UNIVERSIDAD NACIONAL DEL ALTIPLANO

ESCUELA DE POSGRADO

PROGRAMA DE MAESTRÍA

MAESTRÍA EN INFORMÁTICA

TESIS

ALGORITMO PARA LA OPTIMIZACIÓN DEL TIEMPO DE EJECUCIÓN EN LA SITUACIÓN DE PROBLEMAS DE PROGRAMACIÓN LINEAL

PRESENTADO POR:

BRAULIO GUTIERREZ PARI

PARA OPTAR EL GRADO ACADÉMICO DE:

MAGISTER SCIENTIAE EN INFORMÁTICA
MENCIÓN EN MATEMÁTICA Y
SIMULACIÓN COMPUTACIONAL

PUNO, PERÚ

2017

UNIVERSIDAD NACIONAL DEL ALTIPLANO

ESCUELA DE POSGRADO

PROGRAMA DE MAESTRÍA

MAESTRÍA EN INFORMÁTICA

TESIS

ALGORÍTMO PARA LA OPTIMIZACIÓN DEL TIEMPO DE EJECUCIÓN EN LA SITUACIÓN DE PROBLEMAS DE PROGRAMACIÓN LINEAL

PRESENTADO POR:

BRAULIO GUTIERREZ PARI

PARA OPTAR EL GRADO ACADÉMICO DE:

MAGISTER SCIENTIAE EN INFORMÁTICA

MENCIÓN MATEMÁTICA Y SIMULACIÓN COMPUTACIONAL

APROBADA POR EL SIGUIENTE JURADO:

PRESIDENTE

.....
Dr. EDGAR ELOY CARPIO VARGAS

PRIMER MIEMBRO

.....
M.Sc. SAMUEL DONATO PÉREZ QUISPE

SEGUNDO MIEMBRO

.....
M.Sc. ELVIS ROQUE CLAROS

ASESOR DE TESIS

.....
Dr. Sc. ALEJANDRO APAZA TARQUI

Puno, 14 de julio del 2017

ÁREA: Modelación matemática.
TEMA: Modelación algorítmica de sistemas dinámicos.

DEDICATORIA

A mi querida hija Joyce Solange Gutiérrez Azaña, fuente de inspiración por quien sigo adelante con la misma convicción de ser mejor persona y profesional día a día. Y a mi amada esposa Ester, por su constante ánimo y apoyo en los momentos difíciles para así desarrollar y concluir este trabajo.

AGRADECIMIENTOS

A Dios, por darme paciencia y tranquilidad en los momentos difíciles durante el desarrollo y conclusión de este trabajo.

A la comisión del Jurado revisor, a todo mis maestros de la Escuela de postgrado, maestría en informática, mención matemática y simulación computacional de la Universidad Nacional del Altiplano por su labor docente.

Al amigo y orientador, Alejandro Apaza Tarqui, por la confianza y hacer alcance de sus sabios consejos, los cuales hicieron posible el desarrollo y culminación de este trabajo.

ÍNDICE GENERAL

CAPÍTULO III

METODOLOGÍA

CAPÍTULO IV

RESULTADOS Y DISCUSIÓN

ÍNDICE DE CUADROS

ÍNDICE DE FIGURAS

ÍNDICE DE ANEXOS

RESUMEN

En la presente investigación construimos un algoritmo para la optimización del tiempo de ejecución en la situación de problemas de programación lineal, este algoritmo está basado en una de las variantes del método de puntos interiores para programación lineal, este algoritmo evoluciona por el interior de la región factible a diferencia del algoritmo Simplex, que evoluciona por sus extremos, disminuyendo considerablemente el tiempo de ejecución en la solución de los problemas. Los algoritmos de puntos interiores surgen, con el trabajo de Karmarkar, como una alternativa de complejidad polinomial al bien establecido método de simplex, para el caso de programación lineal. En 1987 Kojima-Misuno-Yoshise presentan un algoritmo de puntos interiores, llamado Primal-Dual, que seguido del trabajo de Mehrotra en 1992, fundamentan las bases de algunos de los algoritmos existentes más eficientes para programación lineal. Este algoritmos combina de otras técnicas numéricas, tales como el método de newton, lagrange y penalización interna, funcionando estas tres técnicas resulta el algoritmos para la optimización del tiempo de ejecución en la situación de problemas de programación lineal. Se hizo la implementación de las tres técnicas numéricas así como la implementación del algoritmo principal en un software y experimentos computacionales que corroboran la eficacia frente a problemas de grandes dimensiones, se generó problemas de programación lineal de 90 restricciones y 110 variables, 220 restricciones y 320 variables, que fueron resueltos con el algoritmo principal.

Palabras clave: Direcciones de búsqueda, método de barrera logarítmica, óptimos alternativos, programación lineal, puntos interiores.

ABSTRACT

In the present research we constructed an algorithm for the optimization of the execution time in the situation of problems of linear programming, this algorithm is based on one of the variants of the method of interior points for linear programming, this algorithm evolves in the interior of the region feasible unlike the Simplex algorithm, which evolves by its extremes, considerably decreasing the execution time in the solution of problems. Inner-point algorithms arise, with the work of Karmarkar, as an alternative of polynomial complexity to the well established simplex method, for the case of linear programming. In 1987 Kojima-Misuno-Yoshise presented an inner-point algorithm, called Primal-Dual, which followed the work of Mehrotra in 1992, laying the foundations for some of the most efficient algorithms for linear programming. This algorithm combines other numerical techniques, such as the newton method, lagrange and internal penalty, these three techniques result in the algorithms for the optimization of runtime in the situation of linear programming problems. The implementation of the three numerical techniques as well as the implementation of the main algorithm in a software and computational experiments that corroborate the efficacy against big problems, generated problems of linear programming of 90 restrictions and 110 variables, 220 restrictions and 320 variables, which were solved with the main algorithm.

Keywords: Alternative optimal,interior points, logarithmic barrier method, linear programming, search directions.

INTRODUCCIÓN

La literatura sobre los algoritmos de puntos interiores del tipo Primal-Dual es variada y muy extensa, existen diferentes formulaciones, diversos resultados de convergencia y complejidad. Además de las buenas propiedades teóricas, estos algoritmos han demostrado tener buen comportamiento práctico y se han utilizado en distintas aplicaciones de manera satisfactoria.

En el capítulo I, se aborda el planeamiento y formulación del problema, asimismo, la finalidad e importancia, los objetivos de la investigación, la hipótesis de estudio.

El capítulo II, contiene el marco teórico de la investigación, los antecedentes del tema y los teoremas de convergencia, así como las tres técnicas numéricas de optimización, cada uno con sus respectivos algoritmos.

El capítulo III, contiene el tipo y diseño de investigación así como la metodología.

En el capítulo IV se describe los fundamentos del algoritmo principal, con todas las justificaciones matemáticas necesarias, la construcción, implementación y la experimentación con problemas de grandes dimensiones.

CAPÍTULO I

PROBLEMÁTICA DE INVESTIGACIÓN

1.1 PLANTEAMIENTO DEL PROBLEMA

Dentro de la optimización, es indiscutible que el algoritmo simplex (1947) representa un clásico en la resolución de problemas de programación lineal. Más aún, el simplex ha servido como fuente en la construcción de métodos especializados en la resolución de otros problemas, tales como programación entera. No obstante, cuando se realiza un análisis teórico con respecto a su eficacia, el algoritmo simplex es considerado de tiempo de ejecución en el orden exponencial, es decir, que para problemas de programación lineal en espacios n dimensionales, puede realizar en el peor de los casos, alrededor de 2^n iteraciones, y por tanto, teóricamente ineficiente. Por otro lado, los resultados obtenidos en la práctica han mostrado que este algoritmo tiene un desempeño razonable y, frecuentemente, apenas requiere pocas iteraciones para resolver problemas de la vida real con un número de variables no muy grande.

El motivo para el mal comportamiento del algoritmo simplex frente a problemas particulares, tales como los propuestos por V. Klee y. Minty, es que se desplaza por los vértices del poliedro definido por la región de factibilidad. A esto se le suman algunas otras desventajas, tales como la implementación computacional sofisticada, peligro de ciclaje ante degeneración e ineficiencia ante problemas de gran tamaño.

En los últimos cuarenta años surgieron otros tipos de algoritmos con una filosofía diferente, desplazarse por el interior de la región factible. Los resultados teóricos fueron sorprendentes, pues mostraron que el problema de programación lineal podía ser resuelto en un tiempo de ejecución polinomial. Dentro de lo más representativos está el algoritmo de las Elipsoides de Khachian y el algoritmo e Karmarkar.

Por los años noventa surgieron algoritmos con la misma estrategia, desplazarse por el interior del poliedro e factibilidad, pero con la diferencia que éstos usaban tanto el primal como el dual para resolver el problema de programación lineal. Estos algoritmos estaban fundamentados en el algoritmo de Newton, el algoritmo de Lagrange y el algoritmo de Barrera, estas herramientas matemáticas fusionadas produjeron lo que hoy en día se denomina algoritmos de puntos interiores Primal-Dual, y actualmente, es considerado la manera más eficiente que se conoce para resolver el problema de programación lineal. Existen varias versiones de este algoritmo, una de ellas constituirá justamente el motivo de este trabajo.

1.2 JUSTIFICACIÓN

Encontrar algoritmos alternativos al algoritmo simplex y que resuelvan con mayor rapidez para hallar las soluciones de problemas de programación lineal. En muchas Universidades, el algoritmo del Simplex es analiza a fondo, debido a su sencillez y a su fácil comprensión. Esta investigación pretende dejar un legado del algoritmo para la optimización del tiempo de ejecución en la solución de problemas de programación lineal, con todas las justificaciones matemáticas, implementado en el lenguaje de programación de MatLab, haciendo sus experimentos para problemas de grandes dimensiones.

Dar al estudiante una herramienta poderosa, para resolver problemas de programación lineal para la optimización.

1.3 OBJETIVOS

1.3.1 Objetivo general

Desarrollar un algoritmo para la optimización del tiempo de ejecución en la solución de problemas de programación lineal.

1.3.2 Objetivos específicos

1. Construir un algoritmo para la optimización del tiempo de ejecución en la solución de problemas de programación lineal, con todas las justificaciones matemáticas necesarias.
2. Implementar el algoritmo para la optimización del tiempo de ejecución en la solución de problemas de programación lineal en MatLab.

3. Realizar experimentos computacionales que corroboren la eficacia del método frente a problemas de gran tamaño.

1.4 HIPÓTESIS

El algoritmo para la optimización del tiempo de ejecución en la solución de problemas de programación lineal es eficiente para problemas de grandes dimensiones.

CAPÍTULO II

MARCO TEÓRICO

2.1 ANTECEDENTES DE LA INVESTIGACIÓN

En 1984, Karmarkar propuso un algoritmo que resultó ser altamente competitivo frente al simplex para resolver problemas de programación lineal de gran tamaño. Trabajos pioneros en métodos de puntos interiores, anteriores al algoritmo de Karmarkar, son el de Frisch, Fiacco y McCormick, el de Khachiyan y otros. El algoritmo de Karmarkar originó multitudes de trabajos alrededor de su idea original que ha sido mejorado en muchos aspectos.

· Aquino (2005). "*Gráficas con Programación lineal*". Universidad de Veracruz. México. En este trabajo de tesis se describen los conceptos básicos de programación lineal, Teoría de grafos y grafos dirigidos, el título de la tesis a primera parte donde dice gráficas se refiere a los grafos.

· Carrasco (2002) "*Dual convergence for penalty proximal point algorithms in convex programming*" Universidad de los Andes. Chile: Consideramos un método iterativo implícito en la programación convexa que combina variantes del algoritmo del punto proximal, con funciones de penalización paramétricas. Se investiga una Multiplicadora que se calcula explícitamente

en términos de la secuencia primitiva generada por el Método iterativo, proporcionando algunas condiciones sobre los parámetros para asegurar la convergencia Hacia una solución dual óptima particular.

· Wesner (2015). "*Técnicas de programación lineal entera para la optimización de la recolección de residuos reciclables en el Municipio de Morón*" Universidad de Buenos Aires. Argentina: Una posible hipótesis para explicar los resultados no tan alentadores obtenidos en la rezonificación puede ser que la zonificación inicial proporcionada por la municipalidad es razonablemente buena, teniendo en cuenta los ajustes que realizaron sobre la forma de los cuadros durante los primeros meses del proyecto de recolección de residuos reciclables. Como observación final, nos gustaría comentar que actualmente la municipalidad de Morón se encuentra usando los recorridos que les proporcionamos para realizar la recolección.

· Gamboa (2016). "*Minimización de funciones convexas sobre la envoltura convexa de un conjunto finito de puntos usando el método de puntos interiores*". Universidad nacional de Trujillo. Perú: En el presente trabajo sobre la minimización de funciones convexas sobre la envolvente convexa de un conjunto finito de puntos usando un algoritmo de puntos interiores, se obtienen las siguientes conclusiones: El algoritmo presentado resulta conveniente para problemas de optimización convexos, llegando a una buena aproximación. El algoritmo puede usarse también para hallar la proyección de un punto de R^n sobre la envolvente convexa de un conjunto finito de puntos. El método de puntos interiores se puede usar en un problema de optimización donde la función objetivo no necesariamente es

cuadrática pues se tiene bien definida la taza de convergencia lineal aún para problemas no-cuadráticos convexos.

- **EL PROBLEMA DE PROGRAMACIÓN LINEAL**

El problema de programación lineal en la forma estándar corresponde al siguiente modelo matemático.

$$\begin{aligned} &\text{minimizar } c^t x \\ &Ax = b \qquad (2.1) \\ &x \geq 0 \end{aligned}$$

donde $A \in \mathbb{R}^{m \times n}$ es de rango m, $b \in \mathbb{R}^m$ y $c \in \mathbb{R}^n$. La expresión $c^t x$ se llama función objetivo, Ax es el producto de la matriz A con el vector x, $Ax=b$ y $x \geq 0$ si y sólo si $x_i \geq 0$, $i=1,...n$, se denominan las restricciones y las condiciones de no negatividad respectivamente. Los problemas de programación lineal se estudian normalmente en esta forma. Típicamente, $n>m$. En resumen un problema de programación lineal se dice que está en forma estándar si y sólo si.

- ✓ Es de minimización.
- ✓ Sólo incluye restricciones de igualdad.
- ✓ El vector b es no negativo.
- ✓ Las variables x son no negativas.

Existen otras formas de escribir el problema de programación lineal, por ejemplo, la forma canónica.

$$\begin{aligned} &\text{Minimizar } c^t x \\ &Ax \geq b \qquad (2.2) \\ &x \geq 0 \end{aligned}$$

Es posible transformar el problema (2.2) en uno de la forma (2.1) mediante la introducción de un vector cuyas componentes son denominados variables auxiliares, del siguiente modo:

$$\begin{aligned} &\text{Minimizar } c^t x \\ &Ax - s = b \\ &x, s \geq 0 \end{aligned} \tag{2.3}$$

donde $s \in \mathbb{R}^m$, tal que $s = Ax-b$, estas variables auxiliares se denominan variables de exceso.

