UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

SOLUCION DE PROBLEMAS DE OPTIMIZACION A TRAVES DEL SERVIDOR DE INTERNET NEOS

T E S I S

QUE PARA OBTENER EL TITULO DE

A C T U A R I A

P R E S E N T A

KATHYA LEDESMA AMAYA

DIRECTOR DE TESIS: DR. PABLO BARRERA SANCHEZ

1-A

DRA. MARÍA DE LOURDES ESTEVA PERALTA
Jefa de la División de Estudios Profesionales de la
Facultad de Ciencias
Presente

Comunicamos a usted que hemos revisado el trabajo escrito:

"Solución de problemas de optimización a través del servidor de internet NEOS"

realizado por KATHYA LEDESMA AMAYA

con número de cuenta 09333038-7 , quien cubrió los créditos de la carrera de: ACTUARIA

Dicho trabajo cuenta con nuestro voto aprobatorio.

Atentamente

Director de Tesis Propietario	Dr. Pablo Barrera Sánchez
Propietario	Dr. Jesús López Estrada
Propietario	M. en I. Rubén Téllez Sánchez
Suplente	M. en C. Guilmer Ferdinand Gónzalez Flores
Suplente	M. en C. José Lino Samaniego Mendoza

Consejo Departamental de Matemáticas

M. en C. José Antonio Flores Díaz

FACULTAD DE CIENCIAS
CONSEJO DEPARTAMENTAL
DE
MATEMÁTICAS

1-3

A mis padres Elvia y Luis

Agradecimientos

Quiero agradecer el apoyo, cariño y paciencia que me brindo mi director de tesis, el Dr. Pablo Barrera, quien a estuvo conmigo en cualquier momento que lo necesite, y junto con él, Guilmer quien me apoyo desde el principio y fin de este trabajo.
Gracias a Lino y Javier, quienes revisaron esta tesis.

Gracias a la Dra. Jetzabeth Ramírez investigadora del Instituto Mexicano del Petróleo quien me brindo el apoyo para la realización de este trabajo.

Gracias a mis padres y mi hermano Luis, "Mamá gracias por apoyarme y creer en mi".
A toda mi Familia, Amaya y Ledesma.

A todos mis amigos que estuvieron siempre presentes: Gris, Alejandro, Judith, Edna, Tatiana, Alejandra, Karla, Bryan, Era, Cruz, Omar, Jean Paul, Herbert, Rogelio, Mauricio, Aarón, Carolina y a mi profesor y amigo Javier Paez, sólo por mencionar algunos gracias.
A Emilio gracias por el apoyo y amor que me brindó.

¿Es que hacemos las cosas sólo para recordarlas?
¿Es que vivimos sólo para tener memoria de nuestra vida?
Por que sucede que hasta la esperanza es memoria
y que el deseo es el recuerdo de lo que ha de venir.

Jaime Sabines

Índice

Introducción

A pesar de los grandes adelantos en la optimización computacional ocurridos durante los últimos 20 años (por ejemplo, los avances en los métodos de punto interior, el cual es una técnica para encontrar la solución de problemas con restricciones), el método Simplex inventado por George B. Dantzig en 1947 es aún la herramienta principal en casi todas las aplicaciones de la programación lineal, utilizada en varias paqueterías de optimización.

Dantzing es considerado como uno de los tres fundadores de la programación lineal, compartiendo dicho honor con: Von Neumann y Kantorovich. A través de su investigación en teoría matemática, computación, análisis económico y aplicaciones de problemas industriales, logro contribuir más que cualquier otro investigador al desarrollo de la programación lineal.

La programación lineal, la optimización no lineal con restricciones y la programación entera, han sido capaces de pasar la prueba del tiempo sin debilitarse, y en nuestros días influyen en las prácticas económicas de las organizaciones y sus administraciones. El científico computacional László Lovász dijo, "Si se tomaran estadísticas acerca de cuál problema matemático usa la mayoría del tiempo computacional en el mundo (sin incluir problemas de manejo de bases de datos, como la búsqueda y ordenamiento), seguramente la respuesta sería la programación lineal". Eugene Lawler de Berkeley dijo lo siguiente: "La programación lineal se usa para asignar recursos, planear la producción, planear el horario de trabajadores, planear la cartera de inversión y formular estrategias de mercados (o estrategias militares). La versatilidad e impacto económico de la programación lineal en el mundo industrial actual es realmente impresionante".

Dantzing: "es interesante notar que el problema original que ocasionó mi investigación está todavía pendiente, es decir, el problema de la planeación dinámica a través del tiempo, particularmente bajo condiciones de incertidumbre. Si este tipo de problemas pudieran resolverse satisfactoriamente,

se podría contribuir (tras una buena planeación) al mejoramiento de este mundo y del ser humano".

El sentido de mencionar lo anterior, es resaltar el avance que ha tenido la optimización computacional hasta nuestros días, y uno de estos avances lo hemos estudiado en esta tesis, cuyo objetivo principal, es presentar una investigación de la solución de Problemas de Optimización a través del `Servidor de Internet NEOS` (Network Enable Optimization System), el cual es una alternativa innovadora de solución de problemas, y que está al alcance de cualquier tipo de usuario que tenga un problema de optimización.

Cabe mencionar que la idea de trabajar en este tema de tesis, surgió de una propuesta de solucionar un problema de optimización que se nos presentó a el Dr. Barrera y a mí. El cual era una formulación derivada de las paredes porosas de los yacimientos de pozos petroleros. Dadas las condiciones poco favorables de trabajo (software disponible), nos llevo a una búsqueda minuciosa en Internet, de algoritmos disponibles y accesibles para cualquier tipo de usuario. De esta forma abordamos el tema de la solución de problemas de optimización a través de un servidor de Internet.

Este trabajo se divide en tres capítulos; en el primero se presenta la definición de un problema de optimización; la teoría y conceptos, los procesos de solución; la formulación del modelo matemático, que resulta importante en el momento de la codificación del problema. Enseguida se analizan los tipos de problemas de optimización y finalmente se estudian algunos ejemplos, que a nuestro parecer son típicos en optimización.

En el segundo se presenta el lenguaje de programación matemática `AMPL`, que sirve para describir los problemas de optimización. La forma en que se explica éste, es a través de un problema de programación lineal en dos variables, donde se formula el modelo matemático, se traduce al lenguaje `AMPL` y se encuentran las soluciones. Otro problema que se presenta es el de la Dieta, el cual es un problema lineal, que es muy utilizado en diversos contextos. Se hace un mejoramiento del modelo y se añaden las restricciones, de tal forma que sea posible utilizar los diversos comandos de `AMPL`. Finalmente se presenta la programación No Lineal, en donde damos ejemplos de costos no lineales, la sustitución automática de variables y expresiones no lineales.

En el tercero se presenta la solución de problemas de optimización a través del `Servidor de Internet NEOS`, por tal razón en éste está la principal aportación de este trabajo. En este capítulo se mencionan las características principales del Servidor, cómo describir un problema de optimización en `AMPL` y `FORTRAN` para enviarlo al `Servidor NEOS`, esto se hace a través de ejemplos

de problemas de optimización con y sin restricciones y la interacción con el Servidor; es decir, la forma de envío y sus alternativas. Por último se analizan algunas de las particularidades de éste.

Hemos tratado de ilustrar de forma gráfica cada uno de los pasos a seguir en la solución de problemas, pero finalmente el desarrollo de la optimización computacional seguirá avanzando y junto con ella los factores técnicos, como son la parte de gráficos, de tal forma que en un futuro seguramente será más sencilla.

Para obtener los modelos completos que se darán como ejemplos en disco, favor de cotactarme a través de la siguiente dirección katledam@hotmail.com.

Capítulo 1

Sobre la Optimización

El objetivo de este capítulo, es formalizar los conceptos más importantes en la descripción de la naturaleza, y alcance en Teoría de la Optimización[20]. Definición del problema, la complejidad de su naturaleza, y los pasos fundamentales en el proceso de su solución. De acuerdo a estas etapas, se da una breve descripción de los tipos de problemas a que nos enfrentamos:

1. Definición del problema
2. Formulación del modelo para optimización
3. Elección de un método para la solución
4. Aplicación del método
5. Su validación
6. Control y retroalimentación de la solución

1.1 El Problema de Optimización

Cuando una persona toma decisiones, y se avoca al problema de elegir una acción, entre un conjunto de alternativas, se ve impulsada a escoger una elección que está en función de una finalidad predeterminada. De un conjunto de objetivos relacionados.

Se asume que cada acción o alternativa puede evaluarse, valiéndose de un método cuantitativo. El grado en el cual se alcanzan las metas u objetivos

del problema, también es posible obtener una medida para el desempeño o lo que comúnmente le llamamos función objetivo. Así el que toma la decisión se apoya en esta información, para seleccionar la alternativa que produzca el mejor resultado . La función objetivo es la que cuantifica la realización o alcance de una meta.

1.1.1 Definición del Problema de Optimización

A partir de un conjunto de alternativas, posiblemente infinito, asociado a un problema se da el nombre de optimización, o también a la acción de elegir una alternativa particular, para la cual la función objetivo es óptima, esto es, la elección de la alternativa con la cual se maximiza o minimiza la función objetivo.

1.1.2 Naturaleza del Problema de Optimización

La naturaleza de un problema de optimización, en la mayoría de las ocasiones es muy compleja. Además en los problemas prácticos, tienen una gran variedad de casos que presentan características diferentes. Para visualizar la complejidad que puede presentarse en la naturaleza del problema, se consideran los puntos siguientes:

1. Cuando el que toma la decisión, se enfrenta al problema de optimizar un objetivo bien definido, puede darse el caso que esa optimización este sujeta a un conjunto de restricciones, o bien a ninguna. El problema puede ser determinista o estocástico.

2. El que toma las decisiones puede estar en comunicación con otras personas, bajo acciones recíprocas y competir. Cada competidor pretenderá hacer decisiones que optimicen sus propias funciones objetivos.

3. Un problema de estados múltiples, puede implicar el tomar acerca de él varias decisiones.

La optimización es de naturaleza compleja, pero que sus modelos tienen características estructurales diferentes, lo que conduce al uso de una gran variedad de técnicas para obtener soluciones. El conjunto de todas estas técnicas, especialmente las incluidas bajo los nombres específicos de Programación Matemática, Teoría de Decisiones, Teoría de Juegos, Programación

Dinámica, Teoría de Redes, Teoría de Control, Cálculo de Variaciones, Combinatoria, etc., constituyen los fundamentos matemáticos de la Teoría General de Optimización.

La Teoría de Optimización es la rama unificada del análisis matemático, que suministra un enfoque formal para resolver problemas de optimización. Determina el valor de la variable o de las variables que maximizan o minimizan a una función o funcional.

1.2 Conceptos en Teoría de Optimización

1.2.1 Procesos de la Solución

Los procesos de solución en los problemas de optimización, no pueden ser idénticos, generalmente difieren de acuerdo con la naturaleza especial del problema; no obstante, siempre es posible distinguir los pasos básicos del proceso de solución de un problema de optimización. Se presentan en la Fig 1.1. los diferentes circuitos indican una posible revisión previa a la etapa.

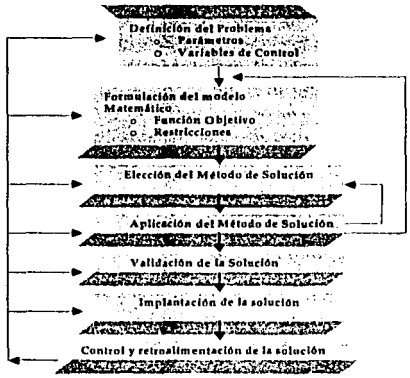

Figura. 1.1 Proceso de Solución de un Problema de Optmización

1.2.2 Definición del Problema

En el proceso de definición, se deben identificar las variables de decisión o de control y especificar la forma de sus relaciones recíprocas, así como su rango de variación implícita o explícitamente. Además, se debe definir la función objetivo en relación a las variables de control convenientes. Finalmente establecer las restricciones que deben cumplir estas variables.

1.2.3 Formulación del Modelo Matemático

Una vez que el problema ha sido definido convenientemente, el paso siguiente es formular un modelo matemático que represente fielmente la estructura esencial del problema y del que fácilmente se pueda lograr la solución por medio de la aplicación de un método bien conocido. El modelo es una abstracción adecuada de la realidad, que preserva la estructura esencial del problema. El análisis proporcionará información, no sólo del problema original, sino también de otros problemas que presenten la misma estructura formal.

El modelo producirá resultados que posteriormente deben ser validados con el problema original, porque si este no ha sido modelado convenientemente, la solución puede conducir a resultados dudosos o completamente erróneos, ejemplo de esta situación es el caso de un modelo de programación lineal del que se obtenga una solución no acotada como consecuencia de haber omitido en el modelo cierta restricción del problema. A continuación se analizan algunas características específicas de los modelos de optimización, con la finalidad de proponer una clasificación adecuada y útil para una identificación posterior de los modelos y métodos que se estudian posteriormente. En los modelos de optimización se distinguen tres componentes principales:

1. El conjunto de variables del problema.

2. La función objetivo que se va a optimizar.

3. El dominio de definición para las variables del problema. Este dominio esta determinado por las restricciones impuestas, a los valores que pueden alcanzar las variables.

La solución óptima es un valor numérico que toman las variables, que satisfacen las restricciones y simultáneamente optimizan la función objetivo.

En la clase de problemas de optimización - los del Cálculo de Variaciones- se busca determinar una función o curva, que satisfaga un conjunto de restricciones y optimice a cierta expresión funcional del conjunto de curvas soluciones factibles.

Además los problemas pueden ser restringidos o sin restricciones. En los problemas con restricciones cuya formulación es sencilla en forma matemática y junto con la naturaleza de las expresiones restrictivas, pueden adoptar muy diversas formas, por ejemplo, pueden ser expresiones algebraicas o trascendentes, igualdades o desigualdades, funciones lineales o no lineales, el dominio de las variables puede estar dado por un conjunto discreto o continuo. En algunos casos, las restricciones también pueden ser derivadas o integrales definidas.

De acuerdo con lo anterior, es posible construir dos árboles que estructuran la clasificación de los Modelos de Optimización, los cuales ilustraremos con las Fig. 1.2 y 1.3.

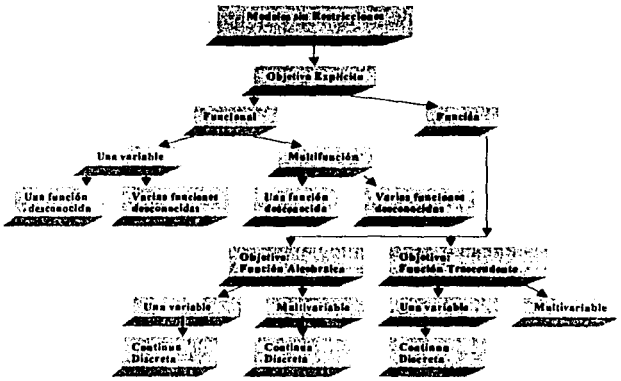

Figura. 1.2 Clasificación de los Modelos de Optimización sin Restricciones.

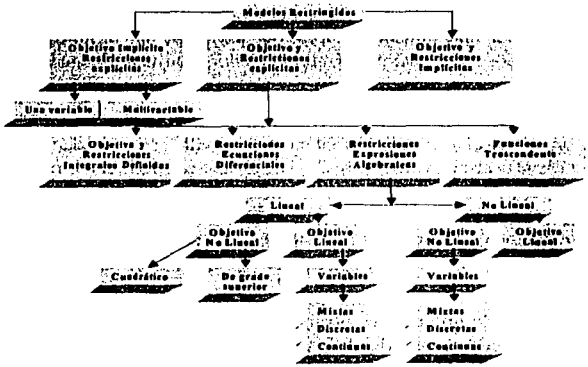

Figura. 1.3 Clasificación de los Modelos de Optimización con Restricciones.

De este modo es posible distinguir ciertas ramas del árbol como representantes de ciertas clases específicas de problemas, cuyo procedimiento de solución es un desarrollo matemático bien conocido.

Por ejemplo, los modelos pertenecientes a la rama de optimización restringida, para los cuales tanto las restricciones como el objetivo se pueden representar en forma algebraica, constituyen la parte de la teoría de optimización conocida como Programación Matemática.

Un segundo ejemplo se refiere a la clase de problemas donde su función objetivo explícita se expresa con una integral definida: caso de un objetivo funcional con otras condiciones adicionales o sin ellas. La solución de estos modelos cae en el dominio del Cálculo de Variaciones Clásico.

Otro ejemplo que vamos a examinar son los modelos restringidos, en los cuales es imposible expresar las funciones restrictivas. Así como las funciones objetivo, concisamente valiéndose de funciones matemáticas. Las técnicas aplicadas pertenecen al dominio de los Modelos de Investigación directa.

En esta clase de modelos se hallan realizaciones de procesos estocásticos, por ejemplo, los fenómenos de espera, analizados mediante una simulación en la computadora, donde se pueden variar los parámetros de entrada y efectuar la simulación para cada conjunto de parámetros y de esta forma estimar una Medición de la Efectividad asociada con la salida de cada corrida.

Si el problema es la elección de un conjunto de parámetros de entrada que optimice la Medida de Efectividad, entonces en este caso se requiere una técnica de investigación directa capaz de hallar el óptimo que colateralmente minimice el número de ensayos o corridas de simulación.

1.3 Tipos de Problemas

Esencialmente se tienen tres tipos de problemas: Problemas de Programación lineal, Problemas sin Restricciones y Problemas con Restricciones; estos últimos dos tipos de problemas comprenden la programación no lineal.

1.3.1 Programación Lineal

La Programación Lineal, es sin duda el mecanismo más natural, para formular una extensa variedad de problemas con poco esfuerzo.

Un problema de programación lineal es caracterizado, como su nombre lo indica, por funciones lineales de variables desconocidas; es decir, la función objetivo es lineal en las variables desconocidas, y las restricciones son igualdades o desigualdades lineales, las cuales están en función de las variables desconocidas.

Las formulaciones de la programación lineal son populares, porque la matemática es más amigable, la teoría es más rica, y el cálculo es más simple para los problemas lineales que para algunos no lineales. Aunque éstas no son las principales razones que hacen popular a la programación lineal. En términos de propiedades matemáticas y computacionales, hay más clases de problemas de optimización, que problemas de programación lineal, que tienen teorías muy elegantes y potentes, para los cuales hay algoritmos muy efectivos que están disponibles. Por lo que la característica que hace más popular a los problemas de programación lineal son la fase de la formulación más que la fase de solución. En la práctica, un gran número de restricciones y funciones objetivos son lineales.

Por ejemplo, si se formula un problema con una restricción, donde la función objetivo es una cantidad de dinero, que es distribuida en dos formas diferentes, sujeta a un presupuesto; la restricción toma la siguiente forma $x_1 + x_2 \leq B$, donde $x_i, i = 1, 2$ son las cantidades distribuidas a la actividad i, y B es el presupuesto.

De manera semejante, si el objetivo es por ejemplo, el peso máximo, entonces esto puede ser expresado como $w_1x_1 + w_2x_2$, donde $w_i, i = 1, 2$ es el peso del producto i. En general, el problema puede ser expresado como

$$\text{máx } w_1x_1 + w_2x_2$$

$$\text{sujeto a} \{ \; x_1 + x_2 \leq B$$

$$x_1 \geq 0, x_2 \geq 0$$

Este es un programa lineal elemental. La linealidad de la restricción del presupuesto es extremadamente natural en este caso y representa simplemente una aproximación para una forma más funcional.

1.3.2 Problemas sin Restricciones

Se puede pensar que los problemas de optimización sin restricciones carecen de propiedades estructurales. Lo que es totalmente contrario, por dos razones. La primera: sí el dominio del problema es abarcar todas las variables de decisión relevantes, entonces puede ser sin restricciones, por otro lado, algunas restricciones representan delimitaciones artificiales del dominio, y cuando el dominio es ampliado, las restricciones desaparecen. Por ejemplo:

Considere el siguiente problema con restricciones

$$\text{mín } f(x)$$

$$\text{sujeto a } h(x) = 0$$

donde $x \in E^n, h(x) \in E^m, m < n$. Aplicando el método de penalidad cuadrática, se puede resolver un problema sin restricciones

$$\text{mín } f(x) + \mu \|h(x)\|^2$$

para alguna μ grande. El Método de penalidad ha surgido nuevamente, ya que por un tiempo fué rechazado, pero ahora ha sido retomado.

Una segunda razón, es que los problemas con restricciones algunas veces es fácil convertirlos en problemas sin restricciones. Por ejemplo, el hecho de que las restricciones sean igualdades, limitar su grado de libertad se hace

simple, esto es esencialmente hacer que algunas variables estén en función de otras.

Aparte de que representan una clase significativa de problemas prácticos, el estudio de problemas sin restricciones provee un paso fuerte para casos más generales de problemas con restricciones. Muchos aspectos para la teoría y algoritmos de optimización, son motivados y verificados, primero por el caso no restringido y después para el caso restringido.

1.3.3 Problemas con Restricciones

No obstante dados los argumentos anteriores, muchos problemas conocidos en la práctica son formulados como problemas con restricciones. Esto es un problema complejo, tal como se presenta por ejemplo en el informe de un sistema de producción de una coorporación grande o como la planeación de una institución gubernamental.

Así, en la planeación de problemas, las restricciones de presupuestos son comúnmente impuestas, en el sentido de que no se pueden incorporar en un problema más global.

La forma matemática para un problema es la siguiente: el conjunto S es un subconjunto del espacio n-dimensional.

$$\text{sujeto a} \begin{cases} \text{mín } f(x) \\ h_i(x) = 0,\ i = 1, 2, ..., m \\ g_j(x) \leq 0,\ j = 1, 2, ..., r, x \in S \end{cases}$$

En esta formulación, x es un vector n-dimensional con entradas desconocidas $x = (x_1, x_2, ..., x_n)$, y $f, h_i, i = 1, 2, ..., m$ y $g_j, j = 1, 2, ..., r$, son funciones en términos de las variables $x_1, x_2, ..., x_n$. La función f es la función objetivo del problema y las igualdades, desigualdades y el conjunto S son las restricciones. En algunos casos se asume que las funciones involucradas en el problema son continuas o tienen derivadas continuas. Esto asegura que pequeños cambios en x deja pequeños cambios en otros valores asociados con los problemas.

1.4 Orden de Magnitud

Una medida de la complejidad de un problema de programación es su tamaño. La otra medida es en términos del número de variables desconocidas o el número de restricciones. Se distinguen 3 clases de problemas en términos de su tamaño : problemas de escala pequeña, intermedia y grande. Esta clasificación no es enteramente rígida, pero refleja las diferencias básicas entre el tamaño de los problemas.

David G. Luenberger en su libro " Introduction to Linear and Nonlinear Programming" a principios de los 70´s [15], señala que problemas de escala pequeña tienen 5 o menos variables desconocidas y restricciones; problemas de escala intermedia tienen de 5 a 100 variables; y los problemas de escala grande tienen en el orden de 1000 variables y restricciones.

Mucha de la teoría asociada con la optimización, en particular en la programación no lineal, se enfoca en obtener una solución que satisfaga condiciones necesarias y suficientes. Esta teoría envuelve principalmente el estudio de los Multiplicadores de Lagrange, incluyendo el Teorema de Kuhn-Tucker y sus extensiones. Esta teoría es aplicable sólo para problemas de pequeña escala, por que las ecuaciones resultantes de las condiciones necesarias pueden ser resueltas de manera eficiente si el problema es lo suficientemente pequeño para realizar el cálculo manual.

Hoy en día, las técnicas de búsqueda pueden ser aplicadas eficasmente a problemas de programación no lineal, Luenberger señaló que entre 100 hasta 200 variables, y para problemas de programación lineal que tienen 400 restricciones y 1000 variables, puede ser aplicado este método.

Los problemas de gran tamaño; es decir, los de escala grande pueden ser resueltos si estos poseen características especiales en su estructura, que pueden ser explotados por un método de solución.

Como es de esperarse, el tamaño de los problemas que pueden ser resueltos con éxito, han ido creciendo con el avance de la tecnología y la teoría.

1.5 Algoritmos Iterativos y Convergencia

La característica más importante de una computadora de gran velocidad, es su habilidad para realizar operaciones repetitivas de manera eficiente. Para explotar estas características, la mayoría de los algoritmos diseñados para resolver problemas de optimización grandes, son iterativos. Generalmente, se

propone un vector inicial x_0, y el algoritmo genera un vector x_1, es decir una nueva aproximación. El proceso se repite y se encuentra una mejor solución x_2, continuando este proceso se genera una sucesión de puntos $x_0, x_1, ..., x_k$, que se aproximan a una solución x^*. Para problemas de programación lineal, la serie que se genera es finita, la solución se encuentra exactamente después de un número finito de pasos. Para problemas de programación no lineal, generalmente no se encuentra la solución exacta, pero la sucesión converge. En la práctica, para problemas no lineales, el proceso termina cuando un punto suficientemente cercano al punto solución, se logra.

La teoría de los algoritmos iterativos puede dividirse en tres aspectos. El primero se refiere a la creación del propio algoritmo, no son arbitrarios, están basados en un examen cuidadoso de los problemas de programación, de su estructura interna, y de la eficiencia computacional. El segundo aspecto es la verificación de que el algoritmo genere una sucesión que converga a la solución. Este se refiere al análisis de la convergencia global, en donde surge la pregunta: ¿si el algoritmo inicia lejos de la solución, en algún momento llegará a ésta?

El tercer aspecto apunta hacia al análisis de la convergencia local, y éste se refiere a la rapidez en el cual la sucesión que se genera converge a la solución.

Una algoritmo puede requerir un tiempo considerablemente grande, no solo para la convergencia, sino para reducir el error de una tolerancia aceptable.

Las propiedades de convergencia de un algoritmo iterativo, pueden ser estimadas al realizar numerosos experimentos computacionales con diferentes problemas o por un simple análisis teórico.

En la siguiente sección se plantearán problemas, que creemos que abarcan algunos aspectos típicos.

1.6 Algunos Problemas de Optimización

1.6.1 Ejemplo 1. Ajuste de Curvas

El problema consiste en determinar la curva de crecimiento de una población de bacterias. Los datos son el número de individuos de una especie particular y_i, en intervalos de tiempo t_i. Ver Tabla 1.1.

t_i	0	4	7.5	25	31	48.75	52	58.5	72.7	72.7	78
y_i	8	6	6	7	8	10	13	18	18	33	38
t_i	95	96	108	112	133	136.75	143	156.5	166.7	181	
y_i	76	78	164	175	280	300	320	405	385	450	

Tabla 1.1, Datos del crecimiento de la población de bacterias

Se espera que el crecimiento de esta población quede bien aproximado por un modelo, cuya expresión analítica es

$$y(t) = \frac{x_1}{1 + \exp(x_2 - x_3 t_i)} \qquad con \quad x_3 > 0,$$

Los parámetros que queremos determinar son x_1, x_2, y x_3. La función que deseamos minimizar es:

$$\sum_{i=1}^{20} (r_i(x))^2$$

donde

$$r_i(x) = \frac{x_1}{1 + \exp(x_2 - x_3 t_i)} - y_i, \text{ para } i = 1, ..., 20$$

Gráficamente se observa en la Fig. 1.4

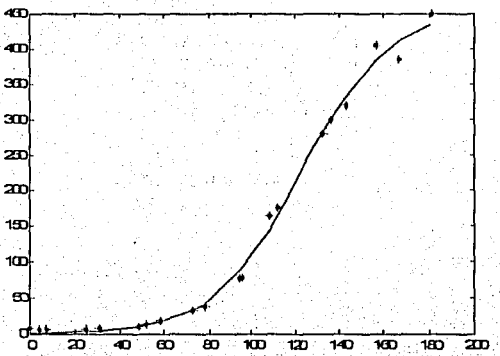

Figura. 1.4 Gráfica del ejemplo 1, (*) datos, (-) curva ajustada.

