YILDIZ TEKNİK ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

KARMARKAR ALGORİTMASI VE ÇOK AMAÇLI PROGRAMLAMAYA UYGULANMASI

Matematik Müh. Abdülkadir TEPECİK

F.B.E. Matematik Müh. Anabilim Dalında hazırlanan

YÜKSEK LİSANS TEZİ

Tez Danışmanı Doç. Dr. Mehmet AHLATÇIOĞLU

İSTANBUL, 1994

İÇİNDEKİLER

ÖZET	*iv*
YABANCI DİLDE ÖZET	*v*
1. GİRİŞ	1
2. GEREKLİ MATEMATİKSEL KAVRAMLAR	
2.1. Cebirsel Ön Bilgi	4
2.2. Amaç Fonksiyonunun Minimize Edilmesi	5
3. LP PROBLEMLERİ VE SİMPLEX METODU	6
4. KARMARKAR ALGORİTMASI	8
4.1. Başlangıç Adımı	10
4.2. Genel İterasyon Adımı	11
4.3. Cebirsel Açıklama	13
4.4. Esas Teorem	14
4.5. Karmarkar Teoremi	15
4.6. Karmarkar Algoritmasının Özeti	16
4.7. Karmarkar Algoritmasının Bilgisayar Yorumu	17
4.8. Uygulamalar	23
5. DEĞİŞTİRİLMİŞ KARMARKAR ALGORİTMASI	31
5.1. Değiştirilmiş Karmarkar Algoritmasının Özeti	35
5.2. Değiştirilmiş Karmarkar Algoritmasının Bilgisayar Yorumu	36
5.3. Uygulamalar	43
6. İÇ-NOKTA ÇALP ALGORİTMASI	48
6.1. İç-Nokta ÇALP Algoritmasının Özeti	55
6.2. İç-Nokta ÇALP Algoritmasının Bilgisayar Yorumu	58
6.3. Uygulamalar	67
7. ÖZET ve İDDİALAR	73
8. KAYNAKLAR	74
ÖZGEÇMİŞ	

Özet

Bu çalışma, üç ana bölümden oluşmuştur. İlk bölümde son yılların oldukça popüler konusu olan, mucidi (N. Karmarkar) tarafından Simplex metodundan 50-100 misli daha hızlı olduğu söylenen Karmarkar Algoritması bütün detayları ile incelenmiştir.

İkinci bölümde , çözümü bilinmeyen standard formdaki LP problemlerini göz önüne alarak, gerçek Primal ve Dual çözümlere yakınsayan yaklaşık primal ve dual çözümler oluşturan, "*Değiştirilmiş Karmarkar Algoritması*" hesaplanmış örneklerle açıklanmıştır.

Üçüncü ve son ana bölümde ise *Affine-Scaling Primal Algoritması* olarak ta bilinen değiştirilmiş karmarkar algoritmasına dayanan bir çok amaçlı LP algoritması sunulmuştur. Herbir amaca ait projekte edilmiş gradientlerin konvex kombinezonunun kullanımı ve bu birleştirilmiş tek doğrultu boyunca yeni iterasyonların nasıl elde edileceği açıklanmıştır.

ABSTRACT

This study consists of three main parts. In the first part, Karmarkar's Algorithm, which was devised by Karmakar, is studied in details. This algorithm is 50-100 times faster than that of simplex method.

In the second part considering LP problem in the standard form of which solution in not known Modified Karmarkar Algorithm which approximates primal and dual solutions as well as finding them is explained giving computational examples.

In the last part, Multiobjective Linear Programming algorithm that is based on modified Karmarkar's algorithm known as the Affine Scaling Primal Algorithm is given. The use of convex combination of the projected gradients is shown for every objective. It was explained that how we can step toward the next iterate along combined direction.