Debemos resaltar aquí que con la introducción de variables auxiliares, el problema original dado en (2.2) es transformado en otro problema de mayor dimensión, pues fueron añadidas *m* nuevas variables, que corresponden a las componentes del vector *s*. En estas condiciones, resolver (2.3) no es lo mismo que resolver (2.2), pero si conseguimos resolver y encontrar una solución $[x,s]^t$ para el problema en (2.3), entonces *x* será una solución para (2.2).

Por otro lado, para transformar un problema del tipo.

$$\begin{aligned} &\text{Minimizar } c^t x \\ &Ax \leq b \\ &x \geq 0 \end{aligned} \tag{2.4}$$

a la forma estándar, podemos introducir un vector auxiliar de variables de holgura, del siguiente modo:

$$\begin{aligned} &\text{Minimizar } c^t x \\ &Ax + h = b \\ &x, h \geq 0 \end{aligned} \tag{2.5}$$

Finalmente, debemos aclarar que la función objetivo en el problema (2.5) es diferente a la función objetivo del problema original, debido a que ahora es de la forma $c^t x + 0h$. Análogamente para el problema dado en (2.3).

Cuando formulamos un problema de programación lineal, algunas veces las variables pueden no satisfacer las condiciones de no negatividad, entonces podemos reemplazar por otras variables no negativas mediante las siguientes sustituciones:

- Si x_i es una variable de tipo irrestricta ($x_i \in \mathbb{R}$), entonces podemos cambiarla por $x_i = x_i' - x_i''$, donde $x_i', x_i'' \geq 0$.
- Si $x_j \geq l_j$, entonces hacemos $\tilde{x}_j = x_j - l_j$, claramente ahora $\tilde{x}_j \geq 0$.
- Si $x_j \leq u_j$, entonces hacemos $\hat{x}_j = u_j - x_j$, observe que $\hat{x}_j \geq 0$.

En lo sucesivo y por defecto, asumiremos también que siempre trabajamos con un problema lineal en la forma estándar, salvo se aclare lo contrario.

Conjunto convexo: Un conjunto K es convexo, si la combinación convexa de dos elementos cualesquiera del conjunto está en K. Es decir, si dados $x_1, x_2 \in K$, para todo $\lambda \in [0,1]$, se verifica que

$$[x_1, x_2] = \lambda x_1 + (1 - \lambda) x_2 \subset K$$

Región Factible: Para el problema de programación lineal en la forma estándar, la región factible es el conjunto $P = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$. La definición es análoga para problemas en otros formatos. En programación lineal, la región factible es un poliedro convexo

Solución Factible: Es un punto situado en la región factible. Es decir, x es solución factible si $x \in P$.

Poliedro: Región definida por la intersección de un conjunto finito de semiespacios. Un poliedro es un conjunto convexo.

Politopo: Poliedro no vacío y acotado

Hiperplano: Un hiperplano en $\mathbb{R}^n$ es un conjunto de la forma

$$h=\{x \in \mathbb{R}^n : px = k\}$$

donde $p \in \mathbb{R}^n$, $p \neq 0$, es un vector columna y k es una constante escalar. Un hiperplano es la extensión de lo que significa una recta en $\mathbb{R}^2$ o un plano en $\mathbb{R}^3$. Debemos notar que el vector p es normal al hiperplano h

Restricción Activa: Dada la desigualdad $p^t x \geq k$, donde $p \in \mathbb{R}^n$ y k es un escalar. Si $x' \in \mathbb{R}^n$ es tal que satisface $p^t x' = k$, entonces se dice que x' hace activa la desigualdad $p^t x \geq k$. En programación lineal, estas desigualdades constituyen las restricciones, en esas condiciones, se dice entonces que x' hace activa dicha restricción. Para el caso de una restricción de igualdad $p^t x = k$, es claro que cualquier x que satisfaga dicha restricción hará activa ésta.

Semiespacio: Un hiperplano divide el espacio $\mathbb{R}^n$ en dos subregiones, llamadas semiespacios. Así, un semiespacio es el conjunto

$$h_1=\{x \in \mathbb{R}^n : p^t x \geq k\}$$

donde p es un vector columna diferente de cero y k es un escalar. El otro semiespacio es el conjunto de puntos

$$h_2=\{x \in \mathbb{R}^n : p^t x \leq k\}$$

donde $h_1 \cup h_2 = \mathbb{R}^n$

Punto extremo: Consideremos una región factible en programación lineal. Un punto extremo $\hat{x} \in \mathbb{R}^n$ es la intersección de *n* hiperplanos linealmente independientes. Es decir, $\hat{x}$ es un punto extremo si hace activa *n* restricciones linealmente independientes. Si $\hat{x}$ fuera también factible, entonces se dice que es un punto extremo factible.

Punto extremo no degenerado: Un punto extremo no degenerado $\dot{x}$ se dice no degenerado, si exactamente hace activa *n* restricciones. Cuando más de *n* restricciones son activas con respecto a $\dot{x}$, entonces el punto se denomina extremo degenerado.

Solución Básica: Consideremos el problema de programación lineal en la forma estándar definida en (2.1) donde existe *n* variables y *m* restricciones, con $m<n$, además asumiremos inicialmente que el rango de *A* es *m*.

$$\text{Minimizar } z = c^t x \tag{2.6}$$

$$Ax = b \tag{2.7}$$

$$x \geq 0 \tag{2.8}$$

Una solución básica *n* - dimensional se obtiene eligiendo desde *A* una submatriz cuadrada *B* de rango *m*, a esta submatriz cuadrada e inversible se le llama matriz básica. Reordenando las columnas de *A*, si fuera necesario, y también las componentes de *x*, particionamos la matriz *A* y el vector de *x* de modo que se mantenga la compatibilidad:

$$A = [B \quad N] \tag{2.9}$$

$$x = \begin{bmatrix} x_B \\ x_N \end{bmatrix} \tag{2.10}$$

Al vector x_N se le denomina vector no básico y a sus *n-m* componentes se les llama variables no básicas. Reemplazando (2.9) y (2.10) en (2.7) y despejando x_B en función de x_N, desde

Obtenemos

$$[B \quad N] \begin{bmatrix} x_B \\ x_N \end{bmatrix} = b$$

$$\begin{aligned} Bx_B + Nx_N &= b \\ B^{-1}Bx_B + B^{-1}Nx_N &= B^{-1}b \\ x_B + B^{-1}Nx_N &= B^{-1}b \end{aligned}$$

$$x_B = B^{-1}b - B^{-1}Nx_N \tag{2.11}$$

Obsérvese que si damos valores arbitrarios a las variables del vector x_N, y evaluamos x_B según (2.11), el vector $x = [x_B, x_N]^t$ satisface automáticamente

$$x = \begin{bmatrix} x_B \\ x_N \end{bmatrix} = \begin{bmatrix} x_B \\ 0 \end{bmatrix} = \begin{bmatrix} B^{-1}b \\ 0 \end{bmatrix}$$

(2.7), pero aún no satisface (2.8). Si hacemos $x_N = 0$ y calculamos nuevamente x_B según (2.11), el vector

$$x = \begin{bmatrix} x_B \\ x_N \end{bmatrix} = \begin{bmatrix} x_B \\ 0 \end{bmatrix} = \begin{bmatrix} B^{-1}b \\ 0 \end{bmatrix}$$

Además será un punto extremo, pues *n* restricciones linealmente independientes son activas en *x*. Esto es, *n-m* restricciones activas a causa

de x_N=0 y m restricciones activas a causa de Ax=b. Este último vector x, calculando de ese modo, se denomina solución básica. Es decir, en programación lineal, una solución básica es un punto extremo.

Solución Básica Factible (SBF): Haciendo x_N = 0 y calculando x_B según (2.11), si x_B=$B^{-1}b$≥0, entonces la solución básica

$$x = \begin{bmatrix} x_B \\ x_N \end{bmatrix} = \begin{bmatrix} B^{-1}b \\ 0 \end{bmatrix}$$

será además factible, pues satisface (2.7) pero ahora también (2.8). En este caso, la solución básica se denomina solución básica factible. En programación lineal, una solución básica factible es un extremo factible.

Solución Básica Factible No degenerada: Es una solución básica factible donde x_B>0. Observe que una solución básica factible no degenerada hace exactamente n restricciones activas, por lo que es un punto extremo no degenerado. Cuando una o más variables básicas toman el valor de cero, la solución básica factible se denomina degenerada. Note que una solución básica factible degenerada hace más de n restricciones activas, lo que significa que es un punto extremo degenerado.

Dirección Extrema: Una dirección extrema de un conjunto convexo es una dirección factible, la cual no puede ser representada como una combinación lineal positiva de dos direcciones factibles distintas (linealmente independientes). Por lo tanto, en programación lineal podemos decir que el conjunto de direcciones extremas generan todas las direcciones factibles.

Dirección de Decrecimiento: Dado x factible y $d \in \mathbb{R}^n$, $d \neq 0$, d es una dirección de decrecimiento partiendo de x, si existe $\varepsilon > 0$ tal que, para todo $\alpha \in ‹ 0,\varepsilon]$

$$c^t(x+\alpha d) < c^t x$$

Para el caso de minimización, una dirección es de decrecimiento si, y sólo si, $c^t d<0$. Obviamente, si para un x factible no existe direcciones de decrecimiento d, tal que el punto $x+\alpha d$ sea factible, entonces x es una solución óptima.

Claramente, podemos verificar que el vector $d=-\nabla(c^t x)=-c\neq 0$, obtenido de la función objetivo, es una dirección de decrecimiento. Esto se debe a que.

$c^t d= c^t(-\mathbf{c})=- \|c\|^2<0$

El vector $-c$ no sólo es un vector de decrecimiento, sino que es el vector que proporciona el máximo decrecimiento. Es decir, si consideramos otra dirección de decrecimiento $\bar{d}$, tal que $\|-c\| = \|\bar{d}\|$, entonces partiendo de un punto factible x y desplazándonos en las direcciones $-c$ y, $\bar{d}$ el punto $x-\alpha d$ es mejor que el punto $x+\alpha\bar{d}$, en otras palabras

$$c^t(x - \alpha d) \leq c^t(x+\alpha \bar{d})$$

Función Convexa: Si K conjunto convexo no vacío y $f : K \subset \mathbb{R}^n \rightarrow \mathbb{R}$ es una función, se dice que f es convexa si:

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2)$$

2.2 EL MÉTODO SIMPLEX

Ticona (2003). Dada un problema de Programación lineal en su forma estándar

$$\text{Minimizar } z = c^t x$$

$$\begin{aligned} Ax &= b \\ x &\geq 0 \end{aligned} \qquad (2.12)$$

$$A \in \mathbb{R}^{m\times n} \text{ de rango } m,\ (m < n),\ b \in \mathbb{R}^m \text{ y } c, x \in \mathbb{R}^n$$

Supongamos que tenemos una base *B* y una matriz *N*, ambas asociadas a la matriz *A*. Además las variables básicas y no básicas x_B y x_N, respectivamente.

$$x = \begin{bmatrix} x_B \\ x_N \end{bmatrix},\ A = [B \quad N] \text{ y } c = \begin{bmatrix} c_B \\ c_N \end{bmatrix}, \text{ esto en } (2.12) \text{ obtenemos}$$

$$\begin{aligned} x_B &= B^{-1}b - B^{-1}Nx_N \\ &= B^{-1}b - \sum_{j\in R} B^{-1}a_j x_j \qquad (2.13) \\ &= \bar{b} - \sum_{j\in R} y_j x_j \end{aligned}$$

donde *R* es el conjunto de los índices de las variables no básicas, donde a_j representa las *j* - columna de *A*. Además $y_j = B^{-1}a_j$ y $\bar{d} = B^{-1}b$, si hacemos $x_N = 0$, el punto *x* se convierte en la solución básica con valor objetivo

$$z_0 = [c_B^t \quad c_N^t] \begin{bmatrix} B^{-1}b \\ 0 \end{bmatrix} = c_B^t B^{-1} b$$

Usando (2.13), la función objetivo en (2.12) puede ser expresado por

$$
\begin{aligned}
z &= c^t x \\
&= c_B^t x_B + c_N^t x_N \\
&= c_B^t \left(B^{-1} b - \sum_{j \in R} B^{-1} a_j x_j \right) + \sum_{j \in R} c_j x_j \\
&= c_B^t B^{-1} b - \sum_{j \in R} c_B^t B^{-1} a_j x_j + \sum_{j \in R} c_j x_j \qquad (2.14) \\
&= c_B^t B^{-1} b - \left(\sum_{j \in R} c_B^t B^{-1} a_j - \sum_{j \in R} c_j \right) x_j \\
&= c_B^t B^{-1} b - \sum_{j \in R} \left(c_B^t B^{-1} a_j - c_j \right) x_j \\
&= z_0 - \sum_{j \in R} \left(z_j - c_j \right) x_j
\end{aligned}
$$

Usando (2.13) y (2.14), el problema (2.12) puede ahora ser visto como

$$\text{Minimizar } z = z_0 - \sum_{j \in R} \left(z_j - c_j \right) x_j$$

$$
\begin{aligned}
x_B + \sum_{j \in R} y_j x_j &= \bar{b} \qquad (2.15) \\
x_j &\geq 0, \; j \in R, \; x_B \geq 0
\end{aligned}
$$

de (2.15). los coeficientes $z_j - c_j$ de las variables no básicas x_j, $j \in R$. se denominan costos reducidos.

2.2.1 Criterio de Optimalidad

El método simplex reconoce que la actual solución básica factible es óptima, cuando todos los costos reducidos a las variables no básicos son menores o iguales a cero, es decir. $z_j - c_j \leq 0$.

2.2.2 Criterio de la razón

Si existe $k \in R$ tal que $z_k - c_k > 0$, incrementando el valor de la variable no básica x_k obtendremos una reducción en la función objetivo, que es justamente lo que deseamos en un problema de minimización. Pero, esto puede ocasionar que alguna de las variables básicas se torne negativa (infactible). Cuando la actual solución básica factible no es óptima, el método simplex sólo modifica una variable no básica en cada paso. Para eso, calcula el costo reducido $z_k - c_k = \max \{ z_j - c : j \in R\} > 0$, elige la variable x_k para incrementarla desde su nivel cero y hace $x_j = 0$, $\forall_j \in R - \{k\}$. Ahora, seleccionada x_k y haciendo $x_j=0$, la función objetivo y las restricciones de (2.15) pueden ser vistas del siguiente modo:

$$z = z_0 - (z_k - c_k)x_k \tag{2.16}$$

$$\begin{bmatrix} x_{B_1} \\ x_{B_2} \\ \vdots \\ x_{B_r} \\ \vdots \\ x_{B_m} \end{bmatrix} = \begin{bmatrix} \bar{b}_1 \\ \bar{b}_2 \\ \vdots \\ \bar{b}_r \\ \vdots \\ \bar{b}_m \end{bmatrix} - \begin{bmatrix} y_{1,k} \\ y_{2,k} \\ \vdots \\ y_{r,k} \\ \vdots \\ y_{m,k} \end{bmatrix} x_k \tag{2.17}$$

para evitar que alguna variable básica se torne negativa, el valor de x_k debe ser elegido mediante el criterio de la razón

$$x_k = \frac{\bar{b}_r}{y_{r,k}} = \min \left\{ \frac{\bar{b}_i}{y_{i,k}} : y_{i,k} > 0, \qquad i = 1, ..., m \right\}$$

2.2.3 Finalización

El simplex finaliza reconociendo que:

- *Una única solución óptima*: si $z_j - c_j < 0$
- *Existen soluciones óptimas alternativas*: Si $z_j - c_j \leq 0$, $\forall j \in$ R, pero alguna variable no básica, digamos x_q, $q \in R$, tiene su correspondiente costo reducido $z_q - c_q = 0$.
- *El problema tiene solución óptima ilimitada*: Dada una solución básica factible, si $z_k - c_k > 0$ y $y_k \leq 0$, para alguna variable no básica x_k, entonces la solución óptima, y por consecuencia, el valor objetivo óptimo, serán ilimitados.