1.6.2 Ejemplo 2. Suavizamiento de Contornos Poligonales

Este ejemplo surge de un problema de suavizamiento de contornos poligonales, que consiste en minimizar la suma de cuadrados de las longitudes de los lados del polígono y cuyo planteamiento es el siguiente: se tiene un conjunto de puntos $P_i = (u_{2i-1}, u_{2i})$ que determinan el polígono:

$P_1 = (0.233403, 0.218586)$
$P_2 = (0.242231, 0.279365)$
$P_3 = (0.257023, 0.336511)$
$P_4 = (0.307364, 0.374008)$
$P_5 = (0.349139, 0.419062)$
$P_6 = (0.404976, 0.444467)$
$P_7 = (0.41471, 0.405226)$
$P_8 = (0.392464, 0.347906)$
$P_9 = (0.366036, 0.292271)$
$P_{10} = (0.345105, 0.234427)$
...etc.

Lo que se quiere determinar son los $Q_i = (x_{2i-1}, x_{2i})$, con $i = 1, ..., 77$, tal que

$$\min f(x) = \sum_{j=1}^{76} \sqrt{(x_{2j-1} - x_{2j+1})^2 + (x_{2j} - x_{2(j+1)})^2}$$

$$\text{sujeto a } \begin{cases} (u_{2i-1} - x_{2i-1})^2 + (u_{2i} - x_{2i})^2 \leq (a_i)^2 \\ \text{con } i \in R = \{1..77\} \end{cases}$$

es decir, se quiere minimizar la distancia entre estos puntos, sujeto a que dado el diámetro a_i de una vecindad, la distancia de P_i a Q_i debe ser igual o menor a este diámetro. Con $a_i = 0.01$, para $i \in R = \{1..77\}$.

Ver Fig. 1.5 Gráfica inicial y Fig. 1.6 Gráfica de las soluciones.

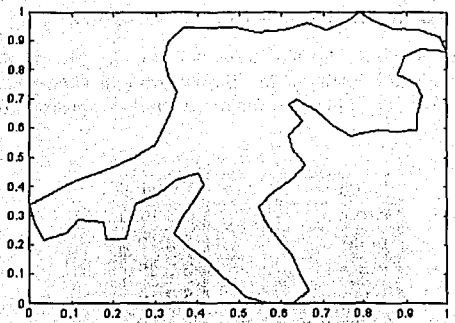

Figura. 1.5 Gráfica de los datos iniciales de "La Habana".

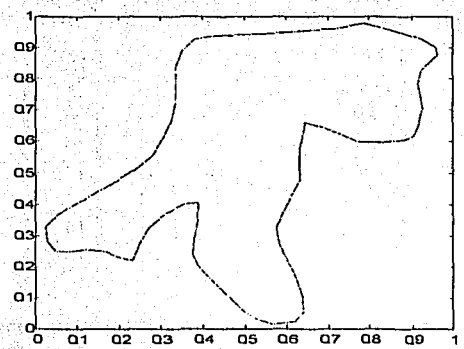

Figura. 1.6 Gráfica de las soluciones de "la Habana".

1.6.3 Ejemplo 3. Problema de la Dieta

Considérese el problema de seleccionar los alimentos para una comida con ciertos requerimientos nutricionales. Supongamos que la cena precocinada de las siguientes clases de alimentos, son aprovechables por los siguientes precios por paquete.

Nombre	Precio
carne	$ 3.19
pollo	$ 2.19
pescado	$ 2.29
jamón	$ 2.89
macarroni y queso	$ 1.89
pastel de carne	$ 1.99
spaghetti	$ 1.99
pavo	$ 2.49

Estos alimentos proveen de los siguientes porcentajes por paquete para el requerimiento mínimo diario de vitaminas A, C, B1 y B2:

	A %	C %	B1 %	B2 %
Carne	60	20	10	15
CHK	8	0	20	20
Pescado	8	10	15	10
Jamón	40	40	35	10
MCH	15	35	15	15
MTL	70	30	15	15
SPG	25	50	25	15
TUR	60	20	15	10

El problema es encontrar la combinación más barata de los paquetes, que además proporcionen en la comida los requerimientos de la semana; es decir, al menos 700% del requerimiento diario de cada nutriente.

Sea X_{Carne} el número de paquetes de carne que es comprado, X_{CHK} el número de paquetes de pollo comprado, y así sucesivamente, entonces el total de costos es:

$$3.19X_{\text{Carne}} + 2.19X_{\text{CHK}} + 2.29X_{\text{Pescado}} + 2.89X_{\text{Jamón}} + 1.89X_{\text{MCH}} + 1.99X_{\text{MTL}} + 1.99X_{\text{SPG}} + 2.49X_{\text{TUR}}$$

El porcentaje total de la vitamina "A" requerida es dado por una fórmula similar excepto que ahora se multiplica cada variable por el porcentaje que da la vitamina "A" en cada alimento.

Así, el porcentaje total de vitamina "A" para el requerimiento diario en la comida:

$$60X_{\text{Carne}} + 8X_{\text{CHK}} + 8X_{\text{Pescado}} + 40X_{\text{Jamón}} + 15X_{\text{MCH}} + 70X_{\text{MTL}} + 25X_{\text{SPG}} + 60X_{\text{TUR}}$$

Esta cantidad debe ser más grande o igual a 700%, de manera similar se obtienen las ecuaciones que involucran las demás vitaminas. De esta forma, el planteamiento es el siguiente:

$$\text{Minimizar } 3.19X_{\text{Carne}} + 2.19X_{\text{CHK}} + 2.29X_{\text{Pescado}} + 2.89X_{\text{Jamón}} + 1.89X_{\text{MCH}} + 1.99X_{\text{MTL}} + 1.99X_{\text{SPG}} + 2.49X_{\text{TUR}}$$

$$\text{sujeto a} \begin{cases} 60X_{\text{Carne}} + 8X_{\text{CHK}} + 8X_{\text{Pescado}} + 40X_{\text{Jamón}} + 15X_{\text{MCH}} + 70X_{\text{MTL}} + 25X_{\text{SPG}} + 60X_{\text{TUR}} \geq 700 \\ 20X_{\text{Carne}} + 0X_{\text{CHK}} + 10X_{\text{Pescado}} + 40X_{\text{Jamón}} + 35X_{\text{MCH}} + 30X_{\text{MTL}} + 50X_{\text{SPG}} + 20X_{\text{TUR}} \geq 700 \\ 10X_{\text{Carne}} + 20X_{\text{CHK}} + 15X_{\text{Pescado}} + 35X_{\text{Jamón}} + 15X_{\text{MCH}} + 15X_{\text{MTL}} + 25X_{\text{SPG}} + 15X_{\text{TUR}} \geq 700 \\ 15X_{\text{Carne}} + 20X_{\text{CHK}} + 10X_{\text{Pescado}} + 10X_{\text{Jamón}} + 15X_{\text{MCH}} + 15X_{\text{MTL}} + 15X_{\text{SPG}} + 10X_{\text{TUR}} \geq 700 \end{cases}$$

$$X_{\text{Carne}} \geq 0, X_{\text{CHK}} \geq 0, X_{\text{Pescado}} \geq 0, X_{\text{Jamón}} \geq 0,$$
$$X_{\text{MCH}} \geq 0, X_{\text{MTL}} \geq 0, X_{\text{SPG}} \geq 0, X_{\text{TUR}} \geq 0$$

Esta última restricción quiere decir que es posible no añadir paquetes de algún alimento, por tanto son ≥ 0.

La solución es $X_{\text{MGH}} = 46.6667$, es la variable que minimiza los costos; es decir, se tienen $46\frac{2}{3}$ paquetes de macarrones y queso.

Se puede checar fácilmente que esto provee el 700% de los requerimientos para las vitaminas "A, B1 y B2" y mucho más vitamina "C" que la necesaria, el costo es $\$1.89 * 46.6667 = \$88.20$.

1.6.4 Ejemplo 4. Refinación de Petróleo

Una refinería divide el petróleo en una serie de compuestos intermedios, la mezcla de éstos compuestos dará un producto final. Dado el volumen disponible de compuestos intermedios, se quiere determinar una mezcla de estos, de tal forma, que los productos resultantes sean lo más provechosos.

Empezaremos definiendo los conjuntos de los compuestos intermedios I, productos finales J, y los atributos K. Los datos relevantes tecnológicamente pueden ser representados por las siguientes variables:

a_i barril del compuesto i disponible para cada $i \in I$

r_{ik} unidades de atributos k contribuidos por barril del compuesto intermedio i para cada $i \in I$ y $k \in K$.

u_{jk} Máximo de unidades permitidas k por barril del producto final j para cada $j \in J$ y $k \in K$

$$\delta_{ij} = \begin{cases} 1 & \text{si el compuesto } i \text{ es permitido en la mezcla del producto } j \\ 0 & \text{en cualquier otro caso} \end{cases}$$

y los datos económicos pueden estar dados por

c_j costo del barril del producto j, $j \in J$

Hay dos colecciones de variables de decisión.

x_{ij} barriles del compuesto intermedio i usado para hacer el producto j para cada $i \in I$ y $j \in J$.

y_j cantidad de barriles del producto j, para cada $j \in J$.

El objetivo es:

$$\text{máx} \sum_{j \in J} c_j y_j$$

la cual es la suma de los costos de J productos. Donde la cantidad de cada compuesto intermedio usado en la mezcla debe ser igual.

$$\sum_{j \in J} x_{ij} = a_i,\ i \in I,$$

la cantidad de productos elaborados, debe ser igual a la suma de la cantidad de componentes que se mezclaron para obtener cada producto.

$$\sum \delta_{ij} x_{ij} = y_j,\ j \in J,$$

para cada producto, el total de atributos que están incluidos en todos los componentes, no bebe exceder del total permitido.

$$\sum_{i \in I} r_{ik} x_{ij} \leq u_{jk} y_j,\ j \in J \text{ y } k \in K,$$

Finalmente tenemos

$$\begin{aligned} 0 &\leq x_{ij} \leq \delta_{ij} a_i \\ 0 &\leq y_j \end{aligned}$$

1.6.5 Ejemplo 5. Portafolio de Inversión (Finanzas)

Este ejemplo resulta del área de finanzas [5] y se refiere a un portafolio de inversión. La formulación está basada en la notación resumida en la Tabla 1.2.

Notación	Definición	Elementos	Dimensión
q	Valor actual de los activos en el mercado.	q_i	$n \times 1$
M	Valuación a Futuro (M_{ji} = valor del instrumento i en el escenario j).	M_{ji}	$s \times n$
p	Probabilidades a priori de los escenarios.	p_j	$s \times 1$
r	Nivel o Tasa de crecimiento del benchmark.	r_j	$s \times 1$
x_L	Límites de la posición inferior.	$(x_L)_i$	$n \times 1$
x_U	Límites de la posición superior.	$(x_U)_i$	$n \times 1$
x	Tamaño o monto de la posición (variables de decisión).	x_i	$n \times 1$
d	Pérdida relativa del portafolio con respecto al benchmark (parte inferior).	d_j	$s \times 1$
u	Ganancia relativa del portafolio con respecto al benchmark (parte superior).	u_j	$s \times 1$

Tabla 1.2. Notación para la formulación del problema

Un portafolio que se compone de posiciones xi en cada activo i alcanza una parte superior en el escenario j de

$$u_j = \text{máx}\left[\sum_{i=1}^{n}(M_{ji} - r_j q_i)x_i,\ 0\right]$$

y una parte inferior en el escenario j de

$$d_j = \text{máx}\left[\sum_{i=1}^{n}(r_j q_i - M_{ji})x_i,\ 0\right]$$

Lo que se quiere es maximizar el rendimiento:

$$\max \sum_{j=1}^{s} 0.25 u_j$$

$$\text{sujeta a} \begin{cases} (0.25)d_j \leq k \\ (xL)_i \leq x_i \leq (xU)_i \end{cases}$$

Para $k = 0, 0.6, 8.25, 25.$ y donde

$$M = \begin{pmatrix} 7 & 21.5 & 19 & 6.5 \\ 7 & 21.5 & 19 & 6.5 \\ 11.6 & 5.2 & 13.8 & 9.4 \end{pmatrix}, \quad r = \begin{pmatrix} 1.20 \\ 1.15 \\ 1.10 \\ 1.05 \end{pmatrix}, \quad q = \begin{pmatrix} 10 \\ 12.5 \\ 8 \end{pmatrix},$$

$$xL = \begin{pmatrix} -2 \\ 0 \\ -1 \end{pmatrix}, \quad xU = \begin{pmatrix} 5 \\ 2 \\ 4 \end{pmatrix}$$

En la Fig 1.7 se muestra la gráfica del valor de la función objetivo de acuerdo a los valores que se listan en la Tabla 1.3.

k	0	0.6	8.25	25
Función Objetivo	0.9375	8.2	23.5	40.25

Tabla 1.3 Valores de la Función Objetivo para cada k

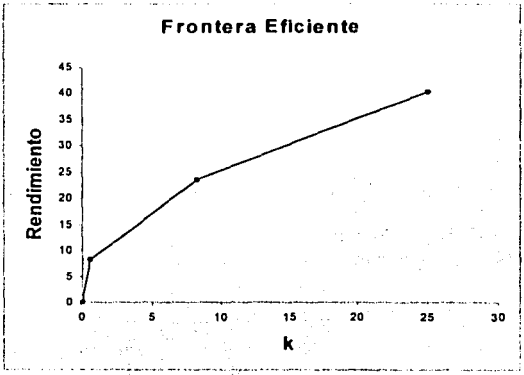

Figura. 1.7 Gráficación del problema de "Portafolios de Inversión".

1.6.6 Ejemplo 6. Eliminación de Restricciones

Considere el siguiente problema

$$\text{mín } f(x_1, x_2, ..., x_{10}) = \sum_{k=1}^{10} k x_k^2$$

$$\text{sujeta a } \begin{cases} 1.5x_1 + x_2 + x_3 + 0.5x_4 + 0.5x_5 = 5.5 \\ 2.0x_6 - 0.5x_7 - 0.5x_8 + x_9 - x_{10} = 2.0 \\ x_1 + x_3 + x_5 + x_7 + x_9 = 10 \\ x_2 + x_4 + x_6 + x_8 + x_{10} = 15 \end{cases}$$

en donde se empleara el método de penalidad. Primero se resolverá como un problema con restricciones y se hará la comparación con los resultados obtenidos para el problema sin restricciones.

Este problema se resolvio para una función de penalidad aproximada, y la función compuesta fue resuelta para varios valores de μ.

Sean

$$\begin{aligned} h_1(x) &= 1.5x_1 + x_2 + x_3 + 0.5x_4 + 0.5x_5 - 5.5 \\ h_2(x) &= 2.0x_6 - 0.5x_7 - 0.5x_8 + x_9 - x_{10} - 2.0 \\ h_3(x) &= x_1 + x_3 + x_5 + x_7 + x_9 - 10 \\ h_4(x) &= x_2 + x_4 + x_6 + x_8 + x_{10} - 15 \end{aligned}$$

Tenemos la siguiente función de penalización:

$$\text{mín } f(x_1, x_2, ..., x_{10}) = \sum_{k=1}^{10} kx_k^2 + [\mu h_1(x)^2 + \mu h_2(x)^2 + \mu h_3(x)^2 + \mu h_4(x)^2]$$

para $\mu = 10, 100, 1000, 10000$.

Ver Tabla 1.4 soluciones

	$\mu = 10$	$\mu = 100$	$\mu = 1000$	$\mu = 10000$	con restricciones
x_1	−0.615661	−1.81409	−1.97887	−1.99597	−1.99788
x_2	3.0577	2.72567	2.67124	2.6655	2.66486
x_3	1.88071	2.31967	2.38089	2.38725	2.38796
x_4	3.0933	3.55611	3.616	3.62218	3.62287
x_5	2.37999	3.14642	3.25284	3.2639	3.26513
x_6	2.49789	2.82002	2.86067	2.86484	2.86531
x_7	2.72409	3.71779	3.85588	3.87022	3.87182
x_8	2.44274	3.06461	3.14887	3.1576	3.15857
x_9	1.8151	2.38516	2.46388	2.47206	2.47297
x_{10}	2.04528	2.60362	2.67963	2.68751	2.68839
Objetivo	388.56251	487.43314	500.88223	502.27630	502.43177

Tabla 1.4 Soluciones de la función objetivo para distintas μ

Lo que se puede concluir es que dado una μ suficientemente grade, es posible obtener una buena aproximación a la función objetivo que se obtiene al resolver el problema con restricciones.

Capítulo 2

Un Lenguaje para Describir los Problemas de Optimización

Una manera de resolver un problema de optimización, es a través de métodos y lenguajes de programación que nos resultan familiares y que tal vez no son los más eficientes.

En mi experiencia como becaria en una Institución, se me asignó la tarea de adecuar unas subrutinas en `FORTRAN`, para encontrar la solución de un problema de optimización derivado de los yacimientos porosos de los pozos petroleros. Por esta información, se sabía que el problema era de Programación No Lineal, y estas subrutinas cumplían con ciertas características del problema, sin embargo no se tenía los suficientes datos del problema para seleccionar un algoritmo apropiado.

El proceso que seguí fue el siguiente: actualizar las subrutinas antes mencionadas, las cuales se encuentran en el libro "Optimization Techniques with `Fortran`"[13]. Por otro lado, estudié un software llamado `VARPRO`, el cual es para estimación de parámetros, ya que en el problema requería de estimar algunos parámetros. También hice uso de paquetería científica como `MATLAB` y `MAPLE`, ya que estos contienen una serie de funciones para resolver problemas de optimización. Posteriormente revicé unas subrutinas en `FORTRAN` de las funciones de Bessel, contenidas en el libro de "Numerical Recipes: The art of scientific computing"[19]., ya que la formulación del problema estaba basada en funciones de Bessel.

Dada la dispersión de la información, bajo el asesoramiento de mi director de tesis, comence una búsqueda minuciosa en internet de algortimos disponibles y accesibles, en particular en la página de internet `www.netlib`[23],

en la cual se puede encontrar algoritmos para mínimos cuadrados, problemas lineales y no lineales; tales como el Método de Newton-Gauss, Marquart-Levengerg, entre otros, disponibles en formato FORTRAN y C. El segundo paso era elegir alguno de estos algoritmos, el cual tenía que ser el adecuado para obtener la solución del problema, esto resultó complejo, puesto que me llevaba a tener que estudiar con detalle cada método (situación que me era difícil, por falta de experiencia en la materia). Otra alternativa era enviar un correo, explicando el problema y a lo mejor alguien lo resolvía. Finalmente esta búsqueda me llevó a encontrar un servidor de internet, para problemas de optimización y un lenguaje de programación.

Lo que se quiere resaltar con este relato, es lo indispensable de explorar las nuevas opciones, trascender los límites de nuestra disciplina, hacer una conjunción con lo ya conocido y no quedarse en una idea. Además de tomar en cuenta que al tener la necesidad de resolver un problema se debe tener en consideración lo siguiente:

1. Identificar que clase de problema
2. Seleccionar un algoritmo adecuado
3. Búsqueda de bibliotecas de software

Ver Fig. 2.1.

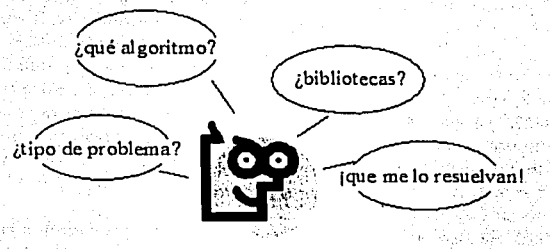

Figura. 2.1 Preguntas que nos hacemos y conclusión

Así, está experiencia condujo al estudio alternativas de solución en internet ,que fueran accesibles para cualquier usuario; es decir, opciones de lenguaje de programación y paquetes que fueran comprensibles de manera inmediata por cualquier persona que quisiera resolver una problema de optimización.

Como ya se mencionó en la introducción, en este trabajo se propone una nueva alternativa de solución de problemas de optimización, a través de un servidor de internet, el cual tiene concentrado una serie de algoritmos, y además un lenguaje de programación matemática, que facilita de una manera muy significativa el acceso a distintos algoritmos y la codificación del problema.

El lenguaje de programación que se estudiará es AMPL[8], y el servidor de internet es NEOS SERVER cuya dirección es:

http://www-neos.mcs.anl.gov/neos[24]

Antes de explicar qué es el NEOS SERVER, se empezará estudiando el lenguaje de programación matemática AMPL, tanto en la parte de programación lineal como no lineal, éste será tomado como uno de los códigos en los que plantearemos los problemas de optimización, formulados en el Capítulo I, que serán resueltos por el Servidor Neos. Cabe mencionar que estas herramientas están disponibles de manera gratuita en la red: para el caso de lenguaje AMPL está disponible una versión para estudiante, la cual contiene ejemplos de modelos, algunos de ellos incluídos en este capítulo y proporcionados en el CD. Para mayor facilidad dejamos alguna notación en inglés, ya que esto facilitara la ejecución y entendiemiento de los modelos.

2.1 AMPL *A Modeling Language for Mathematical Programming.*

El lenguaje de modelación AMPL está diseñado e implementado para ayudar al usuario de computadoras a desarrollar y aplicar modelos de programación matemática.

Por el año de 1940 se utilizó el término "programación", éste fue implementado al encontrar ciertas variables que podían representar el número o nivel de una actividad y cuyo valor sería después determinado. Esto se hizo a través de describir matemáticamente las restricciones en la planeación

o programación de un cierto problema, esto es, se plantearon un conjunto de ecuaciones y desigualdades que involucran las variables; estas ecuaciones y desigualdades son las restricciones. Posteriormente se demostró que esta forma de operar era compleja al especificar restricciones, ya que muy pocas soluciones satisfacían el problema o en el peor de los casos las soluciones eran imposibles de encontrar. Para un mejor resultado, se añadió una función objetivo que dependiera de las variables, la cual estaría restringida a las anteriores ecuaciones planteadas, de tal forma que se pudiera obtener el costo o ganancia y decidir si una solución era mejor que la otra, no importando si muchas soluciones satisfacían las restricciones, era suficiente encontrar una que minimizara o maximizara la función objetivo; es decir, en términos de programación matemática llegó a ser usada para describir el máximo o mínimo de una función objetivo sujeta a restricciones, que dependían de varias variables.

Un problema fue cuando la función objetivo y sus restricciones eran lineales. La programación lineal ha llegado a ser muy importante ya que hay una gran variedad de problemas que pueden ser modelados como un programa lineal y existen una gran variedad de métodos que los resuelven rápidamente, aunque este dependa de cientos de variables. El primer método para resolver problemas de programación lineal fue el Algoritmo Simplex. A pesar de la amplia aplicación de la programación lineal, el asumir linealidad en ciertos problemas resultaba irreal, de esta forma se introdujeron funciones no lineales y derivables; que representan funciones objetivo y restricciones.

2.2 El Lenguaje de Programación AMPL

La programación matemática no es tan simple como correr algún algoritmo en una computadora e imprimir la solución. La serie de etapas que lleva consigo son:

- Formular el modelo que consiste en un sistema abstracto de variables, funciones objetivo y restricciones que representan la forma general del problema que va ser resuelto.
- Colección de datos que definen una o más instancias de problemas específicos.
- Generar una función objetivo específica y las ecuaciones de sus restricciones al modelo y los datos.

- Resolver el problema.
- Analizar los resultados.
- Refinar el modelo y los datos como sea necesario y repetir el proceso.

El lenguaje AMPL ha entrado recientemente en el campo algebraico de lenguajes de modelación para programación matemática; se caracteriza por la semejanza de sus expresiones aritméticas a la notación algebraica y por la generalidad de su aplicación y la notación de sus expresiones. AMPL también extiende su notación algebraica para expresar estructuras comunes de la programación matemática, tales como restricciones de redes de flujo. AMPL ofrece un medio interactivo para aplicar y resolver problemas de programación matemática. Como ya se vio anteriormente, la programación matemática es una técnica para resolver ciertas clases de problemas de optimización, maximizar ganancias o minimizar costos, sujetos a restricciones sobre recursos, capacidades, suministros, demandas, etc. AMPL es un lenguaje específicamente para problemas de optimización, y es muy cercano a la forma en la que se describiera un problema matemáticamente, de tal forma que es muy fácil de pasar de una descripción algebraica a AMPL.

En la siguiente sección se dará un ejemplo de un problema lineal, cuya descripción algebraica será simple, de tal forma que la codificación en AMPL,sea vista desde su forma más simple, a una más compleja.

2.2.1 Un Programa Lineal en dos variables

Una compañía de acero debe decidir como distribuir el tiempo de un proceso de fabricación de la próxima semana . La fabricación toma barras de acero como entrada, y puede producir cualquiera de los productos determinados, los cuales llamaremos bandas y bobinas. Estos dos productos tienen una salida distinta en la línea de rodamiento; es decir, resultan las siguientes cantidades:

Toneladas por hora:

Bandas	200
Bobinas	140

También se tienen distintas ganancias por tonelada :

Bandas	$25
Bobinas	$30

Para complicar el asunto el número de producción ascendió rápidamente.

Max en toneladas: bandas 6000 y bobinas 4000.

La pregunta es como sigue: Si 40 horas de producción son destinadas esta semana, ¿cuántas toneladas de bandas y cuantas de bobinas pueden ser producidas para obtener la mejor ganancia?.

Mientras se dan los valores numéricos, para las cantidades de producción y ganancias por unidad, las toneladas de bandas y bobinas que son producidas, son desconocidas. Estas variables son las variables de decisión, cuyo valor debe ser determinado, de tal forma que se maximicen las ganancias.

El problema lineal tiene como objetivo especificar las ganancias y límites de producción, como una fórmula explícita que envuelvan las variables, cuyo valor pueda ser determinado sistematicamente.

Denotemos las variables de la siguiente forma; sea X_B el número de toneladas de las bandas y X_C para las toneladas de bobinas.

El total de horas para producir estas toneladas esta dada por:

$$(\text{horas de ton por bandas})\, X_B + (\text{horas por ton de bobinas})\, X_C$$

Este número no puede exceder de 40 horas, por lo que las restricciones estarán dadas por lo siguiente:

Si las horas por tonelada es recíproco a las toneladas por hora dadas anteriormente. Obtenemos

$$(1/200)\, X_B + (1/140)\, X_C \leq 40$$

los límites de producción son:

$$\begin{aligned} 0 &\leq X_B \leq 6000 \\ 0 &\leq X_C \leq 4000 \end{aligned}$$

Cabe mencionar que es ilógico hablar de estas variables de holgura cuando son menores que cero en términos computacionales y reales. Como se planteo anteriormente, la forma en que estará dada las ganancias es :

(ganancia x tonelada de bandas) X_B+(ganancia x tonelada de bobinas) X_C.

