UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
A C A T L A N

MEMORIA DINAMICA EN PROGRAMACION LINEAL CON MATRICES DISPERSAS

T E S I S P R O F E S I O N A L

QUE PARA OBTENER EL TITULO DE

LICENCIADO EN MATEMATICAS APLICADAS Y COMPUTACION

P R E S E N T A :

EDUARDO DOMINGUEZ CAMBRON

MEXICO, D. F. 1992

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

I N D I C E

OBJETIVOS GENERALES

El procesamiento de datos hoy en dia, a medida que las empresas se desarrollan, tiende a ser muy largo y tardado, ya que las actividades que se realizan dentro del plan de trabajo tienden a multiplicarse, y su relación con otras tiende a ser menor cada vez. Por tal motivo, la empresas buscan nuevas técnicas de procesamiento que puedan ser utilizadas con las herramientas ya existentes dentro de su medio.

Uno de estos problemas frecuentemente es el de analizar actividades representadas por medio de gráficas, donde se toman en cuenta caracteristicas tales como la dirección del flujo, las capacidades, orden, etc., lo cual hace representables las actividades, no sólo como gráficas, sino también como matrices, que teniendo muchas variables y muy poca relación entre ellas, arrojan la caracteristica principal de una matriz dispersa : la mayoria de los coeficientes de las inecuaciones ó elementos de la matriz son iguales a cero, que matemáticamente, tiene un tratamiento muy especial.

Uno de los objetivos que persigue esta investigación es el diferenciar el uso de métodos para matrices dispersas a comparación de los de las matrices densas.

Lo anterior conduce a la necesidad de implementar una estructura capaz de realizar un procesamiento similar al utilizado en Programación Lineal con las matrices densas, que proporcione las mismas ventajas al hacer cálculos con matrices dispersas, pero con una característica más de la cual se pueda tomar ventaja. Una característica primordial de la matrices dispersas, como ya se mencionó, consiste en que la mayoría de sus elementos son iguales a cero, de modo que si se procesara como matriz densa convencional, habría que desperdiciar la memoria que ocuparan tales datos. Así, por un lado se incurre en la necesidad de más memoria de la indispensable, y por otro, se incurre en un procesamiento innecesario, es decir, que se realizan operaciones donde no se requieren (en los elementos iguales a cero).

De esta forma, habiendo analizado una matriz dispersa con procesos para matriz densa, se obtienen resultados iguales, pero con dos desventajas consecuentes: mayor gasto de memoria y mayor tiempo de proceso.

El objetivo principal del uso de técnicas para matrices dispersas, y de la investigación, es que tratando de obtener los mismos

resultados, se logrará un ahorro de memoria ocasionado por el aprovechamiento de la característica de dispersidad en las matrices, por medio de un algoritmo con procedimientos similares a los utilizados para matrices densas, pero con una implementación que permita el manejo de matrices dispersas.

El siguiente de los objetivos surge cuando, ya habiendo implementado el algoritmo para matrices dispersas, se pueda suministrarle al método simplex una solución inicial factible que esté próxima a la solución óptima bajo ciertas condiciones. Esto se logra tomando del planteamiento de Programación Lineal (P.L.), una sub-matriz cuadrada la cual pueda ser resuelta por algún método para sistemas de ecuaciones lineales. El resultado se le suministra al método de P.L. para que la búsqueda de la solución óptima sea (o trate de ser) más rápida dependiendo de la estructura del problema.

En suma, todos los objetivos son encaminados al uso de matrices dispersas en la programación lineal por medio de técnicas para el tratamiento de la dispersión.

INTRODUCCION.

Una gran variedad de actividades de la vida cotidiana pueden ser representadas por medio de matrices. En éstas, se toman en cuenta las características de la actividad, y llevándolas a un plano experimental (análisis del modelo matemático), se puede llegar a una solución óptima real, sin necesidad de arriezgar mayores presupuestos, tiempo, ni material humano. Por ejemplo, cuando una empresa busca implementar un nuevo sistema de aprovechamiento de materia prima. Si se decidiera implementarlo sin una previa investigación, seguramente tendría que experimentar con algunos sistemas hasta encontrar el mejor. Esto implicaría un gasto de materia prima en el transcurso de su búsqueda. Con un modelo matemático se busca experimentar en un plano ficticio, y el resultado obtenido será o estará próximo de ser el óptimo y no se habrá necesitado más que una buena representación del problema a analizar.

Comúnmente la matriz que describe alguna actividad se compone de un arreglo regular de números que describen dicha actividad. Ahora bien, cuando dentro de esta actividad hay un diverso número de sub-actividades que, en su conjunto, llegan, o tratan de llegar, a un

fin buscado bajo planes de trabajo, algunas veces, dichas sub-actividades no tienen ninguna relación con alguna otra de ellas. Cuando lo anterior es frecuente, dado que la actividad que se desarrolla es demasiado compleja, su planteamiento en una matriz no es particular. Especificamente , se trata de las matrices llamadas "Dispersas ó Huecas", éstas son reconocidas por la caracteristica de tener la mayoria de sus elementos iguales a cero. Más especifico, si "γ" es el número de elementos diferentes de cero en la matriz, entonces para que esta sea dispersa se debe cumplir la siguiente caracteristica:

$$\gamma \quad << \quad \frac{n * n}{2} \quad ,$$

lo que significa que ese número "γ" debe de ser "mucho menor" de la mitad de números contenidos (ceros o no) en dicha matriz.

El interés en las matrices dispersas es el poder explotar sus caracteristicas.

Hablando computacionalmente, se puede ahorrar memoria, y algunas veces, procedimientos.

Hablando en términos funcionales, muchos de los problemas que se

presentan en la práctica tienen su planteamiento en este tipo de matrices.

En términos de proceso matemático, es posible asumir que se pueden mantener implícitos los elementos iguales a cero correspondientes a la matriz, de modo que los coeficientes que se registran se les conoce como " entradas " y se refieren a los elementos que pueden ser manejables explícitamente dentro de los cálculos.

Para el mayor aprovechamiento de las matrices dispersas, es necesario relacionar tres ingredientes fundamentales : La Matriz, El Algoritmo y La Computadora.

La Matriz bien definida para su proceso, basado en la implementación de un algoritmo en una computadora, puede llevar a un camino el cual ahorre tiempo y memoria en procesamiento de datos, lo cual corresponde a la finalidad que se tiene al usar distintamente los métodos para matrices dipersas.

Se hace notar que si el significado benéfico de dispersión en una matriz no se convierte en reducción de costos, se puede asegurar el hecho de que un problema que no es factible de determinar su solución (dadas sus dimensiones), ahora puede ser resuelto. Se habla específicamente de los problemas que pueden tener decenas ó centenas de

ecuaciones.

Para la resolución de ciertos tipos de problemas que se plantean con matrices dispersas se utiliza una herramienta de la Investigación de Operaciones conocida como " Programación Lineal ". Específicamente el " Método Simplex " y el método auxiliar de éste conocido como " Dos Fases ", que se basan en el pivoteo sobre elementos seleccionados, para encontrar así, los valores óptimos de una función objetivo, la cual se maximizará ó se minimizará según las especificaciones del problema.

A este método se le suma la posibilidad de poder inicializarlo mediante el suministro de una solución factible dada por alguno de dos tipos de métodos utilizados para resolver sistemas ecuaciones simultáneas : Método Directo y Método Iterativo . La solución suministrada por el procedimiento anterior, debe de ser una solución que se encuentra justamente en la orilla del conjunto de soluciones factibles determinado por el problema, por lo que al método simplex únicamente le faltará buscar la solución óptima en los puntos adyacentes a esta solución. Esto se debe a una de las características principales del método Simplex : " la solución óptima se encuentra en la orilla (específicamente en una esquina) del conjunto de soluciones factibles ".

En general, la problemática que se presenta sugiere el uso de la Programación Lineal con métodos auxiliares para resolver problemas cuyo

planteamiento matemático origina una matriz dispersa. Se plantea la alternativa de poder ahorrar memoria en cálculos de dicha naturaleza mediante el uso de Coordenadas en la matriz, lo que permite almacenar únicamente los elementos que serán procesados, es decir, los elementos diferentes de cero.

Se analizará la importancia de la Programación Lineal, sus métodos más importantes y la búsqueda de la solución al problema de Maximización ó Minimización para alguna actividad restringida por características propias del problema. Se tratarán las matrices dispersas desde su representación gráfica, su interpretación, sus ventajas y su uso, al igual que su comparación con respecto a las matrices densas.

Se parte del supuesto de que los métodos de Programación Lineal (los analizados en la presente investigación) suelen ser homogéneos en su aplicación. Es decir, no importando el tipo de matriz que este involucrada, el método es realizado de igual forma.

Ahora bien, en el caso específico de una matriz dispersa, se cuenta con un grado de pureza el cual es evaluado de acuerdo al número de elementos diferentes de cero en la matriz y a las dimensiones de ésta. Por ejemplo, el grado de pureza indica que los números iguales a cero en una matriz densa debe de ser mayor del 50 % (Esto se refiere

precisamente al término relativo << visto con anterioridad).

De acuerdo a lo anterior, un método de Programación Lineal con características de dipersidad, originaría un ahorro de memoria, y posiblemente, mayor rapidez en la obtención de resultados.

Por último, una sub-matriz cuadrada tomada del planteamiento original, debe ser resuelta por alguno de dos métodos usados para resolver sistemas de ecuaciones simultáneas : Método Directo (Gauss-Jordan) y Método Iterativo (Kaczmarz). Los resultados obtenidos deben ser suministrados al método de Programación Lineal, y analizar su comportamiento, tanto el proceso convencional, como el proceso de inicialización.

Como se puede observar, el marco teórico en que se desarrolla el presente trabajo se encuentra estrechamente relacionado con las materias abarcadas a lo largo de la carrera de Matemáticas Aplicadas y Computación, siendo alguna de ellas la Investigación de Operaciones, la Teoría de Gráficas, la Programación Avanzada , el Algebra Lineal , la Optimización y la Computación , que son analizados en dicha carrera, juegan un papel muy importante en el desarrollo de la presente investigación.

Los temas antes mecionados serán fácilmente identificados dentro de su propio contexto y se refiere bibliográficamente al final de esta investigación.

CAPITULO PRIMERO

EL PROBLEMA GENERAL DE LA PROGRAMACION LINEAL

C A P I T U L O P R I M E R O

EL PROBLEMA GENERAL DE LA PROGRAMACION LINEAL .

1.1 Desigualdades Lineales.

Dentro del contexto del Algebra Lineal, se da una atención especial lo que respecta a matrices. Estas se componen de un conjunto de renglones, y cada uno de estos representa una ecuación, una inecuación o parte de ella. En esta sección se dirigirá la atención a las " inecuaciones " mejor conocidas como " desigualdades ". Estas describen la situación de alguna actividad con respecto a algún límite, el cual es impuesto por el medio ambiente, por capacidades ó por conveniencias y convenciones.

La parte principal de este capítulo está encaminada a la introducción algebraica y a la comprensión, en términos geométricos, de las desigualdades lineales. Una desigualdad, divide espacios n-dimensionales en dos sub-espacios, uno donde la desigualdad es satisfecha y otra donde no lo es (sub-espacios). Esta característica,

dentro del planteamiento del problema lineal, representa una restricción donde los valores factibles a la solución únicamente se podrán encontrar en el sub-espacio donde la restricción si es satisfecha. Esto sucede para cada una de las restricciones.

Además, existen otros tipos de restricciones, que son fundamentales en la Programacion Lineal, y éstas deben de cumplir con la no-negatividad de las variables involucradas. De esta forma, los valores que puedan tomar las variables involucradas están restringidos por dos caracteristicas: cumplir con las desigualdades del problema y que siempre sean valores mayores o iguales a cero. En términos geométricos se tienen n subespacios más, demarcados por los límites derechos de los ejes coordenados (únicamente la parte positiva de los subespacios).

A la región de intersección de los subespacios generados, se le conoce con el nombre de conjunto factible ó región de factibilidad. De modo que este conjunto factible está compuesto por una familia de soluciones factibles que satisfacen simultaneamente todas y cada una de las desigualdades lineales.

Con lo anterior es fácil visualizar que un sistema de la forma "Ax = b", con "m" desigualdades y "n" incógnitas, describen la intersección de "m" diferentes planos (descritos por las

desigualdades). En un caso particular, cuando el número de desigualdades es igual al número de incógnitas, la intersección de estos planos se encuentra en un punto el cual la solución es

$$x = A^{-1} b .$$

Así, un sistema de "m" desigualdades describe la intersección de "m" subespacios. Si a éstas intersecciones se le suman "n" subespacios más que describen las condiciones de no-negatividad de las variables, en consecuencia, la intersección del total de subespacios será más reducida.

La parte esencial de conocer el Algebra de desigualdades lineales no es el identificar cual es el conjunto de todos los puntos factibles, sino el encontrar aquel punto que proporcione maximización ó minimización de una cierta " Función de Costos ", " Función de Utilidades " ó " Función Objetivo ". En suma, el problema de la P.L. es encontrar el punto de intersección de subespacios que pertenezca al conjunto factible y que además minimice o maximice el costo de la actividad representada.

Se puede intuir que el punto óptimo corresponde a un vector donde alguna ó todas las variables involucradas encuentran su coeficiente ó valor correspondiente a dicho punto, y que además este punto

corresponda a una esquina del conjunto factible. Lo anterior es garantizado geométricamente, porque las líneas que dan la función de costos son movidas de lugar en lugar hasta que se intersecta al conjunto factible en su punto máximo ó mínimo según sea el caso. Así el primer contacto (en caso de minimización) debe ocurrir a lo largo de su frontera inferior, ó el último contacto en el caso de maximización, a lo largo de su frontera superior.

Se puede dar el caso que la línea de la función objetivo que optimice el problema no solamente toque una esquina como antes se mencionó, sino que puede pertenecer a toda una orilla correspondiente a dos puntos adyacentes. Cuando esto sucede se dice que el problema es de solución múltiple, es decir, cualquiera de los puntos pertenecientes a dicha orilla darán la misma solución óptima al problema.

Un aspecto importante resulta cuando se tiene un problema de minimización y éste no cuenta con frontera superior (no está acotado en la parte superior). Si se deseara maximizar el problema podría no tener solución, por el motivo de que el conjunto factible podría irse tan alto (al no existir límite superior) y la solución a maximizar costo sería infinito ó indeterminado.

Otra característica que se puede presentar es cuando los subespacios generados por las desigualdades del problema no se

intersectan. En este caso el problema tampoco tiene solución por ser su conjunto de factibilidad vacio, es decir, que ningún punto perteneciente al subespacio de una restricción satisface simultaneamente a alguna otra.

De esta forma se puede categorizar todo problema de programación lineal por su estructura : que el conjunto factible sea vacio, que la función de costos quede ilimitada ó indeterminada en el conjunto factible, que exista un único valor para el programa lineal ó que exista un número infinito de soluciones.

Indudablemente el camino más fácil para resolver este tipo de problemas consiste en cambiar las desigualdades a " igualdades " introduciendo variables de holgura (restricciones con menor o igual) ó variables superfluas o artificiales (restricciones con mayor o igual). Por ejemplo, se tiene una restricción $7x + 3y \geq 5$,

entonces

$$w = 7x + 3y - 5$$

ó

$$7x + 3y - w = 5$$

donde $w \geq 0$.

Esta última ecuación, de igual forma, debe cumplir con las restricciones de no-negatividad para "x" y para "y", y como "w" debe de ser positivo, entonces la nueva restricción es válida.

En "La Metodología de Dantzig", en el siguiente capítulo, se explicará con detalle este tipo de conversión de desigualdades a igualdades.

Esta característica de tener todas las desigualdades como igualdades será muy importante, ya que el método de programación lineal que se utilizará (método Simplex), las usará como tales para iniciar sus procedimiento.

Hay que hacer notar que las variables que surgan de dichas conversiones se sumarán a las restricciones de no-negatividad, de tal forma que estas variables auxiliares tampoco podrán ser negativas (Ver tema 2.1).

1.2 Descripción del Programa Lineal .

Para llegar más a fondo al planteamiento del problema, hay que tener una visión clara de lo que es la estructura del problema lineal,

y no hay mejor camino para describirlo que en forma vectorial y matricial.

Se escribe un problema de programación lineal de la siguiente forma:

$$A\,\alpha \;\leq\; \text{ó} \;\geq\; b$$

donde A es una matriz de (m * n), y cada uno de sus "m" renglones describen una desigualdad.

La matriz "A" y el vector "b" son dados por las restricciones. Por ejemplo, la restricción $7x + 3y \geq 5$ implicaría que $A = [\,7\,,\,3\,]$ y $b = [\,5\,]$.

La función de costos será también lineal y estará dada de la siguiente forma :

$$Cx = c_1 \alpha_1 + c_2 \alpha_2 + \quad . \; . \; . \; + c_n \alpha_n$$

En resumen, un programa lineal es aquel que optimiza (ya sea maximizar ó minimizar) un problema planteado de la forma siguiente:

F.O. $Z = CX$

sujeto a

$Ax \leq$ ó $\geq b$

$x \geq 0$

donde

Z	es la función objetivo
$x \geq 0$	son condiciones de no-negatividad
X	es el vector de actividades
c	es el vector de costos unitarios
b	es el vector de recursos
A	es la matriz de coeficientes tecnológicos

Una característica del problema surge cuando al tener "n" incógnitas y "m" desigualdades se genera el vector solución de actividades de dimensión "m" (por existir solo "m" subespacios), donde se registará la solución, es decir, un vector tal que,

$$X = [\ x_1, x_2, \ldots, x_m\ ]$$

(estos aspectos se analizarán más a fondo en el tema 2.1).

Este vector tendrá que ser factible, por lo tanto debe satisfacer las "m" desigualdades y además deberá cumplir con las condiciones de

no-negatividad.

El problema ahora es encontrar un vector " X " factible que minimice ó maximice (según sea el caso) la función objetivo, de tal forma que el vector sea óptimo y cumpla con todas las restricciones.

Geométricamente, las condiciones de no negatividad restringen la solución al cuadrante positivo para todas las variables. Por ejemplo, si existen dos variables entonces sus soluciones estarán en un cuarto del plano, si son tres, entonces sus soluciones estarán en un octavo del espacio, etc. Generalizando, un vector tiene una oportunidad en 2^n, siendo "n" el número de variables, de ser no-negativo.

Las otras "m" restricciones producen "m" subespacios más, y los vectores factibles son aquellos que cumplen con las "m + n" restricciones generadas. En otras palabras, el conjunto de vectores factibles están incluidos en la intersección de los " m + n " subespacios existentes, y cuyo conjunto factible podría ser, como ya se mencionó, limitado, ilimitado ó vacío.

Ahora bien, la función de costos " CX " trae al problema un conjunto de familias de soluciones. Por ejemplo, un miembro de dicha familia es uno que pasa a través del origen, esta es la solución

correspondiente a " CX = 0 " y si el vector "X" satisface esta ecuación y las restricciones, entonces la función objetivo tendría el valor de " cero ".

Los otros planos " CX = k ", donde k es una constante positiva, dan todas las demás posibles soluciones. De este modo, con el valor de la función objetivo, "barre" el conjunto de soluciones factibles incluídas en la itersección de los " n + m " subespacios, para determinar el punto que optimice la función objetivo (solución óptima).

El propósito principal es precisamente determinar el vector X que sea óptimo. Este propósito se podría lograr encontrando todas la esquinas del conjunto factible y calcular sus costos, y aquel que proporcione un valor más atractivo será el óptimo.

En la práctica, el procedimiento anterior es teóricamente imposible; pueden existir miles de esquinas, y no se podrían calcular todos sus valores correspondientes. Para eso, se usará el Método Simplex que fué desarrollado por el Dr. George Dantzig (Década de los 40's). Su metodología se tratará en el siguiente capítulo.

A continuación se presentan los conceptos más importantes dentro de la descripción y desarrollo del problema lineal.

1.3 Concepciones Basicas de Programacion Lineal.

Uno de los aspectos más importantes es el de tener bien establecido que una solución factible es aquel vector columna que satisface todas las restricciones del planteamiento, incluyendo las de no-negatividad. Esta solución factible puede ser básica, cuando esta no contiene más de "m" componentes positivas. A su vez, esta solución factible básica puede ser no-degenerada cuando precisamente "m" componentes del vector son positivos o iguales a cero, o de igual forma puede ser degenerada al existir menos de "m" componentes positivos en el vector.

Todas las soluciones que son factibles pertenecen a un conjunto homogéneo llamado " región de factibilidad " la cual contiene todas las soluciones (ó vectores) que cumplen con todas las restricciones. Un concepto importante a tener en consideración es el siguiente:

Tomando en cuenta que se tiene una región de factibilidad, y que ésta contiene a todas las soluciones factibles posibles, entonces cualquier punto dentro de esta región podrá alcanzar cualquier otro punto (también dentro de la región) en linea recta sin necesidad de salir de esta, es decir que la región de factibilidad es un conjunto convexo.

Así, ya definido lo que es un conjunto convexo de soluciones factibles, se asume que la función objetivo de un programa lineal encuentra su solución óptima en un punto extremo (ó en la frontera) de su conjunto convexo generado.

Ahora bien, se supone la existencia de un conjunto K ∈ "m" de columnas de la matriz que sean linealmente independientes tal que,

$$\alpha_1 a_1 + \alpha_2 a_2 + \ldots + \alpha_k a_k = b$$

donde a_i son vectores columna de A,

$\alpha_i \geq 0$ (óptimo) $i=1, \ldots, k \leq m$,

y $b = (b_1, b_2, \ldots, b_m)$

entonces el punto de solución óptima será

$$X^* = (X_1, X_2, \ldots, X_k, 0, 0, \ldots, 0)$$

y será también un punto extremo del conjunto de soluciones factibles.

Un aspecto de la programación lineal que es importante, es el hecho de tener un punto extremo (vector de longitud "n") del conjunto de las soluciones factibles,

$$X = (\alpha_1, \alpha_2, \ldots, \alpha_n)$$

donde "n" es el número de incógnitas, de modo que las columnas de A son asociadas a cada componente de X que sea positivo y dichas columnas serán linealmente independientes, con un máximo de "m" componentes de X positivas y las restantes serán ceros.

En otras palabras, si el número de incognitas en el problema es de "n", el cual es mayor que el número de restricciones "m" (sin contar las de no-negatividad), entonces los puntos extremos que pertenezcan al conjunto factible estarán compuestos de "m" componentes mayores que cero (dado por las restricciones) y " n - m " componentes iguales a cero.

Por las caracteristicas de los problemas de programación lineal expuestas hasta el momento, se pueden obtener los siguientes axiomas:

» Una solución factible básica corresponde a un punto extremo del conjunto convexo de soluciones factibles.

» Cada punto extremo del conjunto de soluciones factibles tiene asociados "m" vectores linealmente independientes de la matriz. Lo que implica que cualquier combinación lineal no-negativa de "m" vectores linealmente independientes de A, sería algún punto especifico en la frontera de la región factible.

» Existe un punto en la frontera del conjunto convexo, que quizá no es único, donde la función objetivo obtiene su valor óptimo.

» Existe un número infinito de puntos en la frontera, uno de los cuales será la solución del problema.

El proceso para determinar la solución óptima, con los conocimientos adquiridos hasta el momento, consistiría en realizar punto por punto una evaluación explícita del valor de la función objetivo. Se puede intuir que se necesitará de mucho tiempo y trabajo antes de encontrar dicha solución. La labor de probar punto por punto hasta encontrar el mejor resultado, puede llevar hasta días debido a que el conjunto de soluciones factibles se le puede considerar infinita. Sabiendo que existen "n" actividades (incógnitas) y "m" restricciones, el número total de combinaciones en la cual se buscaría la solución estaría dado por la expresión

$$\binom{n}{m} = \frac{n!}{m!\,(n-m)!} \quad .$$

lo cual, para cualquier n y m, sería un número extremadamente grande de puntos para evaluar.

El procedimiento anterior suele ser extenuante, pero es efectivo

e importante para poder entender la representación de la programación lineal en el plano geométrico.

Con las herramientas anteriores se puede hacer ya un análisis del comportamiento de un método de programación lineal que resulta ser más complicado, pero más rápido y eficiente; este es " El Método Simplex ".

CAPITULO SEGUNDO

PROGRAMACION LINEAL (METODO SIMPLEX) Y MATRICES DISPERSAS

C A P I T U L O S E G U N D O

PROGRAMACION LINEAL (METODO SIMPLEX) Y MATRICES DISPERSAS .

2.1 Metodologia de Dantzig .

Como se mencionó en el capitulo anterior, el principal problema en programación lineal es calcular o encontrar un vector X^* que optimice la función objetivo, y esto podía ser logrado encontrando todas las esquinas del conjunto factible, y calculando sus valores correspondientes a la función objetivo, poder encontrar el punto ó esquina que optimizara la función. De igual forma se mencionó que este trabajo podría ser proporcionalmente imposible, ya que el número de esquinas en el conjunto son demasiadas como para poder analizar cada una de ellas.

Para resolver este problema, el Dr. George Dantzig ideó un método que permite analizar los puntos sin necesidad de calcular todos los valores correspondientes, y que lleva con toda seguridad, a la solución óptima del problema. Este método es conocido como Método Simplex.

Geométricamente el método simplex se compone de dos fases: la

fase I simplemente localiza una orilla del conjunto factible, y la fase II, considerada la parte central del método, donde se va de esquina a esquina a lo largo de las orillas del conjunto factible. En otras palabras, en una esquina cualquiera, hay "n" orillas de las cuales de debe escoger alguna para segir el camino.

Algunos conducirán a la X^* óptima, y otros a un camino gradual hacia este. Dantzig escoge la orilla que garantiza el mayor incremento de la función objetivo en la elección de la siguiente esquina u orilla. Así, la nueva orilla seleccionada conducirá a una nueva esquina con un valor de la función objetivo más conveniente.

Se puede tener la seguridad de que este método no tiene posibilidad de regresar a un valor que empeore la solución que se tenga en ese momento.

El método termina cuando una esquina en particular ha sido alcanzada, y todas sus orillas adyacentes correspondientes son caminos erróneos ó caminos que empeoran la solución del problema, entonces dicha esquina se consideraría óptima. Por lo tanto el vector X^* actual, correspondería a la esquina óptima, el resultado de la función habrá sido inmejorable y el método se detendrá.

Aparentemente, el método es sencillo, pero los problemas reales

comienzan cuando se intenta analizar el algoritmo algebraico del método.