2.2.4 El algoritmo Simplex

Elegir una matriz básica B tal que se$x = \begin{bmatrix} B^{-1}b \\ 0 \end{bmatrix}$ón básica factible.

Paso 1: Calcular $x_B = B^{-1}b = \bar{b}$. hacer $x_N = 0$ y calcular $z = c_B^t x_B$.

Paso 2: Calcular $w = c_B^t B^{-1}$ y hallar $z_k - c_k = \max_{j \in R} \{z_i - c_j\}$, don $z_j - c_j = wa_j - c_j$

1. Si $z_k - c_k \leq 0$, detenerse, la SBF óptima es $x = \begin{bmatrix} B^{-1}b \\ 0 \end{bmatrix}$
2. Caso contrario, ir al paso 3.

Paso 3: Calcular $y_k = B^{-1}a_k$

1. Si $y_k \leq 0$, detenerse. Existe soluciones óptimas ilimitadas
2. Caso contrario, ir al paso 4.

Paso 4: Calcular

$$x_k = \frac{\bar{b}_r}{y_{r,k}} = \min\left\{\frac{\bar{b}_i}{y_{i,k}} : y_{i,k} > 0, \quad i = 1, ..., m\right\}$$

1. Actualizar la base B. cambiando aB_r por a_k.
2. Actualizar R. cambiando k por el índice B_r.
3. Volver al paso 1.

2.3 TEOREMAS DE CONVERGENCIA

Teorema 2.1 *La región factible* $P = \{ x \in \mathbb{R}^n : Ax = b, x \geq 0\}$, *de soluciones de un problema de programación lineal, es un conjunto convexo*

Prueba. Sea P la región factible del problema (2.1), sea $x_1, x_2 \in P$ arbitrarios, definamos $x = \lambda x_1 + (1 - \lambda)x_2$ con $\lambda \in [0, 1]$, queremos probar que $x \in P$ y $x \geq 0$.

i. Como $Ax_1 = b$ y $x_1 \geq 0$, $Ax_2 = b$ y $x_2 \geq 0$, entonces.

$Ax = A(\lambda x_1 + (1 - \lambda) x_2) = \lambda Ax_1 + (1 - \lambda)Ax_2 = \lambda b + (1 - \lambda)b = b$

ii. Ahora, $x_1 \geq 0$, $x_2 \geq 0$. El hecho de que $\lambda \in [0,1]$ implica que $\lambda \geq 0$ y que $(1-\lambda) \geq 0$, de aquí se tiene que $\lambda x_1 \geq 0$, $(1-\lambda)x_2 \geq 0$, por lo que $x = \lambda x_1 + (1-\lambda) x_2 \geq 0$. Por lo tanto P es un conjunto convexo.

Definición 2.1 *Sea* $P \subset R^n$ *un conjunto convexo y* $f : R^n \rightarrow R$ *una función convexa. La siguiente formulación matemática dada por.*

$$\text{Minimizar } f(x)$$
$$x \in P \qquad (2.18)$$

*se denomina problema de programación convexa***.**

El problema de programación convexa tiene importantes propiedades que lo distingue de problemas más generales, uno de los resultados más interesantes está dado en el siguiente teorema.

TEOREMA 2.2 *En un problema de programación convexa.*

a) Todo minimizador local es minimizador global.

b) El conjunto de minimizadores es convexo.

c) Si f es estrictamente convexa, no puede haber más de un minimizador.

Teorema 2.3 *El problema de programación lineal en la forma estándar es un problema de programación convexa*

Consideremos ahora el problema de programación lineal en la forma estándar

$$\text{Minimizar } c^t x$$

$$x \in P \tag{2.19}$$

donde $P = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$, $b \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times n}$ es de rango m, con $m < n$.

Teorema 2.4 *Un extremo factible x de P es una solución básica factible para (2.19). Inversamente, si x es una solución básica factible de (2.19), entonces x es un punto extremo factible de P.*

Prueba. Como *x* es un punto extremo de *P*, existe *n* hiperplanos linealmente independientes pasando por *x*. Ahora, como *x* es factible, satisface *Ax* = *b*, de donde ya existen *m* restricciones linealmente independientes y activas en *x*, pues hemos asumido que el rango de la matriz *A* es *m*.

Naturalmente, los restantes *n* - *m* hiperplanos linealmente independientes pasando por *x* deben ser del tipo $x_i = 0$, donde x_i es una componente del vector *x*. Denotemos tales componentes x_i como parte del vector x_N, y las

columnas *i* de *A* asociadas a cada x_i por la matriz *N*. Mientras que las demás componentes x_j de *x* la hacemos parte del vector x_B, y las respectivas columnas *j* de *A* por la matriz *B*. Reordenando, si fuera necesario, de modo que no se altere el problema (2.19), particionamos *x* de *A* de modo tal que:

$$\begin{aligned} x &= \begin{bmatrix} x_B \\ x_N \end{bmatrix} \\ A &= [B\ N] \end{aligned}$$

Por hipótesis, el vector *x* pertenece a *P* y satisface $Ax = b$ y $x \geq 0$. Es decir, satisface el sistema de $n \times n$:

$$Bx_B + Nx_N = b \tag{2.20}$$

$$Ix_N = 0$$

donde $I \in \mathbb{R}^{(n\ -\ m)\times(n\ -\ m)}$ es la matriz identidad. El sistema (2.20) implica que $x_N = 0$ y $B_{m\times m}$ inversible, debido a la independencia lineal de las ecuaciones. Luego, $x_B = B^{-1}b$. Como *x* es factible, se tiene que $B^{-1}b > 0$, esto significa que

$$x = \begin{bmatrix} B^{-1}b \\ 0 \end{bmatrix}$$

es una solución básica factible.

Inversamente, si *x* es una solución básica factible, considerando (2.20), *x* satisface $Ax = b$, es decir, hace *m* restricciones linealmente independientes activas, pues B^{-1} existe. Además, $n - m$ restricciones linealmente

independientes son activas debido a $x_N = 0$, lo cual nos dice que x, definido de ese modo, hace n restricciones linealmente independientes activas.

Podemos concluir con respecto al problema dado en (2.19) que: La colección de puntos extremos factibles corresponde a la colección de soluciones básicas factibles. Ambos son no vacíos desde que la región factible sea no vacía.

Para resolver el problema de programación lineal en la forma estándar, basta analizar los puntos extremos del poliedro de factibilidad, pues si una solución óptima existe, entonces existirá un extremo óptimo. Además, sólo tenemos que verificar que tal punto extremo es un minimizador local.

2.4 DUALIDAD Y CONDICIONES DE OPTIMALIDAD

Por lo general, la dualidad y las condiciones de optimalidad son usadas para la implementación del algoritmo de métodos que son variaciones del Simplex, el algoritmo para la optimización del tiempo de ejecución en la solución de problemas de programación lineal

2.4.1 Dualidad

Dado el problema de programación lineal en la forma estándar

$$\begin{aligned} &\text{Minimizar } c^t x \\ &Ax = b \\ &x \geq 0 \end{aligned} \tag{2.21}$$

Definiendo el dual de un problema de programación lineal en la forma estándar.

$$\begin{aligned} &\text{Maximizar } b^t y \\ &A^t y + z = c \\ &z \geq 0 \end{aligned} \tag{2.22}$$

Dado un problema de programación lineal, denominado problema primal, existe otro problema de programación lineal, denominado

problema dual, íntimamente relacionado con él. Se dice que ambos problemas son mutuamente duales.

Cuadro 1. Relación variable-restricción

Problema de Minimización		Problema de Maximización
$[coeficientes]$	$\Leftrightarrow$	$[coeficientes]^t$
número de variables	$\Leftrightarrow$	*número de restricciones*
número de restricciones	$\Leftrightarrow$	*número de variables*
Variable		**Restricción**
≥ 0	$\Leftrightarrow$	$\leq$
≤ 0	$\Leftrightarrow$	$\geq$
Irrestricta	$\Leftrightarrow$	$=$
Restricción		**Variable**
$\geq$	$\Leftrightarrow$	≥ 0
$\leq$	$\Leftrightarrow$	≤ 0
$=$	$\Leftrightarrow$	Irrestricta

Fuente: Zhang, A quick introducction to linear programming, 2007

2.4.2 Propiedades

Lema 2.1 *Si x, y son soluciones factibles del problema de minimización y maximización, respectivamente, entonces* $b^t y \leq c^t x$

Según Zhang (2007).

Prueba. Consideremos la forma canónica de dualidad, sean

$\bar{x}$ e $\bar{y}$ soluciones factibles del problema de minimización y maximización, respectivamente. Veamos que las desigualdades del problema primal así como de su dual.

$$A\bar{x} \geq b, \qquad \bar{x} \geq 0 \tag{2.23}$$

Además

$$A^t\bar{y} \leq c, \qquad \bar{y} \geq 0 \tag{2.24}$$

Multiplicando por $\bar{y} \geq 0$ en (2.23) y por $\bar{x} \geq 0$ en (2.24), obtenemos

$$\bar{y}^t A\bar{x} \geq \bar{y}^t b = b^t\bar{y}$$

y

$$\bar{x}^t A^t\bar{y} \leq \bar{x}^t c = c^t\bar{x}$$

de donde $b^t\bar{y} \leq \bar{y}^t A\bar{x} \leq \bar{x}^t A^t\bar{y} \leq \bar{x}^t c \leq c^t\bar{x}$ ■

2.4.3 Condiciones de Optimalidad de Karush-Kuhn-Tucker

Ya que en este trabajo hemos dado más importancia al problema de programación lineal en su forma estándar. Consideremos el problema de programación lineal en esta forma, al cual denominaremos primal:

$$\begin{aligned} &\text{Minimizar } c^t x \\ &Ax = b \\ &x \geq 0 \end{aligned} \tag{2.25}$$

Por lo que estas condiciones KKT para el problema (2.25) están definidas por la factibilidad del prima y su dual, como también de la complementariedad.

Por lo que las condiciones de optimalidad de Karush-Kuhn-Tucker para el problema de programación lineal en la forma estándar.

Si $x \in R^n$ es una solución óptima del problema de programación lineal estándar si, y sólo si, existen vectores w $\in R^m$ y v $\in R^n$

$$Ax = b, \quad x \geq 0 \qquad \text{(Factibilidad Primal)}$$

$$A^t w + v = c, \quad w \ \ irrestricto, \quad v \geq 0 \qquad \text{(Factibilidad Dual)}$$

$$v^t x = 0 \qquad \text{(Complementariedad)}$$

Estas condiciones de Karush Kunh Tucker representan condiciones necesarias y suficientes para un minimizador global del problema (2.25).

2.5 EL MÉTODO DE NEWTON, LAGRANGE Y PENALIZACIÓN INTERNA

En esta sección describiremos el método de Newton, en el estudio de sistemas de ecuaciones no lineales, el método de Lagrange y Penalización interna, los cuales serán empleados en la construcción del algoritmo.

2.5.1 El método de Newton

Uno de los fundamentales problemas computacionales en ciencia e ingeniería es resolver un sistema de ecuaciones no lineales de múltiples variables.

Consideremos el problema de hallar $x \in R^n$ tal que $F(x) = 0$, donde

F: $R^n \rightarrow R^n$, a saber, hay n variables y n restricciones y asumiremos que no toda las n restricciones son lineales, esto es:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$F(x_1, \ldots x_n) = \begin{pmatrix} f_1(x_1, \ldots, x_n) \\ f_2(x_1, \ldots, x_n) \\ \vdots \\ f_n(x_1, \ldots, x_n) \end{pmatrix}$$

Comenzamos asumiendo que *F* es continuamente diferenciable en $x \in \mathrm{R}^n$. Recordemos de los cursos de cálculo en varias variables que en ese caso cada componente f_i es continuamente diferenciable en *x* y la derivada de *F* en *x* se define como el Jacobiano de *F* en el punto x, denotado por $F'(x) = J(x)$.

$$F'(x) = \nabla F(x)^t = J(x) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix} \in \mathbb{R}^{n \times n}$$

Como el caso unidimensional, el método por elección para resolver el problema presentado arriba es el método de Newton.

El método de Newton consiste en resolver un problema difícil mediante sucesivas resoluciones de un problema fácil, se espera que cada solución del problema fácil sea una mejor aproximación

para el problema difícil. Así, sea $x^{(k)}$ una aproximación inicial a una solución del sistema $F(x) = 0$ (problema difícil), si tomamos una función L_k tal que $L_k(x) \approx F(x)$, para todo x en una vecindad de $x_{(k)}$, entonces esperamos que la solución del sistema $L_k(x) = 0$ (problema fácil) sea una mejor aproximación que $x_{(k)}$para una solución de $F(x) = 0$.

El método de Newton es de carácter iterativo y está basado en la aproximación lineal de la función F en torno al punto actual $x^{(k)}$

$$L_k = F(x^{(k)}) + J(x^{(k)})(x - x^{(k)}) \qquad (2.26)$$

El próximo punto $x^{(k+1)}$, está definida como la solución de

$$L_k(x) = 0 \qquad (2.27)$$

Si $Jx^{(k)}$ es no singular, entonces (2.27) tiene solución única.

$$0 = F(x^{(k)}) + J(x^{(k)})(x^{(k+1)} - x^{(k)}) \qquad (a)$$

despejando $x^{(k+1)}$ se obtiene

$$\begin{aligned} x^{(k+1)} &= x^{(k)} - \frac{1}{J(x^{(k)})} F(x^{(k)}) \\ x^{(k+1)} &= x^{(k)} - J^{-1}(x^{(k)}) F(x^{(k)}) \end{aligned}$$

Esta última expresión se denomina puro Newton. Obtener $x^{(k+1)}$directamente debería ser evitado para fines computacionales, ya que calcular $J^{-1}(x^{(k)})$ es considerado costoso. Despejando de (a) tenemos

$$
\begin{aligned}
J(x^{(k)})\underbrace{(x^{(k+1)} - x^{(k)})}_{d^{(k)}} &= -F(x^{(k)}) \\
x^{(k+1)} - x^{(k)} &= d^{(k)} \\
x^{(k+1)} &= x^{(k)} + d^{(k)}
\end{aligned}
$$

En estas condiciones, una iteración de Newton consiste en calcular $x^{(k+1)}$ previamente conocido $x^{(k)}$, mediante la resolución del sistema lineal:

$$J(x^{(k)})d^{(k)} = -F(x^{(k)}) \tag{2.28}$$

$$x^{(k+1)} = x^{(k)} + d^{(k)} \tag{2.29}$$

Lo que dice (2.29) es que una vez conocido el paso de newton $d^{(k)}$, es posible calcular $x^{(k+1)}$. De un modo más directo, el método de Newton puede ser visto del siguiente modo: Calcular $x^{(k+1)}$ una vez conocido $x^{(k)}$, mediante la siguiente regla:

$$x^{(k+1)} = x^{(k)} - J^{-1}(x^{(k)})F(x^{(k)}) \tag{2.30}$$

En conclusión, el método de Newton es aplicada originalmente para resolver un sistema de ecuaciones $F(x) = 0$, donde $F : R^n \rightarrow R^n$ admite primera derivada continua.