Esta es la función objetivo a maximizar. Por lo tanto el planteamiento del problema es:

$$\begin{aligned} \text{máx} \quad & 25X_B + 30X_C \\ \text{sujeto a} \quad & \begin{cases} (1/200)\,X_B + (1/140)\,X_C < 40 \\ 0 \leq X_B \leq 6000 \\ 0 \leq X_C \leq 4000 \end{cases} \end{aligned}$$

a partir de los datos proporcionados, es fácil obtener la ganancia por hora de fabricación de cada producto; es decir, la ganancia por hora

$25(200) = 5000$
$30(140) = 4200$

Por lo tanto obtenemos una ganancia mayor con las bandas. En total para maximizar las ganancias se pueden producir hasta un máximo de 6000 toneladas, lo cual implica 30 horas y el tiempo restante para producir bobinas, produciendo un total de 1400 toneladas.

$$25(6000) + 30(1400) = 192000.$$

Gráficamente tenemos la Fig. 2.2. (Las figura 2.3 y 2.4 se incluyen en esta gráfica)

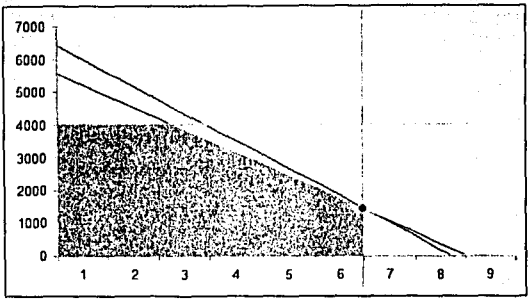

Figura 2.2 Región factible y punto óptimo

Este problema traducido al lenguaje AMPL está presentado en el siguiente listado.

```
>ampl
ampl: var XB;
ampl: var XC;
ampl: maximize profit: 25 * XB + 30 * XC;
ampl: subject to Time: (1/200) * XB + (1/140) * XC <=40;
ampl: subject to B_limit : 0 <= XB <= 6000;
ampl: subject to C_limit: 0 <= XC <= 4000;
ampl: solve;
MINOS 5.4 : optimal solution found.
2 iterations, objective 192000
ampl: display XB, XC;
XB=  6000
XC=  1400
ampl: quit;
>
```

El prompt esta denotado por `ampl:`, a partir de éste se escribe el programa. Como se muestra en el listado, se empieza declarando cada una de

las variables como en la forma algebraica; es decir, se especifican las variables de decisión y posteriormente se proporciona la función objetivo a maximizar y sus restricciones.

Para facilidad de procesamiento, la variable es denotada por la afirmación `var:`, y cada restricción es indicada por "`subject to`" y un nombre, que en este ejemplo son `Time`, `B_limit` y `C_limit`. La multiplicación requiere el carácter (`*`) y el menor o igual es utilizado como (`<=`).

Cabe mencionar que `AMPL` utiliza varios solvers, la forma en que se utilizaran dependerá de la computadora que se tenga; para este problema `MINOS 5.4` será el que obtenga el valor de cada variable, aunque cualquiera puede ser utilizado.

2.3 Un Modelo de Programación Lineal

Dada la dificultad de escribir ciertos problemas con la notación anterior, es posible dar una descripción compacta de la forma general de los problemas; a lo cual llamaremos modelo. Utilizando notación algebraica para la función objetivo y las restricciones.

El siguiente esquema muestra una forma simbólica.

- Conjuntos: Los productos
- Parámetros: La producción y las ganancias
- Variables: Los valores de las soluciones que se quieren determinar
- Una función objetivo: Es el objetivo que se desea maximizar.
- Restricciones: Son los límites que tienen las variables.

El modelo describe un número infinito de problemas de optimización relacionados, donde uno de éstos es el ejemplo antes mencionado; para cada colección de datos que se asocian al problema se obtendrá un modelo. El modelo básico de producción está dado por la forma algebraica.

Definición de variables: X_j = toneladas del producto que se fabrican, para cada $j \in P$

$$\text{Maximizar} \quad \sum_{j \in P} c_j X_j$$

$$\text{Sujeto a} \quad : \quad \begin{cases} \sum_{j \in P} (1/a_j) X_j \leq b \\ 0 \leq X_j \leq u_j, \text{ para cada } j \in P \end{cases}$$

P = un conjunto de productos
a_j= toneladas por hora del producto j, para cada $j \in P$
b = posibles horas de la fabricación
c_j= ganancia por tonelada del producto j, para cada $j \in P$
u_j= toneladas máximas del producto j, para cada $j \in P$

Este planteamiento es más complejo que el problema original; sin embargo, éste puede ser utilizado cuando tenemos más de dos productos, más de 40 variables restringidas inferior y superiormente, lo cual implicaría que la función objetivo consta de 40 variables o más. Este tipo de problemas puede llegar a ser muy complicado.

2.4 El Modelo de Programación Lineal en AMPL.

El objetivo principal de AMPL es acercarse lo más posible a la forma matemática que presentan los problemas; es decir, existen construcciones para cada componente básica del modelo planteado: conjuntos, parámetros, variables, funciones objetivo y restricciones, así como también expresiones aritméticas, suma sobre conjuntos, etc..

Un ejemplo del modelo básico.

```
set P;
param a{j in P};
param b;
param c{j in P};
param u{j in P};
var X{j in P};
maximize ganancia: sum {j in P} c[j]*X[j];
subject to Time: sum{j in P} (1/a[j])*X[j]<=b;
subject to Limit{j in P}: 0 <=X[j] <= u[j];
```

Este listado se guarda como un archivo con extensión .mod. El comando set declara un conjunto, como set P de los productos, los elementos de este conjunto serán proporcionados por otro archivo.

Con el comando param declaramos los parámetros, los cuales pueden ser un simple valor escalar como param b, o una colección de valores indexados por un conjunto, tales como param a {j in P}, se lee : "existe un a_j para cada j en P", lo cual significa que "a" es una colección de valores de los parámetros, uno por cada elemento de P.

Cuando manejamos la notación a_j, en AMPL el índice se traducirá como a[j]. Las variables se declaran var X {j in P}; esto nombrará una colección de variables por cada elemento de P, el valor de estas variables es aquel que se quiere determinar.

La función objetivo estará declarada como

maximize ganancia: sum { j in P} c[j] * X[j];

ganancia es simplemente el nombre que se da a la función, por tanto este es arbitrario. Posteriormente se tiene la suma, la cual se efectuará después que el producto de las variables; es decir, tenemos una suma de productos. Por último tenemos las restricciones:

```
subject to Time: sum {j in P} (1 / a[j]) * X[j] <=b;
subject to Limit {j in P}: 0 <= X[j] <= u[j];
```

Time es la suma sobre el conjunto P que no debe exceder del parámetro b. Limit es una familia de restricciones, existe una restricción por cada elemento del conjunto P, cada X[j] es limitada inferiormente por cero y superiormente por el parámetro u[j]. La forma de indexar un conjunto de parámetros o variables es de la forma {j in P} como en el ejemplo:

```
subject to Limit {j in P};
```

Con esta instrucción se da un restricción por cada elemento del conjunto P; análogo a esto lo realiza:

```
sum { j in P} c[j]*X[j];
```

Es decir, tenemos un producto por cada elemento de P, los productos se sumaran.

El esquema del modelo en AMPL resulta ser general, de tal forma que es posible aplicarlo a cualquier modelo con este tipo de solución. La forma de declarar los conjuntos, parámetros y variables puede ser en el orden que desee

el usuario. Al final de cada instrucción es necesario poner punto y coma, como se presenta en el ejemplo. Como es claro, la forma en que indicaremos $\sum$ será por la palabra sum y los conjuntos estarán dentro de llaves. Para realizar la multiplicación es necesario el carácter *. En lo que se refiere a la sintaxis, AMPL toma como diferentes a las palabras time, Time y TIME.

Ahora, ya indicada la forma en que se debe codificar el problema, es necesario proporcionar los datos correspondientes de manera independiente a través de una tabla como la siguiente:

```
set P := bands  coils;
param :       a       c      u:=
bands       200      25    6000
coils       140      30    4000
param b :=  40;
```

Esta tabla es con base en el problema de dos productos, de tal forma que se declaran los elementos del conjunto P, que en este caso son bandas y bobinas, posteriormente se indican los parámetros, los cuales tomarán distintos valores según el producto; finalmente se define el parámetro b; el símbolo (:=) se lee como "se define como".

2.5 Mejoramiento del Modelo

Hay una forma de poder reducir términos en los nombres, parámetros, variables y restricciones del tal manera que la escritura del modelo sea más sencilla. Las expresiones para indexar son P in PROD, las restricciones sobre las variables son reemplazadas y declaradas en el momento de definir la variable; es decir, después de la expresión var, se hace lo mismo para los parámetros, dentro de la declaración se incluye si estos son positivos o negativos. También se tiene una forma de escribir comentarios de tal forma que el lector pueda entender mucho mejor el problema, los comentarios se indican al inicio con (#).

```
set PROD;       # productos
param rate {PROD}>0;     # toneladas producidas por hora
para avail >= 0;         # horas requeridas en la semana

param profit {PROD};     # ganancia por tonelada
```

```
param market {PROD};     # limite de las toneladas
                         # vendidas en la semana

var Make {p in PROD} >=0, <=market [p];   # toneladas producidas
maximize total_profit: sum {p in PROD} profit[p] * Make [p];
# Función objetivo: total de ganancias por todos los productos

subject to Time: sum {p in PROD} (1/rate[p])*Make[p] <= avail;
# Restricciones: total de horas usadas por
# todos los productos que no pueden
# exceder de las horas disponibles.

Los datos estan proporcionados por la siguiente tabla:

set PROD   :=  bands  coils;
param :  rate   profit market:=
bands    200      25     6000
coils    140      30     4000;
param avail  :=40;
```

Esta nueva forma de escritura en AMPL permite modificaciones, de tal manera que se puede ajustar el modelo a un determinado problema. Estos listados se guardan en un archivo con extensión `.mod` y los datos con `.dat`, éstos archivos formarán parte de la sesión de AMPL.

Después (si es que el modelo no tiene ningún error) se despliega el solver que se utilizó, el número de iteraciones que realizó, que en este caso son 2, y el valor de la función objetivo 192000.

Con los comandos `model` y `data` se especifican los archivos que serán leídos. Al igual que antes damos la instrucción de `solve`; y finalmente se da la instrucción de "`display`" el cual es un comando de salida, inmediatamente después se escribe la variable que deseamos obtener.

```
ampl: model profit1.mod;

ampl: data profit1.dat;
ampl: solve;

ampl: display Make;
Make [*] :=
```

```
bands     6000
coils     1400
;
```

2.6 Mensajes de Error

El lenguaje AMPL detecta varias clases de errores, que se presentan en el desarrollo del modelo, los cuales son reportados como sigue:

Si el error fue de sintaxis dentro de la declaración de las variables, por ejemplo Make para los límites en la declaración de la variable Make, el mensaje de error es:

```
steel.mod, line 8 (offset 250):
syntax error
context: var Make {p in PROD}  >>> 0 <<<  <= market [p];
```

El lenguaje distingue nombres como make y Make, por tanto si se tiene el error de escribir en ganancia[p]*make[p], se recibirá el mensaje

```
steel.mod, line 11 (offset 353):
make is not defined
context : maximize total_profit:
sum { p in PROD} profit [p]* >>> make [p] <<<;
```

El error es indicado por >>> y <<<. Errores tales como usar variables antes de ser declaradas son marcados, de igual forma el punto y coma al final de cada instrucción. Los errores para los datos son señalados de la misma forma que para el modelo.

Si existen errores en los valores de los datos, el mensaje de error será enviado en el momento de escribir ''solve''; como por ejemplo si el tiempo esta dado en forma negativa -40 aparece el mensaje:

```
ampl: model steel.mod; data steel.dat; solve;
error processing param avail:
failed check: param avail is not >=0;
currently the check -40>=0;
```

2.6.1 Límites Inferiores en el Modelo

Siguiendo con el ejemplo anterior de la Producción de bandas y bobinas, dado que no siempre podemos plantear de forma precisa un problema de programación lineal, es posible añadir datos que ayuden a obtener una solución mucho más satisfactoria. Es decir, se añaden parámetros, en éste problema se anadieron las placas como nuevo producto, esta modificación se realiza dentro del archivo de datos de la siguiente forma:

```
set PROD := bandas bobinas placas;
param :  rate ganancia  market;
bandas   200      25       6000
bobinas  140      30       4000
placas   160      29       3500;
param avail := 40;
```

Nuevamente se llama al archivo del modelo y de los datos junto con el comando `solve` y como antes se obtendrá la solución.

```
ampl: model steel.mod; data steel2.dat; solve;
```

Posteriormente aparecerá, el solver que se utilizó, el número de iteraciones que realizó y la función objetivo evaluada.

Es necesario poner un límite inferior sobre la cantidad de producción, y entonces se introduce un nuevo producto. Las placas producidas absorben la capacidad que tenemos para fabricar y que no están tomadas por las bandas, de tal forma que obtenemos una ganancia menor al producir bandas pero mayor a bobinas.

Modificado el modelo, se incrementan las ganancias en determinado tiempo, entonces es posible dejar de producir estos productos, pero en realidad esto no es así, por esta razón se añade al modelo un límite sobre la producción; es decir, se añadirá una colección de parámetros nombrados "`commit`" que representaran este límite sobre la producción, impuesto por las ventas comprometidas, el cambio es ≥ 0 por `>= commit[p]` en la declaración de las variables `Make[p]`. Ver Listado 1 (Apéndice A).

Después de haber obtenido la producción comprometida, es mucho más negociable producir bandas hasta el límite de mercado y posteriormente producir placas con el tiempo restante disponible.

2.7 Añadir Restricciones de los Recursos a el Modelo

El proceso de producción de cierto tipo de acero no es simple, ya que en algunas ocasiones es necesario tener un proceso de recocimiento en el acero, de tal forma que si se empieza el proceso por 200 toneladas por hora, tenemos una producción de bandas, bobinas o placas en velocidades previamente dadas. Otro factor impuesto es que hay solamente 35 horas de recocimiento, tomando en cuenta que estas están incluidas en las 40 horas disponibles.

Para esta clase de situaciones se introdujo al modelo, un conjunto de jornadas (STAGE) de producción. La forma de añadir esto se muestra en el Listado 2 (Apéndice A).

Dado que hay una diferencia en el número de horas disponibles en cada jornada, el parámetro "avail" es ahora indexado por "STAGE", análogamente sucede para la velocidad en la producción en cada jornada, por tanto el parámetro "rate" es indexado sobre ambos "PROD" y "STAGE". En la restricción Time, la forma de indicar la velocidad de la producción del producto "p" en las jornadas "s", nos referiremos como `rate [p,s]`; es decir, en AMPL es posible manejar subíndices de manera combinada, en notación común nos referimos a a_{ps}.

La forma algebraica de escribir este problema es:

$$\text{sujeto a} \quad \sum_{p \in P}(1/a_{ps}) * X_p \leq b_s, \text{ para cada } s \in S$$

En la versión AMPL es:

```
subject to Time { s in STAGE}:
sum {p in PROD} (1/rate[p,s])*Make[p] <=avail [s];
```

La tabla de valores se esta manejando combinaciones de dos subíndices, se incluye el nuevo conjunto STAGE, por lo que se requiere una tabla para este conjunto y otra para el conjunto PROD, como se hizo anteriormente. Ver Listado 2 (Apéndice A).

Ya hecho los cambios, nuevamente se da la instrucción para resolver el problema

```
ampl: reset;
ampl: model ganancia4.mod, ganancia4.dat, solve;
```

y finalmente se pide que desplieguen los valores obtenidos.

```
ampl: display Make.lb, Make, Make.ub, Make.rc;
ampl: display Time;
ampl: quit;
```

La primera instrucción "reset" le indica al programa que borre cualquier otro modelo que anteriormente corrió, de tal forma que pueda leer el nuevo modelo; al finalizar la corrida es necesario salir de AMPL con la instrucción "quit" y entrar nuevamente si se desea correr un nuevo modelo.

Al final del ejemplo anterior se desplegaron los valores duales o sombras de los precios (Make.lb, Make, Make.ub, Make.rc), asociados a la restricción de "Time". El valor dual de una restricción, se refiere al valor de la función objetivo que podría ser perfeccionado si la restricción fuera satisfecha por una pequeña cantidad. Por ejemplo, aquí se esperaría que en algún punto el tiempo aumentara en la etapa recocimiento, que puede producir $1800 de ganancia extra por hora, y también aumentaría el tiempo adicional de producción generando $3200 por hora; decreciendo estos tiempos puede decrecer la ganancia correspondiente. En los comandos de salida como "display", AMPL interpreta el nombre de la restricción sólo como refiriéndose a los valores duales asociados.

También se desplegaron muchas cantidades asociadas con las variables Make. Primero hay un límite inferior Make.lb y un límite superior Make.ub, los cuales en este caso son lo mismo que commit y market. También se muestra el costo reducido Make.rc, el cual tiene el mismo significado con respecto a las restricciones . Se observa que otra vez que en algún punto, el crecimiento de una tonelada en el límite inferior (lo comprometido a producir) para la producción de bobinas, podría reducir ganancias por $1.86. Los niveles de producción para bandas y placas están dentro de sus límites, entonces su costo reducido es esencialmente cero (recordar que $e^{-15} es\ 10^{-15}$), y cambiando sus niveles no hay efecto.

Comparando esta sesión con una previa, observamos que el tiempo adicional de recocimiento restringido, reduce ganancias alrededor de $ 4750.00 y fuerza un cambio sustancial en la solución óptima; mucho más alta es la producción de las placas y más baja la de las bandas, sin embargo la base lógica del óptimo no es obvia.

2.8 Dietas, Combinaciones y Listados

2.8.1 Minimización de Costos

Los ejemplos anteriores se enfocaron en maximizar ganancias. En el siguiente ejemplo se considerará un modelo de programación lineal en donde el objetivo es minimizar costos, las restricciones del modelo son límites inferiores sobre las sumas de ciertas características en la solución.

El problema es de la dieta, en el cual se debe encontrar una combinación de alimentos que satisfagan los requerimientos en la cantidad de varias vitaminas, al igual que en el ejemplo anterior comenzaremos dando un ejemplo pequeño y posteriormente el modelo general, que podrá ser formulado para cualquier clase de problema.

2.8.2 Programa Lineal para el Problema de la Dieta

Considérese el problema de seleccionar los alimentos para una comida con ciertos requerimientos nutricionales. Supongamos que las siguientes clases de cenas son sugeridas por sus precios por paquete.

Lenguaje AMPL	Nombre	Precio
Carne	carne	$ 3.19
CHK	pollo	$ 2.19
Pescado	pescado	$ 2.29
Jamón	jamón	$ 2.89
MCH	macarroni y queso	$ 1.89
MTL	pastel de carne	$ 1.99
SPG	spaghetti	$ 1.99
TUR	pavo	$ 2.49

Estas cenas proveen los siguientes porcentajes por paquete para el requerimiento mínimo diario de vitaminas A, C, B1 y B2:

	A	C	B1	B2
Carne	60%	20%	10%	15%
CHK	8	0	20	20
Pescado	8	10	15	10
Jamón	40	40	35	10
MCH	15	35	15	15
MTL	70	30	15	15
SPG	25	50	25	15
TUR	60	20	15	10

El problema es encontrar la combinación de paquetes más barata , que den en la comida los requerimientos de la semana; es decir, 700% de los requerimientos diarios para cada nutriente.

Sea X_{Carne} el número de paquetes de carne que es comprado, X_{CHK} el número de paquetes de pollo comprado, y así sucesivamente.

Entonces el total de costos es:

$$3.19X_{\text{Carne}} + 2.19X_{\text{CHK}} + 2.29X_{\text{Pescado}} + 2.89X_{\text{Jamón}} + 1.89X_{\text{MCH}} + 1.99X_{\text{MTL}} + 1.99X_{\text{SPG}} + 2.49X_{\text{TUR}}$$

El porcentaje total de la vitamina "A" requerida está por una fórmula semejante, excepto que ahora se multiplica cada variable por el porcentaje que da la vitamina "A" en cada alimento.

Porcentaje total de vitamina "A" para el requerimiento diario en la comida es:

$$60X_{\text{Carne}} + 8X_{\text{CHK}} + 8X_{\text{Pescado}} + 40X_{\text{Jamón}} + 15\ X_{\text{MCH}} + 70X_{\text{MTL}} + 25X_{\text{SPG}} + 60X_{\text{TUR}}$$

Esta cantidad debe ser más grande o igual a 700%. De manera similar se obtienen las ecuaciones que involucran las demás vitaminas.

De ésta forma, el planteamiento es el siguiente:

$$\text{Minimizar } 3.19X_{\text{Carne}} + 2.19X_{\text{CHK}} + 2.29X_{\text{Pescado}} + 2.89X_{\text{Jamón}} + 1.89X_{\text{MCH}} + 1.99X_{\text{MTL}} + 1.99X_{\text{SPG}} + 2.49X_{\text{TUR}}$$

$$\text{sujeto a} \begin{cases} 60X_{\text{Carne}} + 8X_{\text{CHK}} + 8X_{\text{Pescado}} + 40X_{\text{Jamón}} + 15X_{\text{MCH}} + \\ \qquad 70X_{\text{MTL}} + 25X_{\text{SPG}} + 60X_{\text{TUR}} \geq 700 \\ 20X_{\text{Carne}} + 0X_{\text{CHK}} + 10X_{\text{Pescado}} + 40X_{\text{Jamón}} + 35X_{\text{MCH}} + \\ \qquad 30X_{\text{MTL}} + 50X_{\text{SPG}} + 20X_{\text{TUR}} \geq 700 \\ 10X_{\text{Carne}} + 20X_{\text{CHK}} + 15X_{\text{Pescado}} + 35X_{\text{Jamón}} + 15X_{\text{MCH}} + \\ \qquad 15X_{\text{MTL}} + 25X_{\text{SPG}} + 15X_{\text{TUR}} \geq 700 \\ 15X_{\text{Carne}} + 20X_{\text{CHK}} + 10X_{\text{Pescado}} + 10X_{\text{Jamón}} + 15X_{\text{MCH}} + \\ \qquad 15X_{\text{MTL}} + 15X_{\text{SPG}} + 10X_{\text{TUR}} \geq 700 \end{cases}$$

$$\begin{aligned} X_{\text{Carne}} &\geq 0, X_{\text{CHK}} \geq 0, X_{\text{Pescado}} \geq 0, X_{\text{Jamón}} \geq 0, \\ X_{\text{MCH}} &\geq 0, X_{\text{MTL}} \geq 0, X_{\text{SPG}} \geq 0, X_{\text{TUR}} \geq 0 \end{aligned}$$

Esta última restricción, simplemente se reafirma el que cada alimento no puede ser menor que cero.

Hecho el planteamiento algebraico, es posible traducir a AMPL como se indica en el siguiente listado:

```
ampl: var Xbeef >=0; var Xchk>=0; var Xfish >=0;
ampl: Xham>=0, var Xmch>=0; var Xmtl >=0;
ampl: 3.19*Xbeef+2.59*Xchk+2.29*Xfish;
ampl: solve;
CPLEX 2.0: optimal solution; Objetive 88.2
ampl: display Xbeef, Xchk, Xfish, Xham, Xmch, Xmtl, Xtur;
```

Todas las variables son cero salvo `Xmch=46.6667`.

El resultado nos lleva a que $X_{\text{MCH}} = 46.6667$, es la variable que minimiza los costos; es decir, se tienen $46\frac{2}{3}$ paquetes de macarrones y queso.

Fácilmente se puede comprobar que esto provee el 700% de los requerimientos para las vitaminas "A, B1 y B2", y mucho más vitamina "C" que la necesaria, el costo es $\$1.89 * 46.6667 = \$88.20$.

Se puede modificar para obtener el requerimiento exacto de 700%; es decir, si se cambia (>=) por (=), en las restricciones. Sin embargo, la función de costo será más alta que la anterior por una diferencia pequeña, pero aproximadamente se tendrán 19.5 paquetes de pollo, 16.3 de macarroni y queso, 4.3 de pastel de carne y así la función de costo será de \$89.99.

2.9 Un Modelo en AMPL para el Problema de la Dieta

Claramente lo que se debe considerar, es que se deben hacer más modificaciones al problema lineal, en el sentido de obtener una dieta más aceptable. Con lo que probablemente se cambie los conjuntos de comas, vitaminas y por lo tanto las restricciones y los límites.

Como anteriormente se hizo con el modelo de producción, se dará un modelo general de tal forma que sea posible acoplarlo con una variedad de datos específicos. Este modelo trabaja con dos elementos: nutrientes y alimentos. Así, al empezar a escribir el código en el lenguaje AMPL, se comienza declarando los conjuntos correspondientes como sigue:

```
set NUTR;
set FOOD;
```

Los parámetros serán los costos de cada alimento y obviamente estos son positivos.

Se mencionó que en AMPL es posible manejar subíndices. Para el caso particular del costo de un alimento se escribe como `cost["carne"]`, pero es más usual utilizar subíndices "j", en vez de especificar cada miembro del conjunto.

Para generalizar el problema, es necesario especificar para cada alimento los límites superiores e inferiores sobre el número de paquetes en la dieta:

```
param f_min {FOOD}>=0;
param f_max{ j in FOOD}>=f_min[j];
```

Notemos que fue necesario introducir este subíndice "j", que corre sobre el conjunto FOOD en la declaración de `f_max`, en el sentido de decir que el máximo de cada alimento debe ser más grande o igual al correspondiente mínimo.

También es conveniente indicar los límites inferiores y superiores sobre la cantidad de cada nutriente en la dieta como sigue:

```
param n_min{NUTR} >=0;
param n_max { i in NUTR}>= n_min[i];
```

Finalmente, por cada combinación de un nutriente y un alimento se necesita un número que represente la cantidad de nutrientes en cada paquete de

alimentos, de tal forma que nuevamente se está manejando un parámetro que depende de dos conjuntos.

```
param amt{NUTR, FOOD}>=0;
```

Por ejemplo `amt[i,j]` se lee como la cantidad de nutriente i en un paquete del alimento `j`.

Las variables que se desean encontrar representan el número de paquetes que se deben comprar de los diferentes alimentos.

```
var Buy { j in FOOD}>=f_min[j],<=f_max[j];
```

El número de paquetes de un alimento `j` que es comprado, será llamado `Buy[j]`; cualquier solución aceptable tendrá que estar entre `f_min[j]` y `f_max[j]`.

El costo total al comprar un alimento `cost[j]`, es la cantidad de paquetes que se compran `Buy[j]` por el costo de cada paquete.

El objetivo es minimizar la suma de estos productos sobre todos los alimentos `j`:

```
minimize total_cost: sum { j in FOOD} cost [j]*Buy [j];
```

Análogamente, la cantidad total de nutrientes `i` provista por un alimento `j`, `amt[i,j]` es la cantidad de paquetes comprados `Buy[j]` por nutriente. El total de nutrientes `i` provistos, es la suma de este producto sobre todos los alimentos `j`.

```
sum { j in FOOD} amt [i,j] * Buy [j]
```

Finalmente, se necesita especificar que cada suma debe estar dentro de los límites apropiados.

Las restricciones son: `subject to diet {i in NUTR}`; esto nos indica que tenemos una restricción nombrada `diet[i]`, que debe ser impuesta a cada miembro del conjunto `NUTR`.

El resto de las declaraciones son las restricciones para cada nutriente `i`. Las variables deben satisfacer:

`n_min[i] <= sum{j in FOOD} amt[i,j]*Bur[j] <= n_max[i]`. Ver Listado 3 (Apéndice A).