Para la interpretación del método, se sabe que una esquina es un punto de intersección de "n" planos diferentes, cada uno dado por una incógnita perteneciente al problema, y que el conjunto factible de un problema de programación lineal es determinado por la intersección de los subespacios de "m" desigualdades generadas por $x \leq$ ó $\geq b$ y "n" desigualdades más, generadas por la no-negatividad.

Si el conjunto factible fuera vacio, entonces es claro que no existiría ninguna intersección de subespacios y en consecuencia ninguna esquina sería generada. Precisamente, esta es la tarea de la fase I; encontrar una esquina que pertenezca al conjunto factible o determinar su existencia.

Por ejemplo, una posible solución sería escoger para el vector X valores ($x_1=0$, $x_2=0$, . . . ,$x_n=0$), de tal forma que el punto solución correspondiente sería el origen. Este punto de intersección es una esquina, y si satisface las "m" restricciones del problema, la fase I termina.

Ya determinada una esquina factible se tendrá que hacer una elección entre las "n" diferentes orillas generadas por los " $n + m$ "

subespacios, de tal forma que la solución se mejore. Esa es la tarea de la fase II : decidir el nuevo camino.

Para poder llevar a cabo la fase II se hace un replanteamiento del problema original, es decir, que las "m" restricciones del planteamiento son transformadas a igualdades mediante las siguientes reglas.

i) las desigualdades de la forma $Ax \geq b$ pueden convertirse en igualdad restandole un vector Y llamado de holgura y sumandole un vector W llamado superfluo o artificial, de tal forma que estos vectores tengan "m" componentes no-negativos.

ii) las desigualdades de la forma $Ax \leq b$ pueden convertirse en igualdad mediante la suma de un vector Y llamado de holgura, este vector contiene "m" componentes no-negativos.

Así, la matriz A de dimension "m x n" se amplía temporalmente, a una de dimensión "m x (n + m)" y también se ajusta el vector X de dimensión "n" a uno de dimensión "n+m". Lo anterior surge de la inclusión de los vectores que convierten a las desigualdades en igualdades. Si existen "m" desigualdades, entonces "m" vectores más serán incluidos, y por la tanto, "m" variables más entran al problema.

Hay que hacer notar que estas variables que surgen de convertir a igualdades las desigualdades son conocidas como "variables auxiliares" y las variables originales son las "variables reales".

El vector de costos necesita también ser extendido temporalmente, por lo que se le suman "m" componentes más iguales a cero, lo que implica que la función de costos será la misma para el nuevo planteamiento. El método simplex no distingue ninguna clase se variables ni de vectores, por lo que el método no sufrirá cambios.

De esta forma, con las modificaciones ya hechas, el método simplex puede comenzar con la fase II. Hay que recordar que la fase I encontró una orilla del conjunto factible donde se intresectaban las " m + n " subespacios, de tal forma que esa orilla es un punto, donde " n - m " componentes del nuevo vector X serán seguramente cero.

Aquí se intuye que esas "n-m" componentes del vector X son las "variables libres" y las "m" componentes restantes son las "variables básicas ", entonces convirtiendo esas variables libres en cero, las "m" ecuaciones de Ax = b determinarán las "m" variables básicas, lo cual correspondería a la definición de una solución factible básica, es decir, exactamente "m" componentes del vector X son positivos, entonces pertenece al conjunto factible.

Pero aún se continúa con la duda de cuál camino seguir para mejorar la solución. Para aclarar esta interrogante, lo expuesto con anterioridad lleva a que, teniendo "m" componentes de X positivas y "n-m" iguales a cero, se busca moverse a lo largo de una orilla hacia una esquina adyacente que mejore la solución.

Se comenzará con el hecho de que, teniendo una esquina factible se desea mejorar la función objetivo, de modo que se buscará una esquina adyacente a esta que cumpla con los requisitos de optimización. Al encontar esa esquina (o vector X de longitud "m") significa que de "m" variables básicas, " m-1 " componentes lo seguirán siendo. Esto quiere decir que al encontrar una nueva esquina se tendrá que incluir en las variables básicas a aquella variable libre que ocasionó el mejor resultado. Además, una de las variables básicas se convertirá en variable libre, ya que sólo pueden existir "m" variables básicas. Así, el nuevo vector X es encontrado y estará compuesto por la nueva variable, y las " m - 1 " variables básicas que ya se tenían. Todos sus coeficientes permanecerán positivos.

En este momento, el problema real es determinar cual variable saldrá de la base y cual entrará.

En términos matriciales se tiene que la matriz A se puede dividir en dos submatrices, de tal forma que A(m,n) = (B(m,m) N(m,n-m)),

donde A(m,n) es la matriz del problema con "m" restricciones y "n" variables. B(m,m) es la submatriz cuadrada de A que contiene los vectores base (variables básicas) y N(m,n-m) es la submatriz de A que no pertenece a la base (variables libres).

El vector X también se divide en dos partes :

$$X(B) \geq 0 \text{ y } X(N) = 0$$

donde : X(B) representa los componentes del vector X que son componentes básicos de longitud "m", X(N) representa los componentes del vector X que son variables libres, de longitud "n-m".

De igual forma sucede con la función objetivo,

$$CX = (C(B), C(N))X$$

pero como $CX = C(B)\,X(B) + C(N)X(N)$

entonces $CX = C(B)X(B)$

por ser $X(N) = 0$.

De lo anterior se supone que

$$X(B) = b$$

es una solución básica factible y corresponde a un punto extremo de la región de factibilidad. Si este punto extremo no es óptimo se deberá

mover a un punto extremo cercano que mejore la solución existente.

Como se mencionó anteriormente hay que cambiar un componente de X(B) por uno de X(N). En otras palabras, hay que quitar un vector columna de la matriz B y reemplazarlo por un vector columna de la matriz N , que mejore el resultado de la función objetivo.

Para realizar lo anterior se supone que una componente de N (o variable libre) va a entrar a la base y será "a(r)" donde a(r) es el r-esima columna de la matriz A y vector columna también de la submatriz N , y que la variable que saldrá de la base B (matriz) será aquel cuyo cociente

$$\frac{X(B)_k}{a(r)_k}$$

donde $a(r) > 0$

X(B) es el vector X actual de solución.

$k = 1,\ldots,m,$

sea el menor de todas las "m" posibles operaciones.

Así, la variable "r-ésima" entrará a la base y la variable "k"

saldrá de la base.

Este proceso ideado por Dantzig garantiza el cambio de un vector columna de B por uno de N, dando como resultado, una mejor solución y el cambio pertinente de los valores de las variables base.

Ahora sólo queda indicar la forma de realizar una buena elección de la columna "r", o dicho en otras palabras, cual columna "a(r)" de la submatriz N de A, entrará a la base ó a la matriz B.

Si con cada cambio del vector X se logra un incremento, lo mejor sería que con cada iteración se buscara el mayor incremento posible. Esto se logrará si se elige la columna a(r), perteneciendo a N, con el componente más negativo correspondiente a la variable "r" en la función objetivo. Así, consecutivamente, el método terminará el proceso cuando todos los componentes en la función objetivo, que pertenezcan a las columnas de N, sean mayores que cero. Esto implicaría que, habiendo alcanzado una esquina, ninguna de las esquinas adyacentes a esta mejorarían el valor de la función objetivo.

Observaciones: Si el número de restricciones es menor al número de variables, entonces no todas las variables incluídas en el problema van a ser básicas, y si el número de variables es menor que el número

de restricciones, entonces todas las variables incluidas en el problema serán básicas, incluyendo algunas variables auxiliares.

El método simplex, en la práctica, suele ser más sencillo que en la teoría, por tal motivo se mencionarán los pasos comúnmente seguidos por el método para plantear y solucionar un problema. Se propone el siguiente ejemplo para describir más claramente los pasos.

El problema es Maximizar $Z = 100x + 200y + 400z$

s.a

$-2x - y + 4z \leq 0$

$x + 2y + 3z \leq 480$

No-negatividad ——————> $x, y, z \geq 0$

PASO 1.

Cuando se tiene un problema de P.L. lo primero que se debe de hacer es expresar la función objetivo de la siguiente forma :

$$Z = cX$$

$$\Rightarrow \quad Z = 100x + 200y + 400z$$

deberá reescribirse como

$$Z - cX = 0$$

$$\Rightarrow \quad Z - 100x - 200y - 400z = 0$$

PASO 2.

Aplicando las reglas *i)* e *ii)* tranformar todas las desigualdades a igualdades. De este paso surgirán las nuevas variables de holgura y las variables superfluas :

$$\text{Max } Z - 100x - 200y - 400z = 0$$

s.a

$$\begin{aligned} -2x - y + 4z + x1 &= 0 \\ x + 2y + 3z + y1 &= 480 \\ x, y, z, x1, y1 &\geq 0 \end{aligned}$$

Es importante tomar en cuenta que el número de variables auxiliares que entran al programa es igual al número de desigualdades existentes, de tal forma que, la primera base B que se formará, será creada por **dichas variables**, siendo todas las variables reales libres (iguales a cero) al comenzar el método. Esto implica que el primer punto a examinar dentro del conjunto factible será el punto extremo de la región factible con las coordenadas dadas por el vector "b" de recursos en las variables auxiliares.

PASO 3.

Construir una tabla con los coeficientes del programa lineal como se muestra a continuación:

		Z	x	y	z	x1	y1		
Fun. Obj. ⟶		1	-100	-200	-400	0	0	0	
var. ⟶ básicas	x1	0	-2	-1	4	1	0	0	vector ⟵ X(B) de recursos
	y1	0	1	2	3	0	1	480	

De tal modo que el punto del cual comenzará el método será de la esquina dada por las cordenadas

$$x = 0 \ , \ y = 0 \ , \ z = 0$$

que tendrá un vector de solución inicial

$$X = (x=0, \ y=0, \ z=0, \ x1=0, \ y1=480)$$

PASO 4.

Seleccionar al vector que entrará a la base con el coeficiente más negativo de la función objetivo. Si es que no existe ningún número negativo en los coeficientes entonces el vector de la base en la tabla será la óptimo. Si existe un empate entre números negativos, seleccionese arbitrariamente cualquiera de ellos.

En el ejemplo se tienen los valores de los coeficientes de la

función objetivo como sigue (-100, -200, -400), por lo tanto el tercer componente es el más negativo (-400) y el vector r = 3 correspondiente a la variable "z" que se convertirá en variable básica.

PASO 5.

Ya seleccionado el vector ó la variable que entrará a la base, seleccione la variable de salida mediante el cociente

$$\frac{X(B)_k}{a(r)_k}$$

sugerido anteriormente. En caso de exista un empate entre cocientes, aplique las pruebas lexicográficas incluidas en el Anexo A. En el caso de que todos los elementos del vector columna a(r) sean iguales a cero, se tendrá el caso de una **solución no acotada.**

En el ejemplo, el vector columna que entra a la base es (4 , 3) y el vector columna base es (0, 480). Se realizan los cocientes y se elige el resultado menor.

$$\text{Min} \quad \underset{k=1}{(0 / 4)} , \underset{k=2}{(480 / 3)} = (0, 160) \Rightarrow k=2 ,$$

por lo que la variable correspondiente a $k = 2$ en la base es la variable auxiliar "y1" que se convierte en libre.

Observación :El valor de la base para $k=1$ es igual a cero. Si se escogiera este valor por ser menor, el problema tendería a hacer un camino muy largo para llegar a una solución debido a que el cero no afectaría en las operaciones a la función objetivo, y no habría cambios que indicarán alguna mejora de resultados. El siguiente valor menor es 160 correspondiente a $k = 2$.

PASO 6.

La inclusión de una nueva variable a la base y la salida de otra origina una coordenada (r,k) la cual será el pivote que, por medio de operaciones matriciales, la columna "r" se convertirá en vector unitario, es decir, tendrá el valor de uno en el elemento k-ésimo y todos los demás elementos en la columna serán ceros. Luego de terminar con las operaciones matriciales regresar al paso 4.

En el ejemplo se tiene que, después de hacer las operaciones matriciales, la tabla se observa como sigue:

	Z	x	y	z	x1	y1	
	1	33.3	66.6	0	0	133.3	64,000
x1	0	-3.3	-3.6	0	1	-1.33	-640
z	0	0.3	0.6	1	0	0.33	160

→ pivote

Este ejemplo fué escogido especialmente para analizar su resultado.

En primera instancia sale de la base la variable "y1" y entra la variable "z" (tercera columna). Despues de realizar las operaciones el valor de "x1" es negativo (en el vector de recursos) y los coeficientes de la función objetivo son todos positivos (33.3, 66.6, 0, 133.3), y como ninguno de ellos es negativo perteneciente a las variables libres en esta iteración, ninguna variable puede entrar a la base (por ser todos positivos), y en este caso el método termina. Por otro lado, la variable "x1" es negativa (-640), por este motivo el problema no tiene solución (no se cumple la no-negatividad).

En este caso, el problema termina sin solución, con un valor de Z = 64,000 y con las variables x = y = 0 y z = 160, pero con el inconveniente de que x1 no cumple con la caracteristica de no-negatividad.

En el caso de que x1 fuera positivo y algún coeficiente de la función objetivo perteneciente a las variables libres fuera negativo, se continuaría con el paso 4, y así sucesivamente.

El método simplex es muy usual cuando las desigualdades del problema son todas menores o iguales a alguna capacidad. Pero cuando

existe alguna desigualdad **mayor o igual** este método no es conveniente. Es decir, que el método simplex no podría operar. Para superar este problema se realizan algunas modificaciones al método. Dichas modificaciones son hechas en el método de **Doble Fase**. Este método utiliza prácticamente el mismo camino que el simplex, únicamente que este método dedica la primera fase para eliminar las variables superfluas que generan las desigualdades de mayor o igual dentro del planteamiento.

Con anterioridad se mencionó que con la existencia de una restricción de este tipo, se deseará convertirla en igualdad, mediante la sustracción de una variable de holgura y la adición de una variable superflua a la vez. Este proceso genera un vector de variables superfluas el cual hay que minimizar (hacerlo cero) en la primera fase del método. Específicamente, el método resuelve, en primera instancia, la primera fase, donde el problema es minimizar las variables supreffluas $W(i)$ con $i = 1 \ldots$ (número de restricciones con mayor o igual), que estará sujeto al mismo planteamiento que en el simplex.

Para explicar esta fase se utilizará el ejemplo siguiente:

El problema es Minimizar $Z = -3x + 5y$

s.a

$$x \leq 4$$
$$y \leq 6$$
$$3x + 2y \geq 18$$
$$x, y \geq 0$$

El planteamiento de la primera fase queda como sigue:

(Primera Fase)

$$\begin{array}{lll} \text{Minimizar} & W = 3x + 2y - z1 & = 18 \\ & \text{s.a.} & \\ & x + x1 & = 0 \\ & y + y1 & = 480 \\ & 3x + 2y - z1 + w1 & = 18 \\ & x\ ,\ y\ ,\ x1\ ,\ y1,\ z1,\ w1 & \geq 0 \end{array}$$

La restricción donde existe variable W(1) deber ser restada a la función W. Esta función W deber ser evaluada en lugar de la función objetivo (Z) con los mismos procedimientos del método simplex. En esto consistirá la primera fase del método de las dos fases.

La tabla entonces comenzará como sigue:

	W	x	y	x1	y1	z1	valor de W
	1	-3	-2	0	0	1	-18 ←
x1	0	1	0	1	0	0	4
y1	0	0	1	0	1	0	6
w1	0	3	2	0	0	-1	18

Aquí se realiza el mismo proceso que en el simplex, y se trata de llegar a que el valor de la función W sea igual a cero, si sucede, se

procede a la fase dos, pero si W es diferente de cero implica que el problema no tiene solución.

Despues de hacer las operaciones, la tabla queda como sigue:

	W	x	y	x1	y1	z1	
	1	0	0	0	0	0	0
x	0	1	0	1	0	0	4
y1	-.5	0	0	1.5	1	0.5	3
y	.5	0	1	-1.5	0	-0.5	3

El valor de W es cero, por lo tanto se sigue con la **fase II** tomando la tabla enterior e ignorando la columna W y se sustituye la función W por la función objetivo Z, quedando la tabla como sigue:

	Z	x	y	x1	y1	z1	
	1	-3	5	0	0	0	0
x	0	1	0	1	0	0	4
y1	0	0	0	1.5	1	.5	3
y	0	0	1	-1.5	0	-.5	3

La segunda fase consiste en hacer unitarias todas columnas correspondientes a las variables incluidas en la base por medio de operaciones elementales. En este caso (x, y1, y) son las variables básicas, de las cuales "x" y "y" no tienen columnas unitarias. El

siguiente paso consiste en que después de haber hecho unitarias las columnas de las variables básicas, si existen elementos negativos en los componentes de la función objetivo se procede a realizar el método simplex tal y como fué expuesto.

La tabla óptima queda como sigue:

	Z	x	y	x1	y1	z1	
	1	0	0	10.5	0	2.5	-3
x	0	1	0	1	0	0	4
y1	0	0	0	1.5	1	0.5	3
y	0	0	1	-1.5	0	-0.5	3

Así, el valor óptimo al problema de Minimizar Z es 3, con X(B) = (x , y1 , y) = (4, 3, 3) y con X(N) = (x1, z1, w1) = (0, 0, 0).

De este modo, el método simplex es realizado con la existencia de restricciones con signo de desigualdad mayor ó igual. Este proceso, aunque es un poco más laborioso, es igual de eficiente que el método simplex.

En conclusión, el método simplex y el de dos fases han tenido un papel muy importante dentro del análisis de actividades. Pero su uso ha empezado a tener dificultad por el hecho de que las empresas se

desarrollan y las actividades que la componen se multiplican, de tal forma que el análisis de todas estas actividades en un sólo proyecto se vuelve muy complicado. Por este hecho, los problemas de grandes dimensiones que pueden ser analizados con el método simplex se vuelven tardados, complejos y cuentan con un gran desperdicio de memoria (implementado en un programa de cómputo) al almacenar todos los datos que describen ó representan dichas actividades.

Para este problema se analizará una nueva forma de utilizar el método simplex para grandes magnitudes, la incursión de las **técnicas de tratamiento de matrices dispersas** será una nueva alternativa para el proceso y el manejo de datos.

2.2 Matrices Dispersas.

Con anterioridad se había mencionado que una matriz dispersa es aquella que contiene muchos de sus coeficientes iguales a cero. El interés que se tiene en el uso de estas matrices es precisamente la ayuda que brinda para registrar sus datos y la rapidez para procesarlos.

En la actualidad existen muchas empresas grandes que en una sola

operación deben de determinar cómo se debe trabajar ó distribuir los recursos para obtener los mayores resultados sin necesidad de sobrepasar las capacidades existentes. Para esto las diversas actividades que se deben tomar en cuenta son incluidas dentro de una matriz. Al realizar lo anterior surge la característica de que muchas de las actividades no tengan ninguna relación con muchas otras, esta situación se representa en una matriz como un cero. Lo importante de esta característica, es que se puede tomar mucha ventaja si existen muchos ceros en la matriz, debido a que estos pueden ser guardados implícitamente.

Para entender más esta estructura de matrices se presenta a continuación un ejemplo:

Se tiene un problema de la forma Ax = b con m=8 (igualdades) y n=8 (variables), descrita por las siguientes ecuaciones,

$$
\begin{aligned}
x3 &= 3 \\
x1 + 2x6 &= 13 \\
x1 + x2 &= 3 \\
x2 - x8 &= -6 \\
x4 - 2x5 + x7 &= 1 \\
-x3 + x6 &= 3 \\
-x5 + x7 &= 2 \\
-2x4 + x8 &= 0
\end{aligned}
$$

Su matriz asociada es la siguiente:

		VARIABLES							
		1	2	3	4	5	6	7	8
N	1	0	0	1	0	0	0	0	0
o.	2	1	0	0	0	0	2	0	0
	3	1	1	0	0	0	0	0	0
E	4	0	1	0	0	0	0	0	-1
C	5	0	0	0	1	-2	0	1	0
U	6	0	0	-1	0	0	1	0	0
A	7	0	0	0	0	-1	0	1	0
	8	0	0	0	-2	0	0	0	1

Aquí, por ejemplo, la ecuación 6 no tiene ninguna relación con las variables 1, 2, 4, 5, 7 y 8.

Visto más funcionalmente, éste sistema únicamente contiene 16 coeficientes diferentes de cero, pero se necesitaría tener 64 localidades registradas. Solamente una cuarta parte de la matriz completa es necesaria, es decir, que únicamente los elementos diferentes de cero pueden explicar las características de la matriz (todos los demás son ceros implícitamente), esto hace pensar que es irrelevante el registrar los coeficientes iguales a cero. Además, al registrar la matriz entera se asume una gran desventaja: cuando se lleva a cabo el proceso de esta matriz, innecesariamente se opera sobre los elementos "ceros", dado que están tomados en cuenta, y en consecuencia, se genera un mayor tiempo de proceso, junto con el gasto inútil de memoria.

Otro tipo de procesamiento de datos auxiliar que se pudiera tener, sería por medio de la implementación del problema en una

gráfica, donde cada par "ecuación-incógnita", es asociado con un vértice y cada coeficiente con una línea.

La teoría de gráficas en este sentido, juega un papel muy importante dentro de la representación de problemas y la representación de las matrices dispersas.

En el plano de las matrices dispersas para representar problemas, se comienza asumiendo que la forma de proceso de la matriz no será completa. Es decir, que los coeficientes iguales a cero serán tomados implícitamente. Así, solamente se guardará el valor numérico de los coeficientes diferentes de cero del sistema. De igual forma se deberá guardar el número de ecuación y el número de incógnita correspondiente al coeficiente. Así, el número de operaciones que realice el algoritmo será proporcional al número de coeficientes diferentes de cero en el sistema.

A continuación se enunciará la importancia de poder plantear una matriz dispersa como una gráfica dirigida ó digráfica.

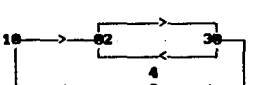
2.2.1 Modelos Gráficos

Se considera que las matrices dispersas y la teoría de gráficas son dos temas que están estrechamente relacionados. El hecho de que una matriz pueda ser representada por una gráfica, significa que los resultados en teoría de gráficas pueden ser utilizados para obtener

resultados en matrices dispersas.

Asi, como la teoria de gráficas es aplicada frecuentemente en todo tipo de industria, en la economia, en planeación, en los flujos de control, etc., en consecuencia, las matrices dispersas se aplican de igual forma. Simplemente se usará, en este caso, la teoria de gráficas como herramienta para visualizar cómo se comportan las matrices dispersas en relación a una gráfica, para lo cual es necesario agregar algunos conceptos básicos de teoria de gráficas.

Una gráfica dirigida ó digráfica en un conjunto de nodos unidos por arcos. Asi, para una matriz cuadrada dada , un nodo es asociado con cada renglón, y si $a(i,j)$ es un coeficiente diferente de cero en la matriz, entonces hay un arco que va del nodo "i" al nodo "j" en gráfica dirigida (los arcos son dibujados como flechas apuntando de "i" a "j"). Por ejemplo, se tiene la siguiente matriz con su respectiva gráfica:



Gráfica A

	1	2	3	4
1	1	1		
2		1	1	
3		1	1	1
4	1			1

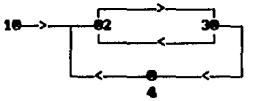
Matriz 1

La linea que sale del nodo 1 al nodo 2 en la gráfica, corresponde al elemento a(1,2) de la matriz.

Una operación fundamental para resolver sistemas de ecuaciones con matrices involucra la suma de múltiplos de un renglón a otro, para hacer todos los elementos inferiores a la diagonal principal de la columna iguales a cero (método de Gauss). Hay que considerar aquí que sumar un múltiplo de la primera fila a la cuarta (en la matriz 1), crea una nueva posición dentro de la matriz. Ese elemento es el a(4,2).

La teoría de gráficas ayuda a visualizar el cambio de coeficientes y la eliminación que tiene lugar. Por ejemplo, se tiene una gráfica G , y se llega a la gráfica Gy eliminando el nodo "y" y sumando un nuevo arco (de x a z) en lugar de una entre los nodos (x,y) y (y,z) que pertenecen a la gráfica G.

Para seguir más de cerca este proceso, se utilizará la gráfica A anterior. Considerando que se suma un múltiplo de la primera fila a la cuarta (en la matriz 1), se borra la relación existente entre el nodo 1 y el nodo 4, mientras que se genera una nueva posición dentro de la matriz. Ese elemento es el a(4,2), es decir que los nodos 2 y 4 que no estaban relacionados ya lo están. El resultado de esta operación cuenta con representación en la digráfica siguiente:



Como se puede ver, el arco (4,2) ha sido sumado. Observe que esta es precisamente la digráfica correspondiente a una submatriz cuadrada que resulta de la eliminación del coeficiente (4,1). La matriz correspondiente a la gráfica anterior se observa a continuación:

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 1 & 1 & & \\ & 1 & 1 & \\ & 1 & 1 & 1 \\ -1 & & 1 & \end{bmatrix} \end{array}$$

Cuando las matrices son relativamente "pequeñas" esta forma de usar la teoría de gráficas para reducir las matrices es efectiva, sin embargo, cuando las gráficas son "grandes" y su reducción representa muchos problemas, la forma más conveniente de procesar los datos es por medio de modelos y técnicas de matrices dispersas. Con su uso se puede lograr un ahorro computacional considerable y en el mejor de los casos, la reducción de tiempos de procesos.

En el siguiente tema se analizará una estructura en particular (la mejor en este caso) utilizada para registrar la información de las gráficas y las matrices, y que además permite el manejo más sencillo de datos.

2.2.2 Análisis Estructural de Matrices Dispersas

Posiblemente uno de los aspectos más importantes que se debe de

tener en cuenta para utilizar matrices dispersas en algún algoritmo es el fácil acceso y manejo de los datos. Para lograr esto, es indispensable que la estructura que se utilice pueda ser manejable dentro del lenguaje de programación que se desee utilizar, y que las operaciones elementales que se requieren puedan ser llevadas a cabo satisfactóriamente.

En este sentido, es indispensable mencionar la importancia que tiene la estructura de datos utilizada para registrar una matriz. Si una matriz dispersa se almacena como una matriz densa, las localidades de memoria que se utilizarán serán n x n, lo cual por ejemplo, si se tiene una matriz de dimensión n=600, entonces se requerirán 360,000 localidades para los coeficientes de la matriz, y lo más alarmante es que sólo los coeficientes diferentes de cero serán necesarios, mientras que los que son iguales a cero solamente ocuparán espacio en memoria y consumirán más tiempo de proceso. Sin embargo, si se utilizara una estructura que solamente registrara los coeficientes no-ceros, por ejemplo 251 de 360,000, el ahorro de memoria sería considerable (cualquier número de coeficientes no-cero inferior a 360,000 generaría un ahorro de memoria en este caso), al igual que el tiempo de proceso, el cual sería correspondiente al número de coeficientes no-ceros en la matriz (coeficientes registrados).