Algoritmo 2.1 (Algoritmo Básico de Newton)

Sea $x^{(0)} \in R^n$ *un punto inicial lo suficientemente cerca de la solución y* $\varepsilon > 0$ *el parámetro de precisión deseado:*

Paso 1: *Si* $\left\|F(x^k)\right\| < \varepsilon$, *detenerse,* $x^{(k)}$ *es la aproximación buscada. Caso contrario, ir al* paso 2.

Paso 2: *Resolver el sistema* $J(x^{(k)})d^{(k)} = -F(x^{(k)})$ *y obtener el paso de newton* $d^{(k)}$. *Ir al* paso 3.

Paso 3: *Calcular* $x^{(k+1)} = x^{(k)} + d^{(k)}$, *hacer* $k \leftarrow k+1$, *volver al* paso 1

2.5.2 El método de Lagrange

Este método transforma un problema de optimización con restricciones de igualdad, en un problema irrestricto. Así, si intentamos resolver el problema.

$$\text{Minimizar } f(x)$$

$$\text{Sujeto a:}$$

$$g_i(x) = 0, \quad i = 1, ..., m$$

He aquí tenemos un problema de optimización con restricciones de igualdad, formando la función Lagrangiana tenemos.

$$L(x, y) = f(x) - \sum_{i=1}^{m} y_i g_i(x)$$

y entonces teniendo esta función irrestricta nos toca minimizar la función *L*(*x*, *y*), para ello podemos utilizar el método de Newton expuesto anteriormente, esto se concreta en resolver el sistema de *n*+*m* restricciones con *n*+*m* variables, es decir.

$$\frac{\partial L}{\partial x_j} = \frac{\partial f}{\partial x_j}(x) - \sum_{i=1}^{m} y_i \frac{\partial g_i}{\partial x_j}(x) = 0 \qquad j = 1, ..., n$$

$$\frac{\partial L}{\partial y_i} = -g(x) = 0 \qquad i = 1, ..., m$$

Este sistema de ecuaciones constituye las condiciones de Karush Kuhn Tucker para el problema de minimización con restricciones de igualdad, las cuales son necesarias para un minimizador local. Pero cuando este problema es un problema de programación convexa, es decir, que tanto la función objetivo como el conjunto definido por las restricciones son convexa, entonces las condiciones de Karush Kuhn Tucker son además suficientes y un minimizador local es global.

2.5.3 El método de Penalización Interna

El método fue propuesto originalmente para tratar problemas con restricciones de desigualdad, donde la región factible tiene interior no vacío.

En general, las restricciones de desigualdad pueden ser convertidas en ecuaciones por la adición de variables no negativas de holgura o exceso.

Consideremos el problema de optimización de la siguiente forma.

$$\text{Minimizar } f(x)$$

$$g(x) \geq 0 \tag{2.31}$$

$$x \in X$$

donde $X \subset \mathbb{R}^n$, $g : \mathbb{R}^n \longmapsto \mathbb{R}^m$, $f, g \in C^0(X)$, y

$$\Omega = \{x \in X : g(x) \geq 0\}$$

tiene interior relativo no vacío. El interior de Ω, denotado por Ω°, definido por

$$\Omega^{\circ} = \{x \in X : g(x) > 0\}$$

suponiendo que el problema (2.31) tiene un minimizador global, podemos transformar (2.31) en un problema irrestricto con función objetivo

$$f(x) + \mu B(x) \qquad (2.32)$$

donde $\mu > 0$ es un escalar denominado parámetro de penalización. La función B se denomina función de barrera.

Algoritmo 2.2 *(Algoritmo Básico de Penalización Interna)*

Dados $\mu_1 > 0$, $x^{(0)} \in \Omega^o$, $k = 1$

Paso 1: Calcular $x^{(k)}$, la solución global del subproblema:

$$\text{Minimizar } f(x) + \mu_k B(x)$$

$$\text{Sujeta a } x \in \Omega^{\circ} \qquad (2.33)$$

Paso 2: Elegir u_{k+1} tal que $0 < u_{k+1} < u_k$

Paso 3: Hacer $k = k + 1$ y volver al paso 1.

Estrictamente hablando, en el problema de penalización (2.33) aparecen las restricciones $g_i(x) > 0$ más allá de $x \in X$. No aun así, como la función objetivo de (2.33) tiende al infinito cuando x tiende a la frontera, estamos autorizados a suponer que un algoritmo

irrestricto no sentirá la menor atracción por los puntos muy próximos al contorno, y que por tanto permanecerá también apartado de puntos externos. A veces, puede ser necesaria alguna modificación leve del algoritmo "irrestricto" para garantizar la permanencia en el interior de Ω. Sabemos por otro lado, que encontrar minimizadores globales acostumbra ser muy difícil y que, usando métodos iterativos, no podemos, de hecho alcanzar exactamente la solución de (2.33). Por eso en la práctica, $x^{(k)}$ será apenas una solución aproximada de (2.33).

El parámetro de penalización μ_k debe aproximarse iterativamente hacia cero, un modo clásico consiste en hacer $\mu_1 = 1$ y $\mu_{k+1} = \mu_k /10$ para $k = 1, 2,\ldots$

Note que el algoritmo (2.2) se exige que el parámetro de penalización μ tiende hacia cero, la razón es que bajo esas hipótesis es posible mostrar que el algoritmo posee la propiedad de otorgar un minimizador global para el problema (2.31).

El problema (2.31) se puede transformar en el problema barrera logarítmica

de minimización sin restricciones

$$\text{Minimizar } f(x) - \mu \sum_{i=1}^{m} \log(g_i(x))$$

donde en este caso se ha empleado la función logarítmica para construir la barrera $\mu \in R^+$.

Adicionando -log(x_j) a la función objetivo ocasionará que el objetivo se incremente infinitamente cuando $x_j \rightarrow 0$, luego, si un vector posee la x_j variable cercana a cero, entonces tal vector no podrá ser una solución óptima para el problema a optimizarse. Sin embargo, dado que nada impide que una solución óptima esté justamente sobre la frontera de la región definida por $x \geq 0$, entonces se hace necesario la introducción de un parámetro de penalización μ que equilibre la contribución de la verdadera función objetivo contra el de la función barrera.

El conjunto de minimizadores $x(\mu)$ es llamado trayectoria central. La siguiente figura adjunta muestra el comportamiento de la función logarítmica cuando decrece el parámetro μ. Se puede observar que a medida que este parámetro tiende a cero, la función se acerca cada vez más a los ejes.

La siguiente (figura 1) muestra el comportamiento de la barrera logarítmica en relación a varias restricciones cuando $\mu \rightarrow 0$. A medida que esto ocurre la pared que forma la barrera logarítmica se asemeja cada vez más a las paredes del polítopo, de modo que en el límite, coincidirán.

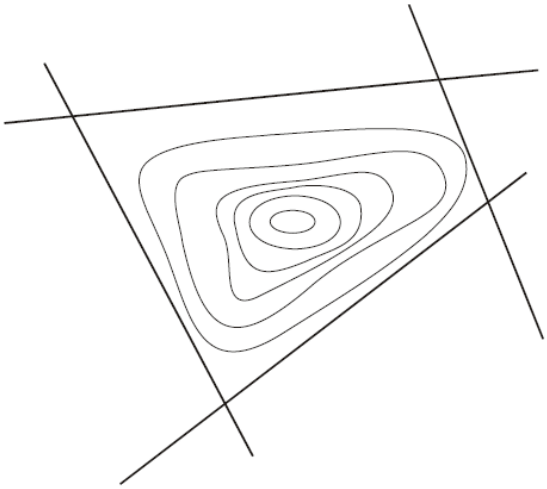

Figura1. Comportamiento de barrera logarítmica

Dado μ, Sea $x^*(\mu)$ un punto que minimiza el problema barrera, bajo hipótesis poco restrictiva se cumple que:

$$\lim_{\mu \longrightarrow 0} x^*(\mu) = x^*$$

donde x^* es un mínimo local del problema original. Este procedimiento permite resolver problemas con restricciones mediante métodos de optimización sin restricciones.

Seguidamente pasaremos a probar las propiedades fundamentales del Algoritmo (2.2)

Lema 2.2 *Sea $(x^{(k)})$ la sucesión generada por el Algoritmo (2.2). Entonces*

a) $f(x^{(k+1)}) + \mu_{k+1} B(x^{(k+1)}) \leq f(x^{(k)}) + \mu_k B(x^{(k)})$

b) $B(x^{(k)}) \leq B(x^{(k+1)})$

c) $f(x^{(k+1)}) \leq f(x^{(k)})$

Prueba. Como la sucesión de los parámetros penalizadores $\{\mu_k\}$ es estrictamente decreciente. por el axioma (2) y debido a que $\{x^{(k)}\}$ es una sucesión de minimizadores globales del problema (2.32), tenemos:

$$f(x^{(k+1)}) + \mu_{k+1}B(x^{(k+1)}) \leq f(x^{(k)}) + \mu_{k+1}B(x^{(k)}) \qquad (2.34)$$

$$\leq f(x^{(k)}) + \mu_k B(x^{(k)})$$

Para mostrar (b) vemos además que

$$f(x^{(k)}) + \mu_k B(x^{(k)}) \leq f(x^{(k+1)}) + \mu_k B(x^{(k+1)}) \qquad (2.35)$$

La afirmación en (2.35) se debe a que $x^{(k)}$ es un minimizador global de (2.33) con respecto al parámetro μ_k. Recordando ahora que, si $a < b$ y $c < d$, entonces $a - d < b - c$, restando (2.35) de (2.34) tenemos

$$\left(\mu_{k+1} - \mu_k\right) B(x^{(k+1)}) \leq \left(\mu_{k+1} - \mu_k\right) B(x^{(k)})$$

como $\mu_{k+1} - \mu_k < 0$, se tiene que $B(x^{(k)}) \leq B(x^{(k+1)})$.

Finalmente, para mostrar (c) usamos el hecho que $x^{(k+1)}$ es minimizador del problema (2.33) con respecto al parámetro μ_{k+1}, además del resultado

obtenido en (b):

$$\begin{aligned} f(x^{(k+1)}) + \mu_{k+1}B(x^{(k+1)}) &\leq f(x^{(k)}) + \mu_{k+1}B(x^{(k)}) \\ &\leq f(x^{(k)}) + \mu_{k+1}B(x^{(k+1)}) \end{aligned}$$

Luego, $f(x^{(k+1)}) \leq f(x^{(k)})$. ■

CAPÍTULO III

METODOLOGÍA

3.1 TIPO Y DISEÑO DE INVESTIGACIÓN

3.1.1 Tipo de investigación

La investigación es de tipo básico y experimental, toda investigación básica se caracteriza porque sus resultados sirven para profundizar los conocimientos del tema de investigación

3.1.2 Metodología

La metodología que se utilizará en este trabajo, fue la investigación del método de Puntos interiores, la implementación de los algoritmos en el lenguaje de programación Matlab.

La técnica que se utilizó en el presente trabajo es la técnica de la lectura analítica y experimental en un lenguaje de programación, y corroborar la eficacia, el tiempo de solución, en la resolución de problemas de pequeña, mediana y gran escala.

Desde su publicación por George B Dantzig en 1947 el método simplex ha demostrado sobradamente ser altamente eficaz para resolver todo tipo de problemas de programación lineal, el hecho de que en determinadas circunstancias su complejidad sea exponencial ha

motivado en los últimos años un elevado número de intentos, tanto teóricos como prácticos, de obtener otros procedimientos con mejor convergencia computacional.

El primero de gran transcendencia teórica es debido a L. G. Khachiyan en 1979. Este autor recopilando una serie de trabajos e ideas de los autores rusos Shor, Yudin y Nemirovskii sobre un método basado en la generación de una sucesión de elipsoides para resolver problemas de programación no lineal, los aplicó e hizo extensivo a programación lineal dando lugar al conocido como método de las elipsoides. Su principal ventaja teórica sobre el método simplex radica en que resuelve problemas de programación lineal en tiempo polinómico; es decir, posee una convergencia computacional teórica polinómica. Las implementaciones prácticas, sin embargo, han demostrado su desventaja respecto al método simplex ya que, por un lado, el número de iteraciones que requiere para llegar al óptimo es muy grande y, por otro, el trabajo que implica cada una de ellas es mucho más costoso que el requerido por el método simplex, pues no se pueden aplicar las técnicas de matrices dispersas tan características de las implementaciones en ordenador de este último. En el año de 1 984, N. K. Karmarkar, de los laboratorios Bell de la compañía AT&T, propuso un nuevo algoritmo de complejidad polinómica para resolver problemas de programación lineal. Lo contrario que el método de las elipsoides, las implementaciones prácticas a que ha dado lugar en los últimos años los presentan como un gran rival del método simplex en cualquier

tipo de problemas y especialmente en los de gran tamaño que surgen de aplicaciones del mundo real.

El método propuesto por Karmarkar es bastante diferente del simplex. Considera el problema lineal dentro de una estructura geométrica de tipo simplicial y una estrategia de movimientos para obtener los distintos puntos del proceso iterativo en el interior del polítopo o poliedro que definen las condiciones. Este polítopo se transforma de tal manera que el punto del proceso iterativo en el que esté en cada procedimiento sea el centro del polítopo o poliedro en el espacio transformado. Este nuevo concepto o enfoque de tratar el problema es decir, llegar al óptimo a través de puntos interiores de la región factible, ha motivado numerosas nuevas ideas dentro del apartado genérico de algoritmos o métodos de puntos interiores, tanto para programación lineal como no lineal.

Método de Punto interior para Programación lineal:

Se denominan métodos de punto interior precisamente porque los puntos generados por estos algoritmos se hallan en el interior de la región factible. Esta es una clara diferencia respecto del método simplex, el cuál avanza por la frontera de dicha región moviéndose de un punto extremo a otro.