Dado el modelo, los datos con los que trabajaremos son las tablas que se mencionaron anteriormente. Los valores de `f_min` y `n_min` son dados como originalmente se indicó; es decir, `f_min` es cero para cualquier alimento y

el `n_min` (mínimo de nutrientes) es 700 %, en lo que se refiere a los valores de `f_min` y `f_max`, no importa el número asignado para obtener una óptima solución. En la tabla para "`amt`", la notación (`tr`) indica que se transpone la tabla, entonces las columnas corresponden al primer índice (nutrientes), y las filas a la segunda (alimentos). Es decir estamos tomando una matriz de la forma `amt["A", "carne"]`; también es posible hacerlo de manera contraria; es decir, `param amt{FOOD, NUTR}` y en tal caso será `amt[j,i]`.

La forma de resolver el modelo con sus datos es utilizando AMPL en el ambiente de MSDOS; es decir, en el momento que aparezca el prompt se hará lo siguiente:

```
ampl: model diet.mod;
ampl: data diet.dat;
ampl: solve;
```

Después de esta última instrucción, se obtendrá la solución óptima como ya se ha visto en anteriores problemas

```
ampl: display Buy;
Buy [*]:=  MCH 46.6667
```

Dado el resultado, lo que pensaríamos es que se desearía tener una dieta más variada; es decir, que contenga entre 2 y 10 paquetes de alimentos, así como el control de sodio y de calorías, lo cual representaría una buena dieta. Para esto se aumenta lo siguiente: la cantidad de sodio y calorías en cada paquete es de igual forma; el total de sodio no debe exceder 40,000.00 m.g, y el total de calorías debe estar entre 16,000.00 y 24,000.00. Todos estos cambios pueden ser hechos a través de pequeñas modificaciones en los datos. Se introduce la columna NA, se salva y nuevamente se corre el modelo junto con los datos.

Al ejecutar éste nuevo modelo, es posible que se obtenga el mensaje de "`infeasible problem`" (problema infactible), indica que una o unas restricciones fuertes no pueden ser satisfechas al mismo tiempo.

En los anteriores ejemplos, se dio uso de distintos valores para investigar la sensibilidad de una solución óptima al cambiar las restricciones. En este caso no es el óptimo el que presenta un problema si no el solver, ya que éste utiliza la solución anterior para satisfacer las restricciones. Para saber con más certeza el problema de infactibilidad que se presenta, podemos desplegar los siguientes datos:

```
ampl: display diet.lb, diet.body, diet.ub;
:   diet.lb   diet.body    diet.ub
A      700      1993.09      20000
B1     700       841.091     20000
B2     700       601.091     20000
C      700      1272.55      20000
CAL  16000     17222.9       24000
NA       0     40000         40000
```

Análogamente, como en el ejemplo de producción, los valores para `diet.lb` y `diet.ub` son límites inferiores y superiores sobre la suma de las variables en la restricción `diet[i]`; es decir, `n_min[i]` y `n_max[i]` (mínimo y máximo de nutrientes), `diet.body` es la suma de variables. Lo que se puede observar es que la dieta no provee de suficiente vitamina "B2", mientras la cantidad de sodio (NA) ha llegado hasta el tope del límite superior. Una opción es subir el límite máximo a 50,000.00 mg., y así una solución factible es posible. La instrucción se puede dar de la siguiente forma:

```
ampl: letn_max[''NA''] := 50000; solve;
```

y desplegará la solución óptima y el solver utilizado

```
ampl: display Buy;
Buy [*] := BEEF 5.3601
CHK  2
FISH 2
HAM  10
MCH  10
MTL  10
SPG  9.30605
TUR  2
;
```

Todo esto se aproxima a tener una buena dieta, aunque el gasto es de $118.06 comparado con $88.20, el original era un caso menos restringido.

Cabe mencionar que el comando "let", permite hacer modificaciones a los datos.

A pesar de que consideramos que hemos obtenido una buena dieta, aún hay un desacuerdo con el hecho de que es necesario comprar 5.36061 paquetes de carne y 9.30605 de spaghetti. ¿ Qué pasa si sólo podemos comprar el paquete entero?. Se pensaría que podemos redondear el resultado; sin embargo, éste no es un camino factible. A continuación se despliega los límites de las restricciones en el óptimo.

```
ampl: display diet.lb, diet.body, diet.ub;
:    diet.lb    diet.body    diet.ub
A       700       1956.29       20000
B1      700       1036.26       20000
B2      700        700          20000
C       700       1682.51       20000
CAL   16000      19794.6        24000
NA        0      50000          50000
;
```

Dado esto, observamos que 6 paquetes de carne y 10 de spaghetti rebasan el límite permitido para el sodio, mientras que 5 de carne y 9 de spaghetti proveen insuficiente vitamina "B2". Aunque podemos encontrar un entero cercano a la solución que satisface la restricción, no hay alguna garantía de que esta solución sea el mínimo costo.

AMPL hace posible introducir la restricción de números enteros, directamente en la declaración de variables:

```
var Buy { j in FOOD} integer >= f_min[j], <= f_max[j];
```

Esto puede ser de gran ayuda, aunque es posible utilizar un solver que pueda utilizar programación entera. En general, el que se pidan enteros y otras restricciones discretas hace al modelo mucho más difícil de resolver.

2.10 Generalización del Listado y Combinación

Normalmente la gente no sigue un modelo de dieta para su alimentación. Sin embargo, estos modelos pueden ser ajustados a situaciones en donde paquetes

de alimentos y preferencias personales, no juegan un papel importante; por ejemplo, las combinaciones de alimentos para animales. El modelo de la dieta, es un ejemplo intuitivo de la formulación de un problema que aparece en muchos contextos.

Supongamos que queremos reescribir el modelo en una forma más general. Los objetos que eran llamados alimentos y nutrientes en el modelo de la dieta, ahora nos referiremos como "inputs" y "outpus" (entradas y salidas). Para cada entrada `j`, debemos decidir usar una cantidad `x[j]` cuyo valor está entre `in_min[j]` y `in_max[j]`; como resultado se llega a un costo igual a `cost[j]*x[j]`, y se crea `io[i,j]*x[j]` que son las unidades de cada salida `i`. La meta es encontrar el costo más pequeño de una combinación de entradas (`inputs`), que den un rendimiento por cada salida (`output`) `i`; es decir, un número entre `out_min[i]` y `out_max[i]`.

Una clase de aplicación común para este modelo, los salidas son filas de materiales que se mezclan entre ellos.

Los "`outputs`" son los resultados de las mezclas. Las filas de materiales pudieron haber sido componentes de un alimento de animal, o también derivados del petróleo crudo, que son mezclados para hacer gasolina, o las clases diferentes de carbón que son mezclados como "`input`" para un horno para cocinar.

Las cantidades pueden ser números de cosas (sodio o calorías de alimento de animal) o medidas más complejas (presión de vapor o grado de octanaje para gasolina), o tal vez algunas propiedades físicas como el peso y volumen.

En otra menos obvia aplicación, los "`inputs`" son las jornadas de trabajo y los "`outputs`" corresponden a horas trabajadas en ciertos días del mes. Para un trabajo particular de la jornada j, `io[i,j]` es el número de horas que una persona trabaja en un jornada j en un día i, `cost[j]` es el salario mensual para una persona que sigue una jornada j, y `X[j]` es el número de trabajadores asignados a esta jornada. Bajo esta interpretación, el objetivo llega a ser el costo total de la nómina de los jornaleros; mientras que para las restricciones se dice que para cada día `i`, el número total de trabajadores asignados, debe estar entre los límites `out_min[i]` y `out_max[i]`. Listado 4 (Apéndice A).

La misma aproximación puede ser usada en una variedad de contextos de otras jornadas, donde las horas, días y meses pueden ser reemplazadas por otros períodos de tiempo.

Aunque la Programación Lineal puede ser muy útil en aplicaciones como éstas, se necesita tener en mente las suposiciones bajo las cuales está el

modelo de Programación Lineal. Anteriormente se había mencionado la suposición de continuidad, con la cual `x[j]` puede tomar cualquier valor entre `in_min[j]` y `in_max[j]`. Esto puede ser más razonable para la mezcla de alimentos que para las jornadas de trabajo.

Otro ejemplo es escribir la función objetivo como:

```
sum { j in IN} cost[j]*X[j]
```

Se asume linealidad en los costos; es decir, el costo de una entrada es proporcional a la cantidad de entrada usada, y que el costo total es la suma de los costos de las entradas individuales.

Escribimos las restricciones como:

```
out_min[i] <=sum {j in IN} io[i,j]*X[j] <= out_max[i]
```

También se asume que la producción de una salida `i` de una entrada particular, es proporcional a la cantidad de entradas usadas, y que la producción total de una salida `i` es la suma de lo que se produce con las entradas individuales.

Esta "linealidad en la producción" asume que no hay problemas cuando las entradas (`input`) son jornadas, y las salidas(`output`) son horas trabajadas. Pero en el ejemplo de la mezcla, la linealidad es una suposición física acerca de la naturaleza de las filas de materiales y los atributos. Por ejemplo en la aplicación de la refinación del petróleo, se identifico que al añadir el plomo como una entrada, éste tenía un efecto no lineal sobre la cantidad conocida como grado de octanaje en el resultado de la mezcla.

El lenguaje AMPL hace fácil expresar modelos discretos y no lineales, pero cualquier desviación de continuidad o linealidad, hace que una solución óptima sea mucho más difícil de obtener.

2.11 Programación No Lineal

Aunque en cualquier modelo que viola las reglas de linealidad (es decir la función objetivo y las restricciones deben de ser funciones lineales) es no lineal, el término programación no lineal es tradicionalmente usado en un sentido más extenso. Al igual que en la programación lineal, en está hay funciones con variables continuas tanto en la expresión de la función objetivo como en las restricciones. Estás funciones deben representar funciones suaves; es decir, que exista la derivada en cualquier punto del dominio de la función.

Matemáticamente una función suave con cualquier número de variables, debe ser continua y estar bien definido su gradiente en cada punto.

Los problemas de optimización de funciones de esta clase, han sido tomados a parte, ya que éstos son entendibles en comparación con otro tipo de problemas no lineales, además de que han tenido una gran variedad de aplicaciones y son manejados muy bien por los algoritmos ya establecidos. La mayoría de los solvers para programación no lineal que usan los métodos, suponen que las funciones son continuas y diferenciables. Aunque con estas suposiciones, los programas no lineales son típicamente mucho más duros de formular y resolver, que los de programación lineal.

AMPL puede expresar sistemas de ecuaciones no lineales o desigualdades, aunque no hubiese una función objetivo para optimizar. Para este caso existen solvers especializados y es posible obtener una solución factible para el sistema de ecuaciones.

2.12 Ejemplos de Costos No Lineales

En el modelo del transporte, se plantea producir bobinas de acero en 3 fábricas en las siguientes cantidades:

GARY	Gary, Indiana	1400
CLEV	Cleveland, Ohio	2600
PITT	Pittsburgh,Pennsylvania	2900

El total de toneladas es de 6900, las cuales deben ser transportadas en varias cantidades, de tal forma que se distribuyan en 7 camiones de distintas fábricas. El problema es ¿cuál es el costo más bajo para transportar las bobinas de las fábricas a las plantas?

Se hace una tabla de costos de transportación por tonelada.

	GARY	CLEV	PITT
FRA	39	27	24
DET	14	9	14
LAN	11	12	17
WIN	14	9	13
STL	16	26	28
FRE	82	95	99
LAF	8	17	20

Para ilustrar veamos la Fig 2.5 (a), en donde se grafica, el costo de transporte contra la cantidad transportada. Como se observa, es una recta con pendiente `cost[i,j]`.

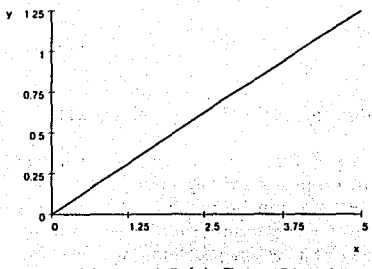

Figura. 2.5 (a) Costo Lineal

Existen otras gráficas que muestran una variedad de otros caminos donde no tenemos linealidad, y en donde el costo de envío pudo depender de la cantidad enviada.

En las siguientes gráficas se da el comportamiento que tienen las variables, por ejemplo:

El costo también tiende a crecer linealmente junto con la cantidad enviada, pero en cierto momento el costo por unidad hace un cambio brusco. Esta clase de funciones se llama linealmente a pedazos, por tanto no es una función suave, véase gráfica de la Fig 2.6 (b).

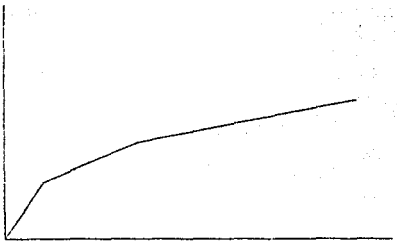

Figura. 2.6 (b) Lineal por segmentos

En la gráfica de la Fig. 2.7 (c) tuvo un brinco. Cuando nada es enviado, el costo es cero, pero en ocasiones hay envíos de todo. El costo es lineal comenzando desde un valor mucho más grande que cero; en este caso hay un costo fijo por ir de i a j, más una variable de costo por unidad enviada. Nuevamente está función no es un problema que pueda ser manejado como un problema lineal, pero tampoco es una función no lineal suave.

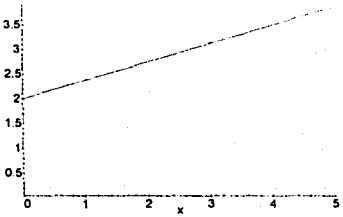

Figura. 2.7 (c) Costo fijo más una variable de costo lineal.

En la gráfica de la Fig. 2.8 (d) tenemos una función suave, aquí se muestra la función de costos cóncava. El incremento en el costo por unidad adicional transportada es más grande que en la primera, pero llega a ser menos en tanto más unidades son transportadas después de un cierto punto, la función de costo es casi lineal. Esto es un alternativa continua para fijar la función

de costo.

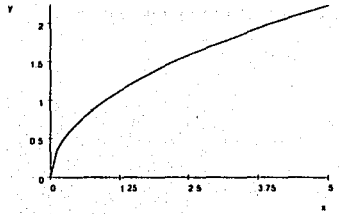

Figura. 2.8 (d) Costos no lineales concavo

En la gráfica de la Fig. 2.9 (e) se representa la función de costo convexa. El costo es casi lineal para pequeños envíos, pero crece hacia infinito cuando los envíos se aproximan a alguna cantidad crítica. Esta clase de función puede ser un modelo para una situación en la cual el costo más bajo de transporte es usada primero, mientras el transporte llega a ser progresivamente más caro en tanto el número de unidades ascienda. Dada esta situación, es claro que el límite superior sobre los envíos es cuando tenemos una cantidad crítica de unidades por transportar.

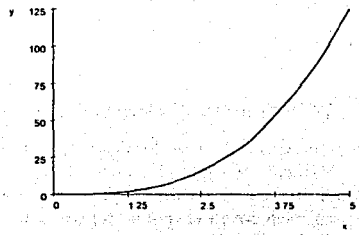

Figura. 2.9 (e) Costos no lineales convexo

Éstas son formas de funciones muy sencillas, pero todo esto dependerá de la clase de situaciones que se estén tratando de modelar.

La gráfica de la Fig. 2.10 (f) demuestra la posibilidad de tener concavidad y convexidad; es decir, estamos combinando características de las anteriores funciones de costo.

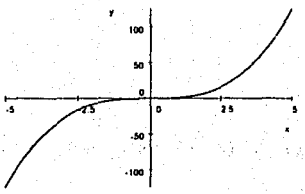

Figura 2.10 (f) Costos no lineales combinados

A diferencia de las funciones lineales, que todas parecen ser la misma excepto por los coeficientes que tienen las variables, las funciones no lineales pueden ser definidas por una infinidad de fórmulas diferentes. Por lo tanto, en la construcción de modelos de programación no lineal se tienen que derivar o especificar funciones, que presenten propiedades en las que se represente la situación que se tiene. Como por ejemplo, en el problema del transporte, por instantes puede ser reemplazado el producto

```
cost[i,j]*Trans[i,j]
```

por

```
rate[i,j]*Trans[i,j]/(1-Trans[i,j]/limit[i,j])
```

donde `rate[i,j]` y `limit[i,j]` son declarados como parámetros. Esta función es esencialmente lineal con pendiente `rate[i,j]` cuando `Trans[i,j]` es pequeño, y se va al infinito cuando `Trans[i,j]` se aproxima a `limit[i,j]`; esto nos lleva a tener un caso como el que se representa en la gráfica de la Fig. 2.9 (e); es decir, la función de costo convexa. Otra forma de aproximar la especificación de una función objetivo no lineal, es concentrarse en la derivadas de las gráficas. En el primer caso la derivada de la función de costo

es constante, esto es por que se puede usar un simple parámetro `cost[i,j]`, para representar el costo por unidad transportada. En el caso de la gráfica de Fig. 2.6 (b) que es lineal por pedazos, la derivada es constante dentro de cada intervalo.

En el caso no lineal, la derivada varía continuamente con la cantidad transportada, por lo que nos será necesario regresar a la formulación lineal del problema del transporte, y cambiar el parámetro de `cost[i,j]` al `Cost[i,j]`;

```
var Cost{ORIG, DEST}          # Costo del transporte por unidad
var Trans{ORIG,DEST}>=   # Unidades por transporte
minimize total_cost:sum{i in ORIG,j in DEST}Cost[i,j]*Trans[i,j];
```

Como se observa, la función objetivo no es muy grande. Se añadieron algunas ecuaciones para especificar como el costo relaciona la cantidad enviada.

```
subject to cost_relation { i in ORIG, j in DEST}:
Cost[i,j] = rate [i,j] / (1-Trans [i,j] / limit [i,j]);
```

Esta ecuación es no lineal ya que envuelve una división por una expresión que tiene una variable. Como notamos `Cost[i,j]` es prácticamente constante en `rate[i,j]` cuando `Trans[i,j]` es pequeño, pero crece rápidamente cuando `Trans[i,j]` se aproxima `limit[i,j]`.

2.13 Otros Recursos de la No Linealidad

Aunque el ejemplo se ha enfocado al costo, cabe mencionar que el hecho de suponer linealidad en cualquier aspecto del modelo, es una herramienta muy fuerte para el problema no lineal

Regresando al modelo de producción, la restricción.

```
subject to Time: sum {p in PROD} (1/rate [p])*Make [p] <=avail;
```

engloba una suposición, en la que el número de horas usadas para hacer un producto crece linealmente con el nivel de producción.

En el modelo del transporte, para el caso donde el costo es lineal, la restricción:

```
subject to Supply { i in ORIG}:
sum { j in DEST} Trans [i,j] = supply[i];
```

engloba solamente una suposición lineal leve, para el hecho de que el total de lo enviado del origen i, es la suma de los envíos a los diferentes destinos.

Estos ejemplos están basados en fenómenos económicos, hay muchos recursos de la no linealidad en modelos de actividades físicas, tal como el refinamiento del petróleo, la transmisión de energía, el diseño estructural o los ejemplos mencionados en el Capítulo I.

En muchas ocasiones en los modelos con funciones no lineales, es difícil poder hacer una relación de tal forma que podamos asumir alguna linealidad, ya que estos suelen ser una consecuencia de relaciones no lineales, en que gobiernan fuerzas, volúmenes, corrientes y así sucesivamente.

Las formas de las funciones no lineales en los modelos físicos, pueden ser fáciles de determinar, puesto que ellos están dados por las leyes de la Física. Por otro lado, estos modelos tienden a envolver formas funcionales e iteraciones más complicadas entre las variables.

2.14 Variables No Lineales

Las variables no lineales son declaradas en el lenguaje AMPL de igual forma que las lineales; sin embargo, hay dos características de las variables que son valores iniciales y sustitución automática, particularmente usadas en los modelos no lineales.

2.14.1 Valores Iniciales de las Variables

Es posible especificar valores para las variables en el lenguaje AMPL, tal como se hace con los parámetros. Antes de la optimización, estos valores iniciales pueden ser desplegados y manipulados por los comandos del lenguaje AMPL. Cuando se da la instrucción de solve; estos valores son pasados a los solvers, los cuales pueden ser usados como un punto de partida en el óptimo. Después de que los solvers han terminado su trabajo, los valores iniciales son reemplazados por los valores óptimos calculados. Todas las características que podamos utilizar para los parámetros en el lenguaje AMPL, es posible utilizarlas en las variables. En la declaración de una variable, se puede especificar valores con una indicación (:=); para el ejemplo del transporte tenemos:

```
var Trans {ORIG, DEST} >=0;   := 1;
```

A cada conjunto `Trans[i,j]` se da un valor inicial de 1 o

```
var Trans{i in ORIG,j in DEST}>=0{limit[i,j]-1};
```

es decir, se inicializa cada `Trans[i,j]` a 1 menos que `limit[i,j]`. Alternativamente, los valores iniciales pueden ser dados en una afirmación de los datos junto con los otros datos para el modelo.

```
var Trans:    FRA   DET   LAN   WIN   STL   FRE   LAF:=
GARY          800   400   400   200   400   200   200
CLEV          800   800   800   600   600   500   600
PITT          800   800   800   200   300   800   500;
```

Cualquier información de los datos para los parámetros, puede ser usada para las variables; es decir, la forma en que asignamos un valor inicial a los parámetros es posible hacerlo de igual manera con las variables. El nombre de la variable comienza por su valor, y el nombre de una restricción comienza por su valor dual asociado. El nombre de la variable o restricción, es simplemente usado en lugar de un nombre de parámetro, lo que se intenta decir es que se tiene el comando "`var`" que puede ser sustituído por "`param`" en el comienzo de cualquier afirmación como por ejemplo en los datos:

```
var Trans:    FRA   DET   LAN   WIN   STL   FRE   LAF:=
GARY          100   100   800   100   100   500   200
CLEV          900   100   100   500   500   200   200
PITT          100   900   100   500   100   900   200;
```

Otro ejemplo, en el modelo de las ganancias de 3 productos, una simple tabla puede dar valores para ciertos parámetros, que en el caso de este modelo son `rate, ganancia, market` y valores iniciales para las variables `Make`:

```
param :    rate  profit  market  Make  :
bandas      200     25    6000   3000
bobinas     140     30    4000   2500
placas      160     29    3500   1500;
```

Los valores iniciales de las variables pueden ser vistos antes, con escribir después de `solve` el comando `display, print, printf`.

Es posible acelerar el trabajo de los solvers al sugerir un buen valor inicial. Esto también es posible hacerlo en programación lineal, pero el efecto es más fuerte en el caso no lineal. La elección de un valor inicial, puede determinar

que el valor de la función objetivo encontrado por el solver es el óptimo. Si no proporcionamos cualquier valor inicial para una variable, entonces AMPL tentativamente asignará el valor cero. Si el solver incorpora una rutina para determinar valores iniciales, entonces éste puede reemplazar los valores de cualquier variable no inicializada, mientras hace uso de los valores de las variables que sí han sido inicializadas. De otra manera, las variables que no han sido inicializadas serán cero. Aunque cero es un obvio punto de partida, éste no es de especial significancia.

2.15 Sustitución Automática de Variables

Como anteriormente se señaló, la sustitución de variables consiste en que ya declaradas las variables, por ejemplo las que representan el costo de transportación, se definen estas variables en términos de otras variables usando restricciones:

```
subject to cost_relation{ i in ORIG, j in DEST}:
Cost [i,j] = rate [i,j] / (1-Trans [i,j] / limit [i,j]);
```

Si la expresión del lado derecho del signo (=) es sustituida, cada vez que aparece `Cost[i,j]`, las variables Cost pueden ser eliminadas del modelo y las restricciones no necesitan ser pasadas al solver.

Hay dos caminos en los cuales se puede llamar a AMPL para hacer tales sustituciones automáticamente:

Primero se escribe:

```
ampl: option substout 1;
```

Se le puede decir a AMPL que busque todas las restricciones "definidas" que tienen la forma anterior, una variable simple a la izquierda del signo igual. AMPL trata de usar muchas de estas restricciones, como sea posible para sustituir variables fuera del modelo.

Cuando esta alternativa es empleada, AMPL busca sustituir como sigue: después de que hemos escrito `solve` y un programa no lineal ha sido generado de un modelo y datos, las restricciones son revisadas en el orden que aparecen en el modelo. Una restricción es identificada como definida, probando que:

- Ésta tiene sólo una variable a la izquierda de un signo (=)

- En la declaración de las variables del lado izquierdo, no se especffica en las restricciones la integrabilidad o si estas están o no acotadas.
- La variables del lado izquierdo no han aparecido en una restricción identificadas como definidas.

La expresión a la derecha del signo (=), es entonces sustituida cada vez que aparezca a la izquierda de la variable, en el orden de las restricciones y la restricción definida es omitida. Todas las anteriores reglas dan a AMPL un camino fácil para rechazar sustituciones circulares, pero esto implica que lo natural es que el número de sustituciones pueda depender del orden de las restricciones.

Un ejemplo más claro es, sí deseamos sustituir especificamente cierta colección de variables, usamos el signo igual en la declaración de las variables:

```
var Cost { i in ORIG, j in DEST}=
rate [i,j]/(1-Trans[i,j]/ limit [i,j]);
```

de esta forma la variable `Cost[i,j]`, será reemplazada por las variables que se encuentran después del igual, siempre y cuando éstas estén previamente declaradas, además de que esta declaración puede aparecer en cualquier orden.

Cabe mencionar que cualquier variable puede ser sustituida, y no afecta matemáticamente el problema de optimización, a menos que tenga un propósito esencial como asociar nombres con expresiones no lineales.

Cuando las mismas expresiones aparecen más de una vez en la función objetivo y restricciones, se sustituye una variable por ésta, de tal forma que pueda hacer al modelo más conciso y más legible.

El lenguaje AMPL también procesa una sustitución eficientemente cuando este genera una representación del programa no lineal para el solver, éste no sustituye literalmente todas las expresiones definidas para cada variable, salva una copia de las expresiones y una indicación de donde su valor está siendo sustituido. Así, la evaluación de las expresiones nunca es repetida innecesariamente.

El sustituir variables reduce el número de restricciones y variables, pero éstas tienden a ser más complejas. Hay circunstancias donde se presentan mejores resultados, si se definen variables que no son sustituidas fuera. Cuando se desarrolla un nuevo modelo es posible que se deba experimentar y determinar cual sustitución da mejores resultados.

2.16 Expresiones No Lineales

Cualquier operador aritmético y funciones aritméticas de AMPL, pueden ser aplicados de igual forma para las variables como a los parámetros (Ver Tabla 2.1).

Estilo usual	Estilo alternativo	Tipo de Operador	Tipo de Resultado
if-then-else		logico, aritmético	aritmético
or	||	lógico	lógico
exist forall		lógico	lógico
and	&&	lógico	lógico
not	!	lógico	lógico
< <= = <> > >=	< <= == != > >=	aritmético	lógico
in not in		objetivo, conjunto	lógico
+ - less		aritmético	aritmético
sum prod min max		aritmético	aritmético
*/ div mod		aritmético	aritmético
+ -		aritmético	aritmético
^	**	aritmético	aritmético

Tabla 2.1 Expresiones No Lineales

El exponente y el `if-then-else` son asociativos derechos, los otros operadores son asociativos izquierdos. Los operadores lógicos de `if-then-else` aparecen después de `if`,y los operadores aritméticos después de `then` y `else`,(Ver Tabla 2.2).