Para iniciar la búsqueda de la mejor estructura, es importante notar la diferencia entre estructuras estáticas y estructuras dinámicas.

Las **estructuras estáticas** simplemente son declaradas desde un principio. Por ejemplo cuando se define una matriz de "m x n", se está reservando espacio en memoria para "m x n" elementos. Aquí, no se tiene la seguridad de que siempre se utilicen estos "m x n" espacios en memoria. Simplemente se da este rango como un máximo de memoria utilizable, lo que no implica que se utilizará toda. Si se necesitara más memoria de la reservada se tendría que volver a redefinir la memoria desde un principio para adaptarla a las necesidades.

El motivo por el cual las estructuras estáticas no son útiles para el presente trabajo es porque, al necesitar un **arreglo** en el cual los números pueden ser directamente guardados y accesados, no se podría determinar la longitud de dicho arreglo y la memoria estaría expuesta a ser o bien desperdiciada o necesitada. Ahora si se utilizara un arreglo bi-dimensional, tri-dimensional ó multidimensional podría resultar demasiado grande para la memoria, incluso la capacidad de almacenamiento podría ser rebasada y no se podrían manejar todos los datos a la vez. Esto representa una desventaja muy grande.

Es fácil reconocer que para los objetivos perseguidos, las estructuras estáticas no ayudarán para un buen desarrollo de la investigación dado que se desperdiciaría mucha memoria (lo cual es antagónico para los objetivos propuestos). En este caso la **estructura dinámica** es ajustada para acomodar cada nuevo elemento a insertar en el límite de la memoria reservada (hasta ese momento) cada vez que se

solicite. Es decir, que para utilizar esta estructura no hay que reservar espacios en memoria al principio de un programa. En lugar de esto, solamente cuando exista la necesidad de insertar un nuevo elemento al conjunto de datos, simplemente se reserva un nuevo espacio en memoria y se anexa en ese lugar. Esto servirá para que se utilice únicamente la memoria necesaria para almacenar los datos.

De esta forma, las listas dinámicas juegan un papel muy importante dentro del proceso de matrices dispersas. En el siguiente capítulo se analizará más a fondo este tipo de estructura.

Para comenzar a describir la estructura para matrices dispersas, se considera primero el uso de un vector de "longitud llena", es decir, que no hay espacio en ese vector que no esté ocupado. En él se registrarán todos los coeficientes no-ceros de la matriz. Con lo anterior, el tiempo de proceso de la matriz estará dependiendo de tres características:

- El Grado de dispersidad que se tenga
- El número de Operaciones a ser realizadas
- Las Características del Hardware y del Software que se utiliza.

Una de las formas más convenientes de representar una matriz

dispersa, es como una lista de *elementos*, donde cada uno de ellos será a su vez, un conjunto triple compuesto por ($a(i,j)$, i, j), donde $a(i,j)$ es el valor del coeficiente, "i" es el número de restricción y "j" es el número de variable en la restricción "i". De tal forma que se tenga una lista de reales con dos enteros, de longitud igual al número de datos.

Por ejemplo,

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \left(\begin{array}{ccccc} 1 & 0 & 0 & -1 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 5 & 0 & 0 & 1 & -2 \end{array} \right) \end{array}$$

tendría la siguiente representación:

elementos	1	2	3	4	5	6	7	8
renglón	1	1	2	3	4	5	5	5
columna	1	4	1	2	2	1	4	5
valor	1	-1	2	1	1	5	1	-2

de tal forma que toda la información necesaria está incluida dentro de la tabla, y así una matriz dispersa queda convertida en una estructura de lista.

El uso de matrices dipersas requiere frecuentemente de una buena estructura de datos. La estructura más adecuada es la de listas de *elementos*, donde un *elemento* sería por ejemplo un *componente* triple (tabla anterior). De igual forma es importante poder realizar operaciones con esas listas, tal como sumar un *elemento* al final de la lista, borrar un *elemento* al final de la lista, insertar o borrar un *elemento* que se encuentra en el centro de la lista, encontrar la posición de determinado *elemento* o encontrar el siguiente *elemento* de cualquiera de ellos. Esto obliga a que la selección de cierto esquema o estructura debe permitir, por lo menos las operaciones anteriores.

Lo más recomendable es usar **listas ligadas lineales**, que son un conjunto de *elementos* ligados unos con otros y siguiendo un orden. Así cada *elemento*, contiene, además de los datos indispensables para distinguir el elemento, un apuntador el cual indica dónde el siguiente *elemento* es localizado. La tabla anterior como lista ligada se presenta a continuación:

elementos	1	2	3	4	5	6	7	8
renglón	1	1	2	3	4	5	5	5
columna	1	4	1	2	2	1	4	5
valor	1	-1	2	1	1	5	1	-2
siguiente	2	3	4	5	6	7	8	NULL

Tabla 1

donde, como se puede observar, el primer elemento apunta al segundo, éste al tercero y así sucesivamente. El último elemento no apunta a ninguno, así que la terminación de la lista está representada por el elemento NULL. Es necesario mencionar que debe existir un apuntador extra que indique dónde se puede encontrar al primer elemento. De esta forma, se puede tener acceso a todos los elementos de la lista, no importando el lugar donde se encuentren. Más adelante se tratará con más detalle esta característica.

Las operaciones fundamentales mencionadas anteriormente pueden ser realizadas con esta estructura. Por ejemplo, si se desea insertar un elemento 7 a la lista de la tabla 1, entre los números 2 y 1 correspondientes a los *elementos* 3 y 4, se hace lo siguiente:

El *elemento* nuevo que contiene el valor 7 será apuntado por el *elemento* 3, de tal forma que el nuevo *elemento* será el elemento 4 y los demás *elementos* no cambian su contenido. Por último, el nuevo elemento apuntará al *elemento* al cual apuntaba el elemento 3 (es decir al elemento 4), de tal forma que no quede ningún *elementos* sin apuntar y sin ser apuntado.

Los pasos fundamentales para realizar la inserción y eliminación de elementos serán explicados más ampliamente en el siguiente capítulo.

En resumen, se considera una matriz dispersa A. El número de elementos no-ceros corresponde a una proporción mínima cuando se compara con el número de ceros. Estos no-ceros son dispersos en toda la matriz, determinando así su estructura. Después, surgen los esquemas que ayudan al tratamiento de dichas matrices. Los elementos no-cero son almacenados en la memoria de la computadora en algún orden establecido usando una forma compacta (lista). Se utiliza un lista ligada. Se considera también la información de la porción a la que pertenece el coeficiente (i,j) respecto a la matriz, donde i es el renglón y j es la columna, además se registra la dirección del siguiente elemento. El orden es importante porque involucra la rapidez con que algún elemento se va a localizar, ya sea por filas o por columnas. Hay que tener un apuntador al primer elemento para poder tener acceso a todos y cada uno de los datos que se encuentren el la lista, de tal forma que se puedan consultar tanto valores, renglones ó columnas desde el primer elemento. Así, la esencia de un lista ligada es que existe un apuntador al primer elemento y a cada elemento es asociado un apuntador el cual puede distinguir el siguiente elemento ó el elemento NULL si es que se trata del último elemento de la lista. Esta lista puede ser barrida ó accesada desde el primer elemento a través de las ligas correspondientes a los demás elementos hasta encontrar el elemento deseado ó el término de la lista (NULL).

Esta estructura tiene requerimientos mínimos de almacenamiento y a la vez es muy conveniente para realizar operaciones tales como suma,

eliminación, permutación y transposición de *elementos*, solución de sistemas de ecuaciones lineales por métodos iterativos y métodos directos (que serán cubiertos en el siguiente capítulo). La principal propiedad de esta estructura es que los valores de los elementos son ligados por filas (ligados por renglones), tomando ventaja de las coordenadas de los coeficientes en la matriz.

Nota: Cuando se usa una lista ligada es importante entender que el orden de los elementos no es determinado por la localización física de los elementos, sino por los apuntadores de cada elemento.

2.3 Caracteristicas de Algoritmos para Dispersidad.

Al tratar de resolver un problema de la vida real propio de programación lineal, primero se busca un modelo matemático, se aplica y luego se resuelve el problema.

Algunos de estos problemas presentan gran dificultad porque contienen mucha información que se debe almacenar en una computadora. Supongase que se tiene un problema con 1,500 variables y 750 ecuaciones, tal seria el caso de una red ferroviaria, donde las variables serian los orígenes, destinos, tiempos de recorrido, cantidad

de gente,etc., y las ecuaciones serían los requerimientos propios de los trenes, los días de viajes, mantenimiento, horarios de corridas, etc.. De tal forma que la matriz A tendría 1'125,000 coeficientes en total, muchos de ellos ceros, que se tendrían que almacenar. Muchas computadoras aún no tienen la capacidad de almacenar cierta cantidad de datos, mucho menos poder procesarla.

Para esto, utilizando técnicas para el tratamiento de matrices dispersas, se busca el aprovechamiento máximo de la estructura del método simplex, que permita la solución a problemas lineales grandes. Para lograr esto se enunciarán dos características importantes acerca de la programación de algoritmos y estructuras.

I) Una característica principal es la relación que existe entre el algoritmo y su flujo de información necesario. Si este flujo no es apropiado para el algoritmo, este no procesará ni arrojará ningún resultado. Es decir, hay que tomar ventaja de las herramientas con las cuales se trabajan (algoritmos) conociendo perfectamente sus propias limitaciones.

II) Siempre existe un nuevo algoritmo para un nuevo problema. Es decir que ese nuevo algoritmo contiene las características de un algoritmo antecesor (obsoleto), y además contiene los nuevos cambios de adecuación al nuevo problema.

Estas especificaciones son aplicables cuando las dimensiones de los problemas son muy grandes y ya no es posible resolverlos con los algoritmos utilizados. En estos momentos se considera que el flujo de información no corresponde a las herramientas utilizadas, por lo que es necesario mejorar los algoritmos para hacer que correspondan con su medio. De esta forma se busca mejorar el algoritmo, de modo que se contengan las mismas características que el anterior, y ademas las nuevas especificaciones de adaptación a su medio. De esta forma los algoritmos van acordes con su flujo de información o medio ambiente, y los resultados obtenidos son más confiables y precisos.

Los algoritmos deben de corresponder, de igual forma, a la estructura del manejo de datos; específicamente, utilizando las listas ligadas para matrices dispersas, existen relativas libertades para poder maniobrar con el algoritmo de tal forma que los requerimientos de la estructura sean satisfechos. Así por ejemplo, una matriz dispersa sólo debe de ser vista como números diferentes de cero, porque son los únicos que van a ser procesados. Entonces el algoritmo tomaría ventaja del conocimiento de los lugares específicos de los coeficientes no-ceros. Por lo tanto, gracias al algoritmo, el número de operaciones realizadas por la computadora durante la ejecución del programa será proporcional al número de elementos no-ceros existentes en la matriz.

Lo anterior sería mucho más práctico que guardar todos los elementos (incluyendo los ceros si la memoria lo permite) de la

matriz y entonces saltar los valores convenientes con una instrucción IF, lo que ocasionaría un tiempo de proceso mayor y un desperdicio de memoria. Por el contrario, conociendo los valores necesarios y sus posiciones, únicamente se realizan las operaciones necesarias en los lugares ya específicos.

Otra característica muy importante de un buen algoritmo para matrices dispersas, es que al haberse generado resultados intermedios durante el proceso, algunos de esos resultados arrojan elementos iguales a cero (los cuales no interesan). El algoritmo debe de ser capaz de mantener la dispersidad de la matriz registrando los elementos no-ceros generados y no registrando los elementos iguales a ceros, es decir, que el algoritmo ocupara la memoria suficiente para los coeficientes no-ceros sin tener ningún desperdicio de memoria (Fill-in). A esta característica se le conoce como **preservación de dispersidad** de la matriz.

En el siguiente capítulo se analizará estas características en algoritmos para resolver sistemas de ecuaciones.

CAPITULO TERCERO

ESTRUCTURA DINAMICA PARA LA SOLUCION DEL PROBLEMA $Ax = b$

C A P I T U L O T E R C E R O

ESTRUCTURA DINAMICA PARA LA SOLUCION DEL PROBLEMA Ax = b .

3.1 INTRODUCCION

Este capitulo estará enfocado a la comprensión de la lógica de las listas ligadas dinámicas en relación con algoritmos encaminados a la realización de operaciones tales como inserción ó eliminación de elementos, tanto al principio y en medio, como al final de la lista, al igual de cómo crear una de ellas.

El lenguaje en que estarán desarrollados los algoritmos será en lenguaje C, dado que éste presenta gran facilidad y flexibilidad para el uso de este tipo de listas, apuntadores, operadores y manejo de memoria. Se analizarán todos y cada uno de los algoritmos de acuerdo a las necesidades y características de un buen algoritmo.

De igual forma, el análisis de técnicas para la solución a un problema planteado como Ax = b es primordial ya que su planteamiento

geométrico debe corresponder al análisis matemático.

Se comenzará con la característica que tiene el método simplex (que será el método que se implementará en el siguiente capítulo) de inicializar sus iteraciones con una solución inicial, donde todas sus variables básicas son iguales a cero, y sus variables auxiliares son iguales al coeficiente correspondiente en el vector de recursos, de tal forma que su punto inicial siempre será el origen.

Por tal motivo se plantea la alternativa de suministrar una solución factible inicial dada por una sub-matriz de A, la cual contenga variables básicas ya solucionadas; esto implicaría, por la teoría del simplex (visto en el capítulo primero), que esa solución pertenecería a una orilla del conjunto factible de soluciones, de modo que el método únicamente tendrá que buscar el las esquinas ú orillas adyacentes para localizar la esquina óptima. Esto, hablando en términos prácticos, podría ahorrar tiempo ya que las iteraciones que el método simplex tendría que realizar serían menos que cuando se inicializara con el origen.

Para esto, se analizarán dos métodos para dar solución a sistema de ecuaciones de los cuales se suministrará dicha solución factible básica resultante. Uno de ellos será un método directo y otro será iterativo. Se enunciarán sus características, ventajas y desventajas.

Estos métodos estarán implementados para poder ser manejados con la estructura de listas ligadas, y de igual forma, se analizarán en términos de dispersidad.

Por último se harán algunas observaciones respecto al uso de los métodos mencionados, a las implementaciones de algoritmos y a la estructuración total con el método simplex.

3.2 Estructura Dinamica .

Se han mencionado hasta el momento tres conceptos importantes en los cuales se fundamenta el desarrollo de la implementación de las matrices dispersas:

- Almacenar únicamente los elementos diferentes de cero.
- Realizar operaciones únicamente en el los elementos diferentes de cero.
- Preservar la dispersidad en el proceso.

La estructura dinámica es por muchas razones la más conveniente en el uso de matrices dispersas, ya que con esta permite que se lleve a cabo cualquiera de las tres características mencionadas con anterioridad.

Específicamente, sólo se registrarán los coeficientes no-ceros, y puesto que no se registran ceros, las operaciones únicamente se realizarán sobre los números registrados. Y con la facilidad de poder insertar ó remover *elemento* de la lista, se puede hablar de una preservación de dispersidad.

Se cumplen las tres características, y además toda la memoria de la que se dispone se ocupará, de modo que no hay desperdicio de memoria. Por esas razones se puede afirmar que la estructura dinámica es la más adecuada para este tipo de procesos. Cabe recordar que la estructura dinámica surge de la necesidad de tener flexibilidad en el manejo de datos, de tal forma que la memoria de la que se disponga será acorde a las demandas. Es decir, se deseaba crear una estructura que se expandiera de modo que la memoria física siempre estuviera llena ó completa.

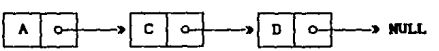
Para lograr lo anterior, la memoria dinámica es expandida cuando se necesita durante el tiempo de ejecución del programa, de tal forma que esta no se reserva en un principio. Este tipo de estructura es muy flexible ya que durante el tiempo de ejecución se podrá disponer de memoria cada vez que se solicite.

Lo dinámico de la estructura permite expander ó contraer tal como se vaya necesitando. Para esto, el uso de operaciones, tales como el anexar, eliminar y reacomodar *elemento* dentro de una lista resulta ser fácil, rápido y eficiente.

Antes de ejemplificar estas operaciones hay que tener en cuenta que cada elemento en la lista ligada contiene un apuntador al siguiente elemento (Ver Cap. 2), y que el último elemento apunta a un marcador ó final de lista llamado NULL.

A continuación se realizarán esquemáticamente las operaciones básicas utilizadas en las listas ligadas.

Se considera la siguiente lista:

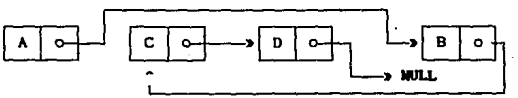


Cadena 1

Para sumar elementos a la mitad de la lista, primero se puede pensar que se deben reacomodar todos los elementos siguientes a la posición donde se quiere insertar para no perder el orden. Sin embargo, para sumar en una lista ligada, simplemente se localiza el elemento a insertar al final de la lista y se reacomodan los apuntadores de los elementos directamente afectados de acuerdo al orden que se requiera.

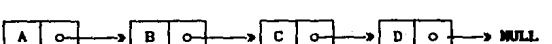
Por ejemplo, se desea sumar un elemento B despues de A en la Cadena 1. Esquemáticamente, se acomoda hasta el final de la lista el

elemento B y simplemente se ajusta el apuntador de A para que apunte al elemento B, y por último, el apuntador de B se debe dirigir al siguiente elemento de A, es decir a C. Este proceso se ve como sigue:



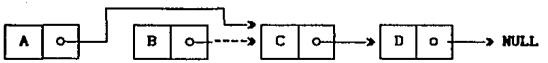
Ahora bien, borrar un elemento resultaría ser una actividad un tanto laboriosa ya que se generarían muchos movimientos por la razón de que, después de borrar el elemento no se desearía tener un espacio vacío a la mitad de la lista, de modo que se recorrerían todos los contenidos de los demás elementos. Esto no sucede con las lista ligadas, ya que para borrar un elemento simplemente se sobrepasa. Es decir, el apuntador que lo señala ya no se dirigirá a el, sino que apuntará al elemento siguiente del que va a ser borrado, y así se elimina con efectividad este elemento.

Por ejemplo se tiene la siguiente lista



y se desea borrar el elemento B. Primero el apuntador de A apuntará a C, de tal forma que a B no se pueda tener acceso, es decir, este

elemento queda aislado. De este modo este elemento ha sido eliminado de la lista, y la cadena se vería como sigue:



En conclusión, el único movimiento de datos necesario para realizar las operaciones anteriores será el movimiento de apuntadores; los elementos permanecen siempre en su localidad original.

El papel que juegan los apuntadores es importantísimo, ya que éstos, siendo el direccionamiento específico de la memoria, son la esencia misma del alojamiento de la memoria dinámica. El lenguaje C tiene la característica de que se pueden manejar apuntadores, de tal forma que, para su declaración, se incluye a la variable un asterisco (*), esto hace entender que la variable definida con un asterisco previo es un apuntador a datos.

En este caso se definirá una estructura que contendrá los datos necesarios (como ya se había discutido) para manejar matrices dispersas. Estos datos son: valor del coeficiente, número de renglón, número de columna y un apuntador al siguiente elemento. La declaración de esta estructura será la siguiente:

```
struct lista_ligada {
     float valor;
     int   renglón;
```

```
        int    columna;
        struct lista_ligada *siguiente;
};
```

Lista 1

Se define la estructura "lista_ligada", "siguiente" es un apuntador que contiene la dirección de una estructura de lista_ligada (este apuntador contendrá la dirección del siguiente elemento). Hay que hacer notar que en esta estructura no se podrá desplegar su contenido con los comandos de impresión " printf " ó realizar operaciones aritméticas directamente sobre ellos.

Por otro lado, la lista 1 únicamente declara una estructura. Si se desearan tener "n" elementos de estos se tendrían que predeclarar "n" variables iguales (memoria estática). Para eliminar este problema, ya que no se desea usar arreglos, se define una variable que apunte a una estructura y que permita ligar ó apuntar, junto con el apuntador interno de cada elemento, al registro contiguo. Este campo es un apuntador a una estructura de tipo lista_ligada llamado ENLACE. La lista quedaría como sigue:

```
struct lista_ligada  {
            float valor;
            int   renglón;
            int   columna;
            struct lista_ligada *siguiente;
      };
typedef struct lista_ligada ELEMENTO;
```

```
typedef ELEMENTO *ENLACE;
ENLACE  c;
```

Lista 2

Para hacer más manejable la estructura se definió una nueva variable ELEMENTO que es de tipo estructura lista_ligada y ENLACE es un apuntador a esta estructura, y "c" es un apuntador de tipo ENLACE, o sea que apunta a una estructura ELEMENTO, y en consecuencia esta variable puede apuntar a cualquier elemento de la lista con la estructura lista_ligada.

En resumen, en lugar de arreglos se tiene un conjunto de direcciones en memoria donde se encuentran guardados los elementos. El apuntador "c", puede dar la dirección de cualquier elemento de la lista. El apuntador que contiene cada elemento contiene la dirección del siguiente elemento. De modo que si se deseara obtener el valor del campo "valor" de algún elemento, simplemente se opera desde el apuntador "c" a la estructura para obtener el valor, es decir que la expresión "c->valor" tiene el contenido del campo valor del elemento al cual esta apuntando "c". Si se desea referirse al elemento siguiente del que se esta apuntado, se utilizará el apuntador interno de cada elemento para referirse a la dirección del siguiente. Esta operación se realiza con la expresión "c->siguiente->valor", donde "c" contiene, en primera instancia, el valor del campo "siguiente" el cual es la dirección del siguiente elemento, de este se extrae el contenido del campo "valor". Este aspecto se explicará detalladamente más adelante.

Ya definida una estructura, muy relacionada con esta se encuentra el manejo de memoria. Hay que hacer notar que en la definición de la lista, en ningún momento se reservó espacio en memoria, esto indica que cada vez que se requiera sedefinirá espacio mediante la instrucción **malloc()**. Esta función reserva espacio en memoria para almacenar el nuevo ELEMENTO. La instrucción se invocaría como sigue :

malloc(sizeof(ELEMENTO))

lo cual reserva el espacio en memoria del tamaño de ELEMENTO, este espacio es alojado a la cabeza del área destinada para trabajo en la memoria RAM.

Otra operación que se puede realizar es la eliminación de elementos (que fué explicada anteriormente). Se había mencionado que simplemente se sobrepasa el apuntador por el elemento que estaba apuntando. De aquí surge un problema, el elemento sobrepasado está ocupando un espacio en memoria de un elemento que ya no se utilizará más, es decir que el espacio que ocupa dicho elemento sobrepasado no se puede utilizar. Para resolver este problema se tendría que liberar dicho espacio en memoria para poder utilizarlo después.

Para realizar lo anterior simplemente se debe apuntar al elemento que se desea liberar (el cual ya fue sobrepasado) y se debe utilizar la instrucción **free()**. Por ejemplo, si el elemento que esta apuntado

por "c" se desea liberar simplemente se ocupa la función de la siguiente forma:

```
free(c)
```

e inmediatamente se podrá disponer de la memoria liberada.

Advertencia : Es muy importante no perder las direcciones de los elementos ya que no hay forma de recuperarlos. Siempre será recomendable tener un apuntador "privado" el cual contenga la dirección del primer elemento, y cuando se necesite dicha dirección, simplemente se iguale un apuntador "auxiliar" a este.

Ya definida la estructura y las operaciones básicas, surge el problema de cómo accesar a la información de la lista únicamente con los apuntadores, ya que no hay otra forma de accesar. Para esto, habiendo un apuntador "c", el cual contiene la dirección del elemento que se desea observar, se utiliza un operador " -> " que en lenguaje C sirve para "apuntar a un campo de una variable"; en este caso la variable es una estructura. La expresión " c -> valor " es equivalente a " (*c).valor ". De esta forma, "c -> valor" contiene el valor del campo "valor" del elemento que se esta apuntando, y "c -> siguiente" contiene la dirección del siguiente elemento la cual es asignada al apuntador, de tal forma que "c" estará apuntando al siguiente elemento

del especificado. En otras palabras, el apuntador "c" contiene la dirección de la estructura actual y " c->valor " es el valor guardado en el campo "valor".

Si "c" apunta a un elemento y se desea saber lo que contiene el campo "valor" del siguiente elemento, se deben usar los apuntadores que se definieron en la estructura para cada elemento, de modo que "c->siguiente->valor" obtiene el contenido del campo "valor" del elemento "siguiente" del que se esta apuntando, pasando por el contenido del campo "siguiente" del elemento apuntado (que contiene la dirección del siguiente elemento). Es decir, que se refiere al contenido del campo "valor" del elemento apuntado por "siguiente" en el elemento apuntado por "c".

Ahora, si se tiene una lista larga de elementos y se desea obtener el nombre del cuarto elemento, se tendrá que usar la expresión "c->siguiente->siguiente->siguiente->valor". Como se puede observar este tipo de busqueda parece ser no muy práctica, especialmente si se desea estar buscando frecuentemente elementos en una lista grande.

Una forma muy fácil para solucionar esto es tener un apuntador que se mueva a lo largo de todos los elementos, examinando sus contenidos hasta encontrar el buscado. Por ejemplo, la siguiente rutina realiza la lectura de una lista completa ya creada y la impresión de su contenido.

```
imp_lista(cc)
ENLACE cc;
{     while(cc != NULL) {
          printf(" valor = %f   renglón = %d  columna = %d",
                  cc->valor, cc->renglon, cc->columna);
          cc=cc->siguiente;
      }
}
```

Lista 3

Como se puede observar la variable "cc" apunta a un elemento y se le extrae la información para imprimirla, después ese mismo apuntador "cc" se iguala a sí mismo pero incrementado al siguiente elemento. Esta operación se realiza hasta que "cc" apunte al final de la lista **NULL**. El apuntador "cc" contiene, en un principio, la dirección del primer elemento y en lo sucesivo, va apuntando a todos y cada uno de los elementos de la lista.