En la figura adjunta se muestra la trayectoria seguida para alcanzar el punto óptimo x^* desde el punto inicial x^0.

a) El método del simplex

b) El método de punto interior

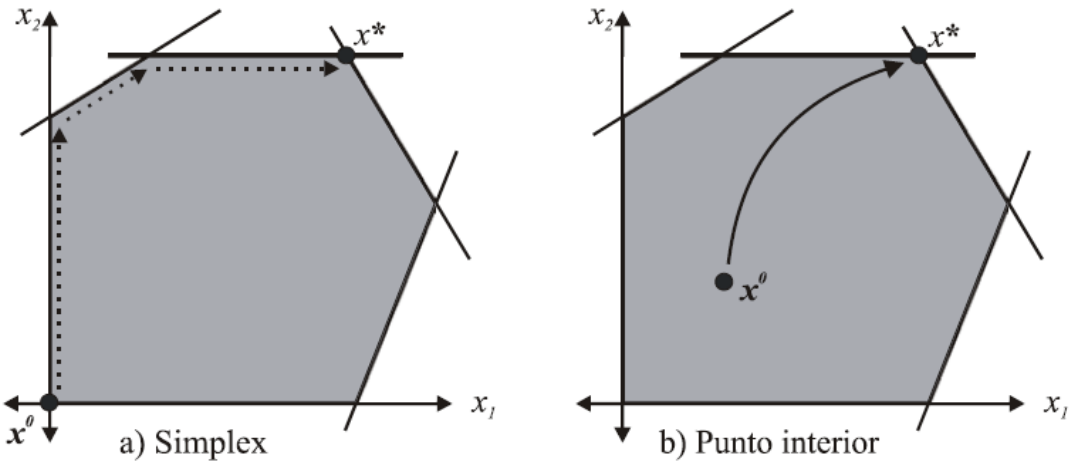

Figura 2. Trayectoria seguida simplex y punto interior

Los procedimientos numéricos de puntos interiores para resolver problemas

de programación lineal basan su estrategia en.

- ✓ Encontrar un punto de partida factible en el interior de la región factible del problema.
- ✓ Definir una dirección de movimiento tal que, conservando la factibilidad del problema, moviéndose a lo largo de ella, se avance hasta un punto en el que se deduzca el valor de la función objetivo.
- ✓ Cuándo pararse. Es decir, cuántas veces se debe realizar la operación descrita en el punto 2 y cómo identificar que se ha alcanzado el óptimo del problema.

La dirección de movimiento *d* calculada en cada iteración del algoritmo no puede ser una dirección cualquiera. Concretamente, *d* debe verificar dos condiciones para ser considerada una buena dirección: Una que preserve la factibilidad del nuevo punto y la otra debe mejorar el valor de

la función objetivo, la dirección de máxima pendiente: en este caso $-c$, por ser lineal la función objetivo.

DESVENTAJAS DEL MÉTODO SIMPLEX

- ✓ Puede realizar un número exponencial de iteraciones, pues se desplaza por los vértices de un poliedro. Para ciertos problemas en espacios de dimensión n, puede tomar alrededor de 2^n iteraciones
- ✓ Implementación computacional sofisticada.
- ✓ Peligro de ciclaje ante degeneración
- ✓ Ineficiencia para problemas de gran tamaño

VENTAJAS Y DESVENTAJAS DEL AOPL

Desventaja

- ✓ No converge a un vértice como lo hace el Simplex

Ventajas

- ✓ Rápido, aproximadamente nL iteraciones
- ✓ Fácil de implementar.

Para la construcción del algoritmo fusionamos tres técnicas numéricas, las cuales son:

El *método de Newton*. Este método numérico es aplicado para resolver sistemas de ecuaciones no lineales y es aplicado en la resolución de problemas de optimización sin la presencia de restricciones.

El *método de Lagrange*. Este método numérico es aplicado frecuentemente para resolver problemas de optimización con restricciones de igualdad.

El *método de Penalización interna*. Este método numérico es aplicado frecuentemente para resolver problemas de optimización con restricciones de desigualdad. Los cuales nos permitirán construir el algoritmo de Puntos interiores Primal dual para programación lineal.

CAPÍTULO IV

RESULTADOS Y DISCUSIÓN

4.1 ALGORITMO PARA LA OPTIMIZACIÓN DEL TIEMPO DE EJECUCIÓN EN LA SOLUCIÓN DE PROBLEMAS DE PROGRAMACIÓN LINEAL

La construcción de este método está soportada por tres técnicas numéricas muy utilizadas en optimización: El método de Newton, Lagrange y penalización interna.

4.1.1 Implementación de las herramientas matemáticas numéricas

a) Newton en el Algoritmo de optimización para PL.

Resumiendo este método más popular para resolver sistemas de ecuaciones no lineales.

Dada una función F_n: $R^v \rightarrow R^v$ y un sistema $F(x) = 0$. El Método de Newton consiste en repetir consecutivamente, para k= 0,1,2,.., hasta que $||F(x^n)|| < e$, donde $e>0$, obedeciendo la siguiente regla:

$$x^{k+1} = x^k - \frac{F_n(x^k)}{F_n'(x^k)}$$

El siguiente programa en Matlab resuelve el sistema $F(x)=0$, si prec representa el parámetro de precisión, $J = F'(x)$ como el Jacobiano del sistema de ecuaciones. y x_0 un punto inicial.

```
function x=newton(x)
iter=0; prec=0.000001;
while norm(Fn(x))>prec
    iter=iter+1;
    x=x-inv(Jn(x))*Fn(x);
    fprintf('iter=%2i; x = [%7.4f%7.4f]\n', iter,x);
    if iter>1000
        error('parece que newton no converge');
    end
end
```

Ejemplo 4.1 Se quiere resolver el sistema de ecuaciones no lineales

$$-x + 2y - 4 = 0$$
$$(x-6)^2 - y + 4 = 0$$

Ejecutando Newton, con los siguientes puntos iniciales $\begin{bmatrix} 1 \\ 2 \end{bmatrix}$ *y* $\begin{bmatrix} 6 \\ 10 \end{bmatrix}$ *tenemos, pero primero definamos el sistema de ecuaciones y el Jacobiano*

$$F(x) = \begin{bmatrix} -x_1 + 2x_2 - 4 \\ (x_1-6)^2 - x_2 + 2 \end{bmatrix};$$

$$F_n'(x^k) = J(x) = \begin{bmatrix} \frac{df}{dx_1} & \frac{df}{dx_2} \\ \frac{dg}{dx_1} & \frac{dg}{dx_2} \end{bmatrix} = \begin{bmatrix} -1 & 2 \\ 2(x_1-6) & -1 \end{bmatrix}$$

Después de ingresar la función y la matríz Jacobiana en el coputador, ejecutamos el programa newton, primero con el punto inicial $\begin{bmatrix} 1 \\ 2 \end{bmatrix}$ *con un parámetro de precisión* 0,000001 *tenemos:*

```
>> x = newton([1  2]')
iter =  1  ;  x  =  [  3.3333    3.6667]
iter =  2  ;  x  =  [ 4.2667    4.1333]
iter =  3  ;  x  =  [ 4.4863    4.2431]
iter =  4  ;  x  =  [  4.4999    4.2500]
iter =  5  ;  x  =  [  4.5000    4.2500]
```

Proyectando en el plano $x_1x_2$podemos observar todo los puntos de iteración, desde el punto inicial, hasta la raíz

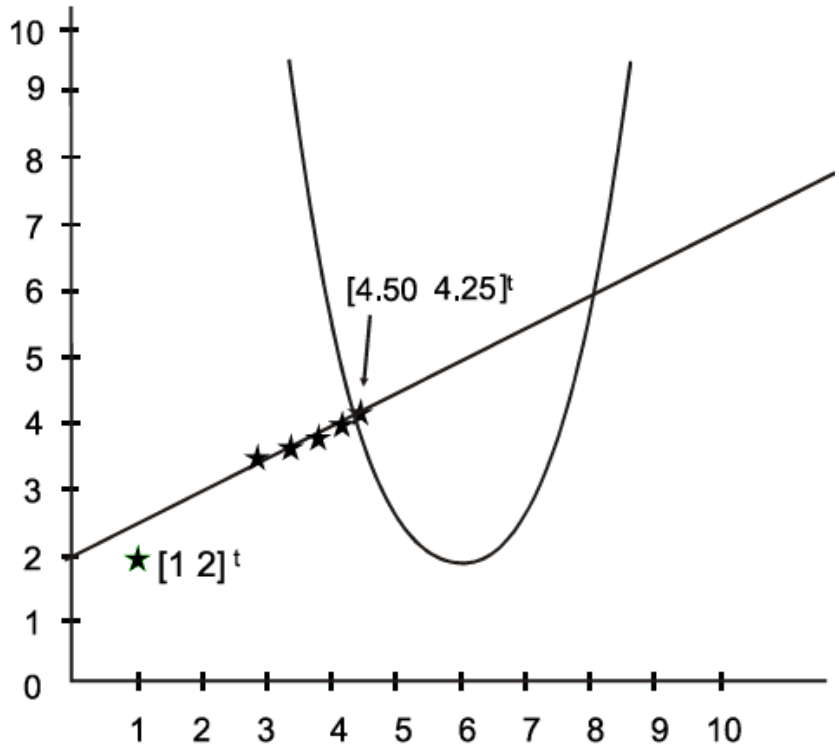

Figura 3. Newton con punto inicialx_0 = [1, 2]^t

Ahora ejecutando newton con el otro punto inicial [9,3]' tenemos

```
>> x = newton([9   3]')
iter =  1  ;  x  =  [8.1818      6.0909]
iter =  2  ;  x  =  [8.0086      6.0043]
iter =  3  ;  x  = [8.0000       6.0000]
iter =  4  ;  x  =  [8.0000      6.0000]
```

que también converge a otra raíz [8,0000 6,0000]' , se observa los puntos de iteración hacia la raíz en el gráfico adjunto.

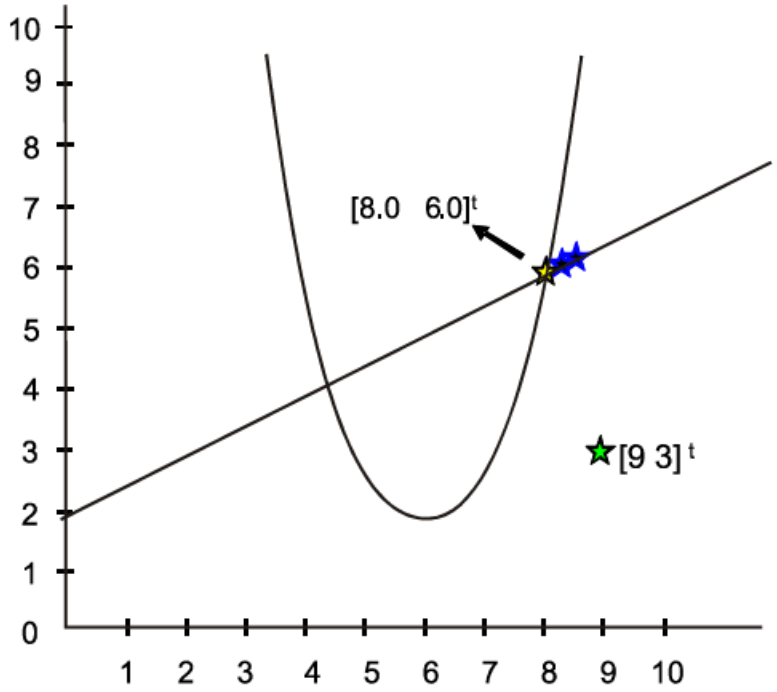

Figura 4: Newton con punto inicial $x_0 = [\,9,\,3]^t$

b) Lagrange en el Algoritmo de optimización para PL.

Para resolver este problema haremos una modificación al programa de newton, e implementaremos en el lenguaje de programación (Matlab) a la que denominamos Lagrange. Teniendo en cuenta que *x* es la variable, y λ que es la variable lagrangiana irrestricta en signo, que lo denotaremos por *y*.

```
function [x,iter]=lagrange(x,y)
n=length(x);m=length(y);iter=0;
    while norm(Fn(x,y))>0.0001
      d=-inv(Jn(x,y))*Fn(x,y);
      dx=d(1:n); dy=d(n+1:n+m);
      a=-1/min(min(dx./x),-1);
      x=x+0.95*a*dx; y=y+0.95*a*dy;
      iter=iter+1;
    end
```

a es el criterio de la razón

Ejemplo 4.2

$$\begin{aligned} &Minimizar && f(x_1, x_2) = x_1 + 2x_2 \\ &s.a && \\ & && g(x_1, x_2) = x_1^2 - 6x_1 + x_2^2 - 8x_2 + 20 = 0 \end{aligned}$$

formando la función Lagrangiana

$$L(x_1, x_2, \lambda) = x_1 + 2x_2 + \lambda\left(x_1^2 - 6x_1 + x_2^2 - 8x_2 + 20\right)$$

Entonces el sistema a resolver es

$$\begin{aligned} \frac{\partial L}{\partial x_1} &= 1 + (2x_1 - 6)\lambda = 0 \\ \frac{\partial L}{\partial x_2} &= 2 + (2x_2 - 8)\lambda = 0 \\ \frac{\partial L}{\partial \lambda} &= x_1^2 - 6x_1 + x_2^2 - 8x_2 + 20 = 0 \end{aligned}$$

donde λ es la variable lagrangiana asociada a la restricción del problema.

Definimos

$$F(x, \lambda) = \begin{bmatrix} 1 + (2x_1 - 6)\lambda \\ 2 + (2x_2 - 8)\lambda \\ x_1^2 - 6x_1 + x_2^2 - 8x_2 + 20 \end{bmatrix}$$

y calculando su respectivo Jacobiano

$$J(x, \lambda) = \begin{bmatrix} 2\lambda & 0 & 2x_1 - 6 \\ 0 & 2\lambda & 2x_2 - 8 \\ 2x_1 - 6 & 2x_2 - 8 & 0 \end{bmatrix}$$

Ejecutando el programa de Lagrange, para un punto inicial de [1 2]' y variable lagrangiana λ = 3

Se Obtuvo.

$$x^{(1)} = \begin{bmatrix} 1.4354 \\ 2.2771 \end{bmatrix}, \quad y^{(1)} = \begin{bmatrix} 1.0406 \end{bmatrix}$$

$$x^{(2)} = \begin{bmatrix} 1.6967 \\ 2.1546 \end{bmatrix}, \quad y^{(2)} = \begin{bmatrix} 0.5294 \end{bmatrix}$$

$$x^{(3)} = \begin{bmatrix} 1.9563 \\ 1.9981 \end{bmatrix}, \quad y^{(3)} = \begin{bmatrix} 0.4964 \end{bmatrix}$$

$$x^{(4)} = \begin{bmatrix} 1.9979 \\ 1.9994 \end{bmatrix}, \quad y^{(4)} = \begin{bmatrix} 0.4997 \end{bmatrix}$$

$$x^{(5)} = \begin{bmatrix} 1.9999 \\ 2.0000 \end{bmatrix}, \quad y^{(5)} = \begin{bmatrix} 0.5000 \end{bmatrix}$$

$$x^{(6)} = \begin{bmatrix} 2.0000 \\ 2.0000 \end{bmatrix}, \quad y^{(6)} = \begin{bmatrix} 0.5000 \end{bmatrix}$$

Después de 6 iteraciones principales ejecutadas por el programa, proyectando $x^{(6)}$ sobre x_1x_2- espacio, podemos ver lo que sucedió en el problema original. Asi como la solución es

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2.0000 \\ 2.0000 \end{bmatrix}$$

Este problema lo podemos visualizar geométricamente según la (figura 5) las restricciones y la función gradiente**.** También se puede ver que el gradiente de la función objetivo y la restricción son linealmente dependientes.