Si en cualquier resultado de una función objetivo o restricción, no se satisfacen las reglas de linealidad, inmediatamente AMPL trata el problema como un programa no lineal; es decir, el proceso que sigue es:

`abs(x)`	valor absoluto $\|x\|$
`acos(x)`	coseno inverso
`acosh(x)`	coseno hiperbólico inverso
`asin(x)`	seno inverso
`asinh(x)`	seno hiérbólico inverso
`atan(x)`	tangente inversa
`atan2(y,x)`	tangente inversa, $\tan^{-1}(\frac{y}{x})$
`atanh(x)`	tangente hiperbólica inversa,
`ceil(x)`	máximo de x (siguiente entero más grande)
`cos(x)`	coseno
`exp(x)`	exponencial
`floor(x)`	mínimo de x (el siguiente entero más pequeño)
`log(x)`	logaritmo natural
`log10(x)`	logaritmo base 10
`max(x,y,...)`	máximo (de dos o más argumentos)
`min(x,y,...)`	mínimo(de dos o más argumentos)
`sin(x)`	seno
`sinh(x)`	seno hiperbólico
`sqrt(x)`	raíz cuadrada
`tan(x)`	tangente
`tanh(x)`	tangente hiperbólica

Tabla 2.2 Operadores

Cuando se escribe "`solve`", el lenguaje `AMPL` trata de pasar a lo largo de ciertas instrucciones que son suficientes para el solver, para evaluar cada expresión en cada función objetivo y restricción junto con las derivadas, si son apropiadas. En caso de usar un solver típico para problemas de programación no lineal, define una función y una restricción en términos de funciones suaves que el solver requiera. La generalidad de las expresiones del lenguaje `AMPL` puede ser engañosa en estas consideraciones. Por ejemplo, si se trata de usar las variables `Flow[i,j]`, representando flujo entre los puntos `i` y `j`, esto induce a escribir expresiones como:

```
cost [i,j]*abs(Flow [i,j])
o
if Flow [i,j] =0 then 0 else base[i,j] + cost[i,j]*Flow[i,j]
```

Éstas no son lineales, pero la primera no es suave (la derivada cambia bruscamente en cero) y la segunda no es continua (su valor brinca rápida-

mente a cero). Entonces si se trata de usar estas expresiones, el lenguaje AMPL no va quejarse y el solver puede regresar y decir que llegó al óptimo, pero los resultados son completamente erróneos.

Funciones que no son suaves, (tales como el valor absoluto, mínimo y máximo), generalmente producen resultados no suaves, lo mismo sucede con `if-then-else`, que son expresiones en las cuales se envuelven variables seguidas de `if`. Aunque hay algunas excepciones en donde es posible preservar funciones sean suaves; por ejemplo, la función x^2 *si*, $x \geq 0$, $-x^2$ *si*, $x < 0$, la cual es escrita en el lenguaje AMPL como:

```
if x[j] >=0 then  x[j] ^2 else -x[j] ^2
```

Ésta función es suave, ya que las funciones seguidas de `then` y `else` son suaves y tienen la misma derivada donde se intersectan en `x[j]=0`.

Otro ejemplo es la función $\log(1+x)/x$ escrita en el lenguaje AMPL `log(1+x[j])/x[j]`, pero el error que puede ocurrir aquí es cuando `x[j]` es cero implica que $\frac{0}{0}$, lo cual es reportado como error; es decir, ésta función no puede ser evaluada exactamente si `x[j]` esta cerca del cero.

Si en vez de esto escribimos:

```
if abs(x[j]) >0.00001 then log(1+x[j])/ (x[j]) else 1-x[j]/2
```

una exactitud alta en la aproximación lineal, es sustituida en pequeñas magnitudes de `x[j]`. Esto no es suave en el sentido matemático, pero es numéricamente lo suficiente cercano para ser considerada como suave y aplicarle algún solver típico.

Aunque las expresiones de el lenguaje AMPL son generalmente suficientes para describir casi todas las funciones no lineales de interés, éstas son limitadas por funciones que tienen una forma matemática conocida .

En el lenguaje AMPL es posible declarar una función como `function`, que será subsecuentemente reconocida en el modelo.

La declaración de `function` especifica un nombre nuevo, requiere de argumentos y describe como algunos programas fuera de el leguaje AMPL, son invocados para calcular valores de las funciones. En la declaración de funciones, puede tener cualquier número de argumentos e itera colecciones de éstos. La conexión entre una función declarada y otro programa, debe ser fijado en un camino que depende en parte del desarrollo en el cual AMPL está comenzado a correr. Como un resultado, las reglas para una función declarada son de cierto modo complejas y técnicas.

2.17 Dificultades de la Programación No Lineal

A pesar de que el leguaje AMPL nos permite formular diversos modelos de optimización no lineal, los solvers no siempre garantizan una solución aceptable, ya que estos son susceptibles a una serie de dificultades que presentan las funciones no lineales.

Se analiza el problema del transporte

```
set ORIG;       # origenes
set DEST;       # destinos

param suply {ORIG} >= 0;
     # cantidad disponible en los origenes
param demand {DEST} >= 0;
     #cantidad requerida en los destinos
check : sum { i in ORIG} supply[i] = sum { j in DEST} demand[j];

param rate { ORIG, DEST} >= 0;  # basado en el costo por envio
param limit { ORIG, DEST} >0;   # limite sobre envio de unidad

var Trans{i in ORIG,j in DEST}>=0,:= 0; #unidades para enviar

minimize total cost: sum { i in ORIG, j in DEST}
rate[i,j]*Trans[i,j]/(1-Trans[i,j]/limit[i,j]);

subject to Supply {i in ORIG}:sum{j in DEST}Trans[i,j]=supply[i];
subject to Demand {j in DEST}:sum{i in ORIG}Trans[i,j]=demand[j];
```

```
param: ORIG: supply :=
GARY 1400 CLEV 2600 PITT 2900;
param: DEST: demand:=
FRA    900
DET   1200
LAN    600
WIN    400
STL   1700
FRE   1100
LAF   1000;
PARAM RATE:  FRA  DET  LAN  WIN  STL  FRE  LAF :=
GARY          39   14   11   14   16   82    8
CLEV          27    9   12    9   26   95   17
PITT          24   14   17   13   28   99   20;
param limit : FRA   DET   LAN   WIN   STL   FRE   LAF  :=
GARY          500   1000  1000  1000  800   500   1000
CLEV          500    800   800   800  500   500   1000
PITT          800    600   600   600  500   500    900;
```

el lenguaje AMPL lee el modelo y sus datos como un programa lineal e invoca un solver, en el mismo camino cuando se escribe `solve`:

```
ampl:  model nltrans.mod;
ampl:  data nltrans.dat;
ampl: solve;
MINOS  5.4 Error evaluanting objective total_cost
cannot compute  8000/0
MINOS 5.4  solution aborted
8 iterations, objetive  0
```

Al obtener este mensaje lo que se observa que hay una división entre cero que pudo presentarse en la expresión:

```
1-Trans[ i,j]/ limit [i,j]
```

es decir, en algún punto es cero. Para este caso, un primer paso es pedir que nos muestre donde `Trans[i,j]` y `limit[i,j]` son iguales.

```
ampl: display { i in ORIG, j in DEST: Trans [i,j] = limit [i,j]};
set { i in ORIG, j in DEST: Trans[i,j] == limit[i,j]}
:=  (GARY, LAF) (PITT, LAN);

ampl:  display Trans ['GARY', 'LAF'], limit ['GARY', 'LAF'];
Trans  ['GARY', 'LAF'] = 1000
limit ['GARY', 'LAF'] = 1000
```

Con la primera instrucción, el leguaje AMPL despliega en qué destino y origen son iguales, posteriormente ya sabiendo esto, se le pide el valor correspondiente a `Limit` y `Trans`; es aquí donde se tiene el problema, ya que al obtener este resultado

```
rate[GARY,LAF]*Trans[GARY,LAF]/(1-Trans[GARY,LAF]/limit[GARY,LAF];
```

$8000/(1-1) = 8000/0$ de tal forma que cuando el solver trata de encontrar la solución óptima, no puede evaluar la función objetivo. Como se utilizó en ejemplos anteriores, con el comando "`let`", es posible cambiar los valores iniciales por cada `Trans[i,j]` que es la mitad de `limit[i,j]`, este camino es sugerido, ya que el comportamiento de un algoritmo de optimización puede ser sensitivo al punto de partida, el solver puede tener mayor éxito con un comienzo diferente:

```
ampl: let { i in ORIG, j in DEST} Trans[i,j] := limit[i,j]/2
ampl:  solve;
MINOS 5.4  the corret point cannot be improved
32 iterations, objective  -7.385903389e+18
```

de esta forma el solver obtiene el valor de la función objetivo, pero ésta resulta ser aún mala, puesto que es menor que -10^{18}, o $-\infty$ para todos los propósitos prácticos.

Si examinamos los valores de `Trans[i,j]/limit[i,j]` en la solución que el solver ha regresado, nos da un indicador de donde ésta la dificultad

```
ampl: display { i in ORIG, j in DEST} Trans [i,j]/ limit[i,j];
Trans [i,j]/limit [i,j][*,*] (tr)
:      CLEV        GARY          PITT:=
DET -6.125e-14   -4.9e-14        2
FRA   0            1.5           0.1875
FRE   0.7          1             0.5
LAF   0.4          0.15          0.5
LAN   0.375        7.03288e-15   0.5
STL   2.9          0             0.5
WIN   0.125        0             0.5
```

Esta tabla nos ayuda a darnos cuenta que muchos pares de transportaciones, tales como `Trans[CLEV,STL]`, significativamente exceden sus límites. Mucho más preciso, `Trans[GARY,FRE]` parecen ser correctos, entonces

`limit[GARY,FRE]`, a partir que su radio, esta dado como 1. Despleguemos estos valores con mayor precisión:

```
ampl:option display_precision  0;
ampl:display Trans[GARY,FRE,limit [GARY,FRE];
Trans[GARY,FRE]= 500.0000000000000028
limit[GARY, FRE]=500;
```

La variable es levemente más grande que el límite, entonces el término del costo tiene un enorme valor negativo, por lo que en la gráfica de la Fig. 2.11 es preciso observar que la función de costo tiende a -∞ del lado derecho de la singularidad en `limit[GARY,FRE]`.

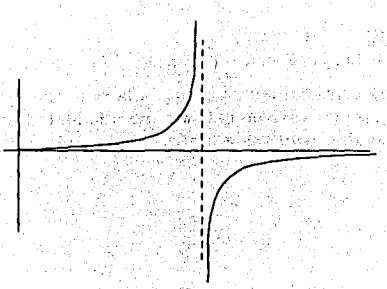

Figura. 2.11 Singularidad en la función de costos

El origen del error en ambas corridas que ocurrieron anteriormente, se debió al suponer que a partir de que la función objetivo que tiende a $+\infty$, como `Trans[i,j]` se aproxima a `limit[i,j]`, el solver guarda `Trans[i,j]` entre cero y `limit[i,j]`. Por lo que debemos forzar una suposición al dar cada `Trans[i,j]` un límite explícito superior, que es levemente menor que `limit[i,j]`, pero sólo lo suficiente; de tal forma que no afecte al óptimo:

```
var Trans { i in ORIG, j in DEST} >=0, <=0.9999*limit[i,j];
```

con esta modificación hemos obtenido mejores resultados para los valores de las variables, con `Trans[i,j]/limit[i,j]` toma un valor menor que 0.96 en cada caso.

Si se cambiara el punto de partida como `limit[i,j]/2` como antes, se encontraría la misma solución, pero el solver necesita solamente la mitad de las iteraciones realizadas. Por lo tanto, es necesario tomar en consideración los problemas que pueden presentarse si las variables salen de su rango. Una forma de observar esto es con el método gráfico, siempre y cuando tengamos la gráfica completa.

2.18 Óptimo Local Múltiple

Otro caso que se debe considerar dado las dificultades de los problemas de optimización, es una función objetivo con una pequeña modificación:

```
minimize total_cost:
```

```
sum { i in ORIG, j in DEST}
rate[i,j]*Trans[i,j]^0.8/(1-Trans[i,j]/limit[i,j]);
```

Se aumento la cantidad transportada a la potencia 0.8, con esto el costo de la función va a ser cóncava en menores cantidades transportadas y convexa en grandes cantidades, como se ve en la gráfica de la Fig. 2.10:

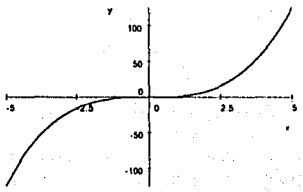

Figura 2.10 (f) Costos no lineales combinados

Nuevamente intentamos resolver este nuevo modelo:

```
ampl: model nltransc.mod;
ampl: data nltrans.dat;
ampl: solve;
MINOS  5.4: solution aborted
0 iterations, objetive 0
```

En ésta ocasión la atención la centraremos en `Trans[i,j] ^0.8`, la cual fue la única expresión que se cambió en el modelo.

El error `pow'(0,0.8)`, la cual denota la derivada de la exponencial, es cero. Cuando `Trans[i,j]` es cero, está función tiene un valor bien definido, pero su derivada con respecto a la variable es infinito. Como resultado, la derivada parcial del total del costo con respecto a cualquier variable en cero, no puede ser ejecutada por el solver, a partir que este las solicita todas las derivadas parciales para aplicar su algoritmo de optimización.

Este es otra variación del problema de la violación de rangos, nuevamente esto puede ser solucionado al imponer límites, de tal forma que la solución este fuera de los puntos que son problema. De esta forma se mueve el límite inferior de cero a un número muy pequeño pero positivo:

```
var Trans { i in ORIG, j in DEST}
>= 1e-10, <= .999*limit[i,j], := 0;
```

Posiblemente también se mueva el punto de inicial lejos del cero, en este ejemplo el solver tiene cuidado y emplea solamente los valores sugeridos como puntos iniciales.

```
ampl: model nltransd.mod;
ampl: data nltrans.dat;
ampl: solve;
MINOS 5.4: optimal solution found.
65 iterations, objetive 427568.1225
ampl: display Trans;

Trans [*,*] (tr)
:        CLEV       GARY       PITT:=
DET     689.601     1e-10      510.909
FRA      1e-10      199.005    700.995
FRE     385.326     326.135    388.54
LAF     885.965     114.035    1e-10
LAN     169.662     1e-10      430.338
STL     469.956     760.826    469.218
WIN      1e-10      1e-10      400
```

por otro lado es posible hacer la modificación en `Trans[i,j]/2` y los resultados son:

```
ampl: let { i in ORIG, j in DEST} Trans [i,j]:= limit [i,j]/2;
ampl: solve;
MINOS 5.4: optimal solution found
40 iterations, objetive 355438.2006
ampl: display  Trans;
Trans [*,*] (tr)
:        CLEV       GARY      PITT:=
DET      540.601    265.509    393.89
FRA      328.599    1e-10      571.401
FRE      364.639    371.628    363.732
LAF      491.262    1e-10      508.738
LAN      301.741    1e-10      298.259
STL      469.108    762.863    468.029
WIN      104.049    1e-10      295.951
```

No sólo la solución es completamente diferente, si no el valor óptimo ha bajado en un 17%. La primera solución no pudo haber minimizado la función objetivo sobre las soluciones que son factibles en las restricciones. Para este caso ambas pueden ser consideradas como soluciones del problema, en el sentido que cada una es un óptimo local. Estos es, cada solución es menos costosa que cualquiera otras soluciones cercanas. Todos los métodos de optimización no lineal comúnmente implementados en los solvers, únicamente localizan óptimos locales empezando en un específico punto de partida, éstos métodos no garantizan encontrar una solución que sea un óptimo global, en el sentido de dar el mejor valor de la función objetivo entre todas las soluciones que satisfacen las restricciones. En general es mucho más difícil encontrar un óptimo global que uno local.

En el caso del ejemplo que se ha trabajado, fue posible encontrar un óptimo local que fuera completamente satisfactorio, lo que implica que si las funciones objetivo y las restricciones satisfacen ciertas propiedades, cualquier óptimo local, es un óptimo global, esto lo garantizamos por la convexidad de la función y la linealidad de las restricciones. En el caso de encontrar más de un óptimo local, es posible acercarse al óptimo global al elegir un valor inicial muy cercano al óptimo, y/o añadir restricciones que limiten la región donde se encuentra el óptimo global.

Para obtener un buen óptimo local, aunque no sea garantía que es un óptimo global, es tratar una serie de puntos de partida sistematicamente,

y tomar el mejor entre las soluciones. Como ejemplo, supongamos que declaramos las variables de la siguiente forma:

```
param alfa >=0,  <=1;
var Trans  { i in ORIG, j in DEST}
>= 1e-10, <=  0.9999*limit[i,j], := alfa * limit[i,j];
```

Por cada elección de alfa, obtendremos un diferente punto inicial y potencialmente una solución diferente, por ejemplo $\alpha \in (0, 1)$ (ver Tabla 2.3):

alfa	total_cost
0	427568.1
0.1	366791.2
0.2	366791.2
0.3	366791.2
0.4	366791.2
0.5	355438.2
0.6	356531.5
0.7	376043.3
0.8	367014.4
0.9	402795.3
1.0	365827.2

Tabla 2.3 Soluciones

La solución para alfa de 0.5 resulta ser la mejor, como anteriormente se vió, aunque resulta notorio el cambio que presenta la función objetivo al cambiar alfa, el segundo valor que se considera es alfa igual a 0.6. Algunos de los métodos más sotisficados para la optimización global, intentan buscar el óptimo a través de puntos de iniciales, pero con un sistema más elaborado, deciden cual es el siguiente punto de partida que se probará.

Otro de los factores importantes que influyen en la eficiencia de un solver no lineal, incluye la formulación del modelo y la elección de las unidades (escala) para las variables.

Como regla, las nolinealidades son más fáciles de manejar cuando aparecen en la función objetivo, en lugar de las restricciones; una de las herramientas eficaces de AMPL es la sustitución automática. Otra regla es, los valores de las variables pueden diferir en al menos un pequeño orden de magnitud.

Los solvers pueden ser engañados cuando algunas variables son en millones y otras en cientos. Algunos solvers escalan el problema para tratar de evitar este problema, aún que el usuario puede ayudar considerablemente a escoger las unidades, en las cuales las variables son expresadas.

Capítulo 3

Servidor de Internet NEOS

Este capítulo se enfocará en explicar cómo utilizar el `Servidor NEOS`(*Network Enabled Optimization System*), para resolver problemas de optimización, utilizando lenguajes de programación como `FORTRAN [6]`y de modelación como `AMPL`, utilizando distintos algoritmos de optimización, los cuales llamaremos `solvers` en todo el desarrollo del capítulo.

En un medio ambiente computacional ideal, el usuario podría formular el problema de optimización y obtener resultados sin preocuparse de los recursos computacionales. Desafortunadamente este medio ideal no es posible, ya que si a la formulación no se le da el cuidado suficiente, un problema razonablemente fácil, puede llegar a ser no manejable. Aún con una apropiada formulación, obtener la solución de un problema de optimización difícil, requiere de un software de optimización sotisficado y acceso a recursos computacionales de gran escala. Modelar procesos físicos en 3 dimensiones, através de un sistema de ecuaciones diferenciables, nos conduce a problemas de optimización que requieren un acceso a recursos computacionales de gran escala. Problemas de optimización discretos y globales se encuetran en esta categoría.

Por tal razón, William Gropp y Jorge J. Moré [10], se interesaron en el desarrollo de ambientes de solución de problemas de optimización, que simplificaran la formulación y el acceso a recursos computacionales.

Este proyecto surgió de una problemática que ellos se plantearon, la cual corresponde a lo siguiente: una vez que el problema ha sido formulado, el primer paso a seguir para resolverlo en un medio computacional típico, es identificar y obtener el software de optimización, el cual puede obtenerse de una librería de software matemático. En algunos casos, el software es de

dominio público y disponible en un sitio de internet.

Una vez que el software ha sido instalado y probado en el medio local, el usuario debe leer la documentación y escribir el código, según lo requiera el software. Usualmente el problema se define en código "FORTRAN o C"; posteriormente debe calcular los valores de la función, derivadas y especificar el modelo. Finalmente el usuario debe compilar, ligar las bibliotecas y ejecutar el código.

De esta manera surge el Servidor de Internet NEOS (ver Fig. 3.1 y 3.2), el cual es un servicio que provee información, software y solución de problemas de optimización. El componente principal del Servidor NEOS es la Guía de NEOS.

El Servidor NEOS es un ambiente que permite al usuario resolver problemas de optimización sobre Internet, en tanto que el usuario solamente debe proporcionar la especificación del problema.

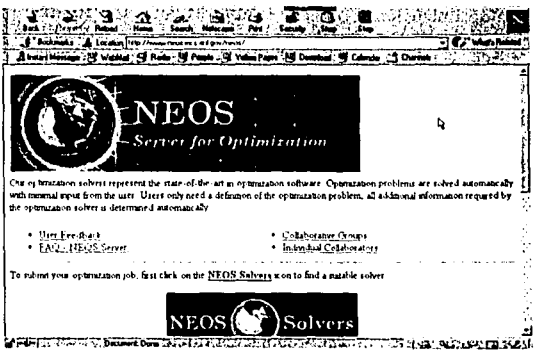

Figura. 3.1 Servidor de Internet NEOS

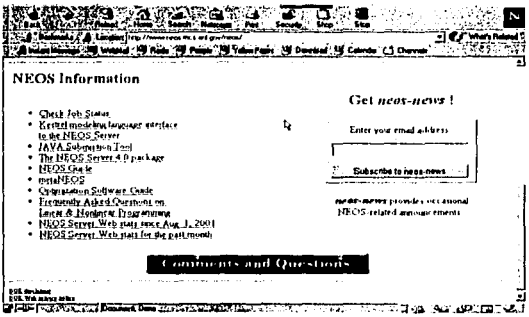

Figura. 3.2 Servidor de Internet NEOS

El usuario del Servidor NEOS no tiene que obtener los algoritmos de optimización, escribir el código o calcular derivadas, ya que este provee una interface que está orientada a recursos computacionales (ver Fig. 3.3).

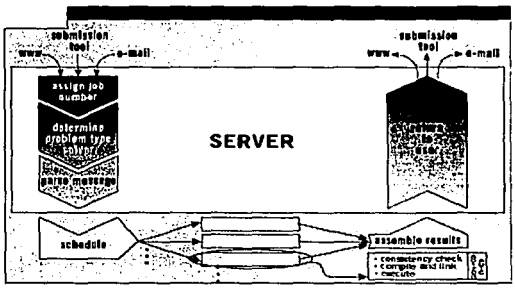

Figura. 3.3 Diagrama del Servidor NEOS

El Servidor NEOS maneja problemas de optimización no lineales y lineales, estos pueden ser con o sin restricciones, es posible resolver problemas no lineales grandes en un tiempo corto, que en software que está instalado en el disco local del usuario, además que el programador debe codificar gradientes

a mano.

A través de NEOS, el usuario tiene acceso a una biblioteca de solvers de optimización con una interface amigable. Por lo que el usuario sólo necesita describir el problema de optimización, toda la información requerida por el algoritmo será determinada automáticamente.

En la página de NEOS se pueden encontrar solvers para distintas áreas, ilustramos esto en las tablas siguientes donde se indica el drive y el lenguaje de programación que se puede utilizar (ver Tablas 3.1 y 3.2).

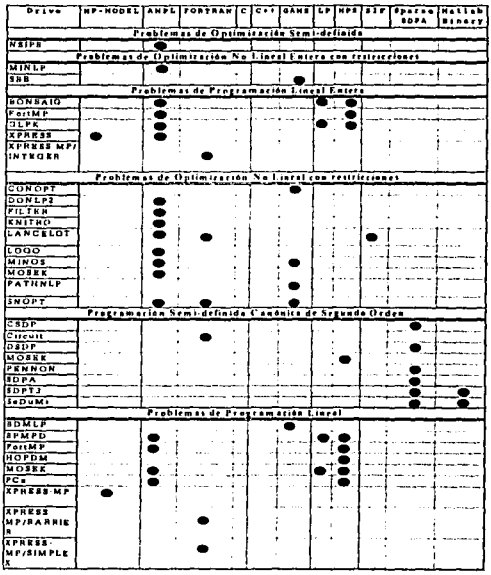

Tabla 3.1 Solvers y Lenguajes de Programación

Drive	MP-MODEL	AMPL	FORTRAN	C	C++	GAMS	LP	MPS	SIF	Sparse SDPA	Matlab Binary
Problemas con restricciones acotadas											
BLMVM		●	●	●							
L-BFGS-B		●									
LANCELOT			●								
TRON		●	●								
L-BFGS-B											
Problemas con restricciones											
CGplus			●								
NMTR			●								
VMLM			●								
Problemas lineales de redes											
NETFLO			●								
RELAX4			●								
Problemas Complementarios											
MILES											
PATH		●	●								
Problemas No Diferenciables											
ACCPM				●							
APPS				●							
BT		●			●						
DFO		●									
NDA		●									
Problemas de programación Estocástica											
AUGMENTED			●								
MSLIP			●								
Optimización Global											
GLOBMIN			●								
Multi-solvers											
AMPL-PRO						●					
GAMS/AMPL						●					

Tabla 3.2 Solvers y Lenguajes de Programación

3.1 Cómo Escribir un Problema de Optimización en AMPL y FORTRAN para enviarlo al Servidor NEOS.

El objetivo de ésta sección es explicar la forma en que se debe codificar un problema de optimización, tal como lo requiere el Servidor NEOS. Esto se ejemplificara con problemas de optimización con o sin restricciones, describiendo cada una de las subrutinas que se requiere, para el caso de FORTRAN, así como la descripción del modelo y datos para AMPL.

3.1.1 Algunos Problemas de Optimización sin Restricciones.

El Servidor NEOS ofrece considerables ventajas, sobre un medio convencional de solvers para la solución de problemas. Considéremos un ejemplo de cómo NEOS, resuelve un problema de optimización de la forma:

$$\min \{f(x) : x \in R^n\}$$

donde $f : \mathbb{R}^n \rightarrow \mathbb{R}$ es una función parcialmente separable; esto es, f puede ser escrita como

$$f(x) = \sum_{i=1}^{nf} f_i(x),$$

donde cada f_i solamente depende de un subconjunto de componentes de x, y nf es el número de elementos de la función.

Han sido desarrollados algoritmos y software que toman ventaja de que una función sea separable, pero estos softwares requieren que el usuario proporcione el gradiente de f, y la estructura de la función separable (una lista de las variables dependientes, de cada elemento de la función f_i).

Código en FORTRAN

Los solvers de NEOS para problemas parcialmente separables , requieren que el usuario especifique el número de variables n, una subrutina `initpt(n,x)` que define el punto inicial, y una subrutina `fcn(n,x,nf,fvec)`, que evalúe los elementos de la función. A partir de que no es necesario proveer el gradiente o la estructura parcialmente separable; ya que el ADIFOR (diferenciación automática para FORTRAN) se encarga de generarlo, el usuario puede concentrarse en las especificaciones del problema. Los cambios para la subrutina `fcn` pueden ser hechos y probados inmediatamente; las ventajas en la facilidad del uso son considerables.

Enseguida señalaremos los puntos que deben ser tomados en cuenta al plantear un problema de optimización:

1. El número de variables.

2. El número de elementos que consta la función objetivo (es decir los elementos que consta la función, si ésta se ve como suma de funciones).

3. Una subrutina que defina el punto inicial.

   SUBROUTINE INITPT(N,X)

   N = ENTERO (entrada),número de variables

   X= DOBLE PRECISIÓN, longitud N (salida), punto inicial

4. Una subrutina que evalue la función objetivo

   SUBROUTINE FCN (N,X,NF,FVEC)

   N= ENTERO (entrada), número de variables

   X= DOBLE PRECISIÓN, longitud N (entrada), vector de variables

   NF= ENTERO (salida), número de elementos de la función

   FVEC= DOBLE PRECISIÓN, longitud NF (salida), elementos de la función.