Los procedimientos anteriores se realizan únicamente cuando la lista ya ha sido creada. A continuación se explicará el procedimiento para crear y ligar el primer elemento de una lista. El procedimiento "comienza" describe el procedimiento a seguir.

```
ENLACE comienza(valor, renglon, columna)
float valor;
int renglon, columna;
{       cabeza = (ENLACE)malloc(sizeof(ELEMENTO));
```

```
        cabeza -> valor = valor;
        cabeza -> renglon = renglon;
        cabeza -> columna = columna;
        fin=cabeza;
        fin -> siguiente = NULL;
        return(cabeza);
}
```

Lista 4

Los parámetros suministrados a la función son los datos requeridos de cada elemento (tratados en el capítulo anterior). Para comenzar con esta función, se genera un espacio en memoria para el nuevo elemento con la función **malloc()** y su requerimiento de espacio. Al principio de esta instrucción aparece (ENLACE), esto quiere decir que se están utilizando **la dirección de el espacio en memoria y no directamente el valor de alguna variable.** Luego hay que tener definidos como globales los apuntadores de tipo ENLACE que son cabeza (que apuntar al principio de la lista) y fin (que apuntará a el último elemento de la misma). Se asignan los valores desde el apuntador "cabeza" para almacenarlos, luego se iguala "fin" con "cabeza" por ser el primer elemento (estas dos variables estarán apuntando a la misma dirección) y luego "fin" apuntará al fin de lista NULL. Se regresará como resultado de la función la dirección donde se alojan los datos del primer elemento (o sea, del único).

El procedimiento para continuar insertando elementos en una lista ya creada es parecido al de la lista 4, pero con el único cambio de que

el nuevo elemento generado será apuntado por un predecesor. Este nuevo elemento apuntará al fin de la lista NULL.

Las operaciones se realizarán como sigue:

```
fin -> siguiente = (ENLACE)malloc(sizeof(ELEMENTO));
fin = fin ->siguiente;
fin -> valor = valor;
fin -> renglon = renglon;
fin -> columna = columna;
fin -> siguiente = NULL;
return(cabeza);
```

En un principio se genera un nuevo espacio en memoria, donde se alojará el nuevo elemento, desde el campo "siguiente" del elemento apuntado por "fin" de lista. En otras palabras, se genera un nuevo elemento ligado al final de la lista. El apuntador "fin" se recorre al elemento final (el creado) y ahí asignan los valores pasados por los parametros. Este elemento apuntará a NULL (al final de la lista). Por último, el valor que regresa la función es la dirección de donde se encuentra el primer elemento.

NOTA : Es muy importante recalcar que, suponiendo que "cabeza" sea el apuntador a la cabeza de la lista, y que ningún otro apuntador contiene la misma dirección, si se realizaran

operaciones directamente con ese apuntador " cabeza", la dirección del principio de lista corre peligro de perderse y no se podrá recuperar. Lo más conveniente es tener la dirección en un apuntador que no se use, y cada vez que se necesite, igualar otro apuntador con este para obtener la dirección y poder realizar las operaciones desde el principio de la lista.

Los anteriores procedimientos únicamente se ocupan cuando no existe la lista y se desea crear. Si se deseara anexar un nuevo elemento en una lista ya creada, únicamente el apuntador del elemento anterior, es dirigido al elemento que se va a incertar, luego el campo "siguiente" del elemento nuevo se direcciona al elemento posterior. El procedimiento se presenta a continuación:

```
Sumar(anterior,nuevo)
ENLACE anterior,nuevo;
{    ENLACE auxiliar;
     auxiliar = anterior -> siguiente;
     anterior -> siguiente = nuevo;
     nuevo -> siguiente = auxiliar;
}
```

Lista 5

Este procedimiento ya se había visto esquemáticamente con anterioridad. Los parámetros de la función corresponden a dos

apuntadores; el apuntador "nuevo" contiene la dirección del elemento que ya ha sido creado y solamente se desea insertar en la lista después del apuntado por "anterior". El apuntador "auxiliar" apunta hacia el elemento siguiente del apuntado por "anterior". De tal forma que el elemento apuntado por "anterior" apuntará ahora al nuevo elemento, y este contendrá la dirección del apuntado por " auxiliar ".

Para borrar un elemento tiene mayor facilidad. Unicamente, como ya se mencionó, se sebrepasa el apuntador del elemento previo al que se quiere eliminar, dirigiéndolo al elemento que apunta el que va a ser borrado. Posteriormente se libera la memoria de dicho elemento para poder ser utilizada después. El procedimiento quedaría de la siguiente forma:

```
elimina(cc,valor)
ENLACE cc;
float  valor;
{  ENLACE borrar,auxiliar;
   auxiliar=cc;
   while(auxiliar->siguiente->valor != valor
                        || auxiliar != NULL)
         auxiliar=auxiliar->siguiente;
   if(auxiliar == NULL )
         printf(" No existe ese elemento ");
   else {
         borrar=auxiliar->siguiente;
         auxiliar->siguiente=auxiliar->siguinte->siguiente;
         free(borrar);
   }
}
```

Lista 6

El apuntador "cc" contiene la dirección del primer elemento de la lista, "borrar" apuntará al elemento a ser borrado y "valor" contiene el valor buscado. El apuntador "auxiliar" se iguala a "cc" para barrer desde el principio de la lista hasta encontrar el elemento, que su siguiente contenga, en el campo "valor", el valor buscado. Este será el elemento que va a ser borrado a menos que se encuentre NULL primeramente. Si encuentra el elemento este será apuntado por "borrar", se ajustarán los apuntadores y se libera la memoria del elemento al cual es apuntado por "borrar".

Todas las necesidades que puedan surgir dentro del uso de listas ligadas pueden ser resueltos deduciendolos de los procedimientos expuestos con anterioridad, dado que estos son primordiales para llevar a cabo el buen funcionamiento de listas ligadas.

Estos tipos de procedimientos serán primordiales en el tratamiento de los algoritmos utilizados en las técnicas para la solución de problemas $Ax = b$. Se implementará y se analizará su comportamiento en algoritmos para solucionar sistemas de ecuaciones lineales, tanto directa como iterativamente, en el siguiente tema.

3.3 Tecnicas Para La Solucion de Sistemas de Ecuaciones Lineales.

Se habia mencionado que el método simplex inicializa sus iteraciones donde todas sus variables reales son iguales a cero, es

decir, comenzará desde el origen. También que una solución factible básica es aquella que contiene estrictamente "m" componentes positivas, donde "m" es el número de restricciones, y que la intersección de los "n+m" subespacios generados por el problema, forman un conjunto convexo donde la función objetivo alcanzaba su valor óptimo y que sería un punto extremo de este.

Ahora, si existía un conjunto de "k" columnas de la matriz A, dado que "$k \leq m$", que fueran linealmente independientes, es decir, que la matriz A se pueda dividir en dos submatrices

$$A(k,m) \mid A(n-k,m)\ X_i = b$$

donde $X_i \geq 0$, con $i = 1 \ldots m$

$b = (b_1, b_2, \ldots, b_m)$

entonces el punto

$$X = (x_1, x_2, \ldots, x_k, 0, 0, \ldots, 0)$$

es un punto extremo del conjunto de soluciones factibles.

Las características anteriores hacen suponer que si se desprende de la matriz A, $k = m$ columnas, se tendría una submatriz de dimensión "m x m" y se determinara su solución por medio de alguna técnica para

sistemas de ecuaciones lineales, esta solución, por lo anterior, se consideraría como un punto extremo del conjunto de soluciones factibles, y como el punto óptimo de la función objetivo se encuentra justamente en una esquina de dicho conjunto de soluciones, suministrando el punto factible encontrado por un método auxiliar, las iteraciones del método simplex podrían reducirse considerablemente, ya que únicamente el método buscaría en las esquina vecinas a ese punto para encontrar el punto óptimo. Más adelante, en el punto 3.4, se analizarán las características de esta solución factible suministrada, con respecto al método simplex y sus consecuencias.

Por el momento, el objetivo será introducir las técnicas para encontrar la solución al sistema de ecuaciónes dado por la sub-matriz de A de dimensión "m x m", para luego implementarlos en el manejo con listas dinámicas.

Una de las técnicas que se analizarán será la **Técnica Directa** que esta basada en la **Eliminación de Gauss**. Otra técnica será la **Técnica Iterativa** que esta basada en los métodos de **Generación por Filas** (Row - Generation).

3.3.1 Técnica Directa

Para poder encontrar una solución efectiva de un sistema de ecuaciones que generan dispersidad por metodos directos, se deben

cumplir tres aspectos importantes:

- Adaptar un método número de un caso denso a un caso disperso.
- Mantener la dispersidad en el sistema
- Realizar una eficiente implementación de un proceso de búsqueda de solución.

Se tiene el problema Ax = b, donde A es una matriz dispersa cuadrada no singular de "n x n" y b es un vector completo.

Los métodos directos están basados en la eliminación de Gauss: las ecuaciones ó los coeficientes de un sistema son modificados en pasos sucesivos hasta que la solución es encontrada.

El método de eliminación gaussiana se basa en la utilización directa de los elementos de la diagonal usados como pivotes en el orden en que van apareciendo. La característica que deben cumplir es que ninguno de los elementos de la diagonal principal debe de ser cero.

Este método consiste en "n" pasos. El propósito del k-ésimo paso es eliminar todos los elementos no-cero de la matriz que se encuentren bajo la diagonal principal en la k-ésima columna. En el primer paso los elementos no-cero de la columna 1 de A, es decir A^1, son eliminados por

substracción de un múltiplo de la fila i, elemento por elemento. El elemento a(1,1) pertenece a la fila (fila 1) que va a ser substraida de las otras filas y el primer elemento de la columna que va a ser eliminada (columna 1) y se le llama **pivote** asumiendo que es diferente de cero.

Antes de la eliminación, la fila 1 es normalizada dividiendo todos sus elementos no-ceros entre el pivote. La matriz resultante A^2 estará formada por los elementos $a(i,1) = 0$ con $i > 1$, y con $a(1,1)=1$. Para el segundo paso el elemento a(2,2) es seleccionado para ser el pivote. De igual modo, se asume que es diferente de cero. La fila 2 es normalizada y todos los elementos bajo la diagonal principal que son no-ceros de la segunda columna son eliminados substruyendoles un múltiplo conveniente de la segunda fila ya normalizada. Hay que hacer notar que siendo el elemento $a(2,1) = 0$, los elementos de la columna 1 no serán alterados. Asi la matriz A^3 es obtenida con $a(i,1)=0$ para $i >> 1$, $a(i,2)=0$ para $i > 2$, y $a(1,1) = a(2,2) = 1$, lo que es similar a ir generando una matriz triangular superior con diagonal unitaria.

Generalizando, al principio del la k-sima iteración se tendría una matriz A^k con ceros en sus primeras $k-1$ columna bajo su diagonal principal y unos el los $k-1$ primeros elementos de la diagonal.

Para ejemplificar lo anterior se supone una matriz de 5 x 5 en la iteración a realizar $k = 4$, la matriz A^5 quedaría como sigue:

$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c}
\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{pmatrix}
1 & x & x & x & x \\
 & 1 & x & x & x \\
 & & 1 & x & x \\
 & & & x & x \\
 & & & x & x
\end{pmatrix}
\end{array}
$$

En el k-esimo paso $a(k,k)$ es seleccionado para ser pivote. La fila k es normalizada y los elementos de la columna k bajo la diagonal son eliminados por la substración de un múltiplo conveniente de la fila normalizada a las filas las cuales tienen no-ceros en la columna k bajo la diagonal.

Este proceso continúa hasta el final del n-ésimo paso donde la matriz A^{N+1} es generada, la cual contiene solamente ceros bajo la diagonal principal y unos en la diagonal. Es decir, se obtiene una matriz triangular superior con sus elementos de la diagonal iguales a uno.

El método de **Eliminación por columnas de Gauss-Jordán** es similar a la eliminación gaussiana por columna, la diferencia radica en que al principio de la k-ésima iteración la matriz A^k tiene ceros en sus columnas de la 1 a la k-1 tanto arriba como abajo de la diagonal. Para representar la diferencia se utilizará el mismo ejemplo de n=5 y k=4, pero ahora habiendo realizado el método Gauss-Jordán.

$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c}
\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{pmatrix}
1 & & & x & x \\
 & 1 & & x & x \\
 & & 1 & x & x \\
 & & & x & x \\
 & & & x & x
\end{pmatrix}
\end{array}
$$

De modo que el k-ésimo paso consiste en la eliminación de los no-ceros de la columna k tanto arriba como abajo de la diagonal. Primero la fila k es normalizada dividiendo todos sus elementos por el coeficiente de la diagonal. Entonces un múltiplo conveniente de la fila k es substraida de todas aquellas filas la cuales tienen un elemento no-cero en la columna k tando arriba como abajo de la diagonal. La matriz A^{k+1} es obtenida, con ceros en sus k columnas iniciales y unos en la diagonal hasta la columna k. Este proceso es terminado hasta el final del paso m, donde la identidad $A^{M+1} = I$ es obtenida, donde I es la matriz unitaria.

Para determinar los valores de las raíces de las ecuaciones únicamente al principio del proceso se aumenta a la matriz A el vector de recursos "b", de modo que las operaciones también afectan a este vector, pero de ninguna manera alguno de los componentes de este vector podrá formar parte de la diagonal. Es decir, este vector sólo será afectado por el proceso del método, y no determinará ningún cambio sobre la matriz. De modo que cuando la matriz es igual a la matriz unitaria se ha encontrado entonces la solución al sistema de ecuaciones, distinguiendo esos valores en el vector "b" que se aumentó al principio del proceso despues de haber sido afectado por las m iteraciones. Asi, la matriz aumentada final tiene la siguiente apariencia:

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc|c} 1 & 2 & 3 & 4 & 5 & \end{array} \\ \left(\begin{array}{ccccc|c} 1 & & & & & b1 \\ & 1 & & & & b2 \\ & & 1 & & & b3 \\ & & & 1 & & b4 \\ & & & & 1 & b5 \end{array}\right) \end{array}$$

Como se puede observar este método (Gauss-Jordan) utiliza todos los coeficientes de la matriz pero únicamente utiliza los componentes que son diferentes de cero para determinar la nueva matriz en cada paso, de modo que es fácil reconocer que se realizan cálculos de más. De hecho, los elementos iguales a cero no existen en la matriz, es decir se encuentran implícitos.

3.3.2 Código del Método de Gauss-Jordán (Método Directo)

El listado que realiza este método se presenta en una forma muy sencilla, dado que sigue la estructura del algoritmo de Gauss-Jordan tal y como se presenta, y además se presenta también en forma estructurada para utilizar en su totalidad llamadas a procedimientos que rigen el desarrollo del método.

Se presenta la rutina que lleva a cabo el método junto con su desarrollo y explicación.

```
1    GaussJordan(lista,limite)
2    ENLACE lista;
3    int limite;
4    {    int arreglar,principal;
```

```
5          principal=1;
6          while (principal <= limite) {
7                    no_solucion='s';
8                    while( no_solucion == 's') {
9                         hazunoprincipal(lista,principal,principal);
10                        if(no_solucion=='s')
11                             buscaderecha(lista,principal,principal);
12                   }
13                   arreglar=0;
14                   while (arreglar <= limite) {
15                        if (arreglar == principal) i++;
16                        if (arreglar > limite) break;
17                        hazunocolumna(lista,principal,arreglar,
18                                                       principal);
19                        arreglar++;
20                   }
21                   principal++;
22         }
23   }
```

Para comenzar con el procedimiento, se suministra el apuntador al primer elemento de la lista y se proporciona un límite (línea 1). Este límite es el que le indica al método cuantas variables se desean calcular. Si por ejemplo se tiene una matriz de 5 x 4 y se desea obtener las raices (ó soluciones) de cuatro variables, hay que calcular las raices de la sub-matriz de 4 x 4.

La variable "principal" representa el número de iteraciones, y en consecuencia, el renglón en turno que se va a normalizar. El método iterará hasta que "principal" sea igual al límite establecido (linea 6).

Realizando la primera iteración, la variable "no_solución" se hace verdadera (línea 7) para que se ejecute la búsqueda del nuevo pivote (línea 8) en la posición (1 , 1). Esto lo realiza la función " HazUnoPrincipal(lista, principal, principal)" (línea 9), donde los dos últimos parámetros representan el renglón y la columna donde estará el pivote. En el caso que de que no exista coeficiente en el lugar indicado el procedimiento hace a "no_solución" = 's', de tal forma que se llamará al procedimiento "BuscaDerecha(lista,principal,principal)" (líneas 10 y 11) que buscará un elemento a la derecha sobre el mismo renglón especificado por el segundo parámentro de la lista. El método que emplea este procedimiento es el algoritmo de Hall, que buscará un elemento a la derecha de la posición requerida (Ver Anexo C). Si encuentra un elemento, entonces permutará la columna donde encuentre elemento con la columna especificada por el tercer parámetro de la función (es decir, en la columna donde no se encontró coeficiente) , de tal forma que el pivote en (1 , 1) ya existirá . Si no se llegaran a encontrar elementos a la derecha, implicaría que el renglón no tiene elementos y por lo tanto el sistema no tiene solución). Esto puede concluirse diréctamente, ya que se considera la existencia de una ecuación que es dependiente a alguna otra, por lo tanto la matriz ya no es cuadrada y es singular. Se repite el procedimiento "HasUnoPrincipal" (línea 9), y como ya hay pivote en la posición requerida, hace a la variable "no_solucion" = 'n' (esta asignación se realiza dentro de esta función. Ver Anexo B) de tal forma que se sale del ciclo con un pivote (línea 12). Si desde el principio existiera elemento en (1, 1) simplemente se hace "no_solución"='n' y saldría del ciclo con un pivote.

El siguiente procedimiento será hacer ceros los elementos no-ceros de la columna donde se encuentra el pivote. Esto lo realiza el procedimiento " HazUnoColumna(lista, principal, arreglar, principal)" (línea 17), donde el segundo y el cuarto parámetro indica la localización del pivote como renglon-columna respectivamente, y el tercer parámetro representa el renglón que se modificará.

La variable "arreglar" marca el número de renglón que se está modificando (línea 13). Aquí si el renglón a modificar es igual al renglón principal (el normalizado) se salta al siguiente renglón (línea 15), y si "arreglar" es mayor que el límite (línea 16), el procedimiento de hacer cero las columnas, termina y el método continuará con el siguiente pivote (líneas 21 y 22).

Así que cuando la variable "principal" sea igual a la variable "límite" (línea 6 y 23), el método terminará con la solución al sistema de ecuaciónes especificado en el vector "b" de recursos.

Los procedimientos y funciones "HazUnoPrincipal", "BuscaDerecha" y "HazUnoColumna" están incluidos en el Anexo B, pero es recomendable analizarlos hasta después de haber repasado el tema 4.1 "La Estructura Interna de Representación" tratado el en siguiente capítulo.

3.3.3 Técnica Iterativa.

Al igual que los métodos directos, los iterativos encuentran el vector solución **x** del sistema **Ax = b**, donde **A** es una matriz dispersa cuadrada de " **m x m** " y **b** es el vector de recursos.

En este tipo de métodos, un vector inicial **x** arbitrario es normalmente usado para iniciar los procedimientos. Este vector **x** es mejorado gradualmente hasta que un promedio suficiente de aproximación es obtenido.

En este caso se analizará un método de "utilización de renglones" que es un procedimiento iterativo, el cual, sin hacer algún cambio en la matriz original A, usa sus renglones, solamente uno a la vez. Estos métodos son importantes y han demostrado su efectividad en problemas con matrices grandes las cuales no contienen ninguna estructura detectable además del alto grado de dispersidad.

El problema **Ax = b** conduciría a un problema de la forma

$$< A_i, x > = b_i,$$

donde $i \in I$ (conjunto de las I filas en el sistema)

$I = 1, 2, \ldots m$

A_i y x pertenecen a el n-dimensional espacio Euclidiano R^N

y $< \circ, \circ >$ se entiende como el producto inerte de vectores.

Cuando se tiene la creencia de que los datos representados por las ecuaciones son inexactos, " hay corrupción de modelos ", discretizaciones, etc., estos métodos son usados para buscar un punto que pertenezca a una vecindad especificada por todos los espacios definidos por las ecuaciones del problema.

Especificamente, el problema es encontrar un vector $x \in R^N$ tal que:

$$b-\delta \leq Ax \leq b+\delta$$

donde δ es la tolerancia requerida, siendo este problema de intervalo impuesto.

El medio ambiente con el cual estos problemas son tratados es distinguido por una combinación de propiedades:

- Dimensiones grandes.
- Dispersidad.
- Carencia de estructura.

En un medio como este, se sugiere el uso de los métodos de "utilización de renglones" porque sus propiedades permiten retener la

dispersidad, resiste la dimensión de cualquier problema y toma ventaja de la dispersidad sin relacionarla con ninguna estructura en particular.

El método iterativo contiene las siguientes propiedades:

- Ningún cambio es hecho en la matriz original.
- Ninguna operación es realizada directamente sobre la matriz.
- En una sola iteración es requerido solamente un renglón de la matriz.
- El algoritmo presenta pocas necesidades aritméticas.

Notaciones para el algoritmo :

La secuencia de índices ó renglones utilizados será $\{ i_k \}$ para $k=0...\infty$. Esto indica cual es la renglón de la matriz A que será utilizado. Así, de la iteración k a la $k+1$ se usa el $i_{k\text{-ésimo}}$ renglón. El control de la utilización de renglones será casi cíclico, es decir,

que con $I = \{ 1, 2, ...,m \}$,

si $i_k \in I$ para cada $k \geq 0$ y existe un entero c tal que para todo $k \geq 0$, i es un subconjunto de $\{ i_{k+1}, i_{k+2}, ..., i_{k+c} \}$, el control es casi cíclico cuando $c = m$.

Por ejemplo, si se está en la iteración 5 ($k = 5$) y existen 10 ecuaciones ($I = 10$), el renglón que se utilizará será el último del conjunto

$$\{ i_6, i_7, i_8, i_9, i_{10}, i_1, i_2, i_3, i_4, i_5 \}$$

es decir, que el renglón 5 será el utilizado.

El método a utilizar será el de **Kaczmarz** y se inicializará, como ya se mencionó, con un vector arbitrario $x^{(0)}$.

La operación a realizarse en cada iteración será la siguiente:

$$x^{(k+1)} = x^{(k)} + \frac{b_{i_k} - \langle a_{i_k}, x^{(k)} \rangle}{\| a_{i_k} \|^2} a_{i_k}$$

donde $x^{(k+1)}$ será la nueva aproximación a la solución óptima.

$x^{(k)}$ es la solución calculada por la iteración anterior.

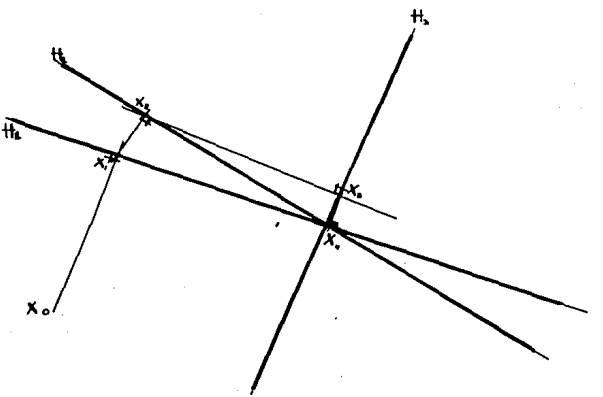
b_{i_k} es el valor del coeficiente de recursos "b" correspondiente a la restricción i-ésima en la iteración k.

a_{i_k} es el renglón i-ésimo de la matriz A en la iteración k.

Interpretación:

Dada una solución $x^{(k)}$ y un espacio H determinado por la i_k-ésima

ecuación, x(k+1) cae en una línea perpendicular a través de x(k) al plano H. Y así sucesivamente hasta aproximarse al punto óptimo. Ver siguiente gráfica.



NOTA : A este método se le puede anexar tolerancias de convergencia, al igual que límites de iteraciones, debido a que exiten problemas cuando, estando en un punto x(k) no exista alguna línea perpendicular que cruce el plano H generado por la actual restricción, de tal forma que en lugar de aproximarse a la solución, ésta se aleje ó se mantenga sin aproximación. En este caso, el método muy difícilmente convergerá. Por tal motivo, para no iterar sin solución, este método cuenta con dos criterios de paro y convergencia : cuando el error en el

vector x ya es lo suficientemente pequeño y cumpla con el mínimo de convergencia, y otro criterio es el número de iteraciones. Si la convergencia es demasiado lenta ó no se obtiene ninguna mejora en la solución, el número de iteraciones máxima se cumple y el método parará sin haber convergido.

3.3.4 Código del Método de Kaczmarz (Método Iterativo)

Este código resulta ser no tan comprensible como lo es el de Gauss-Jordan, dado que se realizan tanto productos innertes, cálculos de normas de vectores y suma de vectores. Este código no se rige por llamadas de funciones, así que todo lo que involucra el método estará incluido en el siguiente listado:

```
1     Kaczmarz( lista, varcalcular, limite)
2     ENLACE lista;
3     int varcalcular, limite;
4     {
5          ENLACE listauxi;
6          float  valor,total,error=0,errante;
7          float  producto,norma,tolerancia;
8          int    utilizando,i,num_iter,iteraciones;
9          scanf("%f",& tolerancia);
10         scanf("%d",& iteraciones);
11         for(i=1;i<=varcalcular;i++)   scanf("%f",&x[i]);
12         errante=0;
13         for(num_iter=0;num_iter <= iteraciones ;num_iter++) {
14              listauxi=lista;
```

```
15              producto=norma=0;
16              utilizando=(num_iter % varcalcular);
17              if( utilizando == 0 ) utilizando=varcalcular;
18              while(listauxi->renglon != utilizando )
19                   listauxi=listauxi->siguiente;
20              while(listauxi->columna  < limite ) {
21                   producto += (listauxi->valor) *
22                        x[listauxi->columna];
23                   norma += (listauxi->valor) * (listauxi->valor);
24                   listauxi = listauxi->siguiente;
25              }
26              while(listauxi->columna != 2000)
27                   listauxi=listauxi->siguiente;
28              total = ((listauxi->valor-producto)/norma);
29              listauxi = lista;
30              while(listauxi->renglon != utilizando )
31                   listauxi = listauxi->siguiente;
32              while(listauxi->columna < limite) {
33                   x[listauxi->columna] += total * (listauxi->valor);
34                   error+=total*(listauxi->valor)*total*
35                                             (listauxi->valor);
36                   listauxi = listauxi->siguiente;
37              }
38              if( (error - errante) < tolerancia ) break;
39              errante=error;
40         }
41    }
```

Los primeros datos que debe conocer el algoritmo son:

i) la tolerancia de aproximación del vector solución,

ii) el número máximo de iteraciones a realizarce y

iii) el vector arbitrario de longitud igual al número de variables a solucionar.