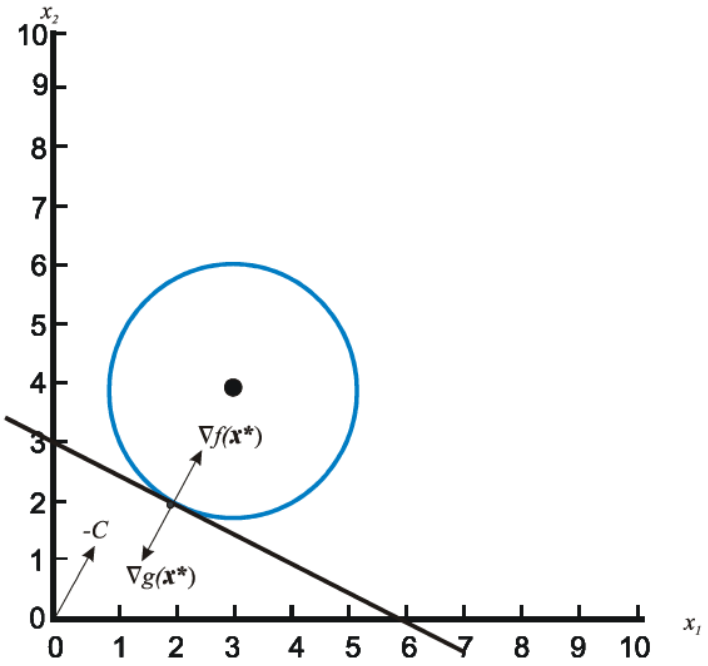

Figura 5. Lagrangiano

c) Penalización Interna en el Algoritmo de optimización para PL.

Con u = μ y *y* = λ la función principal la llamaremos Penalización. El código en MatLab de la función principal es;

```
function [x,iter]=Penalización(x,y)
n=length(x);
m=length(y);
iter=0;
    while norm(Fu(x,y,u))>0.0001
      d=-inv(Jn(x,y,u))*Fn(x,y,u);
      dx=d(1:n);
    dy=d(n+1:n+m);
      a=-1/min(min(dx./x),-1);
      x=x+0.95*a*dx; y=y+0.95*a*dy;
     u = u/10;
      iter=iter+1;
    end
```

Ejemplo 4.3 Considere el siguiente problema

$$Minimizar\ (x_1+2)^2+(x_2-3)^2$$
$$x_1, x_2 \geq 0$$

El minimizador irrestricto debería ser (- 2,3)' pero el minimizador para este problema es (0,3)' como se observa en la (figura 6)

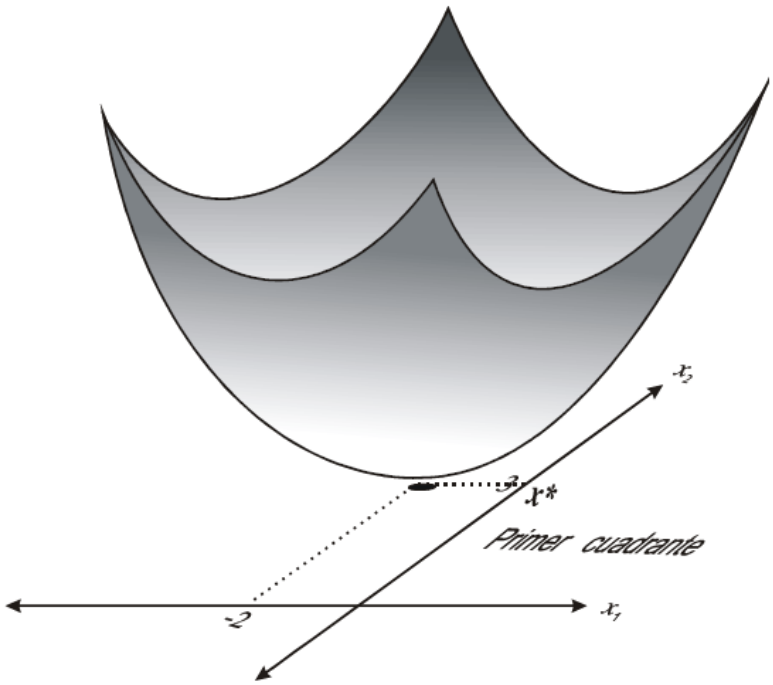

Figura 6. Minimizador irrestricto [-2, 3]^t

aplicamos penalización interna para μ >0 tenemos

$$B(x \mid \mu) = (x_1+2)^2+(x_2-3)^2-\mu\left(\log(x_1)+\log(x_2)\right)$$

derivamos parcialmente con respecto a x_1 y x_2

$$\frac{\partial B}{\partial x_1} = 2(x_1+2)-\frac{\mu}{x_1}=0$$
$$\frac{\partial B}{\partial x_2} = 2(x_2-3)-\frac{\mu}{x_2}=0$$

Resolviendo el sistema tenemos

$$x_1(\mu) = \sqrt{\frac{\mu}{2}+1}-1$$
$$x_2(\mu) = \sqrt{\frac{\mu}{2}+\frac{9}{4}}+\frac{3}{2}$$

Observamos además

$$\lim_{\mu\longrightarrow 0} x_1(\mu) = 0$$
$$\lim_{\mu\longrightarrow 0} x_2(\mu) = 3$$

$$\lim_{\mu\longrightarrow 0}\mu\sum_{j=1}^{n}\log(x_j) = 0$$

Paraμ = 100, tenemos

$$x_1(\mu) = 6{,}1414$$
$$x_2(\mu) = 8{,}7284$$

El cual está cerca del punto óptimo, que es (0,3)' y gráficamente también podemos observar en la (figura 7).

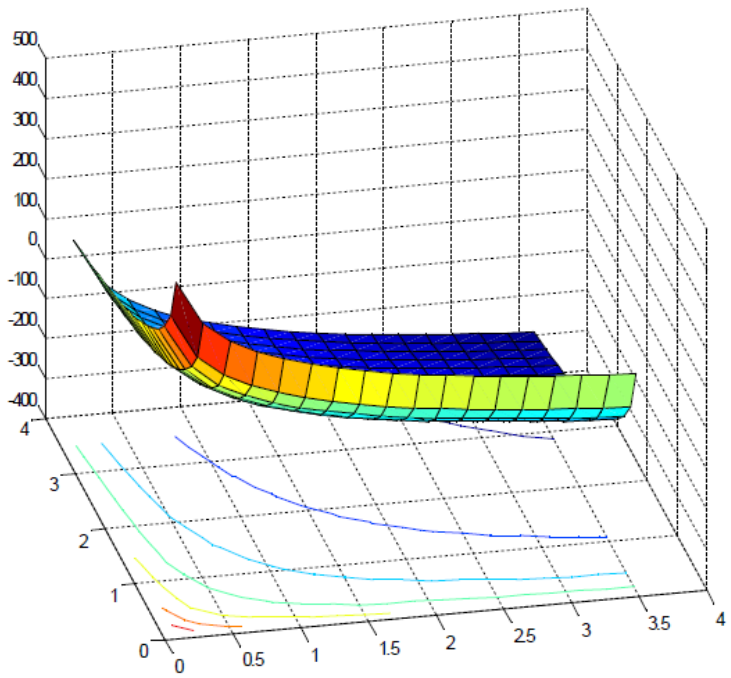

Figura 7. B(*x*| μ) para μ=100

Si disminuimos el parámetro de penalización, para μ = 0.01, entonces tendremos que los nuevos puntos

$$\begin{aligned} x_1(\mu) &= 0{,}0025 \\ x_2(\mu) &= 3{,}0017 \end{aligned}$$

Tal punto se encuentra muy próximo al punto óptimo, y gráficamente también podemos apreciar en la (figura 8).

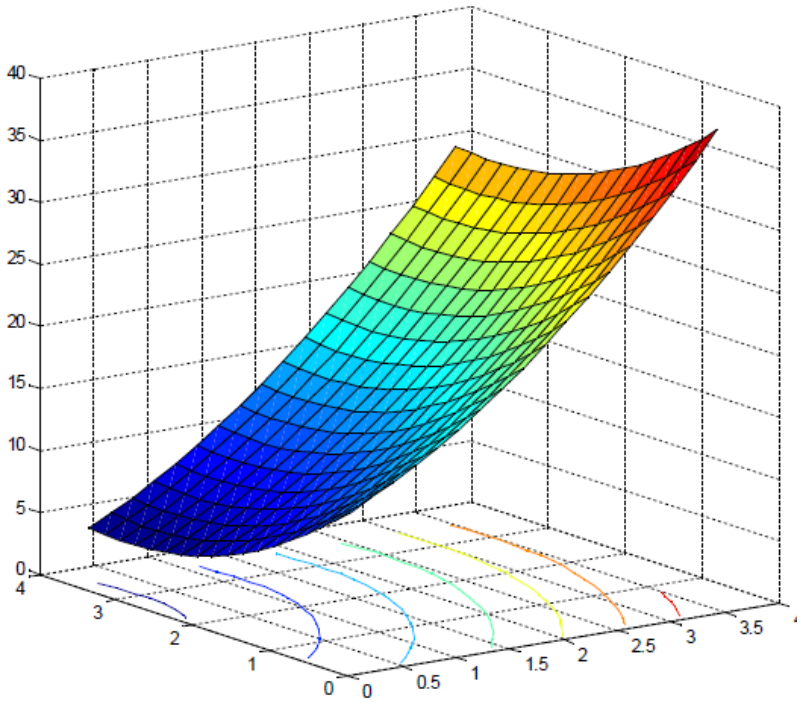

Figura 8: B(x | μ) para μ=0.01

4.1.2 Construcción del algoritmo

Consideremos los problemas Primal y Dual para programación lineal en forma estándar

$$\begin{aligned} &\text{Minimizar} \quad c^t x \\ &\text{Sujeto a:} \\ &\qquad Ax = b \\ &\qquad x \geq 0 \end{aligned} \tag{P}$$

Donde $c, x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ y $A \in \mathbb{R}^{m \times n}$ $(m < n)$

$$\begin{aligned} &\text{Maximizar} \quad b^t y \\ &\text{Sujeto a:} \\ &\qquad A^t y + z = c \\ &\qquad z \geq 0 \end{aligned} \tag{D}$$

Pero debemos tener en cuenta lo siguiente.

H1. El conjunto $P = \{x \in \mathbb{R}^n : Ax = b, \quad x > 0\}$ no vacío

H2. El conjunto $T = \{y \in \mathbb{R}^m, z \in \mathbb{R}^n : A^t y + z = c, \quad z > 0\}$ no vacío

H3. La matriz A es de rango m

En estas condiciones, de acuerdo con el teorema de la dualidad, los problemas (P) y (D) tienen solución óptima, coincidiendo los valores de sus funciones objetivos. Además el conjunto de soluciones óptimas de ambos problemas están acotados. La estrategia de este método resuelve ambos problemas en forma simultánea.

Para *x* factible y (*y, z*), el *duality gap* (intervalo de dualidad)

$$\begin{aligned} duality\ gap &= c^t x - b^t y \\ &= x^t c - x^t A^t y \qquad (Ax = b) \\ &= x^t (c - A^t y) \\ &= x^t z \qquad (c - A^t y = z) \\ &\geq 0 \qquad (x, z \geq 0) \end{aligned}$$

El *duality gap* es cerrado en el óptimo *x**

$$c^t x^* - b^t y^* = (x^*)^t z^* = 0$$

Por lo que las condiciones de complementariedad son:

$$x_j^* z_j^* = 0, \qquad j = 1, 2, ..., n$$

Las condiciones de no negatividad serán manejadas por el método de penalización interna, y las restricciones de igualdad por el algoritmo de Lagrange. Finalmente la función irrestricta resultante por el método de Newton.

Introduciendo primeramente la función Barrera a los (P) y (D)

$$\begin{aligned} \text{Minimizar} \quad & c^t x - \mu \sum_{j=1}^{n} \log(x_j) \\ & Ax - b = 0 \end{aligned} \tag{4.1}$$

$$\begin{aligned} \text{Maximizar} \quad & b^t y + \mu \sum_{j=1}^{n} \log(z_j) \\ & A^t y + z - c = 0 \end{aligned} \tag{4.2}$$

Vale recalcar que las barreras, $-\sum_{j=1}^{n} \log(x_j)$ y $+\sum_{j=1}^{n} \log(z_j)$, no permiten que los puntos interiores de (P) y (D) se aproximen a la frontera de la región factible. Las barreras al estar presentes en la función objetivo del problema de minimización aumentan la función objetivo infinitamente a medida que el punto *x* se acerca a la frontera. Pero en programación lineal el óptimo *x** está en un extremo, y aparentemente la barrera nos impediría de llegar a *x**. Para contrarrestar este efecto usamos un parámetro de penalización equilibrante $\mu > 0$.

A continuación formando las funciones lagrangianas correspondiente de (4.1) y (4.2) tenemos:

$$L_P(x, y) = c^t x - \mu \sum_{j=1}^{n} \log(x_j) - y^t (Ax - b) \tag{4.3}$$

$$L_D(x, y) = b^t y + \mu \sum_{j=1}^{n} \log(z_j) - x^t (A^t y + z - c) \tag{4.4}$$

Dados los puntos x, z ∈ R^n. si definimos $e = [1\ 1\ 1\ ...\ 1]^t \in R^n$ y

$$X = \begin{bmatrix} x_1 & 0 & \cdots & 0 \\ 0 & x_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & x_n \end{bmatrix} \quad \text{y} \quad Z = \begin{bmatrix} z_1 & 0 & \cdots & 0 \\ 0 & z_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & z_n \end{bmatrix}$$

Notando que

$$X^{-1} = \begin{bmatrix} \frac{1}{x_1} & 0 & \cdots & 0 \\ 0 & \frac{1}{x_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{x_n} \end{bmatrix} \quad \text{y} \quad Z^{-1} = \begin{bmatrix} \frac{1}{z_1} & 0 & \cdots & 0 \\ 0 & \frac{1}{z_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{z_n} \end{bmatrix}$$

Calculando las derivadas parciales respectivas de (4.3) y (4.4), tenemos:

$$\frac{\partial L_P}{\partial x} = c - \mu X^{-1}e - A^t y = 0 \qquad (4.5)$$

$$\frac{\partial L_P}{\partial y} = b - Ax = 0 \qquad (4.6)$$

$$\frac{\partial L_D}{\partial x} = c - A^t y - z = 0 \qquad (4.7)$$

$$\frac{\partial L_D}{\partial y} = b - Ax = 0 \qquad (4.8)$$

$$\frac{\partial L_D}{\partial z} = \mu Z^{-1}e - x = 0 \qquad (4.9)$$

De (4.7) despejamos $c = A^t y + z$ y sustituimos en (4.5)

$$\begin{aligned} c - \mu X^{-1}e - A^t y &= 0 \qquad (c = A^t y + z) \\ A^t y + z - \mu X^{-1}e - A^t y &= 0 \\ z &= \mu X^{-1}e \\ Xz &= \mu e \\ XZe &= \mu e \qquad (z = Ze) \end{aligned}$$

Esto se puede escribir

$$x_j z_j = \mu \qquad j = 1, ..., n$$

Por lo que las condiciones de optimalidad simultáneas, para un mismo μ de los algoritmos (P) y (D) se pueden escribir como:

$$\begin{aligned} Ax &= b \\ A^t y + z &= c \\ XZe &= \mu e \end{aligned} \tag{4.10}$$

En forma equivalente están dadas por

$$\begin{aligned} Ax &= b \\ A^t y + z &= c \\ x_j z_j &= \mu, \qquad j = 1, ...n \end{aligned} \tag{4.11}$$

Observamos que cuando $\mu = 0$, las condiciones de optimalidad para el problema de programación son satisfechas. Estas condiciones de optimalidad en la forma estándar son:

$$\begin{aligned} Ax &= b, & x \geq 0 \\ A^t y + z &= c, & z \geq 0 \\ x_j z_j &= 0, & j = 1, ...n \end{aligned}$$

Si la solución del problema del sistema de ecuaciones lineales (4.11) se designa mediante $[x(\mu), y(\mu), z(\mu)]^t$, para cada $\mu > 0$. El *duality gap* es

$$\begin{aligned} g(\mu) &= c^t x(\mu) - b^t y(\mu) \\ &= x^t(\mu)c - x^t(\mu)Ay(\mu) && (Ax(\mu) = b) \\ &= x^t(\mu)\left(c - Ay(\mu)\right) \\ &= x^t(\mu)z(\mu) && \left(c - A^t y(\mu) = z(\mu)\right) \\ &= n\mu \end{aligned}$$

Al tender $\mu \rightarrow 0$, g (μ) converge a cero, lo que implica que $x(\mu)$ y $y(\mu)$ converge a la solución de los problemas (P) y (D), respectivamente.