   Para este tipo de problemas el usuario debe proveer la función objetivo en forma parcialmente separable; es decir, representar a la función objetivo como suma de sus elementos. Se espera que cada elemento sea una función que depende de las variables desconocidas.

   Ejemplo:

Este es un problema de optimización no lineal sin restricciones, donde se desea minimizar una suma de cuadrados.

Una subrutina que defina el punto inicial:

```
C EJEMPLO 1:   (POWELL)
C PUNTOS INICIALES
      SUBROUTINE INITPT(N,X)
C
      INTEGER N
      DOUBLE PRECISION X(1)
      X(1)= 1.0D+0
      END
```

Una subrutina que evalue la función objetivo:

```
C SUBRUTINA DE LA FUNCION A MINIMIZAR
      SUBROUTINE FCN(N,X,F)
      INTEGER N
      DOUBLE PRECISION F
```

```
DOUBLE PRECISION X(1), L(1)
L(1)= 0.1D+0
F=(X(1)+1)**2+(L(1)*X(1)**2+X(1)-1)**2
END
```

Código en AMPL

Para el caso del lenguaje AMPL no hay dificultad, ya que simplemente es plantear el problema en este lenguaje tal como lo explicamos en el capítulo II, y añadir los datos si el problema lo requiere. Se puede utilizar cualquier editor o el que cuenta el lenguaje AMPL.

El archivo que contiene el problema se guarda como `.mod` y si se proporciona un archivo con los datos, este tendrá que ser guardado con la extensión `.dat`.

Ejemplo:

Este es un problema de optimización no lineal sin restricciones, en donde se requiere ajustar una serie de datos.

```
param n>0 integer;
param m>0 integer;
param r>0 integer;

set R1:= {1..n};
set R2:= {1..m};
set R3:= {1..r};

param t {i in R1};
param Y{ j in R2};

var x{ i in R3};

minimize F: sum{i in R1, j in R2}
(Y[j]-(x[1]/(1+exp(x[2]-x[3]*t[i])))) ^2;

data;
var: x:=
1     413
2     5.134
```

```
3     0.0444;
param n:=20;
param m:=20;
param r:=3;
param t:=
1     0
2     4
3     7.5
4     25
5    31;
etc.
param Y:=
1     8
2     6
3     6
4     7
5     8;
etc;
```

3.1.2 Algunos Problemas de Optimización con Restricciones

El Servidor NEOS puede resolver problemas con restricciones, cuando éstas son cotas en las variables:

$$\begin{aligned} &\text{mín } f(x) \\ &\text{sujeto a } \{x_l \leq x \leq x_u \end{aligned}$$

y el problema con restricciones no lineales

$$\begin{aligned} &\text{mín } f(x) \\ &\text{sujeto a } \begin{cases} x_l \leq x \leq x_u \\ c_l \leq c(x) \leq c_u \end{cases} \end{aligned}$$

Para FORTRAN, la especificación de los límites x_l y x_u, está dada en una subrutina en la que se indican las restricciones, mientras que para los problemas

no lineales con restricciones es necesario una subrutina que especifique los límites de las restricciones c_l y c_u, y la función c no lineal $c : \mathbb{R}^n \to \mathbb{R}^m$, que es nombrada como `cfcn(m,x,c)`.

Código en FORTRAN

Se requiere que el usuario haga cinco archivos donde cada uno debe contener una subrutina en FORTRAN; de forma opcional es posible dar un archivo con cualquier otra especificación de parámetros para el solver, si así lo requiere el usuario. Las indicaciones para este proceso de solución son las siguientes:

1. El número de variables.
2. El número total de restricciones.
3. El número de elementos que consta la función objetivo, (es decir los elementos que consta la función, si ésta se ve como suma de funciones).
4. Una subrutina que defina el punto inicial.

   `SUBROUTINE INITPT(N,X)`

   `N` = ENTERO (entrada),número de variables

   `X`= DOBLE PRECISIÓN, longitud `N` (salida), punto inicial
5. Una subrutina que evalúe la función objetivo

   `SUBROUTINE FCN (N,X,NF,FVEC)`

   `N`= ENTERO (entrada), número de variables

   `X`= DOBLE PRECISIÓN, longitud `N` (entrada), vector de variables

   `NF`= ENTERO (salida), número de elementos de la función

   `FVEC`= DOBLE PRECISIÓN, longitud `NF` (salida), elementos de la función en x.

   Nuevamente el usuario debe dar la función objetivo en forma parcialmente separable.
6. Una subrutina que evalúe las restricciones

   `SUBROUTINE CFCN (N,X,M,C)`

   `N`= ENTERO (entrada), número de variables

X= DOBLE PRECISIÓN, longitud N (entrada), vector de variables

M=ENTERO (salida), número general de restricciones

C= DOBLE PRECISIÓN, longitud m (salida), restricciones evaluadas en x.

7. Una subrutina que defina los límites sobre las variables

   SUBROUTINE XBOUND (N,XL,XU)

   N= ENTERO (salida), número de variables

   XL=DOBLE PRECISIÓN, longitud N (salida), límite inferior

   XU= DOBLE PRECISIÓN, longitud N (salida), límite superior

   (Sí $x(i)$ no tiene límite inferior, entonces $xl(i)$ no debe ser fijo, análogamente, si $x(i)$ no tiene límite superior, entonces $xu(i)$ no debe ser fijo).

8. Una subrutina que defina los límites sobre las restricciones

   SUBROUTINE CBOUND (M,CL,CU)

   M= ENTERO (salida), número de restricciones

   CL=DOBLE PRECISIÓN, longitud M (salida), límite inferior

   CU=DOBLE PRECISIÓN, longitud M (salida), límite superior

   (Sí la $i-ésima$ restricción no tiene límite inferior, entonces $cl(i)$ no debe ser fijo, análogamente sí la restricción no tiene límite superior, entonces $cu(i)$ no debe ser fijo).

   Ejemplo:

   Problema de optimización no lineal con restricciones, en el cual se desea minimizar una suma de cuadrados.

- Una subrutina que defina el punto inicial.

```
C Número de variables 26
C Número de restricciones
C
C Asignación del punto inicial
 SUBROUTINE initpt(N,X)
 INTEGER N
 DOUBLE PRECISION X(26)
```

```
 X(1)=0.0
 X(2)=0.0
 X(3)=0.2
 X(4)=0.1
 X(5)=0.9
 X(6)=0.1
 X(7)=1.0
.
.
.
 end
```

- Una subrutina que evalúe la función objetivo

```
C Esta subrutina evalua la función objetivo
 SUBROUTINE fcn(N,X,NF,FVEC)
 INTEGER N,NF
 DOUBLE PRECISION X(*), FVEC(*)
 INTEGER J
 DO 35 J=1,NF
 FVEC(J)= DSQRT((X(2*J-1)-X(2*J+1))**2+(X(2*J)-X(2*(J+1)))**2)
35 CONTINUE
 RETURN
 end
```

- Una subrutina que evalúe las restricciones

```
 C Esta subrutina evalua las restricciones
  SUBROUTINE cfcn(N,X,M,C)
 INTEGER N,M
 DOUBLE PRECISION Z(26)
 DOUBLE PRECISION X(*),C(*)
 INTEGER I
 Z(1)= 0.0D+0
 Z(2)= 0.0D+0
 Z(3)= 2.0D-1
```

```
.
.
.
DO 15 I=1,13
C(I)=DSQRT((Z(2*I-1)-X(2*I-1))**2+(Z(2*I)-X(2*I))**2)
15 CONTINUE
RETURN
END
```

- Una subrutina que defina los límites sobre las variables

```
C Subrutina que asigna las cotas en las variables
 SUBROUTINE xbound(N,XL,XU)
 INTEGER N
 DOUBLE PRECISION XL(*),XU(*)
 INTEGER I
 DOUBLE PRECISION COTA
 PARAMETER (COTA=1.0D+0)
 DO 30 I=2,N-1
 XL(I)=0.0D+0
 XU(I)=COTA
30 CONTINUE
 XL(1)= 0.0D+0
 XU(1)= 0.0D+0
 XL(N)= 0.0D+0
 XU(N)= 0.0D+0
 RETURN
 END
```

- Una subrutina que defina los límites sobre las restricciones

```
C Esta subrutina asigna los límites sobre las restricciones
 SUBROUTINE cbound(M,CL,CU)
 INTEGER M
 DOUBLE PRECISION CL(*),CU(*)
 INTEGER I
 CU(1)= 0.0D+0
 CU(2)= 2.0D-1
 CU(3)= 2.0D-1
 CU(4)= 1.0D-2
 CU(5)= 2.0D-1
 CU(6)= 2.0D-1
 CU(7)= 1.0D-2
 CU(8)= 2.0D-1
 CU(9)= 2.0D-1
 CU(10)= 1.0D-2
 CU(11)= 2.0D-1
 CU(12)= 2.0D-1
 CU(13)= 0.0D+0
 DO 25 I=1,13
 CL(I)= 0.0D+0
25 CONTINUE
 RETURN
 END
```

Código en AMPL

Para este tipo de problemas de optimización con restricciones, es sencillo añadir éstas, ya que sólo es traducir el problema de optimización en el

lenguaje de AMPL. Esto se realiza realiza simplemente añadiendo las restricciones, las cuales estan señaladas por el comando s.t.. El siguiente ejemplo es el mismo problema que se dió para FORTRAN, por lo que hay que hacer notar la facilidad en que pueden ser planteados los problemas en el lenguaje AMPL, sin olvidar las ventajas de FORTRAN.

Ejemplo:

```
# Problema:
# Minimizar una función no lineal
# Número de variables 26
# Número de restricciones 13
param n>0 integer;
param m>0 integer;
param m1>0 integer;
param b<0 integer;
set R3 :={1..m1};
set R1 :={1..n};
set R2 :={1..m};
param a{j in R2};
param u{i in R1};
var x{i in R1};
minimize f:  sum {j in R3}
((x[2*j-1]-x[2*j+1])^2+(x[2*j]-x[2*(j+1)])^2)^1/2;
s.t.  distancia { i in R2} :
(u[2*i-1]-x[2*i-1])^2+(u[2*i]-x[2*i])^2 <= (a[i])^2;
data;
param n:=26;
param m:=13;
param m1:=12;
param a:=
1 0
2 .2
3 .2...
(ver cd anexo)
param:  u :=
1 0
2 0
3 .2 ...
(ver cd anexo)
```

```
solve;
printf {i in R1}:'' %2.15f\n'',x[i]>salida.out;
```

3.2 Interacción con el Servidor NEOS

Existen tres mecanismos para enviar problemas: `e-mail`, `NEOS Submission Tool` y la interface del `Servidor Web NEOS`(ver Fig.3.4). Las interfaces están diseñadas de tal forma que el envío de problemas es intuitivo, y requiere de la mínima cantidad de información. Las interfaces solamente difieren en la forma que la información es especificada y enviada al `Servidor NEOS`.

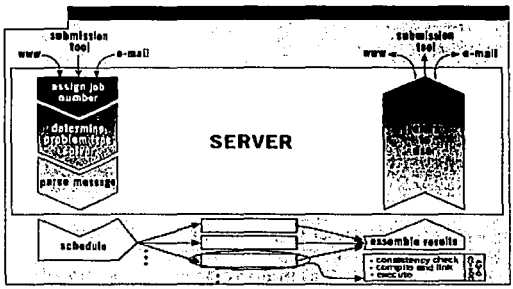

Figura. 3.4 Diagrama del `Servidor NEOS`.

La interface `e-mail` es relativamente primitiva, pero es muy usada, ya que la mayoría de los usuarios cuentan con fácil acceso al `e-mail`.

La herramienta de envío `NEOS (NEOS Submission Tool)`, es una opción rápida basada en una interface para estaciones de trabajo `UNIX`, que proveen un fácil acceso a los algoritmos de optimización disponibles en el `Servidor NEOS`. Esta herramienta se puede obtener del URL

`http://www.mcs.anl.gov/otc/server/submission_tool.html.`

`NEOS Submission Tool`, permite al usuario enviar los problemas al `Servidor NEOS`, directamente de su network local. Una vez que esta herramienta es instalada, como lo mencionamos anteriormente, el usuario tiene acceso a todos

los servicios provistos por el Servidor NEOS,la instalación es inmediata , pero si ésta falla, el remedio usual es correr el script Perl h2ph que cambia los archivos de C dentro de archivos de Perl [22]. Correr el script h2ph es simple, pero éste debe ser hecho por el instalador de Perl, quien es usualmente el administrador del sistema.

El envío de problemas vía NEOS Submission Tool es simple (ver Fig.3.5). El usuario primero debe escoger el tipo del problema de optimización. Una vez que el tipo de problema es seleccionado, el usuario debe escoger un drive.

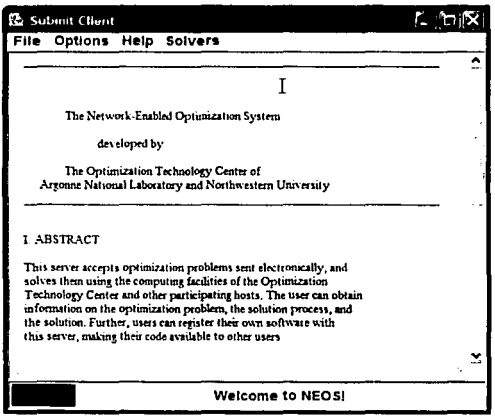

Figura. 3.5 Ventana inicial de Submission-tool

El problema de optimización es especificado por la forma de envío: por ejemplo si se quiere enviar el problema de optimización sin restricciones, se elige el solver correspondiente, el usuario necesita especificar el lenguaje que utilizó para enviar el problema (lenguaje C, FORTRAN, AMPL, etc.), el número de variables n, el número de funciones parcialmente separables nf , los archivos para los puntos iniciales y una subrutina de las evaluaciones de la función, en caso de que el problema sea planteado en FORTRAN (ver Fig. 3.6 y 3.7).

Una ventaja de esta interface (no característico de la interface Web), es

que las subrutinas pueden encontrarse en directorios distintos y el usuario proveer la ruta.

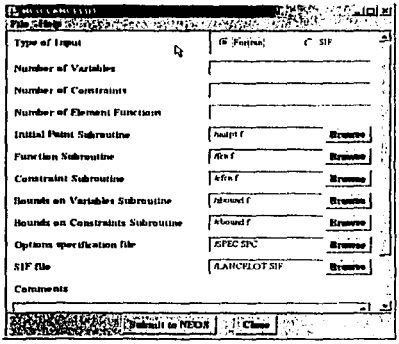

Figura. 3.6 Forma de envío Submission Tool FORTRAN

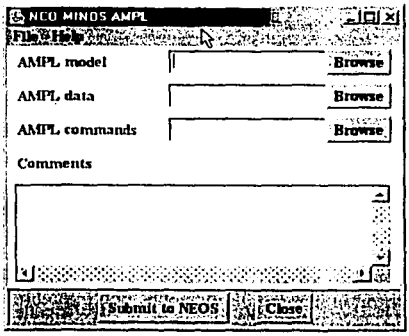

Figura. 3.7 Forma de envío Submission Tool AMPL

Al hacer un envío típico, el usuario recibe información del progreso del

trabajo y la solución. En estas salidas, NEOS muestra que ha contactado una estación de trabajo disponible y transfiere todos los datos a la estación de trabajo. El algoritmo revisa los datos y compila el código del usuario. Si algunos errores son detectados en esta etapa, el compilador regresa unos mensajes de error y la ejecución termina.

Si el código del usuario compila correctamente, las herramientas de diferenciación automática (ADIFOR [2]para FORTRAN) son usadas para generar el gradiente. Una vez que el gradiente es obtenido, el código del usuario es ligado con una biblioteca de software y comienza la ejecución. Los resultados son desplegados en la ventana generada por la herramienta de envío (ver Fig. 5.8).

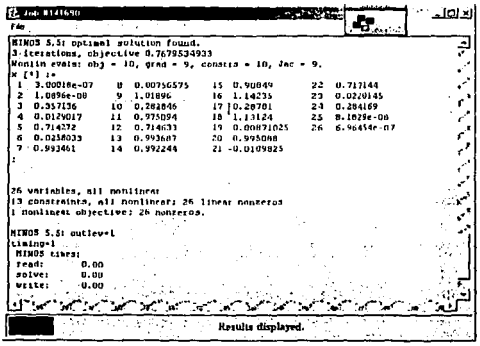

Figura. 5.8 Resultados vía Submission Tool

Como ya se mencionó, otra forma muy intituiva de enviar problemas es vía e-mail, para este caso no se debe incluir símbolos como <,> o paréntesis como [ ], se sigue el patrón siguiente:

Ejemplo 1:

Para este ejemplo seleccionamos como solver a MINOS y el lenguaje de programación es AMPL.

```
TYPE NCO
SOLVER MINOS-AMPL
```

```
BEGIN.MOD
Cuadrado.mod ( Aquí se escribe el modelo)
END.MOD

BEGIN.DAT
Esto es opcional, ya que recordemos que los datos pueden ser
incluídos en el modelo
END.DAT

BEGIN.COM
Este es para una indicación en especial que se desee dar
el usuario;es decir, como indicar alguna salida para los
resultados.
END.COM

BEGIN.COMMENT
Comentarios.txt
END.COMMENT

END-SERVER-INPUT
```

Este e-mail se envía a neos@mcs.anl.gov. Cabe mencionar que para cualquier solver seleccionado, se debe seguir el mismo patrón.

Ejemplo 2:

Este ejemplo es con el formato FORTRAN, y para resolver el problema de optimización no lineal con restricciones se eligió a LANCELOT.

```
TYPE NCO
SOLVER LANCELOT
INPUT= Tipo de salidas
N= Número de variables
M= Número de restricciones
NF= Número de elementos de la función
BEGIN.INITPT
Subrutina del punto inicial
END.INITPT
BEGIN.FCN
Subrutina de la función
END.FCN
```

```
BEGIN.CFCN
Subrutina de las restricciones
END.CFCN

BEGIN.XBOUND
Subrutina de los límites sobre las variables
END.XBOUND

BEGIN.CBOUND
Subrutina de los límites sobre las restricciones
END.CBOUND

BEGIN.SPEC
Archivo con otras especificaciones
END.SPEC

BEGIN.COMMENT
Comentarios.txt
END.COMMENT
END-SERVER-INPUT
```

Enviar un problema al Servidor NEOS no garantiza éxito alguno, pero los usuarios de NEOS pueden resolver problemas de optimización difíciles sin preocuparse de muchos detalles, que son típicos en un medio de computación.

3.3 Ejemplos Implementados en NEOS

Los siguientes problemas son algunos de los planteados en el Capítulo I, los cuales abarcan problemas de programación lineal y no lineal. Primero se plantean en su forma algebraica, además de presentar las gráficas de los datos iniciales, si es el caso. Posteriormente se codificarán en el lenguaje AMPL y FORTRAN. Finalmente se presenta la forma que en son enviados y resueltos en NEOS. Para algunos problemas se presenta la gráfica de las soluciones.

3.3.1 Problema de Optimización No Lineal con Restricciones "Cuadrado"

Este ejemplo surge del suavizamiento de contornos poligonales. Se tiene un conjunto de 13 pares de puntos que describen el contorno de un cuadrado,

el problema consiste en minimizar su perímetro; por lo tanto, se desea minimizar $f(x)$ sujeta a 13 restricciones, donde $f(x)$ y las restricciones son de la forma siguiente:

$$\min f(x) = \sum_{j=1}^{26}((x_{2j-1} - x_{2j+1})^2 + (x_{2j} - x_{2(j+1)})^2)^{1/2}$$

$$\text{sujeto a } \left\{ \begin{array}{c} (u_{2i-1} - x_{2i-1})^2 + (u_{2i} - x_{2i})^2 \leq (a_i)^2 \\ \text{con } i \in R = \{1..13\} \end{array} \right.$$

donde x_i son los valores que se desean obtener y u_i son los puntos iniciales. Donde a toma los siguientes valores:

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}
0	0.2	0.2	0.01	0.2	0.2	0.01	0.2	0.2	0.01	0.2	0.2	0

La gráfica del problema inicial se muestra en la Fig.3.9:

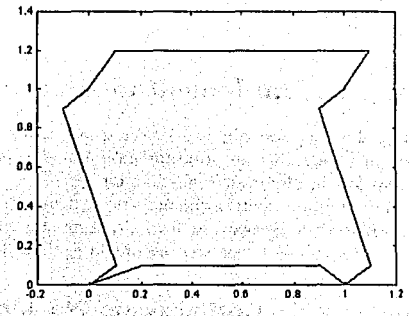

Figura. 3.9 Gráfica de los datos iniciales "Cuadrado".

- Formato en AMPL

```
# Problema:
# Minimizar una función no lineal
# Número de variables 26
# Número de restricciones 13
param n>0 integer;
param m>0 integer;
param m1>0 integer;
param b<0 integer;
set R3 :={1..m1};
set R1 :={1..n};
set R2 :={1..m};
param a{j in R2};
param u{i in R1};
var x{i in R1};
minimize f:  sum {j in R3}
((x[2*j-1]-x[2*j+1])^2+(x[2*j]-x[2*(j+1)])^2)^1/2;
s.t.  distancia { i in R2} :
(u[2*i-1]-x[2*i-1])^2+(u[2*i]-x[2*i])^2 <= (a[i])^2;
data;
param n:=26;
param m:=13;
param m1:=12;
param a:=
...( ver cd anexo)
```

- Formato FORTRAN

```
C Número de variables 26
C Número de restricciones
C
C Asignación del punto inicial
 SUBROUTINE initpt(N,X)
 INTEGER N
 DOUBLE PRECISION X(26)
 X(1)=0.0
 X(2)=0.0
 X(3)=0.2
 X(4)=0.1
```

```
 X(5)=0.9
 X(6)=0.1
 X(7)=1.0
 .
 .
 .
 end
C Esta subrutina evalúa la función objetivo
 SUBROUTINE fcn(N,X,NF,FVEC)
 INTEGER N,NF
 DOUBLE PRECISION X(*), FVEC(*)
 INTEGER J
 DO 35 J=1,NF
 FVEC(J)= DSQRT((X(2*J-1)-X(2*J+1))**2+(X(2*J)-X(2*(J+1)))**2)
35 CONTINUE
 RETURN
 end
C Esta subrutina evalúa las restricciones
 SUBROUTINE cfcn(N,X,M,C)
 INTEGER N,M
 DOUBLE PRECISION Z(26)
 DOUBLE PRECISION X(*),C(*)
 INTEGER I
 Z(1)= 0.0D+0
 Z(2)= 0.0D+0
 Z(3)= 2.0D-1
 .
 .
 .
 DO 15 I=1,13
 C(I)=DSQRT((Z(2*I-1)-X(2*I-1))**2+(Z(2*I)-X(2*I))**2)
 15 CONTINUE
```

```
      RETURN
      END
      C Subrutina que asigna las cotas en las variables
       SUBROUTINE xbound(N,XL,XU)
       INTEGER N
       DOUBLE PRECISION XL(*),XU(*)
       INTEGER I
       DOUBLE PRECISION COTA
       PARAMETER (COTA=1.0D+0)
       DO 30 I=2,N-1
       XL(I)=0.0D+0
       XU(I)=COTA
      30 CONTINUE
       XL(1)= 0.0D+0
       XU(1)= 0.0D+0
       XL(N)= 0.0D+0
       XU(N)= 0.0D+0
       RETURN
       END
     C Esta subrutina asigna los límites sobre las
C restricciones
     SUBROUTINE cbound(M,CL,CU)
     INTEGER M
     DOUBLE PRECISION CL(*),CU(*)
     INTEGER I
     CU(1)= 0.0D+0
     CU(2)= 2.0D-1
     CU(3)= 2.0D-1
```

```
      CU(4)= 1.0D-2
      CU(5)= 2.0D-1
      CU(6)= 2.0D-1
      CU(7)= 1.0D-2
      CU(8)= 2.0D-1
      CU(9)= 2.0D-1
      CU(10)= 1.0D-2
      CU(11)= 2.0D-1
      CU(12)= 2.0D-1
      CU(13)= 0.0D+0
      DO 25 I=1,13
      CL(I)= 0.0D+0
   25 CONTINUE
      RETURN
      END
```

Recordemos que cualquier editor pueder ser utilizado para escribir el problema.

Ya que se tiene el archivo que contiene el modelo se envía al **Servidor NEOS**, para este caso utilizaremos la forma `www.form` (ver Fig. 3.10 y 3.11).

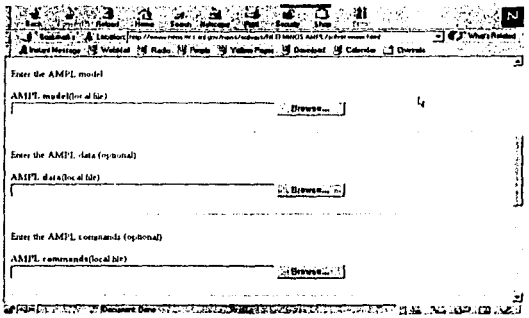

Figura. 3.10 Página de la forma de envío Web "Cuadrado"

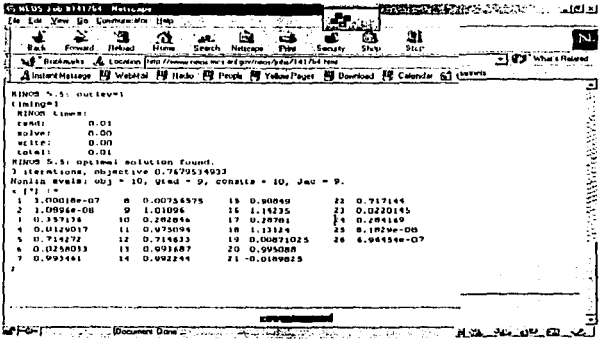

Figura. 3.11 Resultados del problema del "Cuadrado" con el solver MINOS

La gráfica de los resultados se presenta en la Fig.3.12.

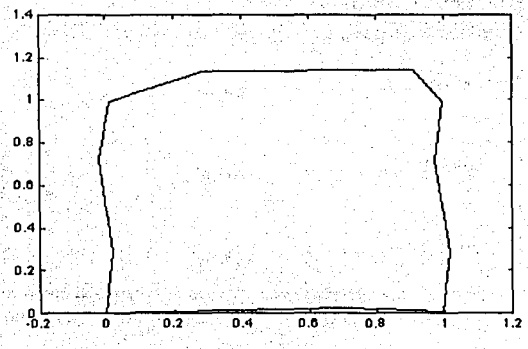

Figura. 3.12 Resultados del "Cuadrado" que nos proporcionó el solver MINOS

3.3.2 Problema de Optimización No Lineal con Restricciones "La Habana"

Este problema resulta de un problema de suavizamiento de contornos poligonales, que consiste en minimizar la suma de cuadrados de las longitudes de los lados del polígono y cuyo planteamiento es el siguiente: se tiene un conjunto de puntos $P_i = (u_{2i-1}, u_{2i})$ que determinan el polígono:

$P_1 = (0.233403, 0.218586)$
$P_2 = (0.242231, 0.279365)$
$P_3 = (0.257023, 0.336511)$
$P_4 = (0.307364, 0.374008)$
$P_5 = (0.349139, 0.419062)$
$P_6 = (0.404976, 0.444467)$
$P_7 = (0.41471, 0.405226)$
$P_8 = (0.392464, 0.347906)$
$P_9 = (0.366036, 0.292271)$
$P_{10} = (0.345105, 0.234427)$
...etc.

