x[1,1]+2*so[2,1]*x[2,1]+6*so[3,1]*x[3,1];
  abajo:=4*so[1,1]*so[1,1]+2*so[2,1]*so[2,1]+6*so[3,1]*so[3,1];
  bo:=-arriba/abajo;
end{DESPEGE};

procedure suma(var c:matriz;a,b:matriz;ren,col:integer);
{Este procedimiento suma matrices}
var
  i,j:integer;
begin
  for i:=1 to ren do
    for j:=1 to col do
      c[i,j]:=a[i,j]+b[i,j];
end{SUMA};

procedure resta(var c:matriz;a,b:matriz;ren:integer);
{Este procedimiento resta matrices}
var
   i:integer;
begin
  for i:=1 to ren do
      c[i,1]:=a[i,1]-b[i,1];
end{RESTA};

procedure transpuesta(var a:matriz;b:matriz;ren:integer);
{Este procedimiento obtiene la transpuesta de la matriz}
var
 i:integer;
begin
  for i:=1 to ren do
      a[1,i]:=b[i,1];
end{TRANSPUESTA};

function fx:real;
{Aquí solamente se evalua la función objetivo}
begin
   fx:=2*sqr(x[1,1])+sqr(x[2,1])+3*sqr(x[3,1]);
end;

procedure actualiza(var a:matriz;b:matriz;ren:integer);
{El procedimiento actualiza un vector}
var
 i:integer;
begin
   for i:=1 to ren do
```

PROGRAMA DPF.PAS... continuación

```
      a[i,1]:=b[i,1];
end{ACTUALIZA};

procedure impresion;
{Este procedimiento imprime en pantalla el resultado}
var
  resultado:real;
begin
  clrscr;
  resultado:=fx;
  gotoxy(26,5);write(' ITERACION ',k);
  for i:=1 to variables do
  begin
    gotoxy(26,6+i);write(' X',i,' = ',x[i,1]:4:4);
  end;
  gotoxy(26,12);write(' Z = ',resultado:4:4);
  readln;
end{IMPRESION};

{_____PROGRAMA PRINCIPAL_____}

begin
   clrscr;

   inicial(d,variables,variables);

   {Se obtienen el punto inicial}
   gotoxy(26,5);write(' DE EL VECTOR INICIAL');
   for i:=1 to variables do
   begin
     gotoxy(26,6+i);write(' X',i,' ');readln(x[i,1]);
   end;
   gotoxy(26,11);write(' ¿ITERACIONES? ');readln(itera);
   GetTime(hora,min,seg,seg100);
 for k:=1 to itera do
 begin
   evalua(gfx,x);

   multiplica(so,d,variables,variables,gfx,1);

   escalar(so,-1,so,variables,1);

   despege;

   escalar(sigma,bo,so,variables,1);

   suma(x,x,sigma,variables,1);

   evalua(gfxn,x);

   resta(delta,gfxn,gfx,variables);

   transpuesta(sigmat,sigma,variables);

  multiplica(nueva,sigma,variables,1,sigmat,variables);

  multiplica(nueva1,sigmat,1,variables,delta,1);
```

PROGRAMA DPF.PAS... continuación

```
  factor:=1/nueva1[1,1];

  escalar(a,factor,nueva,variables,variables);

  transpuesta(deltat,delta,variables);

  multiplica(nueva,d,variables,variables,delta,1);

  multiplica(nueva,nueva,variables,1,deltat,variables);

  multiplica(nueva,nueva,variables,variables,d,variables);

  multiplica(nueva1,deltat,1,variables,d,variables);

  multiplica(nueva1,nueva1,1,variables,delta,1);

  factor:=-1/nueva1[1,1];

  escalar(b,factor,nueva,variables,variables);

  suma(nueva,a,b,variables,variables);

  suma(d,nueva,d,variables,variables);

  actualiza(gfx,gfxn,variables);

  impresion;
 end;{for k}
 GetTime(hora1,min1,seg1,seg1001);
 gotoxy(26,15);write('TIEMPO INICIAL:',hora,': ',min,': ',seg,': ',seg100);
 gotoxy(26,17);write('TIEMPO FINAL  :',hora1,': ',min1,': ',seg1,': ',seg1001);
 gotoxy(32,19);writeln('F I N');