μ es una aproximación del intervalo de dualidad, y debemos aclarar, que si su valor decrece muy rápidamente se puede perder convergencia. El intervalo de dualidad puede incluso aumentar. Si su valor disminuye muy lentamente se requiere muchas iteraciones.

Pero, por cuestiones numéricas, nosotros debemos comenzar fijando un $\mu > 0$ inicial y aproximarlo iterativamente hacia cero.

Paso de Newton

El método Primal-Dual se basa en tomar como direcciones de movimiento las de Newton, resultantes de resolver el sistema (4.10). Dada los puntos iniciales x^0, $z^0 \in R^n$, $y^0 \in R^m$, donde $x^0 > 0$ y $z^0 > 0$.

Fijemos adecuadamente un valor inicial para el parámetro de penalización $\mu = \mu_0$. El sistema (4.10) lo podemos representar por $F(x^0, y^0, z^0) = 0$

$$F(x^0, y^0, z^0) = \begin{bmatrix} Ax^0 - b \\ A^t y^0 + z^0 - c \\ XZe - \mu_0 e \end{bmatrix} = 0 \qquad (4.12)$$

La matriz Jacobiana de $F(x^0, y^0, z^0) = 0$, lo podemos representar por

$$J(x^0, y^0, z^0) = \begin{bmatrix} A & 0 & 0 \\ 0 & A^t & I \\ Z & 0 & X \end{bmatrix}$$

Para realizar una iteración de Newton, resolvamos el sistema $J(x^0, y^0, z^0)dw = -F(x^0, y^0, z^0)$ donde $dw = [dx, dy, dz]^t$ es el paso de Newton.

$$\begin{bmatrix} A & 0 & 0 \\ 0 & A^t & I \\ Z & 0 & X \end{bmatrix} \begin{bmatrix} dx \\ dy \\ dz \end{bmatrix} = -\begin{bmatrix} Ax^0 - b \\ A^t y^0 + z^0 - c \\ XZe - \mu_0 e \end{bmatrix}$$

$$= \begin{bmatrix} b - Ax^0 \\ c - A^t y^0 - z^0 \\ \mu_0 e - XZe \end{bmatrix}$$

definamos los vectores residuales Primal y Dual

$$\begin{aligned} r_p &= b - Ax^0 \\ r_d &= c - A^t y^0 - z^0 \\ r_c &= \mu_0 e - XZe \end{aligned}$$

Es decir, se resuelve el sistema

$$\begin{bmatrix} A & 0 & 0 \\ 0 & A^t & I \\ Z & 0 & X \end{bmatrix} \begin{bmatrix} dx \\ dy \\ dz \end{bmatrix} = \begin{bmatrix} r_p \\ r_d \\ r_c \end{bmatrix} \tag{4.13}$$

Obsérvamos que si $x^k \in P$ y $[y^k, z^k]^t \in T$. $k = 0, 1, 2, ...$, entonces $r_p = 0$ y $r_d = 0$.

El sistema (4.13) puede ser resuelto indirectamente, y puede ser visto como.

$$Adx = r_p \tag{4.14}$$

$$A^t dy + dz = r_d \tag{4.15}$$

$$Zdx + Xdz = r_c \tag{4.16}$$

multiplicando (4.15) por *X* y restando con (4.16), obtenemos

$$XA^t dy - Zdx = Xr_d - r_c \tag{4.17}$$

multiplicando la ecuación (4.17) por AZ^{-1}e involucrando (4.14) llegamos a un sistema cuadrado para *dy* esto es

$$AZ^{-1}XA^t dy = AZ^{-1}(Xr_d - r_c) + r_p$$

calculada *dy* se puede calcular fácilmente *dz* y *dx*, de (4.15) y (4.16), en conjunto tenemos.

$$\left.\begin{aligned} dy &= (AZ^{-1}XA^t)^{-1}(AZ^{-1}(Xr_d - r_c) + r_p) \\ dz &= r_d - A^t dy \\ dx &= Z^{-1}\left(r_c - Xdz\right) \end{aligned}\right\} \tag{4.18}$$

Criterio de la razón

Mediante (4.18) tenemos calculada el paso de Newton $(dx, dy, dz)^t$. Ahora usamos *dx* como una dirección en el *x* − espacio y (*dy, dz*) como una dirección en el (*y, z*) − espacio. Seguidamente debemos elegir la longitud de los pasos en los dos espacios de modo que podamos desplazarnos prudentemente, manteniendo *x*> 0, *z*> 0, esto es posible utilizando:

$$\alpha_P = \min_{1 \le j \le n} \left\{ -\frac{x_j}{dx_j} : \quad dx_j < 0 \right\}$$
$$\alpha_D = \min_{1 \le j \le n} \left\{ -\frac{z_j}{dz_j} : \quad dz_j < 0 \right\}$$

El siguiente punto (x^1, y^1, z^1) será calculado mediante

$$x^1 = x^0 + \beta \alpha_P dx$$
$$y^1 = y^0 + \beta \alpha_D dy$$
$$z^1 = z^0 + \beta \alpha_D dz$$

donde $\beta \in <0, 1>$ es un parámetro que nos permite controlar que las posibles soluciones no lleguen a la frontera de la región factible, ni mucho menos salgan de la región. En la práctica se obtuvieron buenos resultados usando $\beta = 0.95$ y/o $\beta = 0.98$, esto completa una iteración de Newton. La técnica sólo resuelve el sistema (4.10) con respecto a un μ_0, para obtener una aproximación a la solución óptima, deberíamos reducir μ_0, en cada iteración.

Los puntos (x^1, y^1, z^1)^t son utilizados como iniciales para la segunda iteración de newton, al repetir esto, el método genera una sucesión de puntos $\{(x^k, y^k, z^k)^t\}$ que converge sólo hacia una solución del sistema (4.10).

Ajuste del Parámetro Penalizador:

Ahora debemos establecer un criterio para determinar cuando el punto actual $x(^k)$ está lo suficientemente próximo a x^*. Penalización sugiere la disminución iterativa de μ con la finalidad de obtener una solución para

el problema de programación lineal estándar, y nada impide la disminución del parámetro de penalización en cada iteración de newton, dado μ_k, el siguiente parámetro penalizador es elegido mediante:

$$\mu_{k+1} = \frac{\left|c^t x^k - b^t y^k\right|}{n^2}, \qquad k = 0, 1, 2, ..$$

donde *n* es el número de variables, esto produce una reducción substancial

en μ a cada paso de newton, las iteraciones continúan hasta que la diferencia

$$\left|c^t x^k - b^t y^k\right|$$

sea suficientemente pequeña, esto se consigue controlando el error relativo

$$\frac{\left|c^t x^k - b^t y^k\right|}{1 + \left|b^t y^k\right|} < \varepsilon \tag{4.19}$$

de (4.19), el denominador $1 + \left|b^t\ y^k\right|$ nos permite mantener una proporción acerca de la aproximación entre los valores objetivos primal y dual, se le sumó 1 para evitar problemas de precisión cuando el costo óptimo se acerca a cero, donde $\varepsilon > 0$ es la precisión deseada. Nosotros utilizamos el valor $\varepsilon = 10^{-6}$.

4.1.3 El algoritmo

Inicialización:

Dados $y \in R^m$ y $x, z \in R^n$, donde $x, z > 0$. Definir la precisión requerida $\varepsilon > 0$, el $n-$ vector columna $e = [1\ 1..,1]^t$ y el valor inicial de $\mu > 0$.

Paso 1: Mientras $\frac{|c^t x^{(k)} - b^t y^{(k)}|}{1+|b^t y^{(k)}|} < \varepsilon$, no satisface, HACER

Paso 2: Definir las matrices diagonales $X = diag(x)$ y $Z = diag(z)$

Paso 3: Calcular
$$\begin{cases} r_p = b - Ax \\ r_d = c - A^t y - z \\ r_c = \mu_0 e - XZ \end{cases}$$

Paso 4: Calcular
$$\begin{cases} dy = (AZ^{-1}XA^t)^{-1}(AZ^{-1}(Xr_d - r_c) + r_p) \\ dz = r_d - A^t dy \\ dx = Z^{-1}(r_c - Xdz) \end{cases}$$

Paso 5: Calcular
$$\alpha_P = \min_{1 \le j \le n} \left\{ -\frac{x_j}{dx_j} : dx_j < 0 \right\}$$
$$\alpha_D = \min_{1 \le j \le n} \left\{ -\frac{z_j}{dz_j} : dz_j < 0 \right\}$$

Paso 6: Asignar
$$\begin{cases} x \leftarrow x + (0.95)(\alpha_P)dx \\ y \leftarrow y + (0.95)(\alpha_D)dy \\ z \leftarrow z + (0.95)(\alpha_D)dz \end{cases}$$

Paso 7: Hacer $\mu_{k+1} = \frac{|c^t x - b^t y|}{n^2}$ y volver al **paso 1**.

4.1.4 Implementación

La implemetación está hecha de un modo particular y con fines experimentales hemos utilizado Matlab 6.0, esto se justifica debido a la interface gráfica y matricial que Matlab otorga, la que denominaremos AOPL (Algoritmo de optimización para programación lineal)

Implementación del algoritmo para la optimización del tiempo de ejecución en la solución de problemas de programación lineal

```
function [x,v,iter] = AOPL(A,b,c)
[m,n]=size(A); x=ones(n,1); y=zeros(m,1); z=x; u=100; e=x
iter=0; t=cputime;
while abs(c'*x-b'*y)/(1+abs(b'*y)) > 0.000001;
X=diag(x); Z=diag(z);
      rp=b-A*x;
      rd=c-A'*y+z;
      rc=u*e-X*z;
dy=inv((A*inv(Z)*X*A'))*(A*inv(Z)*(X*rd-rc)+rp);
dz=rd-A'*dy;
dx=inv(Z)*(rc-X*dz);
        aP=-1/min(min(dx./x),-1);
        aD=-1/min(min(dz./z),-1);
  x=x+0.95*aP*dx;
  y=y+0.95*aD*dy;
  z=z+0.95*aD*dz;
        u=abs(c'*x-b'*y)/n^2;
  iter=iter+1;
   end
v=c'*x;
disp('tiempo de ejecución (seg):%f'); disp(cputime-t);
```

El programa resuelve problemas de programación lineal en la forma estándar

Minimizar $c\,x$

$$Ax = b$$

$$x \geq 0$$

debemos advertir que α_P =aP y α_D =aD, ambos ejecutan el criterio de la razón. El parámetro con que iniciamos es μ =u=100, para reducir μ a cada iteración se usó u = abs(c'*x - b'*y)/n^2. Con un parámetro de precisión de ε =0.000001, y los puntos iniciales x = [1 1 . . .1]^t, y = [0 0 . . .0]^t y z = [1 1. . .1]^t. Una vez ingresada la matriz *A* y los vectores *b* y *c*, llamamos AOPL.

4.1.5 Experimentos computacionales

Al ejecutar el programa con los ejemplos siguientes y ver su comportamiento de estos, hemos usado el sistema operativo Windows-XP (Pentium- III), con un PC compatible y usando procesador celeron(R) cpu 2.40 GHz, con 256 MB de memoria RAM.

Usamos el Matlab 6.0 para windows, Matlab es el nombre abreaviado de "MATrix LABoratory". MATLAB es un programa para realizar cálculos numéricos con vectores y matrices. Como caso particular puede también trabajar con números escalares, tanto reales como complejos. Una de las capacidades más atractivas es la de realizar una amplia variedad de gráficos de dos y tres dimensiones. Matlab tiene también un lenguaje de programación propio.

Cada resultado obtenido fué comparado con la ejecución del algoritmo Simplex, utilizando la técnica M-grande.

Observación 4.1 *Para generar problemas inmensos se construyó un programa en Matlab, a lo que llamamos generaPL al ingresar las dimensiones m y n, el programa produce un ejemplar de programación lineal.*

Primer experimento en el computador: Considere el siguiente problema

$$\begin{aligned} \textit{Minimizar} \quad & -2x_1 - 7x_2 \\ s.a \quad & 4x_1 + 5x_2 \leq 40 \\ & 2x_1 + x_2 \geq 8 \\ & 2x_1 + 5x_2 \geq 20 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Su forma estándar dada por

$$\begin{aligned} \textit{Minimizar} \quad & -2x_1 - 7x_2 + 0x_3 - 0x_4 - 0x_5 \\ s.a \quad & 4x_1 + 5x_2 + x_3 = 40 \\ & 2x_1 + x_2 - x_4 = 8 \\ & 2x_1 + 5x_2 - x_5 = 20 \\ & x_1, x_2, x_3, x_4, x_5 \geq 0 \end{aligned}$$

Y su correspondiente Dual de problema es

$$\begin{aligned} \textit{Maximizar} \quad & 40y_1 + 8y_2 + 20y_3 \\ s.a \quad & 4y_1 + 2y_2 + 2y_3 + z_1 = -2 \\ & 5y_1 + y_2 + 5y_3 + z_2 = -7 \\ & y_1 + z_3 = 0 \\ & -y_2 + z_4 = 0 \\ & -y_3 + z_5 = 0 \\ & z_1, z_2, z_3, z_4, z_5 \geq 0 \end{aligned}$$

Para un punto inicial factible

$$x^0 = \begin{bmatrix} 5 \\ 3 \\ 5 \\ 5 \\ 5 \end{bmatrix}$$

La (figura 9) muestra la sucesión de puntos generadas por el algoritmo, hasta llegar al punto óptimo que es el punto [0, 8]'. Se puede observar que todos los puntos están en la región de factibilidad.