Lo que se quiere determinar son los puntos $Q_i = (x_{2i-1}, x_{2i})$, con $i = 1, ..., 77$, tal que

$$\min f(x) = \sum_{j=1}^{76} \sqrt{(x_{2j-1} - x_{2j+1})^2 + (x_{2j} - x_{2(j+1)})^2}$$

$$\text{sujeto a } \left\{ \begin{array}{c} (u_{2i-1} - x_{2i-1})^2 + (u_{2i} - x_{2i})^2 \leq (a_i)^2 \\ \text{con } i \in R = \{1..77\} \end{array} \right.$$

es decir, se quiere minimizar la distancia entre estos puntos, sujeto a que dada una vecindad de diámetro a_i, la distancia de P_i a Q_i debe ser menor o igual a este diámetro (ver Fig. 3.13). Con $a_i = 0.01$, para $i \in R = \{1..77\}$.

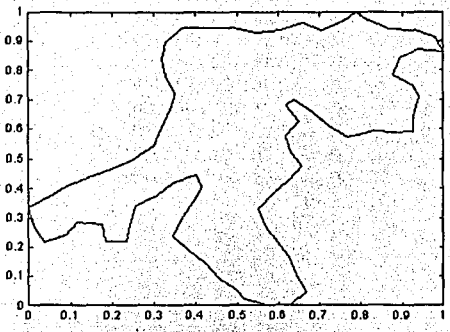

Figura. 3.13 Gráfica de los datos iniciales de "La Habana"

- Formato AMPL

```
# Problema La Habana
# Minimizar una función no lineal
# Número de variables 154
# Número de restricciones 77
param n>0 integer;
param m>0 integer;
param m1>0 integer;
set R3 :={1..m1};
set R1 :={1..n};
set R2 :={1..m};
param a{j in R2};
param u{i in R1};
var x{i in R1};
minimize f:  sum {j in R3}
((x[2*j-1]-x[2*j+1])^2+(x[2*j]-x[2*(j+1)])^2)^1/2;
s.t.  distancia { i in R2} :
```

```
(u[2*i-1]-x[2*i-1])^2+(u[2*i]-x[2*i])^2 <= (a[i])^2;
data;
var:  x:=
```

(ver cd anexo)

- Formato FORTRAN

```
C Número de variables 154
C Número de restricciones 77
C
C Asignación del punto inicial
 SUBROUTINE initpt(N,X)
 INTEGER N
 DOUBLE PRECISION X(*)
 INTEGER I
 DO 10 I=1,N
 X(I)=0.0D+0
10 CONTINUE
 RETURN
 end
C Asignación de los límites de las variables
C Esta subrutina evalua la función objetivo
 SUBROUTINE fcn(N,X,NF,FVEC)
 INTEGER N,NF
 DOUBLE PRECISION X(*), FVEC(*)
 INTEGER J
 DO 35 J=1,NF
 FVEC(J)= DSQRT((X(2*J-1)-X(2*J+1))**2+(X(2*J)-X(2*(J+1)))**2)
35 CONTINUE
 END
C Esta subrutina evalua las restricciones
 SUBROUTINE cfcn(N,X,M,C)
 INTEGER N,M
 DOUBLE PRECISION Z(154)
 DOUBLE PRECISION X(*),C(*)
 INTEGER I
 z(1)= 0.0D+0
 z(2)= 0.0D+0
```

```
      z(3)= 0.242231D+0
      z(4)=0.279365D+0
      z(5)= 0.257023D+0
      .
      .
      .
      z(153)=0.233403D+0
      z(154)=0.218586D+0
      DO 15 I=1,77
      C(I)=DSQRT((z(2*I-1)-X(2*I-1))**2+(z(2*I)-X(2*I))**2)
     15 CONTINUE
      RETURN
      END
     C Subrutina que asigna las cotas en las variables
     SUBROUTINE xbound(N,XL,XU)
      INTEGER N
      DOUBLE PRECISION XL(*),XU(*)
      INTEGER I
      DOUBLE PRECISION COTA
      PARAMETER (COTA=1.0D+20)
      DO 30 I=3,N-2
      XL(I)=-COTA
      XU(I)=COTA
     30 CONTINUE
      XL(1)= 0.233403D+0
      XU(1)= 0.233403D+0

      XL(2)= 0.218586D+0
      XU(2)= 0.218586D+0

      XL(N-1)=0.233403D+0
      XU(N-1)=0.233403D+0
      XL(N)=0.218586D+0
      XU(N)=0.218586D+0
      END
     C Esta subrutina asigna los límites sobre las restricciones
      SUBROUTINE cbound(M,CL,CU)
```

```
      INTEGER M
      DOUBLE PRECISION CL(*),CU(*)
      INTEGER I
      DO 25 I=1,77
      CU(I)= 0.1D-1
      CL(I)= 0.0D+0
   25 CONTINUE
      RETURN
      END
```

La forma de introducir los datos, la ilustraremos en la Fig. 3.14. Al enviar estos archivos el Servidor NEOS despliega los resultados como se muestra en la Fig.3.15. Posteriormente se presenta la gráfica de los resultados obtenidos (Fig. 3.16).

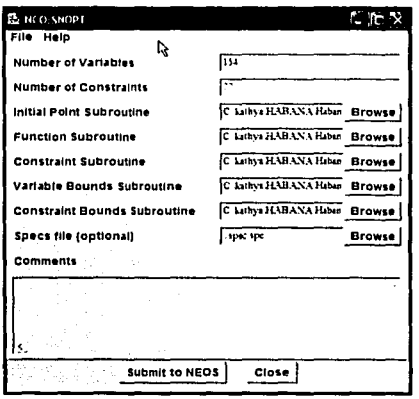

Figura. 3.14 Forma de introducir los datos

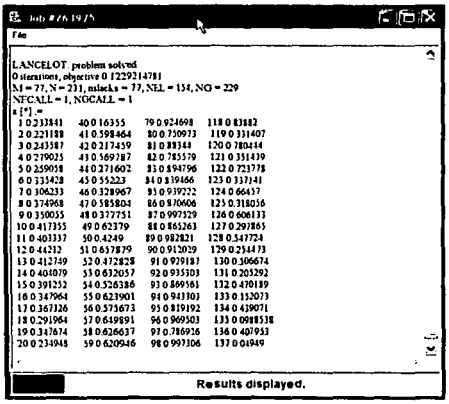

Figura. 3.15 Resultados de " La Habana"

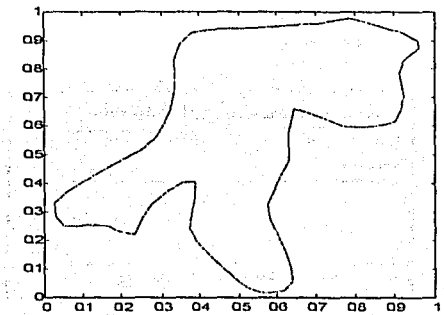

Figura. 3.16 Gráfica final de "La Habana".

Los siguientes ejemplos fueron tomados de "Cuadrados Mínimos No Lineales, Desarrollos Recientes,I," del Dr.Pablo Barrera y M. en C. Ernesto Olvera [1]. Los cuales fueron utilizados para la revisión, análisis y experimentación numérica de los métodos más clásicos y eficientes para resolver el problema de mínimos cuadrados no lineales, el cual es vinculado al problema de ajustes de curvas.

3.3.3 Problema de Optimización No Lineal sin Restricciones "Mínimos Cuadrados 1"

Sea

$$\begin{aligned} f_1(x) &= x + 1 \\ f_2(x) &= \lambda x^2 + x - 1 \end{aligned}$$

Este es un problema de suma de cuadrados de $f_1(x)$ y $f_2(x)$, consiste en:

$$\text{mín } r(x) = f_1(x)^2 + f_2(x)^2$$

con un punto inicial $x_0 = 1$ y para $\lambda = 0.1, 0.3, 0.6, 0.8$. Para $\lambda = 0.1$, la gráfica se presenta en la Fig. 3.17.

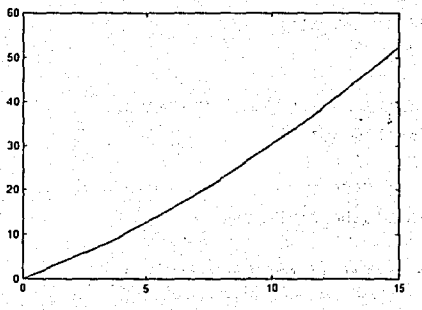

Figura. 3.17 $r(x) = f_1(x)^2 + f_2(x)^2$

- Formato en AMPL:

```
# Cuadrados mínimos No Lineales
# Ejemplo 1:  (Powell)
# Consideremos r1(x)=x+1 y r2(x)=0.1x^2+x-1
# Para l= 0.1,0.3,0.6,0.7
var x;
minimize F: (x+1)^2+( 0.1*x^2+x-1)^2;
data;
var x:=1;
```

- Formato en FORTRAN

```
C PROBLEMA 1:  (POWELL)
C PUNTOS INICIALES
SUBROUTINE INITPT(N,X)
INTEGER N
DOUBLE PRECISION X(1)
X(1)= 1.0D+0
END
C SUBRUTINA PARA FUNCION A MINIMIZAR
SUBROUTINE FCN(N,X,F)
INTEGER N
DOUBLE PRECISION F
DOUBLE PRECISION X(1), L(1)
L(1)= 0.1D+0
F=(X(1)+1)**2+(L(1)*X(1)**2+X(1)-1)**2
END
```

El problema es resuelto por la forma Web, y se muestra en la Fig. 3.18.

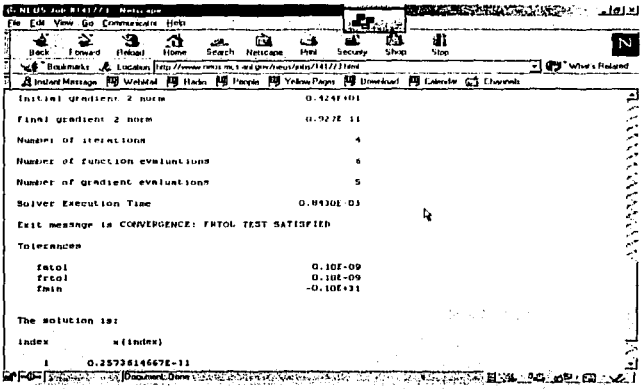

Figura 3.18 Resultados, forma Web "Min. Cuadrados 1".

3.3.4 Problema de Optimización No Lineal sin Restricciones "Mínimos Cuadrados 2"

Sea

$$\begin{aligned} f_1(x) &= \exp(x) - 2 \\ f_2(x) &= \exp(2x) - 4 \\ f_3(x) &= \exp(3x) - b \end{aligned}$$

para distintos valores de b y con un valor inicial $x_0 = 1$.

$$\min \left[f_1(x)^2 + f_2(x)^2 + f_3(x)^2 \right]$$

- Formato AMPL:

```
# Ejemplo 2:  ( Dennis)
# Para b= 8,3,-1,-4
var x;
minimize D: (exp(x)-2)^2+(exp(2*x)-4)^2+(exp(3*x)+4)^2;
data;
var:  x:= 1;
```

- Formato FORTRAN:

```
C EJEMPLO 2:  (DENNIS)
C PUNTOS INICIALES
SUBROUTINE INITPT (N,X)
INTEGER N,I
DOUBLE PRECISION X(N)
X(1)= 1.0D+0
END

C SUBRUTINA DE LA FUNCION OBJETIVO DEL EJEMPLO 2
SUBROUTINE FCN(N,X,F)
INTEGER N
DOUBLE PRECISION F
DOUBLE PRECISION X(1),B(1)
B(1)= 8.0D+0
F= ( EXP(X(1))-2)**2+(EXP(2*X(1))-4)**2+(EXP(3*X(1))-B(1))**2
END
```

Para obtener la solución a este problema se eligió la opción de `Submission Tool` y el solver VMLV (ver Fig.3.19)

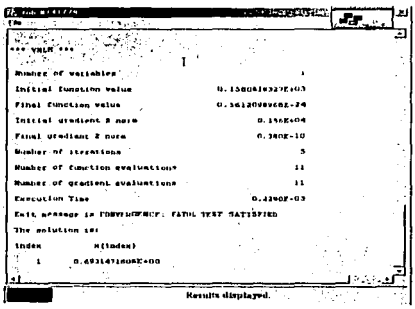

Figura. 3.19 Resultados "Mínimos Cuadrados 2".

3.3.5 Problema de Optimización No Lineal sin Restricciones "Ajuste de curvas 1"

El problema consiste en determinar la curva de crecimiento de una población de bacterias. Los datos son el número de individuos de una especie particular y_i, a intervalos de tiempo t_i. Ver Tabla 3.3.

t_i	0	4	7.5	25	31	48.75	52	58.5	72.7	72.7	78
y_i	8	6	6	7	8	10	13	18	18	33	38
t_i	95	96	108	112	133	136.75	143	156.5	166.7	181	
y_i	76	78	164	175	280	300	320	405	385	450	

Tabla 3.3 Datos del crecimiento de la población de bacterias

Se espera que el crecimiento de esta población quede aproximado por un modelo, cuya expresión analítica es:

$$y(t) = \frac{x_1}{1 + \exp(x_2 - x_3 t_i)} \qquad con \quad x_3 > 0,$$

Los parámetros que queremos determinar son x_1, x_2, y x_3. La función que deseamos minimizar es:

$$\sum_{i=1}^{20} (r_i(x))^2$$

donde

$$r_i(x) = \frac{x_1}{1 + \exp(x_2 - x_3 t_i)} - y_i, \text{ para } i = 1, ..., 20$$

La gráfica de los datos iniciales se presenta en la Fig. 3.20.

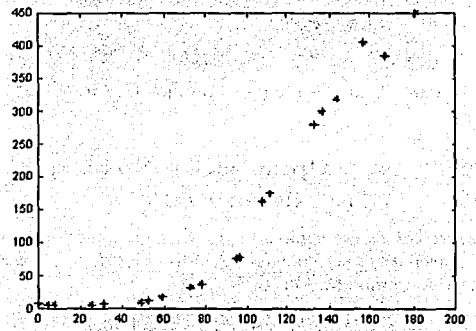

Figura. 3.20 "Ajuste de curvas".

- Formato AMPL:

```
# Ejemplo 3:
param n>0 integer;
param m>0 integer;
param r>0 integer;
set R1:={1..n};
set R2:={1..m};
set R3:={1..r};
param t{ i in R1};
```

```
param Y{ j in R2};
var x{ i in R3};
minimize F: sum{i in R1}
 (Y[i]-( x[1]/(1+exp(x[2]-x[3]*t[i]))))^2;
data;
var:  x:=
1 413
2 5.134
3 0.0444;
param n:=20;
param m:=20;
param r:=3;
param t:=
1 0 ...
```

(ver cd anexo)

- Formato FORTRAN:

```
C EJEMPLO 3:
SUBROUTINE INITPT(N,X)
INTEGER N
DOUBLE PRECISION X(N)
X(1)= 413.0D+0
X(2)= 5.134D+0
X(3)= 0.0444D+0
END
C SUBRUTINA DE LA FUNCION DEL EJEMPLO 3
SUBROUTINE FCN(N,X,F)
INTEGER N,M
DOUBLE PRECISION F
DOUBLE PRECISION X(N), Y(M),T(M)
M=20
T(1)= 0.0D+0
T(2)= 4.0D+0
T(3)= 7.5D+0
T(4)= 25.0D+0
T(5)= 31.0D+0...
```

(ver cd anexo)

```
      DO 35 I=1,M

      F(I)= (Y(I)-(X(1)/(1+(EXP(X(2)-X(3)*T(I)))**2

      35 CONTINUE

      END
```

En la Fig. 3.21 se muestra la solución del problema vía `Submission Tool` y en la Fig. 3.22 se presenta la gráfica de los resultados.

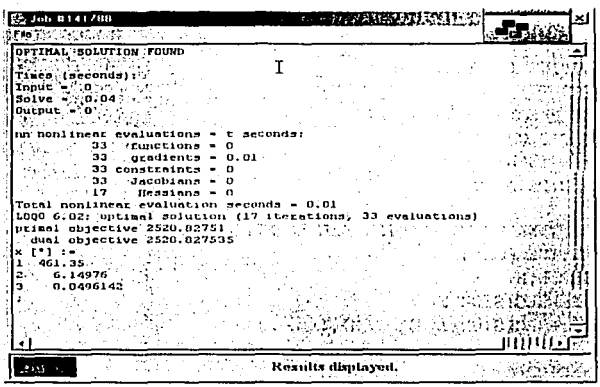

Figura. 3.21 Resultados, *Submission Tool* "Ajuste de curvas 1".

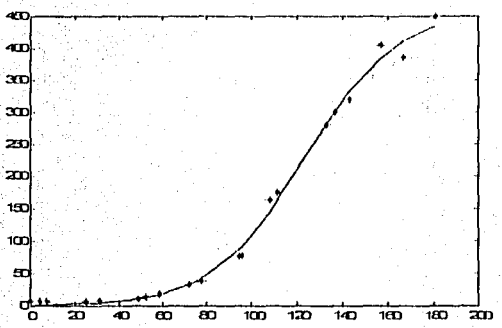

Figura. 3.22 Resultados, (*) valores iniciales, (- - -) curva ajustada.

3.3.6 Problema de Optimización No Lineal sin Restricciones "Ajuste de curvas 2"

Los siguientes datos fueron proporcionados a M. R. Osborne por el Dr. A.M. Sargeson of the Research School Chemestry in the Australian National University (ver Tabla 3.4 y Fig 3.23)[17].

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
t_i	0	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160
y_i	0.844	0.908	0.932	0.936	0.925	0.908	0.881	0.850	0.818	0.784	0.751	0.718	0.685	0.658	0.628	0.603	0.580
i	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	
t_i	170	180	190	200	210	220	230	240	250	260	270	280	290	300	310	320	
y_i	0.558	0.538	0.522	0.506	0.490	0.478	0.467	0.457	0.448	0.438	0.431	0.424	0.420	0.414	0.411	0.406	

Tabla 3.4 Datos

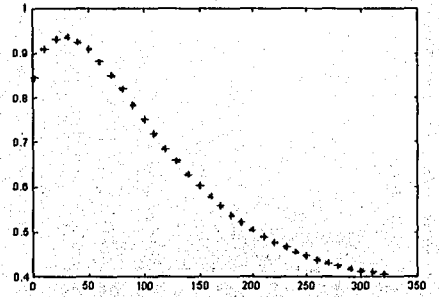

Figura. 3.23 Datos iniciales, "Ajuste de curvas 2"

El problema consiste en ajustar el modelo

$$y(t) = x_1 + x_2 \exp(-x_4 t) + x_3 \exp(-x_5 t)$$

a los datos proporcionados; es decir:

$$\min f(x) = \sum_{i=1}^{33} (y(t) - y_i)^2$$

- Formato AMPL:

```
# El problema es ajustar el modelo
# Y(t)= x1+x2 exp(-x4t)+x3exp(-x5t)
# a los siguientes datos
param n>0 integer;
param m>0 integer;
param r>0 integer;
set R1:={1..n};
set R2:={1..m};
set R3:={1..r};
```

```
param t{ i in R1};
param Y{ j in R2};
var x{ i in R3};
minimize G:
sum{ i in R1, j in R2}
Y[j]-(x[1]+x[2]*exp(-x[4]*t[i])+x[3]*exp(-x[5]*t[i]))^2;
data;
var:  x:=
1 0.5
2 1.5
3 -1.0
4 0.01
5 0.02;
param n:=33;
param m:=33;
param r:=5;
param t:=
1 0...
param Y:=
1 0.844
2 0.908
(ver cd anexo)
```

- Formato FORTRAN:

```
C SUBRUTINA DEL EJEMPLO 4
SUBROUTINE INITPT(N,X)
INTEGER N
DOUBLE PRECISION X(N)
X(1)= 0.5D+0
X(2)= 1.5D+0
X(3)= -1.0D+0
X(4)= 0.01D+0
X(5)= 0.02D+0
END
C SUBRUTINA DE LA FUNCION DEL EJEMPLO 4
SUBROUTINE FCN(N,X,F)
INTEGER N,M
```

```
DOUBLE PRECISION F
DOUBLE PRECISION X(N), Y(M),T(M)
M=33
T(1)= 0.0D+0
T(2)= 10.0D+0
T(3)= 20.0D+0
T(4)= 30.0D+0
T(5)= 40.0D+0...
Y(1)= 0.844D+0
Y(2)= 0.908D+0
Y(3)= 0.932D+0
Y(4)= 0.936D+0
Y(5)= 0.925D+0...
(ver cd anexo)
DO 35 I=1,M
F(I)=(Y(I)-(X(1)+X(2)*(EXP(-X(4)*T(I)))+
& X(3)*(EXP(-X(5)*T(I))))**2
35 CONTINUE
END
```

En las siguientes figuras se muestra la forma de envío y los resultados obtenidos (ver Fig.3.24 y 3.25).

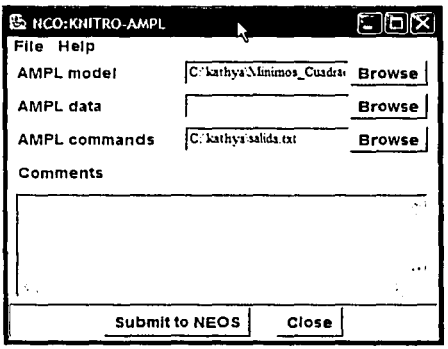

Figura. 3.24 Forma de envío Submission Tool

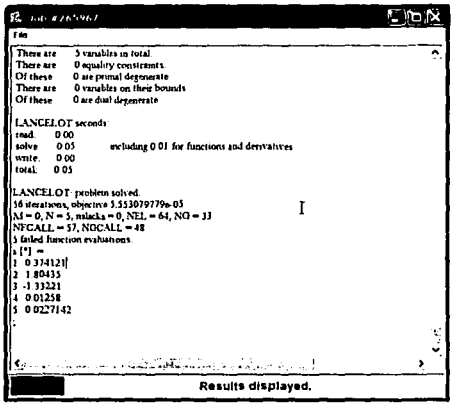

Figura 3.25. Resultados

3.3.7 Problema de Optimización No Lineal sin Restricciones "Mínimos Cuadrados 3"

El siguiente problema fué propuesto por Powell [18] y consiste en minimizar la función $f(x)$, cuyas componentes son:

$$\begin{aligned} r_1(x) &= x_1 \\ r_2(x) &= \frac{10x_1}{x_1 + 0.1} + 2x_2^2 \end{aligned}$$

$$\text{mín } f(x) = r_1(x)^2 + r_2(x)^2$$

- Formato AMPL:

```
# Ejemplo 5:  Este es un problema propuesto por Powell
# r1(x)=x1, r2(x)=10*x1/(x1+0.1)+2*x2^2
#
param n>0 integer;
set R1:={1..n};
var x{ i in R1};
minimize J: (x[1])^2+ ((10*x[1]/(x[1]+0.1))+2*x[2]^2)^2;
data;
param n:=2;
var:  x:=
1 3
2 1;
```

- Formato FORTRAN:

```
C SUBRUTINA DEL EJEMPLO 5:  ESTE ES UN PROBLEMA PROPUESTO POR
C POWELL
SUBROUTINE INITPT(N,X)
INTEGER N
DOUBLE PRECISION X(N)
X(1)= 3.0D+0
X(2)= 1.0D+0
END
C SUBRUTINA DE LA FUNCION DEL EJEMPLO 5
SUBROUTINE FCN(N,X,F)
INTEGER N,
DOUBLE PRECISION F
DOUBLE PRECISION X(N)
F= (X(1))**2+(10*X(1)/(X(1)+0.1)+2*X(29)**2)**2
END
```

Los resultados se presentan en la Fig. 3.26.

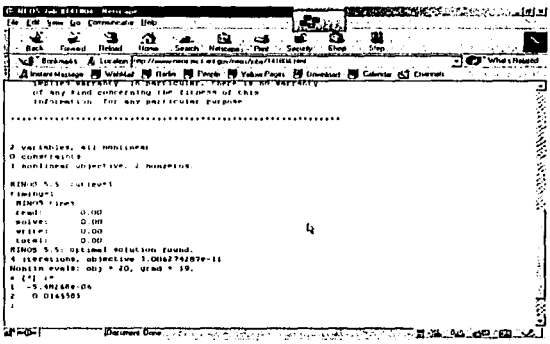

Figura. 3.26 Forma Web "Mínimos Cuadrados 3".

3.3.8 Problema de Optimización Lineal Con Restricciones "Refinación del Petróleo"

Una refinería divide el petróleo en una serie de compuestos intermedios, la mezcla de estos compuestos dará un producto final. Dado el volumen disponible de compuestos intermedios, se quiere determinar una mezcla de estos, de tal forma que los productos resultantes sean lo más provechosos.

Empezaremos definiendo los conjuntos de los compuestos intermedios I, productos finales J y los atributos K. Los datos relevantes tecnológicamente pueden ser representados por las siguientes variables:

a_i barril del compuesto i disponible para cada $i \in I$

r_{ik} unidades de atributos k contribuidos por barril del compuesto intermedio i para cada $i \in I$ y $k \in K$.

u_{jk} Máximo de unidades permitidas k por barril del producto final j para cada $j \in J$ y $k \in K$

$$\delta_{ij} = \begin{cases} 1 & \text{si el compuesto } i \text{ es permitido en la mezcla del producto } j \\ 0 & \text{en cualquier otro caso} \end{cases}$$

y los datos económicos están dados por

c_j costo del barril del producto j, $j \in J$

Hay dos colecciones de variables de decisión.

x_{ij} barriles del compuesto intermedio i usado para hacer el producto j para cada $i \in I$ y $j \in J$.

y_j cantidad de barriles del producto j, para cada $j \in J$.

El objetivo es:

$$\text{máx} \sum_{j \in J} c_j y_j$$

la cual es la suma de los costos del conjunto J de productos. Donde la cantidad de cada compuesto intermedio usado en la mezcla debe ser igual.

$$\sum_{j \in J} x_{ij} = a_i, \; i \in I,$$

La cantidad de productos elaborados, debe ser igual a la suma de la cantidad de componentes que se mezclaron para obtener cada producto.

$$\sum \delta_{ij} x_{ij} = y_j, \; j \in J,$$

Para cada producto, el total de atributos que están incluidos en todos los componentes, no bebe exceder del total permitido.

$$\sum_{i \in I} r_{ik} x_{ij} \leq u_{jk} y_j, \; j \in J \text{ y } k \in K,$$

Finalmente tenemos

$$\begin{aligned} 0 &\leq x_{ij} \leq \delta_{ij} a_i \\ 0 &\leq y_j \end{aligned}$$

El problema esta escrito en el lenguaje AMPL y se anexa en el CD. Utilizamos como solver LANCELOT. Los resultados son los siguientes, ver Fig 3.27, 3.28 y 3.29.