Cabe hacer la aclaración de que el tiempo que tarde el método en encontrar la solución estará en función del número de iteraciones ó del intervalo de convergancia que se proporcionen.

El número de variables a solucionar está dado por el segundo parámetro de la función "kaczmarz" (linea 1). El método iterará el número de veces que se especificó mediante la variable "iteraciones" (lineas 10 y 13) y lo primero que hace es igualar "listaaux" con "lista" (linea 14), ya que esta contiene el apuntador al primer elemento y se necesitará "barrer" la lista cada vez que se necesite un renglón diferente sin perder la dirección del primer elemento.

Las siguientes instrucciones (lineas 16 y 17) llevan el control casi ciclico que se necesita para utilizar los renglones.

```
utilizando=(num_iter % varcalcular);
if( utilizando == 0 ) utilizando=varcalcular;
```

La variable "utilizando" es la que contendrá el número de renglón que se usará en cada iteración, y está se obtendrá determinando el

remanente (ó módulo) del número de iteración con respecto al número de variables a calcular, y si este valor es cero, "utilizando " tomaría el valor del número de variables utilizadas (linea 17) (parte del control cíclico). Para aclarar este punto se propone el ejemplo utilizado en teoría. Si se está en la iteración num_iter = 5 y el número de variables a calcular es varcalcular = 10, el renglón a utilizar será:

utilizando = 5 % 10 = 5

lo que implica que se utilizará la ecuación número 5. Ahora, si num_iter = 30 y varcalcular = 10 entonces

utilizando = 30 % 10 = 0

y como if(utilizando == 0) es verdadero, entonces utilizando = 10, por lo que aquí se usará la ecuación número 10.

Ya determinado el renglón a utilizar se dirige al apuntador "listauxi" hasta el primer elemento de dicho renglón (líneas 18 y 19), esto permitirá calcular el producto inerte del renglón utilizado con el vector de solución actual x(k) generando así un escalar llamado "producto" (líneas 21 y 22). De igual forma, aquí se calculará la

norma de la ecuación que se está utilizando (línea 23), guardando su resultado en el escalar "norma". Estos dos valores serán calculados únicamente con las variables que puedan ser utilizadas, la variable "límite" demarcará hasta cual variable se debe tomar en cuenta para calcular estos escalares (línea 20). Así, únicamente falta conocer el valor del componente b del renglón que se utiliza. Para esto el mismo apuntador "listauxi" se dirigirá hasta un lugar donde la columna tenga el valor de 2000 (líneas 26 y 27). Aclarando, existe una estructura que guarda la lista internamente y el valor de la columna de todos los elementos del vector "b" de la lista será siempre 2000 (para mayor detalle consultar "La Estructura Interna de Representación", capítulo siguiente). Así, se manda el apuntador hasta la columna con valor 2000 y se obtiene el valor de "b" con la expresión " **listauxi->valor** " (línea 28).

Ya teniendo todos los datos necesarios para la evaluación se puede realizar ya la operación

$$\text{total} = \frac{\text{(listauxi->valor)} - \text{producto}}{\text{norma}}$$

(línea 28), luego se vuelve a dirigir el apuntador al primer elemento del renglón utilizado (líneas 29, 30 y 31) para proceder a multiplicar el escalar "total" por todos sus elementos (línea 33), y

el vector resultante es la medida de variación en que intervino dicho renglón para acercarse a la solución. De modo que esta variación se la suma al vector actual x(k) para generar así el vector x(k+1) (línea 33).

Estas operaciones se realizan de acuerdo a la elección del renglón que se utilizará, y con cada uno de los renglones se apoyará cada vez que sea utilizado para aproximar la solución arbitraria a la solución real.

Con cada iteración se genera una variación del vector solución, este cambio se compara con el cambio anterior y si la diferencia elevada al cuadrado es más pequeña que la tolerancia que se había impuesto (líneas 34, 35 y 38) con la variable "tolerancia", el método termina, si no lo es, se sigue iterando hasta cumplir, ya sea con el número de iteraciones ó con la tolerancia permitida.

Aparentemente suele ser complicado su análisis, pero con la práctica este método se vuelve muy claro, práctico y manejable.

3.4 Observaciones .

El método Gauss-Jordan presenta problemas cuando no encuentra elemento en la diagonal principal. Esto representa una desventaja para

este método, ya que, como la finalidad principal es manejar matrices dispersas no se podrá asegurar la existencia de algún elemento en los lugares específicos. Para esto el algoritmo de Hall (Anexo C) ayudará en la búsqueda de un elemento; trayéndolo de la posición donde lo encuentre (sobre el mismo renglón), sin alterar la estructura actual de la matriz. La única condición es que ese coeficiente pertenezca al mismo renglón. Así, con la ayuda de este algoritmo, el método de Gauss-Jordan resulta ser más eficiente ya que cuando no existe pivote se llama al algoritmo de Hall y se resuelve el problema. Este método de Hall es tan fuerte que si no llega a encontrar un elemento a la derecha sobre el mismo renglón, con toda seguridad se puede afirmar que el sistema no tiene solución.

Por otro lado, el método de Kaczmarz es mucho más efectivo en el aspecto de estructura de matrices. Unicamente habrá que dar una solución arbitraria y la solución será encontrada dependiendo de la tolerancia de error ó en el número de iteraciones que se le dé.

Es muy importante hacer notar que este método presenta problemas de convergencia cuando la solución inicial arbitraria es lejana a la solución óptima. Esto ocasiona que los incrementos de la solución inicial hacia la solución real sean muy pequeños ó que tiendan a alejarse. En este caso, probablemente el método no convergerá ó la solución no podrá ser alcanzada a menos que se realice un gran número de iteraciones.

Estos dos métodos serán usados para suministrarle una solución inicial al método simplex. Cada uno de estos métodos presentan las siguientes características en relación al Simplex :

- El método de Gauss por el hecho de que realiza operaciones en toda la matriz de acuerdo a los pivotes utilizados, resulta ser más compatible al relacionarlo con el método simplex, dado que los dos presentan el mismo proceso de eliminación. En otras palabras, los dos métodos trabajan homogeneamente con eliminación gaussiana, de modo que resultan ser compatibles.

- Por el contario, el método de Kaczmarz resulta ser más complicado al unificar sus procedimientos con los del método simplex. El primer problema que se presenta es que, al no realizar ningún cambio en la matriz A, los únicos datos que se obtienen al terminar el proceso iterativo, son el vector "b" que sería igual a la solución alcanzada por el método, una matriz A intacta y una matriz A' imaginaria que se compone de columnas unitarias correspondientes a las variables resueltas por el método. Esto puede no resultar problemático cuando la dimensión de la sub-matriz solucionada tiene el mismo valor que el número de variables totales, es decir, cuando la base puede contener a todas las variables reales.

El problema comienza cuando el número de restricciones es menor que el número de variables, es decir, la base no puede contener a todas las variables reales. El método de Kaczmarz no es homogeneo con respecto al método simplex ya que, si alguna restricción contiene por ejemplo 6 variables, de la cuales este método encuentra la solución de únicamente 4 (por tener 4 restricciones), existen dos variables que no están definidas (variables 5 y 6). Hasta este punto se sabe que existe una sub-matriz unitaria, una sub-matriz de las variables 5 y 6 (que no es posible determinar cual ha sido su cambio después del proceso, dado que no se tomaron en cuenta.) y un vector "b" correspondiente a la solución de las cuatro variables de la sub-matriz. El problema principia cuando se intenta suministrarle los datos obtenidos al método simplex. En este caso, pueden existir dos caminos :

1) la información es malentendida debido a que los coeficientes correspondientes a las variables 5 y 6 no se conocen. En este punto cualquier valor alterará el planteamiento de la matriz y la solución ya no correspondería al problema original.

Para visualizar esta problemática se representan los datos obtenidos por el método de Kaczmarz y los coeficientes desconocidos en el proceso:

$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}
\begin{array}{c}
\begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\
\left(\begin{array}{cccccc|c}
1 & & & & ? & ? & b1 \\
 & 1 & & & ? & ? & b2 \\
 & & 1 & & ? & ? & b3 \\
 & & & 1 & ? & ? & b4
\end{array}\right)
\end{array}
$$

2) en este caso simplemente habrá que tomar las variables resueltas como guia para el método simplex, es decir, que el método comenzará haciendo las columnas unitarias correspondientes a las variables resueltas, de tal forma que la matriz no altera su estructura.

Este procedimiento no es muy conveniente ya que al tiempo que tarde en inicializar se le sumará el que tarde en hacer las columnas unitarias y además el que tarde para encontrar la solución óptima, en este caso sería mas conveniente realizar el método sin inicialización, ya que únicamente realizará el proceso de encontrar columna y hacerla unitaria. Es fácil observar que se ahorra la mitad del proceso de esta forma.

En conclusión, cuando se tenga el caso de que el número de restricciónes sea menor que el número de variables, no utilizar el método de inicialización de Kaczmarz. Este problema no sucede con el método Gauss-Jordan porque este, aunque no estén involucradas las variables 5 y 6 en la sub-matriz que se va a analizar, estas si sufren los cambios convenientes ocasionado por el pivoteo, de modo que el problema no se altera.

Para visualizar más claramente el flujo y proceso interno de los datos de la lista dinámica en los métodos anteriores y en el método simplex, se analizará la estructura interna de representación dentro del proceso en el siguiente capítulo, además de la fusión de los dos métodos anteriormente cubiertos, sus consecuencias y aplicaciones.

CAPITULO CUARTO

IMPLEMENTACION DEL METODO SIMPLEX Y SUS APLICACIONES

C A P I T U L O C U A R T O

IMPLEMENTACION DEL METODO SIMPLEX Y SUS APLICACIONES.

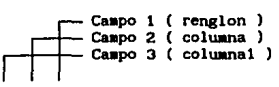
4.1 La Estructura Interna de Representacion.

4.1.1 Introducción

En esta sección se hará referencia a la forma en que una lista ligada es almacenada en memoria y poder así distinguir los componentes del problema de programación lineal que representa, es decir, sus renglones, sus columnas, el vector "b" de recursos, la función objetivo, la función W (en caso de utilizar el método de Doble Fase), las variables reales, las variables auxiliares, la base y los demás elementos utilizados para porder operar el método simplex y el de la doble fase.

Un aspecto muy importante dentro del método simplex, corresponde al uso de la base, ya que esta indicará cuales variables son las que se encuentran dentro de la solución y en que orden aparcerán. Un mal manejo de esta podría ocasionar malas interpretaciones de los datos.

La estructura de la base es exactamente igual a la de la lista ligada, únicamente que esta contendrá una estructura con los siguientes campos: renglón, columna, columna1, y un apuntador al siguiente. Cada uno de estos campos resultan ser muy importantes para cada método debido a que en cada uno de ellos, si hay un cambio de pivote (cambio de variable base) o un intercambio de columnas (cambio en el orden de las variables base), en esta lista base se verá reflejado. El campo "renglón" indicará el renglón al cual pertenece la solución de la variable de acuerdo al problema. El campo "columna" indicará las variables que en un principio formaban parte de la base. Es muy importante mantener este campo intacto, ya que las operaciones que se realicen con la ayuda de la base, se rigen por el orden que contenga este campo. Al campo "columna1" se le puede llamar "reflejo de los procedimientos", ya que este comienza con los mismos valores del campo "columna" y se va modificando de acuerdo a los cálculos y los métodos que se van utilizando. Hasta el final del proceso, este campo es el que contendrá la solución básica óptima generada por todos los cambios. Por ejemplo, si se realizó un intercambio (la columna 4 por la 1) por la razón de que no existía pivote en esa posición (columna 1), la base pasará de ser



(1, 1, 1) —→ (2, 2, 2) —→ (3, 3, 3) → 1, 2, 3

Lista Base Vars. Base

a la nueva base despues del cambio que será

(1, 1, 4) —» (2, 2, 2) —» (3, 3, 3) » 4, 2, 3
Lista Base Vars.Base

Esto quiere decir que antes del intercambio las variables 1, 2 y 3 pertenecían a la base, y después, la base se formó de las variables 4, 2 y 3. El orden es importante ya que definirá cual es el valor que corresponde a las variables en la base. Estos valores forman parte del vector "b" resultante (como ya se mencionó, este orden lo mantiene el campo "reglón"). De modo que los dos primeros campos : "renglón" y "columna", son tomados como un control de la lista.

Más adelante se seguirá haciendo mención del comportamiento de la base de acuerdo a los métodos y procedimientos seguidos.

4.1.2 Registro de Restricciones

Ahora se fijará la atención en la forma en que la lista se irá registrando en la memoria. Se irán creando los elementos y se realizará una distinción en cada uno de ellos.

Se llegó a la conclusión de que los campos incluidos en cada *elemento* de la lista contendría cuatro campos:

1) valor (coeficiente)
2) renglón (# de restricción)

3) columna (# de variable en la restricción)
4) siguiente (apuntador al siguiente elemento)

Se comenzará accesando restricción por restricción, de tal forma que todos los elementos de la primera restricción tienen el valor del campo "renglón" = 1, la segunda restricción "renglon"=2 y así sucesivamente.

Con cada elemento que se accesa se suministran dos datos: el valor del coeficiente (campo "valor") y el número de variable al cual corresponde (campo "columna"). Dado que el valor del renglón ya se conoce por el número de restricción, este ya no se accesa. El número máximo de variables reales que se puede utilizar es de hasta 999. Es decir, el número máximo de una columna, para la lectura de restricciones, será de 999.

Terminando de accesar las variables con sus coeficientes se tendrá que especificar que tipo de restricción es, si es = (igualdad), ≤ (menor o igual) ó ≥ (mayor o igual). Cada uno de estos signos trae consigo una consecuencia sobre la restricción. Esas consecuencias serán explicadas en el tema " Tipos de Restricciones".

Luego se propociona el valor correpondiente al lado derecho de la

restricción, es decir, el componente del vector "b" correspondiente a la restricción. De este únicamente se pedirá su valor, ya que el número de renglón ya se conoce, además se le asignará un valor de columna igual a 2000. Este será el valor más alto que pueda tomar una columna, y cuando esto suceda, se podrá tener la plena seguridad de que éste es un elemento del vector "b".

4.1.3 Tipos de Restricciones

Se mencionó anteriormente que existen tres tipos de restricciones: = , ≤ y ≥, donde cada una de ellas causarán un efecto diferente sobre las desigualdades.

También hay que recordar que si el número de restricciones del problema es *m*, y entonces solo *m* variables terminarán en la base, de tal forma que con cada restricción se suma una nueva posición para la base.

El cambio que se origina en la desigualdad con la existencia de cualquiera de los signos = , ≤ y ≥, esta estrechamente relacionado con la conversión de una desigualdad en igualdad en el método simplex.

El signo de igualdad (=) es el único que no causa ningún tipo

de cambios en la restricción. Si se accesa la restricción como igualdad la restricción quedará tal y como fué accesada. La base, si aún no ha sido creada comienza con el primer elemento asignando los siguientes valores:

renglón = número de restricción
columna = 999 + número de restricción
columna1 = 999 + número de restricción

Pero si la base ya se creó, únicamente se suma un nuevo elemento con las características anteriores.

Por otro lado, hay que recordar que el método simplex comienza sus iteraciones con las variables auxiliares dentro de la base, por tal motivo se le asignarán a los campos "columna" y "columna1" del nuevo elemento de la base el valor de 999+número de restricción. Cabe aclarar que el rango de columnas de las variables auxiliares comienza desde 1000.

El signo de menor ó igual (≤) causará un cambio en la restricción, ya que, recordando el procedimiento para convertir una desigualdad a igualdad (Cap. 2), cuando la desigualdad es menor ó igual se suma una variable de holgura, de tal forma que a la

restricción accesada se le suma una nueva variable con el coeficiente igual a 1, el renglón igual al número de restricción y el valor de la columna igual a (999 + número de restricción). Esto se debe a que esta variable es auxiliar y se debe de distinguir de las reales. Estas variables auxiliares tendrán como rango de variación en su campo "columna" de 1000 hasta 1999. Este rango está compartido con las variables superfluas generadas por el signo de mayor o igual. En el caso de menor o igual se le sumará un nuevo elemento a la base con las siguientes características:

renglón = número de restricción
columna = 999 + número de restricción
columna1 = 999 + número de restricción

El signo de mayor ó igual (≥) causará también un cambio en la restricción, ya que se resta una variable de holgura y se le suma una variable superflua (Ver Cap. Segundo) a la desigualdad para convertirla en igualdad, de tal forma que a la restricción accesada se le resta una nueva variable con el coeficiente igual a -1, el renglón igual al número de restricción y el valor de la columna igual a (999 + número de restricción) y se le suma otra variable con el coeficiente igual a 1, el renglón igual al número de restricción y el valor de la columna igual al de variable restada.

Aquí se le sumará un nuevo elemento a la base con las siguientes características:

renglón = número de restricción

columna = 999 + número de restricción

columna1 = 999 + número de restricción

Más adelante se realizará un ejemplo para hacer más claros estos cambios.

Hay que aclarar que el rango de las columna para variables reales es de 1 a 999, el rango para las columnas de las variables auxiliares es de 1000 a 1999 y la columna del vector "b" será 2000. Como se puede observar, los rangos son exclusivos, de tal forma que cualquier variable queda identificada dentro de su rango. De igual forma, **a cada restricción le corresponde una variable auxiliar**. Se puede intuir que esta estructura es apta para resolver problemas con hasta 999 incognitas y con hasta 999 restricciones.

4.1.4 Función Objetivo

Ya teniendo almacenadas las restricciones y generada la base, hay que accesar la función objetivo.

Primero se especificará el tipo de problema (Maximización ó Minimización). Se recomienda que si todas las restricciones del

problema son menor ó igual (≤) se utilice la maximización.

La estructura para la función objetivo será la misma que se uso para las restricciones. Se accesarán los coeficientes y sus respectivas columnas. Las caracerísticas especiales que tendrá serán:

- El valor del elemento "b" será igual a cero con su columna= 2000.
- El valor que tomará el campo "renglón" de los coeficientes será siempre cero (0), ya que los renglones de las restricciones comienzan en uno.

En el caso de Minimización, se generará un nuevo renglón con valor -1 donde se contendrán los valores de la función W a minimizar en la primera fase del método Dos Fases, que se tratará en el siguiente tema.

De esta forma ya se tiene completo y listo para ser procesado el planteamiento de programación lineal con matrices dispersas como lista dinámica.

A continuación se propone un ejemplo de aplicación para seguir el proceso de lectura y registro (visto teóricamente) de un problema de programación lineal.

Se presentará un planteamiento y su forma de representación interna utilizando los conceptos anteriores.

Problema :

$$
\begin{aligned}
\text{Min } Z = {} & 5x1 + 7x2 + x3 + 3x4 + 5x5 \\
& \text{s. a} \\
& 7x1 - 4x3 \geq 34 \\
& x1 + 7x4 \leq 23 \\
& 2x2 - 9x5 = 18 \\
& xi \geq 0
\end{aligned}
$$

Representación Interna :

7[1,1]		-4[1,3]		-[1,1000]	= 34
[2,1]		+7[2,4]		+[2,1001]	= 23
	2[3,2]		-9[3,5]		= 18
5[0,1]	+7[0,2]	+1[0,3]	+3[0,4]	+5[0,5]	= 0

donde por ejemplo el elemento 7[2,4] significa:

coeficiente = 7

restricción = 2

variable = 4

Como existen solamente 3 restricciones, solo habrá tres elementos en la base, los cuales serán

(1, 1, 1000) —» (2, 2, 1001) —» (3, 3, 1002)

lo que implica que (1000, 1001, 1002) corresponden a la base inicial que estará sujeta a modificaciones.

4.2 Simplex : Maximizacion, Minimizacion e Implementacion .

Ya definada la estructura interna, se puede realizar un análisis metódico del Simplex con respecto a la estructura, aplicando todas sus caracteristicas y casos. Se implementará, de igual forma, el uso del suministro de soluciones factibles visto en el capitulo anterior. La aplicación de la teoria vista en el capitulo segundo será fundamental para este tema. Se comenzará con el análisis de problemas del tipo Maximizar, sus formas de inicialización y espectos importantes. Después se analizará el problema del tipo Minimización con las mismas caracteristicas, y habiendo obtenido los elementos y caracteristicas del método, se realizará una implementación total de todos esos aspectos que complementan el método simplex.

4.2.1 Problemas de Maximización

El método simplex, cuando se trata de maximización, suele ser más sencillo de analizar dado que todas las restricciónes son de tipo

menor o igual. En el caso de que se tratará de **minimizar** un planteamiento con estas caracteristicas, el punto minimo corresponderia al origen porque todas las variables reales tendrian el minimo valor factible dentro del conjunto convexo en dicho punto.

La forma de tratar un problema de maximización con la caracteristica especificada (todas las restricciones ≤) será la siguiente:

Algoritmo:

Paso 0) Se invierten los signos de los elementos de la función objetivo negativos.

Paso 1) Por medio del procedimiento "MasNegativo(lista, 0)" (linea 2 siguiente lista) se localiza el elemento más negativo en el renglón cero, es decir, en la función objetivo (recordar que la función objetivo está almacenada en el renglón cero) (Ver procedimiento "mas_negativo" en el Anexo B).

Paso 2) Si no hay más negativos terminar (linea 3 siguiente listado). Si existen más negativos, se llama al procedimiento " NuevoPivote(lista, base,columna)" (linea 4) donde se realizará la busqueda de la variable a salir de la base. El tercer parametro indica la columna en donde será realizada la búsqueda del pivote. En caso de existir un empate, el procedimiento contiene las reglas lexicográficas incluidas, de tal forma que se realiza la elección habiendo o no empate

(para más información referirse al Anexo B). Este procedimiento detecta cuando la solución es "no acotada" (cuando todos los elementos de la columna a(r) de N son menores de cero. Ver Cap.2), de tal forma que cuando un contador interno de elementos menores que cero es igual al número de restricciones, el método se detiene anunciando la solución no acotada (lineas 5,6,7 y 8).

Paso 3) La base se actualiza de acuerdo a la fila *k* elegida para salir y a la columna r elegida para entrar (linea 9). En la base, el elemento con el campo "renglón" que sea igual al renglón escogido, su valor en "columna1" será cambiado al valor de la columna elegida. Por otro lado, se normaliza el renglón dividiendolo entre el valor correspondiente al cruce de la columna *r* y la fila *k* (el cruce de estos dos valores corresponderá al nuevo pivote). Este paso se realiza en el procedimiento "HazUnoPrincipal(lista, renglón, columna)" incluido en el Anexo B (linea 10).

Paso 4) Ya teniendo el renglón normalizado, se procede a hacer la columna correspondiente al pivote igual a cero, con excepción del elemento correspondiente a la fila *k*. Este paso es realizado por el procedimiento "HazUnoColumna(Lista, Renglon, Renglon_Arreglar, Columna)" (Lineas 12-16. Ver Anexo B). Hecho esto, regresar al **paso 1.**

Este proceso se realiza hasta que no existan elementos negativos en el renglón cero (en la función objetivo). A continuación se

presenta el listado que realiza los cuatro pasos anteriores.

Listado Método Simplex (Maximización Sencilla) :

```
1     while(1) {
2          masnegativo(lista,0);
3          if (masnegativos == 'n') break;
4          nuevopivote(lista,base,columna);
5          if(negativos == restricciones) {
6               printf(" Sol. No  Acotada !!! ");
7               break;
8          }
9          cambiabase(base,renglon,columna);
10         hazunoprincipal(lista,renglon,columna);
11         renglon_arreglar=0;
12         while(renglon_arreglar <= restricciones) {
13              if(renglon_arreglar == nuevopivotes ) i++;
14              hazunocolumna(lista,renglon,renglon_arreglar,columna);
15              renglon_arreglar++;
16         }
17    }
```

Como se puede observar, este procedimiento se detendrá cuando la variable "MasNegativos" sea igual a 'n' (linea 3).

Acontinuación se presenta el caso de inicializar el método simplex, cuando el problema es de maximización.

INICIALIZACION: Caso Maximización (Gauss-Jordan) :

Ahora se supone que se desea inicializar este método con una solución factible generada por el método Gauss-Jordan (método directo). El problema no se hace mayor, ya que ambos métodos mantienen una estrecha relación porque sus procesos se basan en la eliminación gaussiana, así que prácticamente se puede hablar de un mismo método. El procedimiento a realizar antes del método simplex será el siguiente:

Algoritmo:

Paso a) Primero se realiza la siguiente operación:

Dimensión = min { # variables, # restricciones },

el valor que contenga "Dimensión" será la dimensión de la matriz cuadrada que resolverá el método de Gauss-Jordan. Así, llamando al procedimiento "Gauss(lista, Dimensión)" (revisado en el capítulo anterior), se tendrá como resultado una sub-matriz de A de tipo unitaria y un vector "b" de soluciones. En este punto se pueden dar tres casos:

1)Dimensión = # variables.- En este caso, la parte resuelta de A es completa de izquiera a derecha. El

método de inicialización no toca las restricciones mayores a "dimensión". A la base se le cambiarían los primeros "dimensión" elementos por los números correspondientes a las variables resueltas (ahora básicas). Por ejemplo:

$$\left|\begin{array}{cc|c} 1 & 2 & 2 \\ 2 & 1 & 2 \\ 3 & 4 & 3 \end{array}\right| \qquad \left|\begin{array}{cc|c} 1 & & 0.67 \\ & 1 & 0.67 \\ 3 & 4 & 2 \end{array}\right|$$

base anterior : (1000, 1001, 1002)
base actual : (1, 2, 1002)

II) Dimensión = # restricciones.- En este caso, la parte resuelta de A es completa de arriba hacia abajo, pero con la característica de que, las variables de (dimensión + 1) en adelante, no son resueltas dentro del sistema, únicamente son alteradas por los cálculos hechos a su respectivo renglón. La base se cambia completa ya que la dimensión es igual al número de restricciones. Por ejemplo:

$$\left|\begin{array}{ccc|c} 1 & 2 & 3 & 2 \\ 2 & 1 & 4 & 2 \end{array}\right| \qquad \left|\begin{array}{ccc|c} 1 & & 1.67 & 0.67 \\ & 1 & 0.67 & 0.67 \end{array}\right|$$

base anterior : (1000, 1001)
base actual : (1, 2)

III) Restricciones = # variables.- En este caso la submatriz

es la propia matriz A, de tal forma que aquí se solucionan todas las variables y se toman en cuenta todas las restricciones. La base se comporta igual que en el caso anterior.