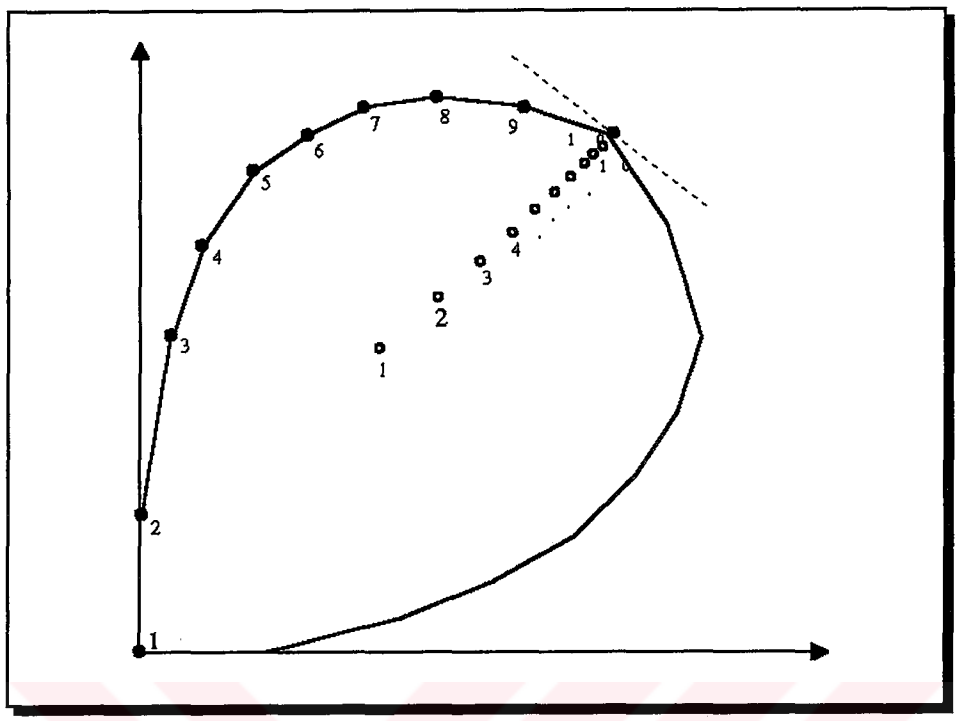
1.Giriş

N.Karmarkar'ın 1984 yılında Lineer Programlar için projektif algoritmasını keşfetmesinden beri birçok makaleye, yoruma ve birçok adaptasyona konu olmuştur. İlk olarak 1967 yılında Rus matematikçi Dikin tarafından ortaya atılan projektif algoritma daha sonra bilgisayarlardan etkili bir şekilde yararlanılarak Karmarkar tarafından yeniden keşfedilmiş ve bundan sonra kendi adıyla anılmaya başlanmıştır.

Karmarkar algoritması, kısıtlar bölgesi(feasible bölge, polytope) nin kesin olarak içinde geliştirilen doğrultuları bulmaya çalışan bir "iç-nokta" işlemidir. Algoritma bu haliyle, feasible bölgenin sınırı boyunca, optimum çözüm bulununcaya kadar bir köşeden başka bir köşeye atlayarak ilerleyen Simplex metodunun zıttıdır. Bu durum Şekil 1.1 de açık bir şekilde gösterilmiştir. Geniş ölçekli (değişken sayısı ve kısıt sayısı bakımından) problemler için simplex metodunun binlerce köşe noktasını taraması gerekecektir. Bu ise çok zaman alacaktır. Eğer feasible bölgenin içinde ilerleyen gelişen bir doğrultu bulunabilirse önemli ölçüde zaman tasarrufu sağlanabilecektir. Bununla birlikte Karmarkar algoritmasıda projeksiyonun hesaplanması için oldukça vakit kaybedecektir. Birçok araştırma bunun üzerinde yoğunlaşmaktadır. Bu hesaplamaların bilgisayarda yorumlanması sırasında seyrek matris tekniklerinin kullanılması oldukça revaçtadır.

Yapılan ilave değişiklikler ve gelişmelerle bu algoritma simplex algoritması için ciddi bir rakip olmuştur. Bu değişikliklerin en önemlileri Vanderbei *et al.* [3], E.R.Barnes *et al.* [8] ve Tom M. Cavalier *et al.* [1,2] tarafından yapılmıştır.

Çok Amaçlı Karar Verme, özellikle Çok Amaçlı Lineer Programlama(ÇALP) son yılların oldukça popüler konularından birisidir ve bu konuda oldukça ilerlemeler kaydedilmiştir. ÇALP şu anda hesaplamalarında yardımcı olarak Simplex metodunu kullanmaktadır. Simplexe dayanan ÇALP algoritmasına karşıt olarak özellikle son bir kaç yılda Karmarkar'ın projektif algoritması da kullanılmaya başlanmıştır.



Şekil 1.1

Bu iki değişik metod arasında tam bir karşılaştırma yapılabilmesi için iç-nokta ÇALP algoritmalarının Simplexi kullanan ÇALP algoritmaları ile aynı seviyeye gelmesi gerekecektir.

Sunduğumuz bu tezimiz adında anlaşıldığı üzere "Karmarkar Algoritmasının Çok Amaçlı Programlamaya Uygulanması" dır. Buradan hareketle tezi üç ana bölümde incelemek mümkündür. Birincisi, iç-nokta algoritmalarının temeli olan "Karmarkar Algoritması", ikincisi "Değiştirilmiş Karmarkar Algoritması" ve sonuncusu da "İç-nokta ÇALP Algoritması" dır. İlkinde bu temel algoritma detayları ile ele alınmış, örnekler üzerinde uygulanmıştır. Yine keza, Karmarkar algoritmasından daha çok kullanılan Değiştirilmiş Karmarkar Algoritması(veya Affine-Scaling Primal Algoritma) da detayları ile incelenmiştir. Daha sonra sunulan metod ise tezin can alıcı bölümüdür. Bu bölümde,

sunulan algoritma, göz önüne alınan amaç fonksiyonlarının herbirinin optimize edilmesi için oluşturulan *iç adım doğrultu vektörlerinden* oluşan bir iterasyonlar dizisi ile gerçeklenmektedir. Daha sonraki adımlarda bütün amaç fonksiyonlarının vektör değerleri göz önüne alınarak bunlarla ilgili adım doğrultu vektörlerine ilişkin tercihe karar verilmektedir.