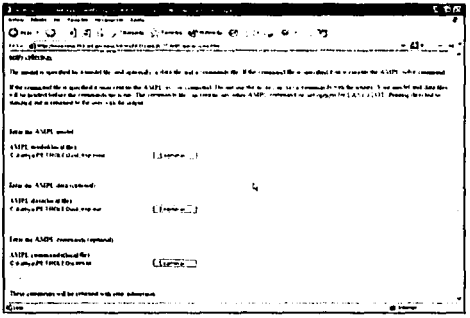

Figura. 3.27 Forma Web, "Refinación del Petroleo"

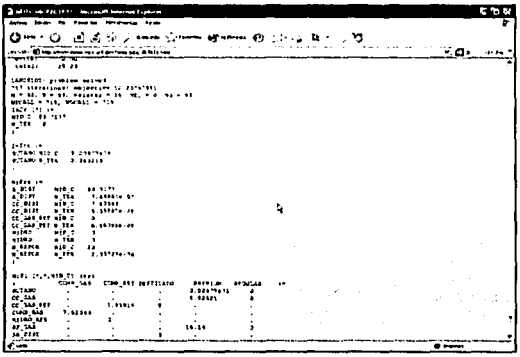

Figura. 3.28 Resultados 1, "Refinación del Petroleo"

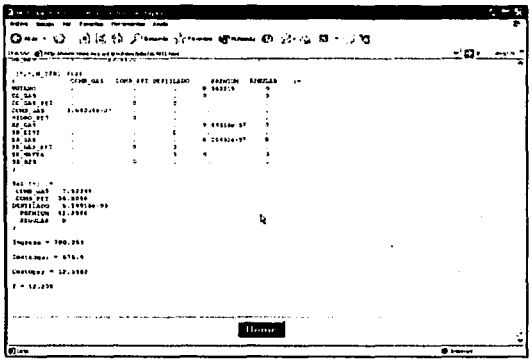

Figura. 3.29 Resultados 2, "Refinación del Petroleo"

3.3.9 Problema de Optimización No Lineal "Portafolios de Inversión"

Un portafolio que se compone de posiciones xi en cada activo i alcanza una parte superior en el escenario j de

$$u_j = \text{máx}\left[\sum_{i=1}^{n}(M_{ji} - r_j q_i)x_i,\ 0\right]$$

y una parte inferior en el escenario j de

$$d_j = \text{máx}\left[\sum_{i=1}^{n}(r_j q_i - M_{ji})x_i,\ 0\right]$$

Lo que se quiere determinar es el tamaño o monto de la posición tal que minimice el problema siguiente:

$$\min \sum_{j=1}^{s} 0.25 u_j$$

$$\text{sujeta a} \begin{cases} (0.25) d_j \leq k \\ (xL)_i \leq x_i \leq (xU)_i \end{cases}$$

Para $k = 0, 0.6, 8.25, 25.$ y donde

$$M = \begin{pmatrix} 7 & 21.5 & 19 & 6.5 \\ 7 & 21.5 & 19 & 6.5 \\ 11.6 & 5.2 & 13.8 & 9.4 \end{pmatrix}, \quad r = \begin{pmatrix} 1.20 \\ 1.15 \\ 1.10 \\ 1.05 \end{pmatrix}, \quad q = \begin{pmatrix} 10 \\ 12.5 \\ 8 \end{pmatrix},$$

$$xL = \begin{pmatrix} -2 \\ 0 \\ -1 \end{pmatrix}, \quad xU = \begin{pmatrix} 5 \\ 2 \\ 4 \end{pmatrix}$$

En la Fig 3.30 se muestra la gráfica del valor de la función objetivo de acuerdo a los valores que se listan en la Tabla 3.5.

k	0	0.6	8.25	25
Función Objetivo	0.9375	8.2	23.5	40.25

Tabla 3.5 Valores de la Función Objetivo para cada k

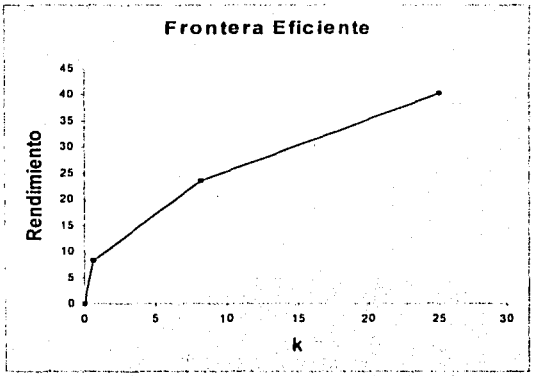

Figura. 3.30 Gráfica del problema de "Portafolios de Inversión".

- Formato AMPL:

```
param n integer;
param m integer;
set I := {1..n};
set J := {1..m};
param M{i in I,j in J};
param r{j in J};
param q{i in I};
param l{i in I};
param up{i in I};
var x{i in I};
var u{j in J}=max (0,sum {i in I} ((M[i,j]-r[j]*q[i])*x[i]));
var d{j in J}= max (0,sum {i in I} ((r[j]*q[i]-M[i,j])*x[i]));
maximize rend:sum {j in J} .25*u[j];
s.t.  ries:sum {j in J}(.25*d[j])<=.60;
s.t.  lim{i in I}:l[i]<= x[i]<=up[i];
data;
param n:=3;
param m:=4;
param M:
 1 2 3 4:=
 1 7 21.5 19 6.5
 2 7 21.5 19 6.5
 3 11.6 5.2 13.8 9.4;
param r:=
1 1.20
2 1.15
3 1.10
4 1.05;
param q:=
1 10
2 12.5
3 8;
param l:=
1 -2
2 0
3 -1;
```

```
param up:=
1 5
2 2
3 4;
```

En las figuras 3.31 y 3.32, se muestra la forma de envío y los resultados obtenidos con el solver LANCELOT.

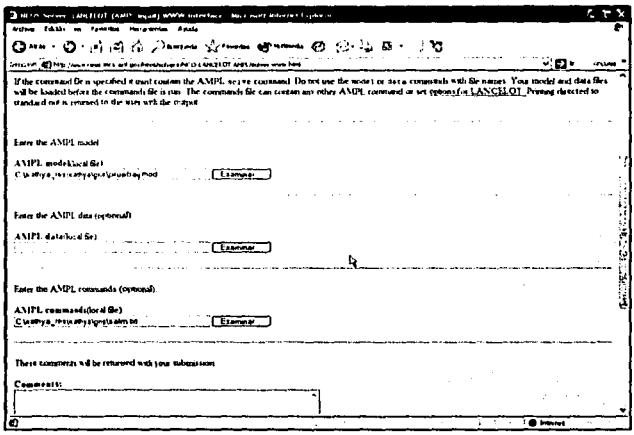

Figura. 3.31 Forma Web, "Portafolio de Inversión"

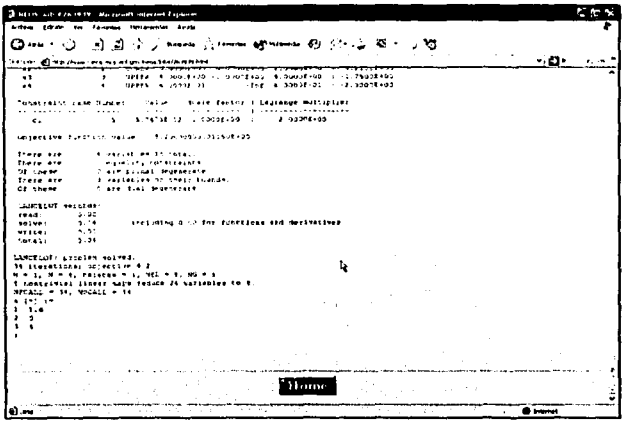

Figura. 3.32 Resultados, "Portafolio de Inversión"

3.3.10 Problema de Optimización No Lineal "Método de Penalización"

Considere el siguiente problema

$$\text{mín } f(x_1, x_2, ..., x_{10}) = \sum_{k=1}^{10} k x_k^2$$

$$\text{sujeta a} \begin{cases} 1.5x_1 + x_2 + x_3 + 0.5x_4 + 0.5x_5 = 5.5 \\ 2.0x_6 - 0.5x_7 - 0.5x_8 + x_9 - x_{10} = 2.0 \\ x_1 + x_3 + x_5 + x_7 + x_9 = 10 \\ x_2 + x_4 + x_6 + x_8 + x_{10} = 15 \end{cases}$$

en donde se empleara el método de penalización. Primero se resolverá como un problema con restricciones y se hará la comparación con los resultados obtenidos para el problema sin restricciones.

La forma de aplicar el Método de Penalización es la siguiente:
Sean

$$\begin{aligned} h_1(x) &= 1.5x_1 + x_2 + x_3 + 0.5x_4 + 0.5x_5 - 5.5 \\ h_2(x) &= 2.0x_6 - 0.5x_7 - 0.5x_8 + x_9 - x_{10} - 2.0 \\ h_3(x) &= x_1 + x_3 + x_5 + x_7 + x_9 - 10 \\ h_4(x) &= x_2 + x_4 + x_6 + x_8 + x_{10} - 15 \end{aligned}$$

Tenemos la siguiente función de penalización:

$$\text{mín } f(x_1, x_2, ..., x_{10}) = \sum_{k=1}^{10} kx_k^2 + [\mu h_1(x)^2 + \mu h_2(x)^2 + \mu h_3(x)^2 + \mu h_4(x)^2]$$

para $\mu = 10, 100, 1000, 10000$. La forma de envío y resultados se muestran en las Fig. 3.33, 3.34 y 3.35. En la Fig. 3.34 se obtienen resultados cuando $\mu = 10$, y en la Fig. 35 los resultados se obtuvieron con $\mu = 1000$.

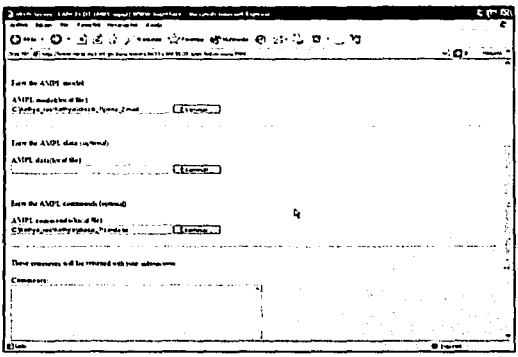

Figura. 3.33 Forma Web, "Penalizacion" con LANCELOT

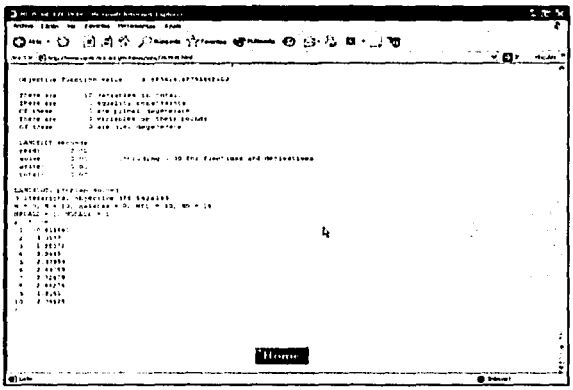

Figura. 3.34 Resultados "Penalización" con $\mu = 10$.

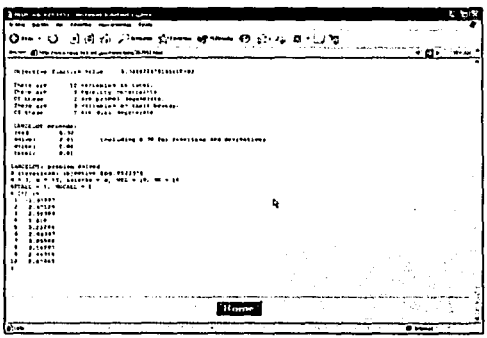

Figura. 3.35 Resultados "Penalizacion" $\mu = 1000$.

En la Fig. 3.36 se presentan los resultados que se obtuvieron con el solver `LANCELOT`, en donde el problema es con restricciones.

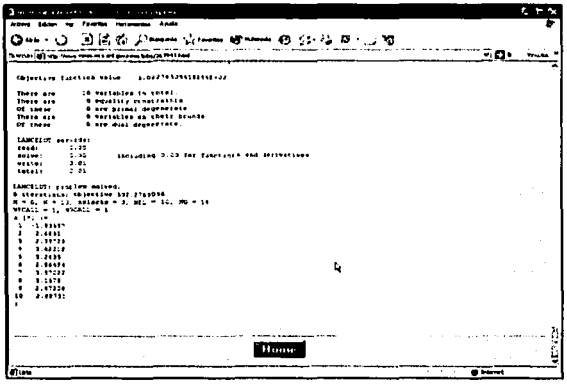

Figura. 3.36 Resultados "Penalizacion", con restricciones

Los resultados para las distintas μ y para el problema con restricciones, se concentran en la Tabla 3.6.

	$\mu = 10$	$\mu = 100$	$\mu = 1000$	$\mu = 10000$	con restricciones
x_1	−0.615661	−1.81409	−1.97887	−1.99597	−1.99788
x_2	3.0577	2.72567	2.67124	2.6655	2.66486
x_3	1.88071	2.31967	2.38089	2.38725	2.38796
x_4	3.0933	3.55611	3.616	3.62218	3.62287
x_5	2.37999	3.14642	3.25284	3.2639	3.26513
x_6	2.49789	2.82002	2.86067	2.86484	2.86531
x_7	2.72409	3.71779	3.85588	3.87022	3.87182
x_8	2.44274	3.06461	3.14887	3.1576	3.15857
x_9	1.8151	2.38516	2.46388	2.47206	2.47297
x_{10}	2.04528	2.60362	2.67963	2.68751	2.68839
Objetivo	388.56251	487.43314	500.88223	502.27630	502.43177

Tabla 3.6 Soluciones de la función objetivo para distintas μ

3.4 Particularidades del Servidor NEOS

3.4.1 Diferenciación Automática

Los algoritmos del Servidor NEOS para problemas parcialmente separables, pueden resolver problemas de optimización no lineal de gran escala, dado que sólo requieren que el usuario provee el código para la evaluación de la función. Esta habilidad fue considerada irrealista recientemente. El mayor obstáculo fue el cálculo del gradiente. Para problemas de pequeña escala se puede aproximar el gradiente por diferencias finitas, por ejemplo:

$$[\nabla f(x)]_i \approx \frac{f(x + h_i e_i) - f(x)}{h_i}, 1 \leq i \leq n$$

donde h_i, es el parámetro de diferencia y e_i es el i-ésimo vector canónico, pero esta aproximación es prohibida para problemas de gran escala, porque se requiere de n evaluaciones de la función para cada gradiente.

Aproximar gradientes por diferencias no sólo es costoso, sino también crece la complejidad del código de optimización. Partir de una elección deficiente para h_i, puede causar una terminación anticipada del algoritmo de optimización lejos del camino de la solución.

Los algoritmos del Servidor NEOS para problemas de optimización no lineal, usan herramientas de diferenciación automática para calcular los Gradientes y Jacobianos requeridos por los algoritmos . En la actualidad se utiliza ADIFOR para el proceso del código en FORTRAN y ADOL-C para C [9].

3.4.2 Problemas de Optimización Global

Lo que pretenden William Gropp y Jorge J. Moré, es extender las capacidades del Servidor NEOS para problemas de optimización global. En general, estos problemas son atacados por algoritmos que requieren un número grande de procesadores. La conexión que ellos están interesados particularmente, es en problemas que surgen de la determinación de estructuras de proteínas. Como ejemplo de estos tipos de problemas se consideran aquellos que comprenden distancias geométricas.

Los problemas de distancias geométricas son especificados por un subconjunto S de todos los pares de átomos, y por las distancias entre los átomos i y j para $(i, j) \in S$. En la práctica, los límites inferiores y superiores sobre

las distancias son dados con valores precisos. El problema de distancias geométricas con límites inferiores y superiores, espera encontrar un conjunto de posiciones $x_1, ..., x_m$ en $\mathbb{R}^3$ tal que

$$l_{i,j} \leq \|x_i - x_j\| \leq u_{i,j}, \ (i,j) \in S,$$

donde $l_{i,j}$ y $u_{i,j}$, son los límites superiores e inferiores sobre las distancias respectivamente.

Investigaciones recientes de las aplicaciones de problemas de distancias geométricas para la determinación de estructuras de proteínas, pueden ser encontradas en Havel [11, 11, 12], Torda y van Gunsteren [21], Kuntz, Thomason and Oshiro[14], Brünger and Nilges[4], y Blaney and Dixon [3].

3.4.3 Detalles Técnicos

El `Servidor NEOS` permite introducir archivos comprimidos; es decir, se puede enviar el archivo que contiene el problema de manera comprimida si así se requiere, pero si el usuario desea comprimir archivos individualmente, en caso de tener archivos muy grandes, estos pueden ser enviados vía `web` o por `Submission Tool`. El `Servidor NEOS` revisa cada archivo, para ver si éste está comprimido. Hay una opción disponible del `Browse` para los archivos comprimidos. El tipo de compresión que el `Servidor NEOS` acepta son de deflación standard para `.gz, .z, .zip` y archivos .Z.

Por otro lado, es muy importante proporcionar una dirección de correo electrónico, para que se puedan obtener los resultados con mayor seguridad.

En lo que se refiere a los errores, unos de los problemas que presenta el `Servidor NEOS` son los siguientes: algunas veces cuando la ejecución toma un largo tiempo sin producir ninguna salida, el `Servidor Web` puede causar el `script CGI` en tiempo fuera. Para este caso, el usuario debe guardar el password y el número que se le asigna a cada trabajo que envió. Posteriormente esto se introduce en la forma `check-status.cgi`, donde es posible que se le proporcione al usuario los resultados finales. Después de un día del que el problema es enviado y resuelto, es posible que el archivo con los resultados sea borrado, por lo que es indispensable que se proporcione la dirección del correo electrónico. Otro de los problemas que puede presentar el `Servidor Neos`, es cuando recibe muchos datos. El mensaje que manda es:

```
"cgi-lib.pl:  Request to receive too much data:
              ********bytes''
```

Este Servidor Web restringe la cantidad de datos que pueden ser enviados; este problema no debería presentarse si se usa e-mail o Submission Tool. Aunque algunos de los solvers tienen limitada la cantidad de datos que pueden ser procesados.

Un mensaje de error que comúnmente se presenta es:

"ERROR: Runtime error encountered",

esto significa que el solver encontró un error de sintáxis o un error de punto flotante, tal como un overflow un subscript fuera de los límites. Normalmente estos tipos de errores son detectados, en el caso contrario es posible enviarles un mensaje a la administración del sistema vía

"neos-commentscreen"

por la página del solver, mencionando el número de trabajo, el mecanismo que se uso para el envío e incluir el mismo envío.

Una característica más, es cuando el solver despliega un envío como basura, esto se debe a que el archivo que se le envió contiene símbolos formateados por Word o Word Perfect. Los archivos deben ser guardados con extensión .txt y con esto es posible quitar los símbolos con formato.

Una de las preguntas importantes que hay que mencionar es, ¿cuánto tiempo puede estar corriendo el problema en el Servidor NEOS?, algunos solvers tienen límites estrictos sobre el número de iteraciones que son permitidas, en la mayoría de los casos, el usuario recibe los mejores resultados en términos de la mejor solución, encontrados en un número limitado de iteraciones. El Servidor tiene un tiempo fuera automático, fijado a una semana en el solver, esto significa que si el problema toma más de una semana en ejecución, no se recibirá resultados.

El tamaño máximo de los archivos, que se pueden enviar vía Web es aproximadamente de 1048576 bytes. El límite para el envío vía e-mail, está fijado por el proveedor del correo electrónico, el cual puede cambiar sin el conocimientos de los administradores. El Submission Tool, tiene un límite no muy extenso, puesto que el Servidor NEOS no envía trabajos más grandes que 50,000,000 bytes, a las estaciones de trabajo donde se encuentran los solvers.

Originalmente estas computadoras que reciben los problemas, son estaciones de trabajo que residen en Argonne National Laboratory, Northestern University and The University of Wisconsin.

Apéndice A

Listados de Programas

LISTADO 1.

```
Modelo steel13.mod:
set PROD; # productos
param rate{PROD}>0; # toneladas producidas por hora
param avail >=0; # horas disponibles en la semana
param profit{PROD}; # ganancia por tonelada
param commit{PROD} >=0;
# límite inferior sobre las toneladas vendidas en la
#semana
param market{PROD} >=0;
#límite superior sobre las toneladas vendidas en la
#semana
var Make {p in PROD} >=commit[p],<=market[p];
#toneladas producidas
maximize total_profit:  sum {p in PROD} profit[p]*Make[p];
# Función objetivo:  Total de ganancias por todos los productos
subject to Time:  sum{p in PROD} (1/rate[p])*Make[p] <=avail;
#Restricciones:  Total de horas usadas por todos los productos
#que no puede
#exceder de las horas disponibles
Datos:
set PROD := bandas bobinas placas;
param:  rate profit commit market :=
bandas  200 25 1000 6000
bobinas 140 30 500  4000
```

```
placas  160 29 750  3500 ;
param avail := 40;
```

LISTADO 2.

```
Modelo steel14.mod
set PROD; # Productos
set STAGE; # Jornadas
param rate {PROD, STAGE}>0; # Toneladas por hora en cada jornada
param avail {STAGE} >=0; # Horas disponibles entre la semana
de cada
#jornada
param profit {PROD}; # ganancia por tonelada
param commit {PROD}>=0;
# límite inferior sobre las toneladas vendidas en la
#semana
param market{PROD} >=0;
#límite superior sobre las toneladas vendidas en la
#semana
var Make {p in PROD} >=commit[p],<=market[p];
#toneladas producidas
maximize total_profit:  sum {p in PROD} profit[p]*Make[p];
# Función objetivo:  Total de ganancias por todos los productos
subject to Time {s in STAGE}:
sum { p in PROD} (1/rate[p,s]*Make[p]<=avail[s];
#En cada jornada:  total de horas usadas por todos los productos

#que no puede #exceder de las horas disponibles
Datos:
set PROD := bandas bobinas placas;
set STAGE := reheat roll;
param rate:  reheat roll:=
bandas  200 200
bobinas 200 140
placas  200 160 ;
param :  profit commit market :=
bandas  25 1000 6000
bobinas 30 500  4000
placas  29 750  3500 ;
```

```
param avail:  reheat 35 roll 40;
```

LISTADO 3.

```
Modelo diet.mod:
set NUTR;
set FOOD;
param cost {FOOD}>0;
param f_min {FOOD} >=0;
param f_max{j in FOOD}>=f_min[j];
param n_min{NUTR} >=0;
param n_max {i in NUTR} >=n_min[i];
param amt {NUTR,FOOD}>=0;
var Buy {j in FOOD} >=f_min[j],<=f_max[j];
minimize total_cost:  sum{j in FOOD} cost[j]*Buy[j];
subject to diet {i in NUTR}:
 n_min[i]<= sum { j in FOOD} amt[i,j]*Buy[j]<=n_max[i];
```

LISTADO 4.

```
Modelo blend.mod:
set IN; #entradas
set OUT; # salidas
param cost {IN}>0;
param in_min {IN}>=0;
param in_max {j in IN}>= in_min[j];
param out_min{OUT}>=0;
param out_max{i in OUT} >=out_min[i];
param io {OUT,IN}>=0;
var X{j in IN} >= in_min[j],<=in_max[j];
minimize total_cost:  sum{j in IN} cost[j]*X[j];
subject to outputs {i in OUT}:
out_min[i]<=sum{j in IN} io[i,j]*X[j]<=out_max[i];
```

Bibliografía

[1] Barrera, Pablo S., Olvera Ernesto(1997), "Cuadrádos Mínimos No Lineales Desarrollos Recientes I ", Reporte de Investigación, Facultad de Ciencias, UNAM, y Escuela de Ciencias, UAEM.

[2] Bischof, C..Carle A. , Khademi P. y Mauer A. (1994), " The ADIFOR 2.0 system for automatic differentiation of Fortran 77 Programs", Preprint MCS-P381, Argonne National Laboratory, Argonne, IL. Also available as CRPC-TR94491, Center for Research on Parallel Computation, Rice University. Houston, TX.

[3] Blaney, J.M. y Dixon J.S. (1994), "Distance geometry in molecular modeling", in Reviews in Computational Chemistry 5 (K.B. Lipkowitz and D.B. Boyd. eds), VCH Publishers (New York), 299-335.

[4] Brüger, A.T. y Nilges M. (1993), "Computational challenges for macromolecular structure determination by X-ray crystallography and solution NMR-spectroscopy", Q. Rew. Biophys. **26**, p.p. 49-125.

[5] Cedillo, R. Gricelda, Bernabé, Ma. A. (2003), "La Frontera eficiente PUT/ CALL", Tesis de Licenciatura, Facultad de Ciencias, UNAM.

[6] Clive G. Page (1995), "Professional Programmer ´s Guide to Fortran 77", University of Leicester, UK

[7] Daniels, Richard W.,(1978), "An Introduction to Numerical Methods and Optimization Techniques", Elsevier North-Holland, Inc.

[8] Fourer, Robert, Gay, David M., Kernighan Brian W. (1993), "AMPL A modeling Language for Mathematical Programming", Boyd & Fraser publishing company, The Scientific Press Series.Servidor de Internet NEOS .

[9] Griewank, A.,Juedes D. y Utke J. (1996), "ADOL-C: A package for automatic differentiation of algorithms written in C/C++", ACM Trans. Math. Software 22, 131-167.

[10] Gropp W. y Moré J. (1997), "Optimization Environments and the NEOS Server", que aparece en Approximation Theory and Optimization, M.D. Buhmann and A. Iseries, eds., páginas 167-182, Cambrige University Press.

[11] Havel, T.F. (1991), "An evaluation of computational strategies for use in the determination of protein structure from distance geometry constraints obtained by nuclear magnetic resonance", Prig. Biophys. Mol. biol. 56, 43-78.

[12] Havel, T.F. (1995), "Distance geometry", in Encyclopedia of Nuclear Magnetic Resonance (D.M. Grant and R.K. Harris, eds), John Wiley & Sons (New York), 1701-1710.

[13] Kuester, J. L., Mize,J.H. (1973), "Optimization Techniques with Fortran", New York: MacGram Hill.

[14] Kuntz, I.D., Thomason J.F. y Oshiro C.M. (1993), " Distance geometry", in Methods in Enzymology 177 (N.J. Oppenheimer and T.L. James, eds), Academic Press (New York), 159-204.

[15] Luenberger, David G.(1973), "Introduction to Linear ann Nonlinear Programming", Stanford University, Addison-Wesley Publishing Company.

[16] Nakamura, Shoichiro (1997), " Análisis Numérico y Visualización Gráfica con MATLAB", The Ohio State University, Prentice-Hall Hispano America, S.A.

[17] Osborne, M. R. (1972) "Some aspects of non-linear least squares calculations". Numerical Methods for nonlinear optimization. Lootsma (ed.) Academic Press.

[18] Powell, M. J. D. (1970) "A new algorithm for unconstrained optimizations". Non linear Programming; Rosen, Mangasarian, Ritter (eds).

[19] Press, W. H. (1989),"Numerical Recipes: The art of scientific computing", Cambridge: Cambridge University.

[20] Tellez, Rubén (2001), " Apuntes de Teoría y Técnicas de Optimización", Facultad de Ingeniería, DEP, Departamento de Sistemas,UNAM.

[21] Torda, A.E. y van Gunsteren W.F. (1992), "Molecular modeling uaing nuclear magnetic resonance data", in Reviews in Computational Chemistry 3 (K.B. Lipkowitz and D.B. Boyd, eds), VCH Publishers (New York), 143-172.

[22] Wall, L., T. Christiansen, Schwartz R.L. (1996), "Programming Perl", O 'Reilly & Associates, Inc.,2nd edn (Sebastopol,CA).

[23] http://www.netlib.com

[24] http://www-neos.mcs.anl.gov/neoscite