En el método simplex con método directo no importa cual sea el caso en que se suministre la lista ya resuelta. Hay que observar que el planteamiento del problema no es cambiado por el proceso de inicialización.

Es importante hacer notar que si algún elemento en el vector "b" de soluciones iniciales es negativo, implicaría que la solución no es factible. En este caso, hay que intercambiar la variable correspondiente al renglón del valor "b" donde se encuentre el negativo por alguna variable que no pertenezca a la base. En el caso de ya no existir mas variables, se puede decir que la submatriz no tiene solución, y lo conveniente es este caso sería no inicializar el método simplex.

El proceso de cambio de solución no-factible (en caso de existir) lo realiza el siguiente listado:

```
1     factible='n';j=1;
2     while(factible != 's') {
3           factible='s';
```

```
4          cc=lista;
5          while(cc )= NULL) {
6               if(cc->columna == 2000 && cc->valor < 0) {
7                    factible='n'; break;
8               }
9               cc=cc->siguiente;
10         }
11         if( (dimension + j) > m) {
12              printf(" No hay más columnas para pivotar ");
13              break;
14         }
15         if(factible == 'n') {
16              renglon=cc->renglon;
17              hazunoprincipal(lista,renglon,dimension+j);
18              ren_mod=0;
19              while(ren_mod <= restricciones) {
20                   if(ren_mod == cc->renglon) ren_mod++;
21                   if(ren_mod > restricciones) break;
22                   hazunocolumna(lista,renglon,ren_mod,dimension+j);
23                   ren_mod++;
24              }
25         }
26         cambiabase(base,renglon,dimension+j++);
27    }
```

El contador "j" contiene el número de elementos encontrados en el vector "b" que sean negativos (lineas 1 y 26). Si encuentra alguno, registra el número de renglón (linea 16) y mueve el pivote hacia la derecha hasta encontrar un no-cero, normaliza la fila con el nuevo pivote (linea 17) y luego vuelve unitaria la columna correspondiente (lineas 19-24). Este procedimiento (movimiento de

pivote) es mucho más sencillo que el intercambio de columnas. La base se va actualizando con cada cambio que suceda (linea 26). El proceso terminará cuando el vector "b" ya sea factible (lineas 5-10) ó cuando ya no encuentra variable para pivotear (lineas 11 ,12 y 13). Si esto sucede, es recomendable usar el método simplex sin inicialización, es decir, no inicializar el método.

Paso b) Este paso consiste en incorporar la función objetivo al planteamiento resultante de la inicialización y hacer que los vectores columna correspondientes a las variables incluidas en la base sean unitarios. Este proceso se realiza mediante la lectura progresiva de la base. Para lograr esto, se ocupa el mismo procedimiento "HazUnoColumna" ocupado en el método de Gauss-Jordan.

El proceso es el siguiente: lee uno por uno los elementos de la base y se ocupan sus campos "renglon" y "columna1" para determinar la posición del pivote, y luego se llama al procedimiento "HazUnoColumna(lista, renglon, columna1)" para volver las columnas correspondientes a unitarias.

Paso c) Realizar Método Simplex con la lista actual.

De esta forma termina el método Simplex inicializado con el método de Gauss-Jordan.

INICIALIZACION: Caso Maximización (Kaczmarz) :

Si se buscara inicializar el método simplex con el método de Kaczmarz (metodo iterativo), el problema se puede complicar debido a que este y el método simplex presentan heterogeneidad de procesos. El procedimiento que se podría seguir sería que cada uno se desarrollara independientemente, y luego por medio de procedimientos, unificar los resultados de uno, y los procedimientos de otro.

El proceso del método de Kaczmarz es iterativo, por lo que no afecta el contenido de la lista. Utilizando esta ventaja, se desarrolla este método en forma independiente del simplex, y ya obtenidos los resultados, se conocerá el vector solución "b" y además, se tendrá, como ya se dijo, la lista original y la existencia de una matriz unitaria imaginaria generada por las variables resueltas por este método. Con estos tres tipos de datos, se procede a modificar la lista de acuerdo a las necesidades del simplex y a los datos con que se cuenta.

El procedimiento inicial a seguir antes del simplex será el siguiente:

Paso a) Primero se realiza la siguiente operación:

Dimensión = min { # variables , # restricciones },

el valor que contenga "Dimensión" será el número de variables que resolverá el método de Kaczmarz. Así, llamando al procedimiento "Kaczmarz(lista, Dimensión, Dimensión+1)" (revisado en el capítulo anterior), se tendrá como resultado un vector "b" de soluciones de longitud "dimensión". En este punto también se dan tres casos:

I) Dimensión = # variables.- Este caso se encontrará la solución correspondiente a las "dimensión" variables con respecto a los "dimensión" primeros renglones. El método simplex se encargará de analizar el total de restricciones. Lo importante es que el sistema no se altere. La base tendrá el mismo cambio que en el caso I) con inicialización Gauss-Jordan.

II) Dimensión = # restricciones.- En este caso, se encontrará la solución de las primeras " dimensión" variables, que corresponden a la base completa. La restantes variables no tendrán solución (no pertenecerán a la base). Este caso puede ser benéfico o perjudicial, ya que, si las variables que optimizan la función objetivo se encuentran en la base, no existirá problema posterior. Pero si hay una variable

que no estando en la base, lo debe de estar para optimizar la función, comenzarán los problemas. Para ver cláramente cual sería el problema se analiza el siguiente problema:

$$\left|\begin{array}{ccc|c} 1 & 2 & 3 & 2 \\ 2 & 1 & 4 & 2 \end{array}\right|$$

Se observa que dimensión = 2 así que resolviendo por Kaczmarz se obtiene que:

$$x1 = 0.67 \quad y \quad x2 = 0.67$$

Esta solución es factible. Pero, que relación existe entre este el resultado y la matriz anterior ?, únicamente que es solución de dos de sus variables. Pero, que pasa con la tercera variable si se quisiera representar el nuevo sistema de la siguiente forma ? :

$$\left|\begin{array}{ccc|c} 1 & & ? & 0.67 \\ & 1 & ? & 0.67 \end{array}\right|$$

No hay forma de determinar (por este método) que valores pertenecen a los coeficientes que se

desconocen, así que el problema estaría incompleto. Hay que recordar que el método de Gauss-Jordan realiza esta operación paso a paso y al final, contiene todos sus coeficientes modificados correspondientes al problema.

Por este motivo es recomendable no usar este método cuando se encuentre en este caso.

III) Restricciones = variables .- En este caso lasolución a las "n" variables es encontrada. La base es completa porque contiene "m" elementos correspondientes a sus restricciones y contendrá a todas las variables reales.

Paso b) Este paso consiste en preparar la lista de acuerdo a los datos que se obtuvieron del método anterior (siempre y cuando no se haya estado en el caso II). La preparación comienza con el borrado de los elementos de los renglones 1 al "dimensión", es decir, que se encuentren en el rango [1..dimensión] y anexando a cada uno de ellos su correspondiente pivote de acuerdo a los elementos de la base. Los renglones que no se encuentren en el rango establecido por dimensión se dejan intactos de tal forma que el problema no se altere. Los componentes del vector "b" son sustituidos por los resultados alcanzados por el método. Los valores de "b" que no pertenezcan al rango, también se dejan intactos, al igual que toda la función objetivo.

Paso c) Realizar Método Simplex con la lista actual. De esta forma termina el método Simplex inicializado con el método de Kaczmarz.

4.2.2 Problemas de Minimización

Los problema de minimización suelen ser más complicados por que las desigualdades incluidas en el planteamiento continen signos tanto ≤ como ≥. Pero por otro lado, resulta ser fácil de explicar, ya que los procedimientos y operaciones son los mismos que se utilizan en el problema de maximización. La problemática radica en que hay que resolver previamente la minimización de las variables superfluas formadas por las restricciones de ≥, para luego aplicar el método simplex.

Para resolver este tipo de problemas es necesario tenerlo planteado en forma de igualdades, recordando que para las ecuaciones que contienen signo de ≥ se le resta una variable de holgura y se le suma una variable superflua. Estas dos variables también debe de ser mayores ó iguales a cero. El método utilizado para resolver este tipo de problemas se le conoce como método de las Dos Fases. Este realiza en su primera fase la minimización de una función auxiliar W, que estará formada por los coeficientes de aquellas restricciones que contienen variables superfluas. Esta fase I se resolverá con el procedimiento del simplex, es decir, que se realizarán operaciones hasta que ya no existan negativos en la función W.

Por ejemplo:

$$\begin{aligned} \text{Min } Z &= 4x + 3y + 7z \\ \text{s. a} & \\ x \quad &+ 2z \quad \geq 23 \\ y &- z \quad \leq 12 \\ x \ , \ y \ , \ z &\quad \geq 0 \end{aligned}$$

la función W estaría compuesta por:

$$W = x + 2z \qquad \text{ó} \qquad W - x - 2z = 0$$

la tabla de la primera fase quedaría como sigue:

		W	x	y	z	w1	y1	
Funcion W	—>	1	-1		-2	1		-23
Var Super—»	w1		1		2	-1		23
	y1			1	-1		1	12

De modo que, realizando las mismas operaciones del simplex, si la función W > 0 implica que no hay solución, y si W = 0 se pasa a la fase dos (Ver Capítulo Segundo), que se comenzaría con la misma tabla pero con la función objetivo en lugar de la función W.

Los pasos a seguir son los siguientes:

Primera Fase (Para visualizar claramente los pasos, referirse al

listado correspondiente más adelante) :

Paso 0) Se crea el renglón -1 correspondiente a la función W y se le agregan los valores de los coeficientes correspondientes a la restricción ≥. Se dispone la función objetivo para que no la altere el proceso de primera fase (línea 9).

Paso 1) Se realiza el mismo procedimiento que en el método simplex. Si al ya no existir elementos negativos en la función W, todos ellos son ceros, entonces seguir con el paso 2. Si no son ceros ó son positivos implica que el problema no tiene solución (línea 18).

Paso 2) Se elimina la restricción con renglón igual a -1. Y termina la primera fase (línea 19).

Segunda Fase :

Paso 3) Se prepara la función objetivo para ser procesada (línea 24).

Paso 4) En este paso las columnas correspondientes a las variables incluidas en la base deberán de hacerse unitarias, esto es debido a que la función objetivo no sufrió cambios y esta se tiene que actualizar a las demandas de la base actual (líneas 21-32).

Paso 5) Ya actualizada la función objetivo se procede únicamente a determinar la existencia de coeficientes negativos en esta (líneas 34 y 35). Este corresponde al primer paso del método simplex. En otras palabras, cuando se llega a este punto únicamente se realiza el método simplex.

Con esto el método de dos fases termina.

Acontinuación se presenta el listado que realiza los pasos anteriores:

Listado Método Simplex (Minimización Sencilla) :

```
                    /*   P R I M E R A   F A S E   */

1     while (1) {                                          ( PASO 1 )
2          masnegativo(lista,-1);
3          if (masnegativos == 'n') break;
4          nuevopivote(lista,base,columna);
5          cambiabase(base,renglon,columna);
6          hazunoprincipal(lista,renglon,columna);
7          ren_mod=-1;
8          while(ren_mod <= restricciones) {
9               if(ren_mod == 0) ren_mod++;
10              if(ren_mod == nuevopivotes ) ren_mod++;
11              if(ren_mod > restricciones) break;
12              hazunocolumna(lista,renglon,ren_mod,columna);
13              ren_mod++;
14         }
15    }
16    cc=lista;                                            ( PASO 2 )
17    while(cc->renglon != -1 && cc->columna != 2000)
```

```
                    cc=cc->siguiente;
18  if(cc->valor != 0) printf(" Problema Sin Solución !!!");
19  borrarestriccion(lista,-1);


                /*   S E G U N D A   F A S E   */

20  bb=base;
21  do {                                              ( PASOS 3 y 4 )
22       renglon1=bb->renglon;
23       columna1=bb->columna1;
24       for(ren_mod=0;;ren_mod++) {
25            if( ren_mod == renglon1 ) ren_mod++;
26            if( ren_mod > restricciones ) break;
27            hazunocolumna(lista,renglon1,ren_mod,columna1);
28       }
29       bb=bb->siguiente;
30       renglon1=bb->renglon;
31       if(bb->renglon > restricciones ) break;
32  } while( bb != NULL );
33  masnegativos='s';                                 ( PASO 5 )
34  while ( masnegativos != 'n' ) {
35       masnegativo(lista,0);
36       if (masnegativos == 'n') {
37            printf(" Termina la Segunda Fase");
38            break;
39       }
40       nuevopivote(lista,base,columna);
41       cambiabase(base,renglon,columna);
42       hazunoprincipal(lista,renglon,columna);
43       ren_mod=0;
44       while(1) {
45            if(ren_mod == nuevopivotes ) ren_mod++;
46            if(ren_mod > restricciones) break;
47            hazunocolumna(lista,renglon,ren_mod,columna);
```

```
48              ren_mod++;
49          }
50      }
```

La variable "base" contiene la dirección del primer elemento de la base, "bb" es un apuntador a la base. La variable "lista" contiene la dirección del primer elemento de la lista, "cc" es un apuntador a la lista. La función W internamente se almacena en la fila -1. En general el proceso que lleva este listado es una composición de rutinas que utilizó el método Simplex y el método de Gauss-Jordán que ya fueron explicados.

La inicialización con el método de Gauss-Jordan para problemas de minimización tiene la misma estructura que utilizada en el proceso de maximización. De igual modo sucede con la inicialización utilizando el método de Kaczmarz. Los casos en los que se puede incurrir en maximización son similares e igualmente válidos para problemas de minimización.

Los mismos problemas sobre las diferencias de procedimientos entre el método Simplex y el método de Kaczmarz son asumidos. Es recomendable que en la práctica se tengan bien definidos los casos en que se presentan los sistemas, de tal forma que no se trabaje inútilmente, ni se pierda tiempo en el análisis del sistema.

La implementación realizada anteriormente se puede ya considerar completa, ya que los casos que pueden existir dentro del método simplex han sido abordados.

Conclusiones Preliminares: Ya se demostró que la estructura dinámica de listas es muy flexible para cumplir con las características especiales que debe tener un algoritmo. El ahorro de memoria fué llevado a cabo registrando únicamente los elementos diferentes de cero, y en consecuencia únicamente se realizarán operaciones en dichos números. El ahorro en este caso fué doble: De memoria y de Proceso. Así, el único aspecto que falta por abordar es la conveniencia de la inicialización externa. Es decir, el objetivo es demostrar que tan conveniente es la inicialización antes del uso del método simplex, si es que realmente ayuda al proceso estandar del método, determinar si es posible realizar menos iteraciones para llegar a la esquina óptima de la región de factibilidad.

Para realizar esta prueba se analizarán y compararán los procesos realizados por el método estandar del simplex y el método simplex con inicialización externa, se analizará también al resultado al que se llege al igual que el número de iteraciones utilizados por cada uno de los métodos.

4.3 Aplicaciones.

Para llevar a la práctica toda la teoria de la investigación, se analizará un problema real desde su planteamiento hasta su solución,

utilizando todos los conceptos expuestos en el presente trabajo.

Planteamiento : En la actualidad existen un gran número de empresas con grandes dimensiones que, dentro de sí mismas, manejan una gran cantidad de actividades, de las cuales pocas están relacionadas. La única característica que se tiene es que todas ellas trabajan para un fin en común: obtención de utilidades, ya sean económicas, políticas, psíquicas, biológicas, etc. Tal es el caso de **Petróleos Mexicanos** que cuenta con una infinidad de actividades dentro de toda la República Mexicana. Muchas de sus actividades son realizadas en las plantas refinadoras de petróleo crudo (Salamanca, Salina Cruz, Cadereyta, Tula y la recientemente desaparecida Azcapotzalco). En estas refinerías se lleva a cabo una actividad muy importante (específicamente la planta refinera de Salina Cruz). Esta actividad es el tratamiento del crudo y la obtención de gasolinas de todo tipo (Amarga, Tratada, Dulce, Catalítica, Magna Sin, Nova y Reformada). En esta planta se cuenta con la característica de que no existen resultados intermedios; sólo se tienen entradas y se obtendrá una salida. Específicamente, las entradas a la planta de refinera de Salina Cruz son: el petróleo crudo y el LPG (Combustóleo), y las salidas son el número de barriles necesarios para cubrir óptimamente, tanto los requerimientos de la empresa como los requerimientos económicos de maximización de utilidades. Este objetivo se logra encontrando el punto donde la diferencia entre la compra y la venta sea mayor a favor de la venta.

Los precios de compra son los siguientes:

LPG (Combustoleo)	25 Dlls / barril
Crudo	14 Dlls / barril

Los precios de Venta son los siguientes:

Gasolina Amarga	18 Dlls / barril
Gasolina Amarga Tratada	20 Dlls / barril
Gasolina Amarga Dulce	20 Dlls / barril
Gasolina Amarga Nova	23 Dlls / barril
Gasolina Amarga Magna Sin	26 Dlls / barril
Gasolina Amarga Reformada	28 Dlls / barril
Gasolina Amarga Catalitica	28 Dlls / barril

Especificaciones :

Dentro del proceso en la planta se tienen que cumplir con un grán número de restricciones, las cuales se ven conformadas por los requerimientos, las capacidades, la disponibilidad, etc. En este caso se cuenta con seis tipos de restricciones:

1) De Balances: Este tipo de restriciones son las que delimitan la distribución de un ducto hacia otros adyacentes. Es decir, el flujo que corra de un ducto a otros será siempre el mismo.

2) De Rendimientos: Este tipo de restricciones delimitan el

grado de pureza en que se debe persistir en un flujo al pasar por una torre de destilación.

3) De Recursos: Este tipo de restricciónes delimitan las unidades de flujo permitidas en un ducto antes de ser procesada en alguna torre, ya sea de destilación, hidrodesulfuradora, combinadoras de azufre ó reformadora.

4) De Indices de Octano : Este tipo de restricciones mantienen el índice de octano en las gasolinas Nova y Magna Sin sobre un nivel preesteblecido.

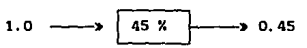
5) De PVR (Presión de Vapor): Este tipo de restricciones mantienen la presión de vapor en los ductos de gasolinas Magna Sin y Nova sobre niveles ya preestablecidos.

6) De TEL (Plomo): Este tipo de restricciones son las que indican en que cantidad se le combinará plomo a los ductos que conformarán la gasolina Nova para que no se rebase un límite ya preestablecido.

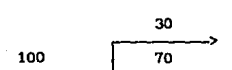
Búsqueda del Planteamiento Matemático :

Ya enunciados los tipos de restriciones y las características de compra-venta, será mas sencillo visualizar el problema desde un punto de vista de Gráfico, es decir, con el diagrama de la planta de refinación (Ver gráfica A más adelante).

Como se puede observar la entrada principal que tiene la planta es del suministro de Crudo la cual empieza a ser refinado. Al final del proceso se obtendrá la gasolina Amarga, la Tratada, la Dulce, la Reformada, la Catalizada, la Magna Sin y la Nova. Cada vez que un ducto entra a una torre de refinación el grado de pureza del fluido decrece en un porcentaje especifico en cada torre. Estos cambios son definidos en las restricciónes de rendimiento.



Se observa que de los ductos principales se desprenden otros, es decir, que los ductos principales son adyacentes a sus divisiones. Esta actividad esta representada en las restricciones de balance.



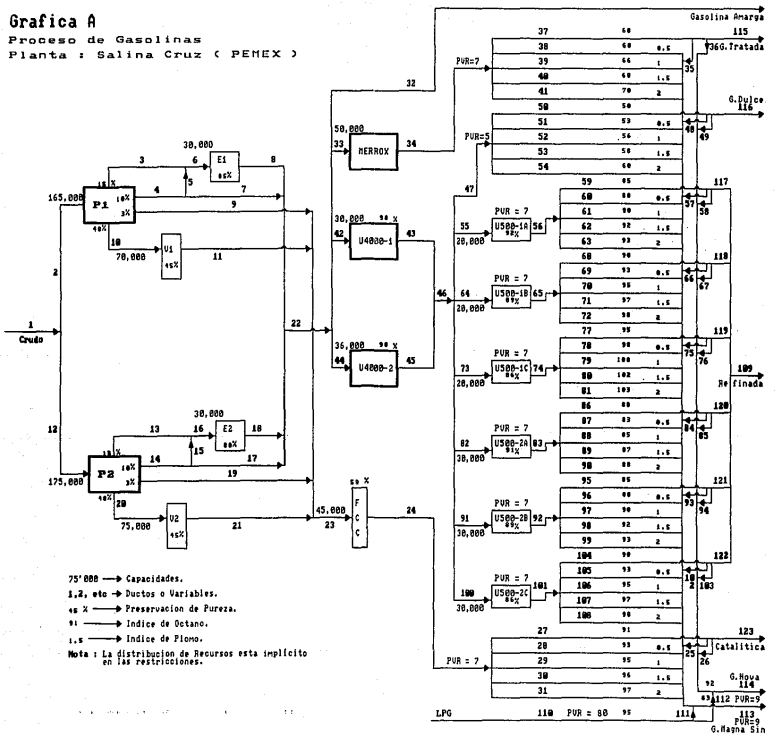
Ahora bien, antes de ser procesado un fluido en una torre debe cerciorarse que sus unidades sean menores o iguales a la capacidad de la torre. Estas caracteristicas estan representadas en las restricciones de recursos.

En la gráfica se puede observar que las gasolinas que más son

Grafica A

Proceso de Gasolinas

Planta : Salina Cruz (PEMEX)



75' 000 ⟶ Capacidades.

1,2, etc ⟶ Ductos o Variables.

45 % ⟶ Preservacion de Pureza.

91 ⟶ Indice de Octano.

1.5 ⟶ Indice de Plomo.

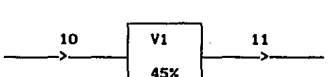
Nota : La distribucion de Recursos esta implicito en las restricciones.

tratadas son la Nova y la Magna Sin, estas dos contienen una cierta cantidad de octano y además necesitan determinada presión en los ductos. De igual forma la gasolina Nova requiere de una cierta cantidad de plomo. Los índices de presión de vapor están especificados en la gráfica, al igual que los índices de plomo para la gasolina Nova. Estas características están especificadas en las restricciones de índices de octano, de presión de vapor y de TEL (índice de plomo).

Este tipo de problema suele ser muy complicado de resolver con métodos propios de Teoría de Gráficas, en este caso se busca realizar un planteamiento matemático, de tal forma que el problema no se complique.

Para lograr el planteamiento, se numeraron los ductos de la gráfica A, de tal forma que los que son adyacentes contienen números consecutivos (dichos números serán las variables que los identificarán).

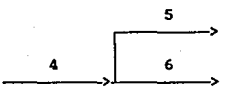
Por ejemplo, para una restricción de rendimiento:



su representación matemática sería:

$$X_{11} = 0.45\ X_{10}$$

para una restricción de balance:



su representación matemática sería como sigue:

$$X_4 = X_5 + X_6$$

para restricciones de recursos:

$$X_2 \leq 165{,}000$$

para restricciones de índices de octano, aplicando restricciones de balance, se observa (en la Gráf.A) que para la gasolina Nova es de 92 y que esta corresponde a la variable 114. Los ductos o variables adyacentes a 114, con su respectivo grado de octanaje, son 26(91), 36(60), 49(50), 58(85), 67(90), 76(95), 85(80), 94(85), 103(90) y 112(95), de tal forma que la representación matemática que describe esta actividad queda como sigue:

$$92X_{114} \leq 91X_{26} + 60X_{36} + 50X_{49} + 85X_{58} + 90X_{67} + 95X_{76} + 80X_{85} + 85X_{94} + 90X_{103} + 95X_{112}$$

Las restricciones de presión de vapor y los de índice de plomo,

se conforman de igual modo que los de índice de octano.

De esta forma el planteamiento de las restricciones del problema queda cubierto, únicamente habrá que delimitar la función objetivo.

Se había mencionado que la función objetivo del problema consistía en maximizar las utilidades de toda la planta. Esta utilidad consiste en obtener el máximo beneficio de las ventas con respecto a las compras. Este beneficio esta dado por el número de barriles de Crudo y de LPG que se adquieran (sus precios unitarios fueron dados con anterioridad) y el número máximo de barriles de gasolina (de cualquier tipo) que se obtengan del proceso. Aquí se esta suponiendo que todos los barriles de gasolina que se producen se venden, de tal forma que, dados los precios de venta de barriles de gasolina de todos sus tipos, se busca encontrar un nivel de producción de gasolinas de tal forma que la utilidad en dolares se maximice.

El planteamiento matemático de la función objetivo será el siguiente:

Max. Utilidad = Precios de Ventas - Precios de Costos

→ **Max. Utilidad** = Distribución de Gasolinas - Precios de Costos

Recordando los precios de compra y observando su respectivo número de ducto ó de variable, se forma el planteamiento matemático de compra:

Crudo (ducto = 1 precio 14 Dlls / barril)
LPG (ducto = 110 precio 25 Dlls / barril)

$$\Rightarrow \quad \text{Compra} = 14X_1 + 25X_{110}$$

Recordando los precios de ventas de gasolina y observando su respectivo ducto ó variable, se conforma el planteamiento matemático de venta:

G. Amarga (ducto = 32 precio 18 Dlls / barril)
G. Tratada (ducto = 115 precio 20 Dlls / barril)
G. Dulce (ducto = 116 precio 20 Dlls / barril)
G. Nova (ducto = 113 precio 23 Dlls / barril)
G. Magna Sin (ducto = 114 precio 26 Dlls / barril)
G. Reformada (ducto = 109 precio 28 Dlls / barril)
G. Catalitica (ducto = 123 precio 28 Dlls / barril)

$$\Rightarrow \text{Venta} = 18X_{32} + 20X_{115} + 20X_{116} + 23X_{113} + 26X_{114} + 28X_{109} + 28X_{123}$$

Ya habiendo obtenido las representaciones para compra y para venta, se conformará completamente la función objetivo.

La función objetivo :

Max. Utilidad = Precios de Ventas - Precios de Costos

queda de la siguiente forma:

$$\text{Max. Utilidad} = 18X_{32} + 20X_{115} + 20X_{116} + 23X_{113} + 26X_{114} + 28X_{109} + 28X_{123} - [\ 14X_1 + 25X_{110}\ ]$$

De esta forma ya está definido el problema en términos de un planteamiento matemático, de modo que el siguiente paso será utilizar la Investigación de Operaciones, que mediante la Programación Lineal (Método Simplex), se podrá dar solución al problema.