 {Ejemplo:
         X1: 36
         X2: 46
         X3: 20
   Iteraciones:  4}
end.
```

Bibliografía

1. Balintfy, J. L. A Hospital Census Predictor Model. en Smalley and Freeman. **Hospital Industrial Engineering**. Reinhold Publishing Corp. New york. 1966. pp. 312-316.

2. Bazaraa, S. Mokhtar y Shetty, C. M. **Nonlinear Programming: theory and algorithms**. John Wiley & Sons, Inc. U.S.A. 1979.

3. Bellman, R. **Dynamic Programing**. Princenton University Press. 1957.

4. Box, G.E.P. y Muller, M.E. A note on the Generation of Random Normal Deviates. **Ann. Math. Stat.** No. 29. 1958 pp. 610-611.

5. Casado, María José. Se Busca un Nuevo Darwin. **Muy Interesante**. Vol. 2 No.18. México, D.F. Febrero 1986. pp. 30-38.

6. Chuck Karr. Genetic Algorithms for Fuzzy Controllers. **IA Expert**. Febrero 1991. pp.26-33.

7. Denning, Peter J. Genetic Algoritms. **American Scientist**. Vol.80 Enero-Febrero 1992.

8. Espejo Munguia, Marcela Eugenia. **Aplicación de la Investigación de Operaciones a una Estación de Radio**. Tesis de Licenciatura (Actuaría). Universidad Nacional Autónoma de México. México, D.F. 1972.

9. Golberg, David E. **Genetic Algoritms**. Reading Mass. Addison-Wesley. 1989.

10. Golberg, David E. **Genetic Algoritms in Search, Optimization, and Machine Learning**. Addison-Wesley Publishing Company. New York, U.S.A. 1991.

11. Grossman, Stanley I. **Algebra Lineal**. México. Iberoamérica. 1983.

12. Hernández, Onésimo y Martínez, Efrain J. Programación Matemática. **IV Coloquio del Departamento de Matemáticas Centro de Investigación y Estudios Avanzados del IPN**. Taxco, Gro. México. Agosto 1985.

13. Hiller, Frederik S. y Lieberman, Geral J. **Introducción a la Investigación de Operaciones**. McGraw-Hill. México, D.F. 1989.

14. Hofstadter, Douglas R. The Genetic Code: Arbitrary? **Metamagical Themas**, New York: Basic Books. 1985. pp. 671-699.

15. Hofstadter, Douglas R. **Godel, Escher, Bahc: An eternal golden braid**. New York: Basic Books. 1979.

16. Holland, John H. **Adaptation in Natural and Artificial Systems**. Ann Arbor. University of Michigan Press. 1975.

17. Kolesar, P. A Markovian Model for Hospital Admission Scheduling. **Management Science**. Vol. 16 No. 6 Febrero 1970. pp. B384-B396.

18. Lander, Eric S.; Landridge, Robert y Saccocio, Damian M. Mapping and Interpreting Biological Información. **Communications of the A.C.M.** No.34 Septiembre 1991. pp.33-39.

19. Lang, Serge. **Algebra Lineal.** Addison-Wesley Iberoamericana. México. 1986.

20. Lazcano-Araujo, Antonio. **El Origen de la vida.** Trillas. México, D.F. 1983.

21. Lawrence Davis (ed). **Handbook of Genetic Algorithms.** New York: Va Nostrand Reinhold. 1991

22. Luenberg, David E. **Programación Lineal y no Lineal.** Addison-Wesley Iberoamericana. México. 1989.

23. Mendoza-Anzaldo, Angélica. Utilización del Método Hill-Climbing para la Optimización de Funciones No Lineales Multivariables. **CIECE 92. 2do. Congreso Interuniversitario de Electrónica, Computación y Eléctrica.** Guanajuato,Gto. México. Del 25 al 29 de mayo de 1992.

24. Negrete, J. Nuevos Algoritmos Géneticos. **Primer Congreso Nacional "La Computación en Nuestros Días".** Veracuz 1991.

25. Prawda, Juan. **Métodos y Modelos de Investigación de Operaciones,** VOL. 1 Modelos Determinísticos. Limusa. México, D.F. 1987.

26. Rechenberg, Ingo. Artificial Evolución and Artificial Intelligence. **Evolution in Machine Learning: principles and techniques.** Ed. Rchard Forsyth. Chapman and Hall. New York, U.S.A. 1989.

27. Singer, S. **A Stochastic Model Variation of Categories Patients within a Hospital.** Johns Hopkins University. Baltimore.

28. Sobol, I.M. **Método de Montecarlo.** Lecciones Populares de Matemáticas. Mir. URRS. 1983.

30. Taha, Hamdy. **Investigación de Operaciones.** Alfaomega. México, D.F. 1991.

31. Thomas, W.H. Model for Predicting Recovery Progress of Coronary Patients. **Health Service Research** (Fall 1968). pp. 185-213.

32. Warner, D. M. y Prawda, J. A Mathematical Programming Model for Scheduling Nursing Personnel in a Hospital. **Management Science.** Vol.19 No. 4 Diciembre 1972. pp. 411-422.

33. White B. F. y Holst W. R. **Ponencia Presentada en el Spring Joint Computer Conference.** Washington, D. C. 1964.