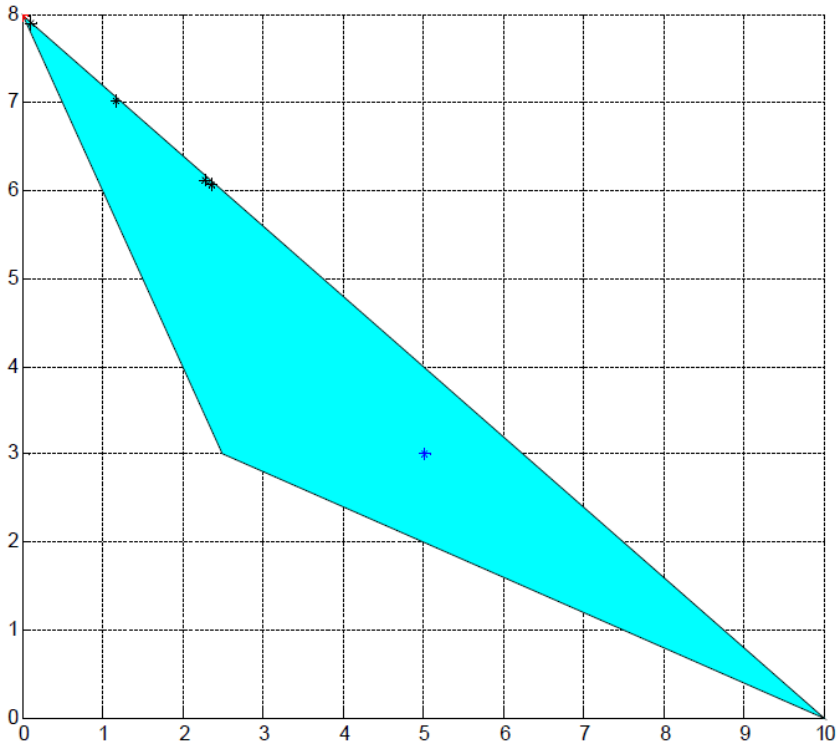

Figura 9. Punto inicial factible [5, 3]^t

Llamando la implementación AOPL, también se obtuvo el punto óptimo [0, 8]' y su valor óptimo se puede obtener al remplazar el punto óptimo en la función objetivo.

iter	$x^0 =$	5	3
1	$x^1 =$	2.35	6.07
2	$x^2 =$	2.27	6.12
3	$x^3 =$	1.18	7.02
4	$x^4 =$	0.12	7.90
5	$x^5 =$	0.01	7.99
6	$x^6 =$	0.00	8.00

Observamos ahora que, dada otro punto inicial infactible x^0 =[1 1 1 1 1]^t, Se observó aquí, a pesar que el punto inicial es infactible [1 1]^t,

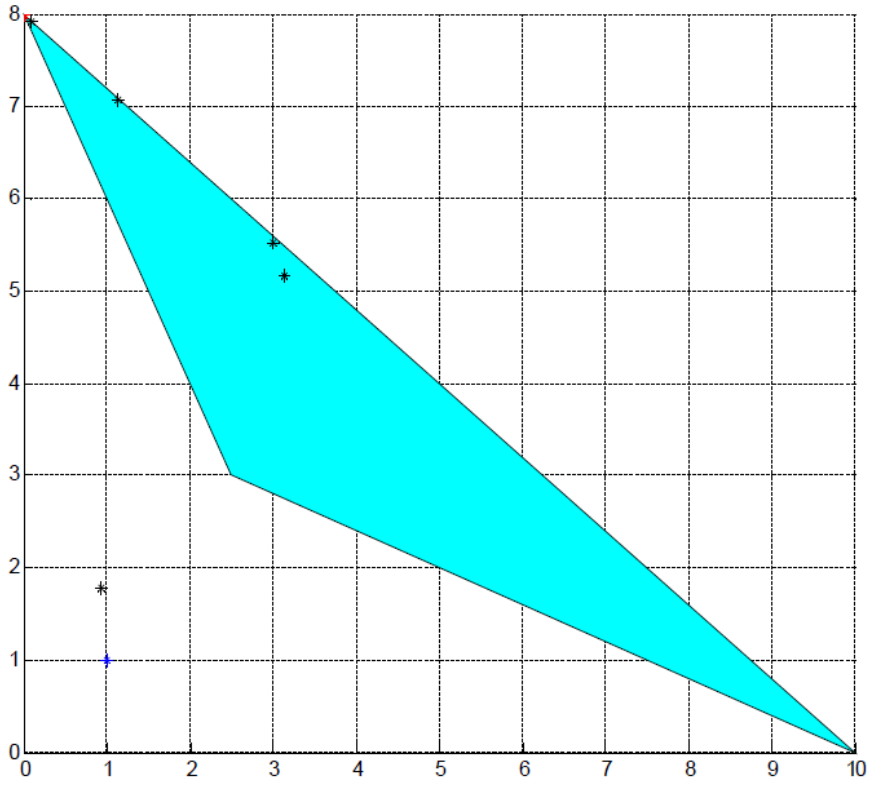

Figura 10. Punto inicial infactible [1, 1]^t

También el algoritmo hace converge a la misma solución óptima del problema la cual es [0 8]^t , veamos la figura adjunta.

Iteraciones con la implementación AOPL

iter	$x^0 =$	1	1
1	$x^1 =$	0.93	1.77
2	$x^2 =$	3.14	5.16
3	$x^3 =$	2.99	5.53
4	$x^4 =$	1.13	7.07
5	$x^5 =$	0.09	7.93
6	$x^6 =$	0.01	8.00
7	$x^7 =$	0.00	8.00
8	$x^8 =$	0.00	8.00

Segundo experimento en el computador:

Un carpintero hace sillas y mesas solamente. Las utilidades proyectadas para los dos productos son S/. 20 por silla y S/. 30 por mesa respectivamente. La demanda proyectada es 400 sillas y 100 mesas, cada silla requiere 2 pies cúbicos de madera, mientras que cada mesa requiere 4 pies cúbicos. El carpintero tiene un total de 1000 pies cúbicos de madera en stock. ¿Cuántas sillas y mesas debe construir para aumentar al máximo sus utilidades?

Sean

$$x_1 \text{ el número de sillas}$$

$$x_2 \text{ el número de mesas}$$

Estas son las variables desconocidas, para éste problema.

El carpintero quiere aumentar al máximo sus utilidades, esto es

$$20x_1 + 30x_2$$

sujeto a las restricciones que:

- El importe total de madera para los dos productos no puede exceder los 1000 pies cúbicos disponibles.
- Los números de sillas y mesas a ser hechas, no deben exceder las demandas, El número de sillas y mesas necesitan ser no negativos. por lo que tenemos:

$$\text{Maximizar } 20x_1 + 30x_2$$
$$2x_1 + 4x_2 \leq 1\ 000$$
$$0 \leq x_1 \leq 400$$
$$0 \leq x_2 \leq 100$$

Añadiendo las variables auxiliares y llevando a su forma estándar, tenemos.

$$\text{Minimizar } -20x_1 - 30x_2 + 0x_3 + 0x_4 + 0x_5$$
$$2x_1 + 4x_2 + x_3 + 0x_4 + 0x_5 = 1\ 000$$
$$x_1 + 0x_2 + 0x_3 + x_4 + 0x_5 = 400$$
$$0x_1 + x_2 + 0x_3 + 0x_4 + x_5 = 100$$
$$x_1, x_2, x_3, x_4, x_5 \geq 0$$

Introduciendo los siguientes datos en el computador

$$A = \begin{bmatrix} 2 & 4 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$$b = \begin{bmatrix} 1000 \\ 400 \\ 100 \end{bmatrix}; c = \begin{bmatrix} -20 \\ -30 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

y ejecutando el programa de AOPL se obtuvo

$$x = \begin{bmatrix} 400.00 \\ 50.00 \\ 0.00 \\ 0.00 \\ 50.00 \end{bmatrix}$$

v= 9499.99

Tercer experimento en el computador:

Aquí se generó un problema de programación lineal con 90 restricciones y 110 variables, resumiendo de estos resultados se tiene el siguiente cuadro.

Cuadro2. 90 restricciones y 110 variables

	m	n	iteraciones	tiempo (seg.)	Valor Objetivo
Simplex	90	110	164	8.03	1597.36
AOPL	90	110	15	0.437	1597.40

Aquí se observa también que el algoritmo Simplex necesita aproximadamente 11 veces más de iteraciones, que las iteraciones de AOPL, el cual solo realizó 15 iteraciones. En cuanto al tiempo de ejecución, la diferencia es de 7.59 segundos

Cuarto experimento en el computador:

Aquí se generó un problema de programación lineal con 220 restricciones y 320 variables y se obtuvo los siguientes resultados.

Resolviendo con el algoritmo se obtuvo en (cuadro 3)

Cuadro 3. 220 restricciones y 320 variables

	m	n	iteraciones	tiempo (seg.)	Valor Objetivo
Simplex	220	320	762	692.28	-185790
AOPL	220	320	21	8.656	-185791.78

En este problema se muestra la eficacia del AOPL respecto al Simplex, en cuanto al tiempo de ejecución y las respectivas iteraciones.

CONCLUSIONES

- Se construyó un algoritmo para la optimización del tiempo de ejecución en la solución de problemas de programación lineal, con todas las justificaciones matemáticas necesarias.
- Se Implementó el algoritmo para la optimización del tiempo de ejecución en la solución de problemas de programación lineal en el software de MatLab.
- Se realizó experimentos computacionales que corroboren la eficacia del método frente a problemas de grandes tamaños

RECOMENDACIONES

Se realizaron también ejemplares con 4000 variables, pero debido a la falta de memoria en el computador no se pudo encontrar la solución. Esta situación puede ser evitada introduciendo un tratamiento numérico al algoritmo, pero este proyecto constituye otro tema de investigación.

BIBLIOGRAFÍA

Bazaraa, M., Sherali, H.,&Shetty, C. (1993). *Nonlinear Programming theory and Algorithms*, México: JohnWiley & Sons.

De la fuente, O. (2016). *Matemáticas y algoritmos numéricos*. España: Círculo rojo.

Fiacco,A., Mccormick, G. (1968) *Nonlinear Programming: Sequential Unconstrained Minimization Technique*. New York: John Wiley and Sons.

Gonzaga, C. (1991).*Large-Step Path-Following Methods for Linear Programming, Part 1: Barrier Function Method*. SIAM Journal on Optimization, Vol. 1, pp. 268

Gonzaga, C. (1992). *Path-Following Methods for Linear Programming*. SIAM Review, Vol. 34, pp. 167-224.

Jansen, B., Roos, C.,Terlaky, C. & Vial, J. (1994). *Primal-Dual Algorithms for linear Programming Based on the logarithmic Barrier Method*.

Karmarkar, N. (1984).*New Polynomial-Time Algorithm for Linear Programming, Combinatorica*, Vol. 4, pp. 373-395

Kojima,M.,Mizuno, S., and Yoshise, A. (1989). *A Primal-Dual Interior- Point Algorithm for a Class of Linear Complementarity Problems, Mathematical Programming*, Vol. 44, pp. 1-26.

Martínez, J., Santos, A. (1995).*Métodos computacionais de optimização*.XX Col\'oquio Brasileiro de Matemática, IMPA (ISBN 85-244-0092-7, 256 pages).

Martinez, J., Santos, A. (1994). *Métodos Computacionales de Optimizacäo*. Brasil: Imec-Unicamp.

Mccormick, G. (1989).*The Projective SUMTMethod for Convex Programming, Mathematics of Operations Research*, Vol. 14, pp. 203-223

Mehrotra, S. (1992). *On the Implementation of a Primal-Dual Interior- Point Method*, SIAM Journal on Optimization, Vol. 2, pp. 575—601.

Monteiro, R., Adler, I. (1989). *Interior Path-Following Primal- Dual Algorithms, Part 1: Linear Programming, Mathematical Programming*, Vol. 44, pp. 27-41.

Leañez, G. (2013). *Estudio comparativo de la relajación lagrangeana y la programación entera-mixta en el problema del pre-despacho de sistemas medianos*. Disponible en.

http://www.repositorio.uchile.cl/handle/2250/115353.

Ruiz, N. (2016). *Contributions to the convergence theory and computational implementation of interior optimization methods for convex problems.*Disponible en http://repositorio.uchile.cl/handle/2250/140605

Roos, C., Vial, J. (1992).*A Polynomial Method of Approximate Centers for Linear Programming, Mathematical Programming*, Vol. 54, pp. 295-305.

Ticona, Percy. (2003). *Optimización Lineal — Parte I*. Perú: Escuela de matemáticas- UNSA

Yin, Zhang. (2007). *A quick Introduction to Linear Programming*.DRAFT. September 24

ANEXOS

Anexo 1. Implementaciones en el MatLab

a) Implementación de newton

```
function x=newton(x)
iter=0; prec=0.000001;
while norm(Fn(x))>prec
    iter=iter+1;
    x=x-inv(Jn(x))*Fn(x);
    fprintf('iter=%2i; x = [%7.4f%7.4f]\n', iter,x);
    if iter>1000
        error('parece que newton no converge');
    end
end
```

b) implementación de lagrange

```
function [x,iter]=lagrange(x,y)
n=length(x);m=length(y);iter=0;
    while norm(Fn(x,y))>0.0001
      d=-inv(Jn(x,y))*Fn(x,y);
      dx=d(1:n); dy=d(n+1:n+m);
      a=-1/min(min(dx./x),-1);
      x=x+0.95*a*dx; y=y+0.95*a*dy;
      iter=iter+1;
    end
```

c) Implementación de penalización interna

```
function [x,iter]=Penalización(x,y)
n=length(x);
m=length(y);
iter=0;
    while norm(Fu(x,y,u))>0.0001
      d=-inv(Jn(x,y,u))*Fn(x,y,u);
      dx=d(1:n);
    dy=d(n+1:n+m);
      a=-1/min(min(dx./x),-1);
      x=x+0.95*a*dx; y=y+0.95*a*dy;
     u = u/10;
      iter=iter+1;
    end
```

d) IMPLEMENTACIÓN DEL ALGORITMO

```
function [x,v,iter] = AOPL(A,b,c)
[m,n]=size(A); x=ones(n,1); y=zeros(m,1); z=x; u=100; e=x
iter=0; t=cputime;
while abs(c'*x-b'*y)/(1+abs(b'*y)) > 0.000001;
X=diag(x); Z=diag(z);
    rp=b-A*x;
    rd=c-A'*y+z;
    rc=u*e-X*z;
dy=inv((A*inv(Z)*X*A'))*(A*inv(Z)*(X*rd-rc)+rp);
dz=rd-A'*dy;
dx=inv(Z)*(rc-X*dz);
```

```
        aP=-1/min(min(dx./x),-1);
        aD=-1/min(min(dz./z),-1);
    x=x+0.95*aP*dx;
    y=y+0.95*aD*dy;
    z=z+0.95*aD*dz;
        u=abs(c'*x-b'*y)/n^2;
    iter=iter+1;
     end
v=c'*x;
```

Anexo 2. Genera problemas de programación lineal

```
function [A,b,c]=generaPL(m,n)
A=round(100*rand(m,n)-100*rand(m,n));
x=round(100*rand(n,1));
b=A*x;
c=round(100*rand(n,1)-100*rand(n,1));
```

Anexo 3. Notación

NOTACIÓN PRINCIPAL EMPLEADA EN ESTE TRABAJO

A	Matriz *mxn*
$x^{(k)}$	Diferentes puntos obtenidos por el algoritmo
x^*	Punto solución
x_i	Componente de vectores
X, Z	Matrices
k	Número de iteraciones
n	Número de variables
SBF	Solución básica factible
AOPL	Algoritmo de optimización para programación lineal
PL	Programación Lineal
KKT	Condiciones de optimalidad.