Planteamiento del Problema Matemático :

El planteamiento del problema en forma matemática el cual se analizará por el método simplex se presenta acontinuación:

$$\textbf{Max. Utilidad} = 18X_{32} + 20X_{115} + 20X_{116} + 23X_{113} + 26X_{114} + 28X_{109} + 28X_{123} - 14X_1 - 25X_{110}$$

s. a

De Balance :

1) $X_1 = X_2 + X_{12}$
2) $X_4 = X_5 + X_7$
3) $X_6 = X_3 + X_5$
4) $X_{14} = X_{15} + X_{17}$

5) $X_{16} = X_{13} + X_{15}$

6) $X_{22} = X_7 + X_8 + X_{17} + X_{18}$

7) $X_{23} = X_9 + X_{11} + X_{19} + X_{21}$

8) $X_{22} = X_{32} + X_{33} + X_{42} + X_{44}$

9) $X_{46} = X_{43} + X_{45}$

10) $X_{24} = X_{27} + X_{28} + X_{29} + X_{30} + X_{31}$

11) $X_{46} = X_{47} + X_{55} + X_{64} + X_{73} + X_{82} + X_{91} + X_{100}$

12) $X_{34} = X_{37} + X_{38} + X_{39} + X_{40} + X_{41}$

13) $X_{47} = X_{50} + X_{51} + X_{52} + X_{53} + X_{54}$

14) $X_{56} = X_{59} + X_{60} + X_{61} + X_{62} + X_{63}$

15) $X_{65} = X_{68} + X_{69} + X_{70} + X_{71} + X_{72}$

16) $X_{74} = X_{77} + X_{78} + X_{79} + X_{80} + X_{81}$

17) $X_{83} = X_{86} + X_{87} + X_{88} + X_{89} + X_{90}$

18) $X_{92} = X_{95} + X_{96} + X_{97} + X_{98} + X_{99}$

19) $X_{101} = X_{104} + X_{105} + X_{106} + X_{107} + X_{108}$

20) $X_{110} = X_{111} + X_{112}$

21) $X_{109} = X_{117} + X_{118} + X_{119} + X_{120} + X_{121} + X_{122}$

22) $X_{114} = X_{36} + X_{49} + X_{58} + X_{67} + X_{76} + X_{85} + X_{94} + X_{103} + X_{26} + X_{112}$

23) $X_{113} = X_{35} + X_{38} + X_{39} + X_{40} + X_{41} + X_{48} + X_{51} + X_{52} + X_{53} + X_{54} + X_{57} + X_{60} + X_{61} + X_{62} + X_{63} + X_{66} + X_{69} + X_{70} + X_{71} + X_{72} + X_{75} + X_{78} + X_{79} + X_{80} + X_{81} + X_{84} + X_{87} + X_{88} + X_{89} + X_{90} + X_{93} + X_{96} + X_{97} + X_{98} + X_{99} + X_{102} + X_{25} + X_{28} + X_{29} + X_{30} + X_{31} + X_{105} + X_{106} + X_{107} + X_{108} + X_{111}$

24) $X_{37} = X_{35} + X_{36} + X_{115}$

25) $X_{50} = X_{48} + X_{49} + X_{116}$

26) $X_{59} = X_{57} + X_{58} + X_{117}$

27) $X_{68} = X_{66} + X_{67} + X_{118}$

28) $X_{77} = X_{75} + X_{76} + X_{119}$

29) $X_{86} = X_{84} + X_{85} + X_{120}$

30) $X_{95} = X_{93} + X_{94} + X_{121}$

31) $X_{104} = X_{102} + X_{103} + X_{122}$

32) $X_{27} = X_{25} + X_{26} + X_{123}$

De rendimiento :

33) $X_3 = 0.15X_2$
34) $X_4 = 0.10X_2$
35) $X_9 = 0.03X_2$
36) $X_{10} = 0.48X_2$
37) $X_8 = 0.80X_6$
38) $X_{11} = 0.45X_{10}$
39) $X_{13} = 0.15X_{12}$
40) $X_{14} = 0.10X_{12}$
41) $X_{19} = 0.03X_{12}$
42) $X_{20} = 0.48X_{12}$
43) $X_{18} = 0.80X_{16}$
44) $X_{21} = 0.45X_{20}$
45) $X_{24} = 0.59X_{23}$
46) $X_{34} = X_{33}$
47) $X_{43} = 0.98X_{42}$
48) $X_{45} = 0.98X_{44}$
49) $X_{56} = 0.92X_{55}$
50) $X_{65} = 0.89X_{64}$
51) $X_{74} = 0.86X_{73}$
52) $X_{83} = 0.91X_{82}$
53) $X_{92} = 0.89X_{91}$
54) $X_{101} = 0.86X_{100}$

De Recursos :

55) $X_2 \leq 165,000$
56) $X_6 \leq 30,000$
57) $X_{10} \leq 70,000$
58) $X_{12} \leq 175,000$
59) $X_{16} \leq 30,000$
60) $X_{20} \leq 75,000$
61) $X_{23} \leq 45,000$

62) $X_{33} \leq 50,000$

63) $X_{42} \leq 30,000$

64) $X_{44} \leq 36,000$

65) $X_{56} + X_{64} + X_{73} \leq 20,000$

66) $X_{82} + X_{91} + X_{100} \leq 30,000$

De Indices de Octano :

67) $92X_{114} \leq 60X_{36} + 50X_{49} + 85X_{58} + 90X_{67} + 95X_{76} + 80X_{85} + 85X_{94} + 90X_{103} + 91X_{26} + 95X_{112}$

68) $83X_{113} \leq 60X_{35} + 68X_{38} + 66X_{39} + 68X_{40} + 70X_{41} + 50X_{48} + 53X_{51} + 56X_{52} + 58X_{53} + 60X_{54} + 85X_{57} + 88X_{60} + 90X_{61} + 92X_{62} + 93X_{63} + 90X_{66} + 93X_{69} + 95X_{70} + 97X_{71} + 98X_{72} + 95X_{75} + 98X_{78} + 100X_{79} + 102X_{80} + 103X_{81} + 80X_{84} + 83X_{87} + 85X_{88} + 87X_{89} + 88X_{90} + 85X_{93} + 88X_{96} + 90X_{97} + 92X_{98} + 93X_{99} + 90X_{102} + 93X_{105} + 95X_{106} + 97X_{107} + 98X_{108} + 91X_{25} + 93X_{28} + 95X_{29} + 96X_{30} + 97X_{31} + 95X_{111}$

De Preción de Vapor :

69) $27X_{114} \leq 18.52X_{36} + 11.18X_{49} + 18.52X_{58} + 18.52X_{67} + 18.52X_{76} + 18.52X_{85} + 18.52X_{94} + 18.52X_{103} + 18.52X_{26} + 715.54X_{112}$

70) $27X_{113} \leq 18.52X_{35} + 18.52X_{38} + 18.52X_{39} + 18.52X_{40} + 18.52X_{41} + 11.18X_{48} + 11.18X_{51} + 11.18X_{52} + 11.18X_{53} + 11.18X_{54} + 18.52X_{57} + 18.52X_{60} + 18.52X_{61} + 18.52X_{62} + 18.52X_{63} + 18.52X_{66} + 18.52X_{69} + 18.52X_{70} + 18.52X_{71} + 18.52X_{72} + 18.52X_{75} + 18.52X_{78} + 18.52X_{79} + 18.52X_{80} + 18.52X_{81} + 18.52X_{84} + 18.52X_{87} + 18.52X_{88} + 18.52X_{89} + 18.52X_{90} + 18.52X_{93} + 18.52X_{96} + 18.52X_{97} + 18.52X_{98} + 18.52X_{99} + 18.52X_{102} + 18.52X_{105} + 18.52X_{106} + 18.52X_{107} + 18.52X_{108} + 18.52X_{25} + 18.52X_{28} + 18.52X_{29} + 18.52X_{30} + 18.52X_{31} + 715.45X_{111}$

De Indice de Plomo :

$$71)\quad X_{113} \leq 0.5X_{38} + X_{39} + 1.5X_{40} + 2X_{41} + 0.5X_{51} + X_{52} + 1.5X_{53} + 2X_{54} + 0.5X_{60} + X_{61} + 1.5X_{62} + 2X_{63} + 0.5X_{69} + X_{70} + 1.5X_{71} + 2X_{72} + 0.5X_{78} + X_{79} + 1.5X_{80} + 2X_{81} + 0.5X_{87} + X_{88} + 1.5X_{89} + 2X_{90} + 0.5X_{96} + X_{97} + 1.5X_{98} + 2X_{99} + 0.5X_{105} + X_{106} + 1.5X_{107} + 2X_{108} + 0.5X_{28} + X_{29} + 1.5X_{30} + 2X_{31}$$

Comentarios :

Como se puede observar existen 71 restricciones, y como el número mayor de ductos númerados es de 123, implica que la matriz que se forma de este planteamiento contendrá 8733 localidades, de modo que si se deseara realizar el examen con memoria estática se desperdiciaría demasiado espacio, por el contario, con memoria dinámica, únicamente se cuenta con 430 elementos diferentes de cero, lo cual corresponde aproximadamente a una veinteava parte del total de elementos en la matriz. En este caso, es muy recomendable el uso de las técnicas para el tratamiento de matrices dispersas con memoria dinámica. La recomendación anterior se reafirma aún más si la necesidad del examen es el procesamiento de una gran cantidad de datos simultaneamente. Es decir, todo el conjunto de datos describen una actividad la cual se optimizará, de tal forma que no interesa tener resultados parciales.

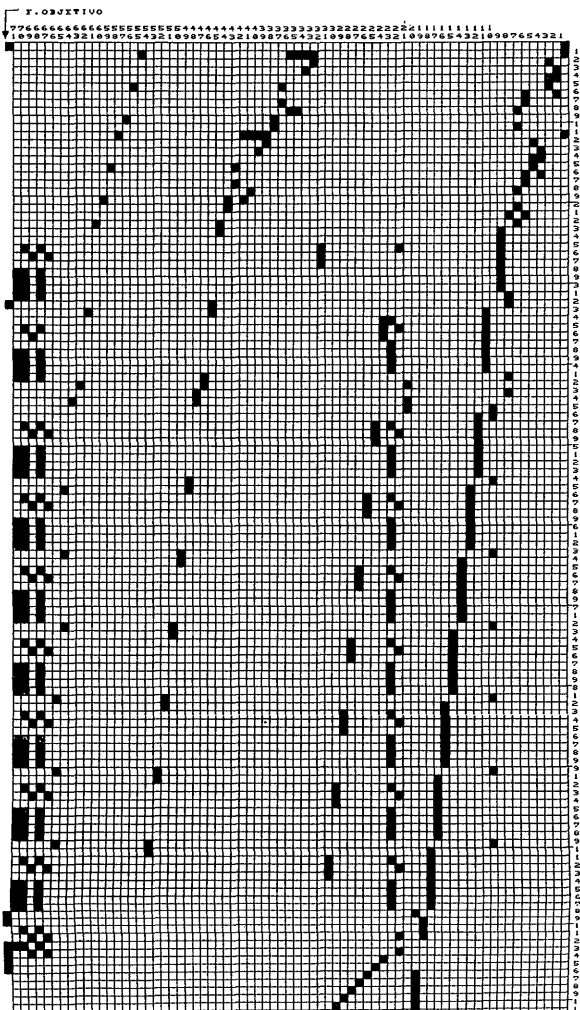
En términos de ahorro de memoria, si se reservaran las 8733

localidades de punto flotante (float) que necesita una matriz densa se requeriría 34932 bytes para registrarla, haciendo mención de que para un float se reserva 4 bytes en memoria.

Ahora bien, recordando la estructura de una lista ligada para registrar los elementos no-ceros, se tenía un elemento "float" (punto flotante), dos elementos "int" (enteros) y un apuntador al siguiente elemento. Para el int se reserva espacio en memoria de 2 byte, para el float se reservan 4 bytes y para el apuntador se reservan 2 bytes en memoria. Así, un elemento de la lista ligada contiene en total 10 bytes. En este problema existen 430 elementos, de tal forma que la memoria que se utilizará en un principio con listas ligadas para representar una matriz dispersa, será de únicamente 4300 bytes, que comparada con 34932 bytes resulta haber un considerable ahorro de memoria (aproximadamente 30000 bytes) en este problema.

El problema que se analiza es considerado de dimensión grande, su representación en matriz numérica se tendría que realizar en varias hojas. A modo de visualizar el problema como matriz densa, se presenta el problema en forma de puntos en la gráfica B, donde cada punto representa la existencia de un elemento no-cero en la matriz. En esta gráfica se puede ver aún más ampliamante el desperdicio de memoria que ocasiona, para un problema como este, el uso de la memoria estática. Incluso se puede asegurar que, para problemas mucho más grandes que este, la memoria destinada para trabajo (aprox. 500 kB dependiendo de

Grafica B



los programas recidentes del DOS) en una computadora con 640 kiloBytes, sería rebasada y el problema no se pudiera solucionar. Si se pensara en matrices dispersas con memoria dinámica y una buena estructura de datos, dicho problema sí podría ser procesado.

Resolución del Problema :

El tratamiento del problema, los algoritmos y las estructuras fueron realizadas en una maquina IBM PC/XT con 640 KB de memoria RAM, microprocesador Intel 8088, sin co-procesador matemático. Estas características corresponden a una computadora personal de pequeñas dimensiónes. Dadas las características de la computadora en que será realizada la prueba, el siguiente paso será dar solución al problema con las herramientas que han sido ajustadas.

Al realizar las operaciones convenientes, los resultados del Método Simplex sin Inicialización Externa se muestran a continuación :

Resultados
Costos
X_1 = 34,000 b
X_{110} = 1,721 b
Ventas
X_{32} = 43,700 b
X_{109} = 38,273.43 b
X_{113} = 114,204.33 b
X_{114} = 80,224.43 b
X_{115} = 27,169 b
X_{116} = 16,782.23 b
X_{123} = 14,223.43 b
Z = 3'045,584.65 Dlls

La tabla anterior indica que la mayor ganancia al suministrar 34,000 barriles de Crudo y 1,721 barriles de LPG se alcanza cuando se distribuye la producción de gasolinas de la siguiente forma:

43,700 barriles de gasolina Amarga
38,273 barriles de gasolina Reformada
114,204 barriles de gasolina Nova
80,224 barriles de gasolina Magna Sin
27,170 barriles de gasolina Tratada
16,782 barriles de gasolina Dulce
14,223 barriles de gasolina Catalítica

obtiendo una utilidad óptima de

Dlls. $ 3' 045, 584. 65

El objetivo del análisis del problema no es precisamente darle solución, sino comparar los resultados con los obtenidos por medio de los métodos de inicialización y observar cual es su conveniencia ó inconveniencia.

Al inicializar el problema con los métodos de Gauss-Jordan y de Kaczmarz, se obtuvieron los resultados de las primeras 71 variables, en las cuales solamente se encuentran X_1 y X_{32} en la función objetivo, de tal forma que el método simplex será indispensable en la solución del problema. Este, tratará de encontrar los valores restantes de la función objetivo, y adecuar los valores ya existentes.

Las iteraciones realizadas por el método simplex sin inicialización fueron 75 (Ver siguiente Tabla), acomparación de las que realizó inicializado que fueron 48. Aquí se puede marcar la conveniencia de la utilización del método simplex sin inicializar cuando el número de variables exceda en mayor grado al número de restricciones, ya que, aunque el número de iteraciones realizadas con inicialización fué menor, a este se le sumará el proceso de encontrar la solución a un sistema de ecuaciones de grado 71 lo que arroja un tiempo mayor de proceso, y además que no existe mucha diferencia entre el número de iteraciones realizadas. Es decir, que la inicialización será conveniente cuando el número de variables a inicializar en el problema sea, ó se aproxime, al número total de variables involucradas.

INICIALIZACION

Gauss-Jordan	*Kaczmarz*	*SIMPLEX*
Interaciones Gauss-Jordan	Iteraciones Kaczmarz	Iteraciones
71	923	75
Variables Resueltas	Aproximación	Vars. Resueltas
X_1 X_{32}	0.001	F.O.
Iteraciones Simplex	Vars. Resuelt.	
48	X_1 X_{32}	
Total Iter.	*Total Iter.*	*Total Iter.*
119	*No Opera Estruc Ver Tema 4.2*	75

Sobre el método de Kaczmarz se puede decir que es muy práctico cuando no se encuentra ninguna estructura en la matriz. En este caso fué muy práctico en la inicialización, pero no en el proceso simplex, esto es debido a los motivos que se mencionaron el tema 4.2 (el número de variables es mayor que el número de restricciones). En este caso, el método simplex inicializado con Gauss-Jordan fué realizado satisfactoriamente, llegando al mismo resultado que el obtenido sin haber inicializado el método.

4.4 Observaciones .

Como se puede observar, dependiendo la estructura del problema será conveniente inicializar ó no el método simplex, pero el resultado de la optimización tendrá que ser el mismo.

En general, si se cuenta con un problema el cual contiene aproximadamente igual número de variables y de restricciones la conveniencia de usar la inicialización es mucha, ya que se puede ahorrar proceso encontrando la solución de casi todas las variables en este paso y luego con el método simplex terminar la búsqueda de la solución de las variables restantes. En este caso es conveniente porque la inicialización se encargó de dar solución a casi todo el problema, lo cual no sucede cuando el número de variables es mucho mayor al número de restricciones (problema solucionado), en este caso la solución es alcanzada, pero lo que realiza la inicialización es

aproximar en menor grado a la solución, ya que el número de variables que se solucionan ó que se toman en cuenta son mucho menor que las variables involucradas en el proceso.

La solución alcanzada por el método simplex y la solución alcanzada con inicialización externa es la misma, lo importante en este caso es determinar perfectamente las limitaciones que se tengan, y principalmente, la estructura del problema. Al respecto, en este problema, si se hubiera buscado una solución rápida (por falta de tiempo), dada la estructura del problema, se hubiera optado por utilizar el simplex sin inicialización externa. Si una limitante no hubiese sido el tiempo, sino la estructura del problema, la inicialización con Gauss-Jordan no hubiera tenido contratiempos, pero para inicializar con el método de Kaczmarz existiría contratiempos, ya que la estructura propia del problema es del tipo dos (Ver tema 4.2) que corresponde a mayor número de variables que de restricciónes, por lo cual no puede ser llevada a cabo la unificación de los resultados con el método simplex. Lo importante en este sentido es saber distinguir el tipo de problema para determinar que camino es el más apropiado, y conociendo esto, las limitantes de tiempo ó de estructura podrán ser sobre llevadas directamente sin mayor problema.

Por último, el grado de pureza de la matriz del problema fué de aproximadamente un veinteavo, de tal forma que las operaciones

únicamente se realizaron en un veinteava parte de la matriz (elementos registrados diferentes de cero) dando como resultado, el ahorro de una grán cantidad de memoria. Por otro lado, el tiempo de proceso se encuentra estrechamente relacionado con el número de elementos no-ceros en la matriz.

Un aspecto importante que se debe de tomar en cuenta en el proceso es el tiempo de la busqueda secuencial de los elementos. La busqueda secuencial de los elementos esta también en función del número de elementos no-ceros en la matriz (recordando que se utilizan listas ligadas), y una busqueda es realizada pasando por los elementos de la matriz hasta encontrar un elemento especifico. Por ejemplo, en una matriz dispersa de grandes dimensiones no existen contratiempos, ya que el número de elementos no-ceros es proporcionalmente pequeño. Lo anterior no sucede cuando la matriz contiene la mayor parte de sus coeficientes diferentes de cero, en este caso, el proceso tenderá a ser mayor por el tiempo de busqueda secuencial.

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES Y RECOMENDACIONES

El objetivo principal de este trabajo fué el encontrar una estructura que permitiera almacenar una matriz dispersa generada por un problema con muchas variables y poca relación entre ellas, de tal forma que el gasto de memoria fuera estrictamente de el necesario, lo cual fué llevado a cabo satisfactoriamente con el uso de listas dinámicas. Por ejemplo, en el problema solucionado, el requerimiento de memoria se redujo aproximadamente en una veinteava parte del total necesitado, lo cual resulta ser bastante satisfactorio.

Se mencionó que una característica muy especial que debían de tener los algoritmos para este tipo de problemas de dispersión, era el mantener la dispersidad en la matriz. Esta característica impide que, mientras es realizado el proceso, se genere espacio en memoria reservado para números que no se utilizarán (números iguales a cero), lo que hace que la estructura del problema sea siempre completa. Esta característica fué una de las más importantes en esta investigación.

El poder realizar operaciones en la lista, tales como inserción, eliminación y creación de *elementos*, permitió la obtención de una mayor

flexibilidad en el manejo de datos, principalmente en la preservación de dispersidad, el fácil acceso a los datos, implementación del método simplex y de los métodos tanto directo como iterativo para solucionar sistemas de ecuaciones.

Cabe mencionar la importancia que tuvo el lenguaje en que se desarrollaron los métodos e implementaciones (lenguaje C), ya que este permitió la facilidad de manejo de todas las estructuras y operaciones necesarias. Un lenguaje con estas características es fundamenteal para todo tipo de proyectos, ya que si en un principio, el lenguaje no hubiese permitido realizar las operaciones fundamentales, el fin perseguido, por lo menos en esta investigación, no hubiera sido alcanzado.

Por otro lado, la relación del método simplex para grandes dimensiones con las matrices dispersas fué el punto fundamental de partida para la obtención de un método simplex estructurado con listas dinámicas capaz de analizar mucha más información que el método convencional, esto es debido al ahorro de memoria ocasionado por el aprovechamiento de la dispersidad de la matriz del problema.

Cabe hacer la aclaración de que estas adaptaciones sirven de igual forma para solucionar problemas "pequeños" con matrices densas. El inconveniente resulta ser, que el tiempo de proceso que se realice

para solucionar el problema será mayor, y estará en función del número de coeficientes no-ceros en la matriz. En estos casos se tendría que pagar el precio de usar listas ligadas. En otras palabras, mayores búsquedas implican mayor tiempo de proceso.

Por último, el aspecto determinante en el uso de inicialización en el método simplex, no fué tan solo una nueva alternativa de uso del método, sino que también representa una forma de explicación parcial de este. Es decir, si la solución suministrada pertenece a una orilla del conjunto factible del problema, el método simplex únicamente buscará en las esquinas adyacentes para encontrar la esquina óptima. Esto facilita mucho más la interpretación geométrica del método.

La necesidad de algún método auxiliar es fundamental para cualquier procedimiento. Se sabe que los métodos obedecen a un determinado esquema, el cual ha sido complementado con el paso del tiempo. En este caso, por ejemplo el método de Gauss-Jordan con una implementación del algoritmo de Hall asegurará la existencia de un pivote ó la singularidad de la matriz, o que por productos innertes de vectores con referencia a un hiperplano se asegure la convergencia del método de Kaczmarz, o que con las reglas lexicográficas se determine la variable que saldrá de la base sin riesgo a ciclar indefinidamente dentro del método simplex. En el caso de esta investigación, el uso de una inicialización servirá como ayuda ó explicación parcial del método simplex.

Para tener una visión general de la investigación, todos los temas que la complementan, formaron parte de una estructura que fué encaminada a la solución de problemas de progamación lineal con matrices dispersas. Desde luego que el concepto de matrices dispersas es tomado desde el punto de vista de que no existen los ceros.

El análisis se llevó con fluidez comenzando desde el concepto de desigualdades lineales, como se répresentan en el campo geométrico, la región factible del problema, la interpretación gráfica (que es indispensable para la mejor visualización del problema), la relación existente entre la metodología del método simplex y la geometría, la descripción, el uso y el tratamiento de la matrices dipersas, la búsqueda de estructuras accesibles, la adecuación de algoritmos, las estructuras dinámicas, las técnicas alternas de solución de sistemas de ecuaciones, la representación de datos de lista, los problemas de implementación y la aplicación. Todos estos temas y conceptos fueron abordados y complementados unos con otros para obtener el resultado de la investigación.

Es fácil observar que tanto los objetivos y análisis propuestos ó expuestos en un principio fueron llevados a cabo. En suma, la optimización de memoria y de proceso fué logrado con el uso de listas dinámicas. La optimización del problema analizado fué llevada a cabo con la implementación de la investigación realizada en el método simplex.

La limitante más marcada de la investigación fué la obtención de los datos del problema a analizar, ya que la fuente de donde se había programado obtener dicha información negó su participación. El problema de la planta refinera de Salina Cruz, analizado en el último capítulo, fué facilitado por el Dr. Antonio Montalvo, Superintendente Gerencial de Transformación Industrial en Petroleos Mexicanos.

La recomendación principal de la investigación está encaminada hacia la diferencia de caracteres de los métodos directos y los métodos iterativos. El caso de que el número de variables sea mayor que el número de restricciones ocasiona una pérdida visual del problema cuando se intenta la inicialización iterativa, por el motivo de que algunas variables no son tomadas en cuenta.

De los métodos iterativos se puede aprovechar el hecho de que la matriz no es alterada por transiciones y de ahí tras una investigación, poder encontrar compatibilidad con el método simplex. Este podría ser un análisis muy interesante que podría ser desarrollado en métodos números.

Otro aspecto de los métodos iterativos es su convergencia. El método no convergerá cuando la solución real del método se encuentre muy alejada de la solución inicial. En este caso se pueden recomendar dos caminos a seguir para asegurar la convergencia: Utilizar una

tolerancia de convergencia igual o cercano a cero y un límite de iteraciones muy grande, ó utilizar un método directo. Se recomienda también que el número de iteraciones que se proponga como límite para converger sea proporcional a la dimensión de la matriz. Es decir, para una buena convergencia utilizar N x 13 iteraciones, donde N es la dimensión de la matriz cuadrada.

Observación Final :

Para finalizar, al igual que el problema analizado, existen miles de problemas en la vida cotidiana que pueden ser solucionados por medio de este tipo de métodos, y a medida que las empresas se desarrollen, se requerirá de un mayor número de herramientas matemáticas más potentes y específicas que ayuden a resolver los problemas que con el tiempo se presenten, en este caso el problema fué; el proceso global de una gran cantidad de datos en un problema de programación lineal.

ANEXOS

A N E X O A : PRUEBAS LEXICOGRAFICAS.

Si al estar iterando con el método simplex sucede el caso de que hay un empate en la decisión de cual columna entrará a la base, aquí arbitrariamente se elige una de ellas y con esto no se causará ningún tipo de problema consecutivo. Pero cuando se tiene un empate en la decisión de cual variable saldrá de la base, entonces no se puede romper el empate arbitrariamente porque se podría ocasionar un ciclaje, que después de varias iteraciones, llevará al mismo punto en el cual se realizó la decisión. Este problema se debe a lo degenerado de las soluciones factibles básicas.

La forma de evitar este tipo de problemas es usar las llamadas reglas lexicográficas, que pueden asegurar que la elección de la variable de salida es la correcta.

Los pasos a seguir son los siguientes:

Paso 1 : Si existe un empate entre las variables que van a salir, construyase un subconjunto de las variables empatadas que pertenezcan a la base.

Paso 2 : Hacer la comparación de cual variable empatada aparece registrada en el primer componente de la base. Si existe esa

variable se borra del subconjunto.

Paso 3 : Si solamente existe en el subconjunto una variable, esa es la variable que saldrá de la base. Si existen dos o más, regrese al paso 2 y comparar pero ahora con el segundo componente de la base, y así sucesivamente.

Ejemplo : Se dice que la base está compuesta por las siguientes variables:

$$B = x5,\ x6,\ x7$$

y que hay un empate entre las variables x5 y x6.

Paso 1 :

$$B(1) = x5,\ x6$$

Paso 2 : La primer componente de la base es x5, y como si existe dentro del subconjunto, se elimina x5 del subconjunto B(1).

Paso 3 :

$$B(1) = x6 \quad .$$

Como ya es el único elemento del subconjunto, la variable x6 es la que sale de la base.

El listado del método se presenta en el anexo .

A N E X O B : PROCEDIMIENTOS PARA SIMPLEX Y GAUSS-JORDAN .

En este Anexo se presentan los procedimientos y funciones más importantes en la realización de los métodos Simplex y Gauss-Jordan.

B.1. Codigo del Algoritmo de Hall .

A esta rutina se le suministra un apuntador al primer elemento de la lista, un apuntador al primer elemento de la base y las coordenadas del lugar donde se desea tener un coeficiente no-cero. El primer paso es buscar un elemento a la derecha de la columna especificada sobre el mismo renglón (lineas 2 y 4). Si se encuentra uno, se actualiza la base cambiando la variable de la columna que se encontró por la variable de la columna donde no existia pivote (lineas 7-13), de tal forma que los valores que quedan en la base son las variables que tendrán solución en el método de Gauss-jordan. Luego intercambia las columnas y termina la función. Hay que hacer notar que cada llamada de este proceso solamente realiza una busqueda de elemento, realiza las operaciones pertinentes y termina.

El código del método es el siguiente:

```
    BuscaDerecha (lista, bases, ren, col)
    ENLACE  lista;
    BASE    bases;
    int     ren,col;
    {
      ENLACE  cc,ccc,nuevo,apuntando;
      BASE    bbb,bb;
      int     colaux,colu1,colu2,renglo1,renglo2,aux1;
      float   auxvalor;
      char cambiocc,cambioccc;
1     cc=lista;
2     while(cc->renglon ! ren ) cc=cc->siguiente;
3     do {
4          if(cc->columna > col ) {
5               if( cc->renglon == ren && cc->columna != 2000 ) {
6                    colaux=cc->columna;
7                    bbb=bb=bases;
8                    while(bb->columna != col ) bb=bb->siguiente;
9                    while(bbb->columna != colaux )
10                        bbb=bbb->siguiente;
11                   aux1=bb->columna1;
12                   bb->columna1=bbb->columna1;
13                   bbb->columna1=aux1;
14                   ccc=cc=lista;
15                   while(1) {
16                        while(cc->columna != col) {
17                             if(cc == NULL) break;
18                             cc=cc->siguiente;
19                        }
20                        while(ccc->columna != colaux ) {
21                             if(ccc == NULL) break;
22                             ccc=ccc->siguiente;
23                        }
24                        if(cc==NULL && ccc==NULL) break;
25                        colu1=cc->columna;
26                        renglo1=cc->renglon;
```

```
27                  colu2=ccc->columna;
28                  renglo2=ccc->renglon;
29                  if(ccc==NULL) renglo2=5000;
30                  if(cc==NULL) renglo1=5000;
31                  if( renglo1 == renglo2) {
32                       auxvalor=cc->valor;
33                       cc->valor=ccc->valor;
34                       ccc->valor=auxvalor;
35                       cambiocc=cambioccc='s';
36                  }
37                  if( renglo1 > renglo2 || renglo1 == 0 &&
38                                           renglo2 > 0 ) {
39                      apuntando=lista;
40                      if(apuntando != ccc)
41                      while(apuntando->siguiente != ccc)
42                          apuntando=apuntando->siguiente;
43                      nuevo=(ENLACE)malloc(sizeof(ELEMENTO));
44                      nuevo->valor=ccc->valor;
45                      nuevo->renglon=ccc->renglon;
46                      nuevo->columna=col;
47                      nuevo->siguiente=ccc->siguiente;
48                      if(apuntando != ccc)
49                           apuntando->siguiente=nuevo;
50                      else c=lista=nuevo;
51                      apuntando=ccc;
52                      free(apuntando);
53                      ccc=nuevo->siguiente;
54                      cambiocc='n';
55                  }
56                  if( renglo1 < renglo2 || renglo2 == 0 &&
57                                       renglo1 > 0) {
58                      apuntando=lista;
59                      while(apuntando->siguiente != cc)
60                           apuntando=apuntando->siguiente;
61                      nuevo=(ENLACE)malloc(sizeof(ELEMENTO));
62                      nuevo->valor=cc->valor;
```

```
63                              nuevo->renglon=cc->renglon;
64                              nuevo->columna=colaux;
65                              nuevo->siguiente=cc->siguiente;
66                              apuntando->siguiente=nuevo;
67                              apuntando=cc;
68                              free(apuntando);
69                              cc=nuevo->siguiente;
70                              cambioccc='n';
71                          }
72                          if(cambioccc=='s') ccc=ccc->siguiente;
73                          if(cambiocc=='s')  cc=cc->siguiente;
74                          cambioccc=cambiocc='n';
75                      }
76                  }
77                  if( renglol != ren ) { noderecha='n'; break; }
78                  break;
79              }
80              cc=cc->siguiente;
81      } while(cc->columna != col || cc != NULL );
82  }
```

Este procedimiento contiene dos apuntadores "cc" y "ccc" los cuales apuntarán respectivamente a la columna donde no se encontró pivote y a la columna donde sí se encontró sobre el mismo renglón (líneas 15-23). Luego se procede a intercambiar las columnas apuntadas por "cc" y "ccc" pudiendo suceder alguno de los 3 casos siguientes por ser dispersa la matriz:

1) Cuando los coeficientes a intercambiar corresponden al mismo renglón. Aquí simplemente se intercambian los contenidos los campos (líneas 31-36).

ii) Cuando el renglón correspondiente al coeficiente de la columna del lado izquierdo es mayor que el renglón del coeficiente del lado derecho. En este caso se genera un nuevo elemento en la columna izquierda con el renglón igual al del elemento de la columna derecha y termina con la eliminación de este último (lineas 37-55).

iii) Cuando el renglón correspondiente al coeficiente de la columna del lado izquierdo es menor que el renglón del coeficiente del lado derecho. En este caso se genera un nuevo elemento en la columna derecha con el renglón igual al del elemento de la columna izquierda y termina con la eliminación de este último (lineas 56-71).

Este procedimiento terminará bajo alguno de estos dos casos : que no encuentre elemento a la derecha del renglón especificado (linea 77) ó que ya no encuentre elementos en las columnas especificadas para seguir intercambiando (linea 24).

B.2. Codigo para hacer Columnas Unitarias .

Este procedimiento, dado un pivote y un número de renglón de la matriz, hace la columna correspondiente al pivote unitaria. Es decir, vuelve la columna de elementos ceros con excepción del elemento pivote que será igual a uno. Este procedimiento necesita un apuntador al primer elemento de la lista (dado por el primer parametro), las

coordenadas del elemento pivote (dadas por el segundo y cuarto parámetros) y el número de renglón que se va a modificar (dado por el tercer parámetro).

Primero se encuentra el pivote (lineas 10,11,18 y 19), si este es cero el procedimiento termina (linea 22), si no es cero sigue con el proceso hasta que opera en todos los elementos del renglón indicado por el tercer parámetro.

El listado del procedimiento se presenta a continuación :

```
 1   HazUnoColumna (lista, ren, renmod, col)
 2   ENLACE lista;
 3   int    ren,renmod,col;
 4   {
 5        ENLACE  lista1,lista2,temp,auxincer,auxc;
 6        float   valor,elemk,elemi,pivotei;
 7        int     linea,columna,columnk,columni;
 8        char    cambiok='a',cambioi='a';
 9        lista2=lista1=lista;
10        while(lista1->renglon != ren )
11           lista1=lista1->siguiente;
12        while(lista2->renglon != renmod )
13           lista2=lista2->siguiente;
14        while( lista2->columna != col ) {
15           if(lista2->columna > col ) break;
16           lista2=lista2->siguiente;
17        }
18        while(lista1->columna != col)
19           lista1=lista1->siguiente;
20        if( lista2->columna > col ) pivotei=0;
```

```
21        else pivotei= lista2->valor;
22        if( pivotei != 0 ) {
23           lista2=lista1=lista;
24           while(lista1->renglon != ren )
25              lista1=lista1->siguiente;
26           while(lista2->renglon != renmod )
27              lista2=lista2->siguiente;
28           while(lista1->renglon == ren || lista2->renglon ==
29                                                renmod) {
30              columnk=lista1->columna;
31              columni=lista2->columna;
32              elemk=lista1->valor;
33              elemi=lista2->valor;
34              if(columnk != columni) {
35                 if (columnk > columni) {
36                    elemk=0;
37                    cambiok='s';
38                    lista2->valor = elemi;
39                 }
40                 if (columni > columnk) {
41                    temp=lista2;
42                    lista2=lista;
43                    if( temp != lista )
44                       while(lista2->siguiente != temp)
45                          lista2=lista2->siguiente;
46                    auxincer=(ENLACE)malloc(sizeof(ELEMENTO));
47                    auxincer->valor = (-pivotei)*elemk;
48                    auxincer->renglon = renmod;
49                    auxincer->columna = columnk;
50                    if(temp != lista)
51                       lista2->siguiente = auxincer;
52                    auxincer->siguiente = temp;
53                    if(temp == lista) lista2=auxincer;
54                    if(temp != lista)
55                       lista2=lista2->siguiente;
56                 }
```

```
57                else lista2->valor = elemi-pivotei*elemk;
58                if( lista2->valor == 0 )
59                   if (lista2->columna== 2000) break;
60                   temp=lista2;
61                   lista2=lista;
62                   if (temp == lista2)
63                      c=c->siguiente;
64                      lista=lista->siguiente;
65                      free(lista2);
66                   }
67                   else {
68                      while( lista2->siguiente != temp)
69                         lista2=lista2->siguiente;
70             lista2->siguiente=lista2->siguiente->siguiente;
71                      free(temp);
72                   }
73                }
74                if(cambiok != 's') lista1=lista1->siguiente;
75                if(cambioi != 's') lista2=lista2->siguiente;
76                cambiok=cambioi='a';
77             }
78          }
79       }
80    }
```

Una vez que se localiza el pivote diferente de cero se procede a localizar los renglones que operarán durante el procedimiento (lineas 24-27). "lista1" buscará el primer elemento del renglón normalizado (principal) y "lista2" buscará el primer elemento del renglón a modificar (secundario) (tercer parametro del procedimiento).

Para modificar el renglón secundario con respecto al principal, por ser la matriz dispersa, pueden suceder tres casos :

i) Cuando los coeficientes de ambos renglones corresponden la misma columna. Aquí simplemente el valor del elemento del renglón a modificar toma el valor de él mismo disminuido por el producto del pivote y el valor del elemento del renglón principal (línea 57).

ii) Cuando la columna correspondiente al coeficiente del renglón que se va a modificar es mayor que la columna del coeficiente del renglón principal. En este caso se genera un nuevo elemento igual al correspondiente del renglón principal, localizado en el renglón a modificar en la columna igual al del elemento del renglón principal (líneas 40-56).

iii) Cuando la columna correspondiente al coeficiente del renglón principal es mayor que la columna del coeficiente correspondiente al renglón a modificar. En este caso no se hace ningún movimiento ya que se toma implícitamente el elemento del principal igual a cero y no le causa cambios al renglón a modificar (líneas 35-39).

Un aspecto muy importante de este procedimiento es que cuando se crea un elemento igual a cero este es borrado de la lista inmediatamente, esta característica mantiene la dispersidad en la lista (líneas 58-66).

Este procedimiento terminará si no existe pivote ó cuando ya no existan elementos en ninguno de los dos renglones que operan en el procedimiento (línea 28).

B.3. CODIGO DE NORMALIZACION DE RENGLONES .

Este procedimiento únicamente realiza la división de los elementos del renglón especificado por el segundo parámetro, entre el valor del coeficiente pivote especificado por el renglón y la columna de los parámetros del procedimiento. De igual forma se le suministra un apuntador al primer elemento de la lista para realizar las operaciones pertinentes.

Este procedimiento realiza al principio una busqueda del elemento pivote (lineas 2 y 3), si no lo encuentra hace la variable global "no_solucion" = 's' (lineas 10-13) y termina el procedimiento. Si lo encuentra hace "no_solucion" = 'n' (lineas 5-9) y continua con el proceso.

El listado se presenta a continuación :

```
    HazUnoPrincipal (lista, renglon, columna)
    ENLACE lista;
    int    renglon,columna;
    {
       float  aux,aux1;
       ENLACE arreglo,apunta;
1      arreglo=lista;
2      while(arreglo->renglon != renglon )
3         arreglo=arreglo->siguiente;
4      while(arreglo->renglon == renglon ) {
5         if (arreglo->columna == columna) {
6            no_solucion='n';
```

```
 7              aux=arreglo->valor;
 8              break;
 9           }
10           if (arreglo->columna > columna) {
11              no_solucion='s';
12              break;
13           }
14           arreglo=arreglo->siguiente;
15        }
16        if(no_solucion == 'n') {
17           arreglo=lista;
18           while(arreglo->renglon != renglon)
19              arreglo=arreglo->siguiente;
20           while(arreglo->renglon == renglon ) {
21              arreglo->valor = arreglo->valor / aux;
22              arreglo = arreglo->siguiente;
23           }
24        }
25     }
```

Cuando el procedimiento encuentra el pivote simplemente divide todos los elementos del renglón entre el valor que contiene el elemento pivote (líneas 20-22).

El procedimiento termina cuando ya no se encuentran elementos en el renglón (línea 20).

B.4. Código para Encontrar el Elemento Más Negativo en un Renglon.

Esta función únicamente encuentra el valor más negativo del

renglón especificado por el segundo parámetro (lineas 3-11). Si no encuentra negativos hace a la variable "masnegativos"='n' (línea 1 . Esta variable permanece igual). Pero si encuentra alguno, asigna el valor de su columna a la variable "masnegacolumna" que es una **variable global** (línea 6). El valor de "actual" comienza valiendo cero y si existe algún elemento menor que el asigna dicho valor a este (línea 7).

La función termina hasta que se termine de revisar el renglón especificado.

```
     MasMegativo (lista, renglon)
     ENLACE lista;
     int    renglon;
     {
        float actual=0.0;
 1      masnegativos='n';
 2      while(lista->renglon != renglon) lista=lista->siguiente;
 3      while(lista->renglon == renglon) {
 4         if( lista->columna != 2000)
 5           if (lista->valor < actual) {
 6              masnegativos='s';
 7              actual=lista->valor;
 8              masnegacolumna=lista->columna;
 9           }
10      lista=lista->siguiente;
11      }
12   }
```

B. 4. Codigo para Encontrar Nuevo Pivote .

Este procedimiento es muy importante, ya que determinará cual variable saldrá de la base. Hay que recordar que una elección inadecuada de esta variable podría ciclar indefinidamente el método, hasta llegar al mismo lugar de donde se había partido, de modo que el problema se degeneraría.

Para evitar este problema se aplicarán las pruebas lexicograficas, (incluidas en este listado) que por medio de comparaciones más profundas, se determinará cual variable saldrá de la base (Ver Anexo A).

Este procedimiento, con el valor de la columna suministrado por el último parámetro (que es la variable que entraría a la base), será la columna donde se buscará la elección más apropiada que determinará la variable que ha de salir de la base, ya sea por la elección directa (dividiendo los elementos del vector de la columna seleccionada para entrar y el vector actual de recursos para determinar el menor) ó por la consulta a las reglas lexicográficas si es que existe empate. El apuntador al primer elemento a la base es suministrado para realizar las comparaciones pertinentes en caso de que exista empate, y las pruebas lexicográficas puedan llevarse acabo. (Para mayor información

acerca de estas pruebas ver Anexo A).

El listado se presenta acontinuación:

```
NuevoPivote (lista, base, columna)
ENLACE lista;
BASE   base;
int    columna;
{
   ENLACE ccc;
   BASE   bbb;
   float  auxiliar,division,elegido,previovalor,actualvalor;
   int iguales,previovalorren,actualvalorren;
   char ya='n';
1  ccc=lista;
3  iguales=negativos=0;
4  while( ccc->renglon != 0 ) {
5     if( ccc->columna == columna ) {
6        actualvalor=ccc->valor;
7        if(actualvalor < 0)
8           negativos++;
9        if(actualvalor > 0) {
10          actualvalorren=ccc->renglon;
11          while( ccc->columna != 2000) ccc=ccc->siguiente;
12          division=ccc->valor/actualvalor;
13          if (ya == 'n') {
14              elegido=division;
15              nuevopivotes=ccc->renglon;
16              ya='s';
17          }
18          else if( division < elegido && division != 0 ) {
19             elegido=division;
20             nuevopivotes=ccc->renglon;
21          }
```

```
22                else if (division == elegido) {
23                   bbb=base;
24                   iguales=1;
25                   while(iguales != 0) {
26                      if(previovalorren == bbb->renglon ) {
27                         nuevopivotes=actualvalorren;
28                         iguales=0;
29                      }
30                      if(actualvalorren == bbb->renglon ) {
31                         nuevopivotes=previovalorren;
32                         iguales=0;
33                      }
34                      if(iguales==1) bbb=bbb->siguiente;
35                   }
36                }
37             }
38          }
39          previovalor=actualvalor;
40          previovalorren=actualvalorren;
41          ccc=ccc->siguiente;
42       }
43    }
```

Este procedimiento busca elemento en la columna especifica para cada renglón (línea 5). Si encuentra dicho coeficiente, este puede caer en cualquiera de los dos casos siguientes:

a) Cuando el valor del coeficiente es menor que cero. Aqui simplemente se incrementa la variable "negativos" en uno (lineas 7 y 8),de tal forma que, fuera de este procedimiento, si el valor de "negativo" es igual al número de renglones, entonces el problema es degenerado.

b) Cuando el valor del coeficiente es mayor que cero. Aquí se busca el valor de "b" correspondiente (línea 11) y se realiza la división pertinente (línea 12). La variable "ya" indica cuando un pivote ya ha sido elegido en primera instancia y el cual podrá irse mejorando. El valor de la división se almacena en la variable "elegido" para poder realizar luego las comparaciones (líneas 13-17). Al repetir el procedimiento y encontrar otro elemento sobre la columna en otro renglón se realiza el mismo procedimiento. Como "ya" = 's' entonces se realiza la comparación de "elegido" con "division" (producto de la búsqueda anterior). En la comparación puede suceder alguno de dos casos siguientes:

i) "division" < "elegido". Aquí "elegido" toma el valor de "division" (líneas 18-21).

ii) "division" = "elegido". Se realizan las pruebas lexicográficas. Se busca el primer elemento de la base (línea 23) y se comparan los renglones de las variables empatadas. Con cada ciclo se recorre un elemento en la base y se vuelve a comparar hasta cumplir con las reglas (líneas 22-26) (ver anexo A).

El valor del renglón elegido quedará almacenado en la variable "nuevopivotes" (líneas 15,20,27 y 31).

El procedimiento termina cuando ya no existen renglones que analizar (línea 9).

A N E X O C : EL ALGORITMO DE HALL.

Cuando una matriz A de n x n tiene algunos de sus elementos iguales a cero en su diagonal, hay que realizar algunos pasos previos de intercambios de filas o columnas de tal forma que la diagonal resulte libre de ceros. El conjunto de no-ceros obtenido es llamado transversal. Hay que tener cuidado con las matrices, ya que una diagonal completa implica una matriz no-singular, pero una matriz singular puede tener también diagonal completa.

Cuando una matriz no puede ser permutada para que la diagonal completa resulte se dice que es una matriz simbólicamente singular. Si el máximo número posible de elementos en la transversal es $k < n$, entonces k es el rango simbólico y n-k es la nulidad simbólica.

El algoritmo de Hall (1956) requiere de "n" pasos: el propósito de la k-ésima iteración es colocar un elemento no-cero en el elemento k-ésimo de la diagonal. Despues de que k iteraciones han sido realizadas la diagonal contiene no-ceros en sus k iniciales posiciones. Entonces en la $k+1$-ésima iteración se pueden tener alguno de los tres casos siguientes:

1) a(k+1,k+1) diferente de cero ⇒ iteración k+1 terminada.

2) Un no-cero puede ser encontrado a la derecha de la columna k+1 sobre el mismo renglón. En este caso, intercambiando columnas se traera ese elemento hasta la columna k+1 (en la diagonal).

La sub-matriz cuadrada formada por las filas y columnas de la uno a la k no se modifica por los intercambios y los elementos de la diagonal permanecen igual.

3) Cuando no se encuentran elementos no-ceros a la derecha, se forma un registro para posiciones visitadas, y se utilizará de la siguiente forma: se registra la primera posición que es k+1, y luego:

Paso 1) Se busca un elemento no-cero en el lado izquierdo de la columna actual. Si se encontró en la columna "f" entonces se busca la posición (f,f) a lo largo de la columna. Se registra "f". Si no se encontró elemento a la izquierda, el método termina y la matriz es singular.

Paso 2) Se busca un elemento no-cero a la derecha de la k-ésima posición en el mismo renglón "f". Si encuentra un elemento (columna "w"), entonces se registra "w" y termina la busqueda. En otras palabras, este proceso termina cuando un elemento registrado se encuentra en el rango de columnas (k+1 ... n). Si no se encuentra

elemento a la derecha en (k+1 ... n) entonces pasar al paso 1 sin cambiar el registro. En estos momentos el registro esta formado por (k+1, f, w). El siguiente paso será :

el renglón k+1 se convierte en renglón f
el renglón f se convierte en renglón w
el renglón w se convierte en renglón k+1,

éste proceso traerá un renglón a la posición k+1 que contenga un elemento a la derecha de la columna k+1, de tal forma que, teniendo esta matriz, *se* analiza nuevamente con los casos 1), 2) y 3) y se realizan las indicaciones convenientes.

Este método es muy sencillo y muy práctico para traer elementos a la diagonal principal, si es que existe. A continuación se realiza un ejemplo para ver el comportamiento del método.

Sea la siguiente matriz de 5 x 5 :

	1	2	3	4	5
1	x				x
2		x			
3	x				
4			x	x	
5					x

Para la posición (1,1) si existe el pivote (caso 1), se continua con el elemento (2,2) que también existe (caso 1), luego el elemento (3,3) que no existe y no tiene elementos a la derecha (no es caso 1 ni caso 2). Este corresponde al caso 3. Se encuentra un elemento en la columna 1, por lo que el registro va como sigue (3 ,1). Por la columna 1 se encuentra el elemento (1,1) y se busca a la derecha de la columna 3 del renglón 1 y se encuentra un elemento en la columna 5 y se registra . Como 5 > k=3, termina el proceso, y se realiza el análisis con el registro = (3, 1, 5, 3).

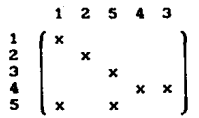
El cambio de renglones se realiza de la siguiente manera :

el renglón 3 se convierte en renglón 1
el renglón 1 se convierte en renglón 5
el renglón 5 se convierte en renglón 3,

de tal forma que la matriz queda como sigue:

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} x & & & & \\ & x & & & \\ & & & & x \\ & & & x & x \\ x & & & & x \end{pmatrix} \end{array}$$

y el caso al que pertenece esta matriz para k = 3 es el caso 2 donde encuentra un elemento a la derecha de la columna 3, e intercambiando las columnas 5 y 3 la matriz resulta como sigue:



así el elemento (3,3) ya existe, y de igual forma se continua hasta el paso 5.

Observaciones .

Si se interactúa éste algoritmo con la eliminación gaussiana resulta ser un método mucho más rápido y eficiente, ya que mientras la eliminación gaussiana barre la matriz de izquierda a derecha, es decir, que todos los elementos existentes se irán registrando a la derecha de la diagonal, el método de Hall únicamente recurrirá a los casos 1, 2 y teóricamente, nunca recurrirá al caso 3.

El listado correspondiente a este método está incluido en el anexo B, bajo el nombre de "BuscaDerecha".

BIBLIOGRAFIA

BIBLIOGRAFIA

CASIO: Computadora Personal PB-2000C, 1991.
" Introducción al Lenguaje de Programación C "
Ed. CASIO.

DUFF, REID & ERISMAN., 1978.
" Direct Methods for Sparse Matrices ", Oxford Science Publications.

FERNANDES, Antonio., 1987.
Articulo: " Dinamic Memory Allocation ", Revista BYTE Enero 1987, p. 169.

KELLEY POHL, 1987.
" LENGUAJE C. Introducción a la Programación "
ADDISON-WESLEY IBEROAMERICANA.

DUFF & STEWART, 1978.
" Sparse Matrix Proceedings "
Symphosium: Filadelphia.

MENDIETA A. , Angeles., 1989.
" Tesis Profesionales ", Ed. Porrúa.

PHIPPS, Thomas Jr., 1986.

Articulo: " The Invertion of Large Matrices ", Ed. BYTE Abril 1986, p. 181.

PISSANETZKY, Sergio.

" Sparse Matrix Technology ", Ed. Accademic Press.

PRAWDA, Juan., 1976.

" Métodos y Modelos de Investigación de Operaciones " Ed. Limusa. Vol. I "Modelos Deterministicos".

CHILDT, Herbert., 1991.

" Programación en TURBO C ", Ed. Boorland-Osborne Mc.Graw-Hill México.

ROSE WILLOUGHBY., 1972.

" Sparse Matrices and their Applications " Ed. Plenum Press.

STRANG, Gilbert.

" Linear Algebra and its Applications ", Ed. Academic Press.

TEWARSON, 1973.

" Sparse Matrices ", Ed. Academic Press.

WINSTON, Wayne L., 1987.

" OPERATION RESEARCH : Applications and Algorithms "

PWS-KENT Publishing Company. Boston.