Eğer bir fayda fonksiyonu varsa, amaç uzayı olarak adlandırılan noktalarla ilgili tercihi bulmak için ayrı ayrı olan bu adım doğrultu vektörlerini birtek adım doğrultu vektörüne dönüştürülerek hangi doğrultuda iterasyona devam edileceğine karar verilir. Fayda fonksiyonunun olmaması durumunda ise seçimi karar verici yapmalıdır. Herhangi bir durdurma koşuluyla karşılana kadar *lokal olarak ilgili* ağırlıklandırılmış katsayıları üretilerek işlem tekrarlanır.

Herhangi bir ÇALP algoritmasının cazip olması için test edilebilir olması hatta test edilmiş olması gerekir. Yani, bir fayda fonksiyonu varsa, bu diğer algoritmik işlemlerle birlikte kullanılarak, sonuçlar kümesinde kara vericinin fayda fonksiyonunu maximum yapan bir optimizasyon probleminin doğru sonucuna ulaşmalıdır. İşte böylesi testler tezde sıkça kullanılmıştır.

2.1 Cebirsel Ön Bilgi

P_+^n ile gösterilen R^n in bir alt kümesi ($P_+^n \subset R^n$) olan n boyutlu orthant, negatif olmayan $x \in R^n$ lerden oluşur. (yani elemanları negatif olmayan reel sayılardan oluşan birbirine dik vektörlerden teşkil edilmiş n boyutlu uzaydır.)

Bütün homojen lineer denklem sistemlerinin $x_1=0$, $x_2=0$, ...,$x_n=0$ olan bir çözümü vardır. Bu çözüme trivial(açık veya sıfir) çözüm, diğerlerine de trivial olmayan çözüm denir.

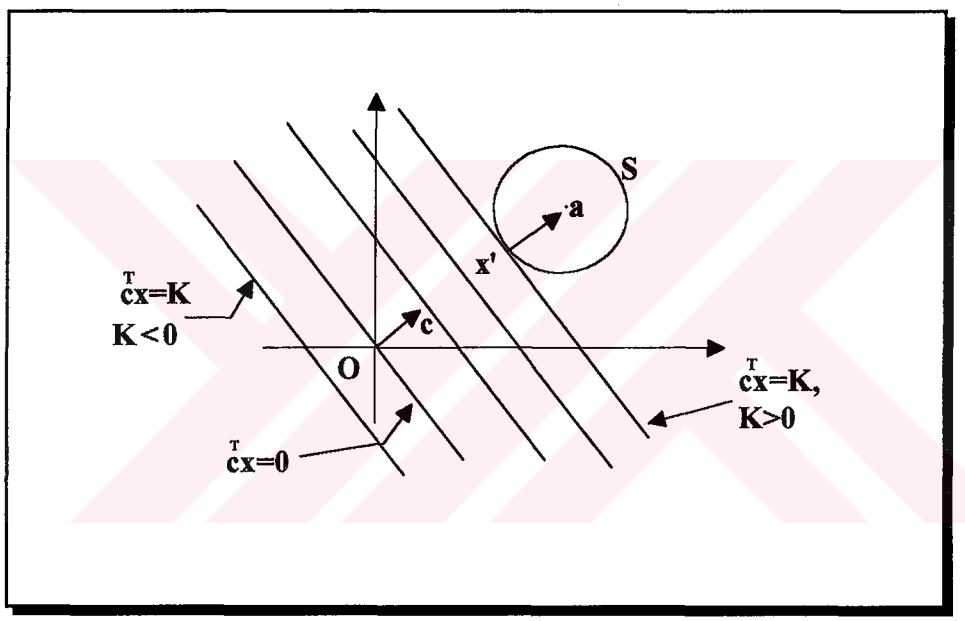
R^n in bir $\Omega = \{x \mid Ax = 0\}$ alt uzayında Ax=0 formundaki lineer homojen denklem sisteminde eğer A matrisi trivial değilse çözümler orijini içerir ve boyutu n den küçüktür.

Ax=b şeklinde homojen olmayan denklem sisteminin çözümleri, uygun homojen denklemlerin çözümlerine dönüştürülür. Bu halde $Ax_0=b$ yi sağlayan bir x_0 özel çözümü verildiğinde Ax=0 olan herhangi x ler için $A(x_0+x)=b$ dir.

N boyutlu S^n küresi R^{n+1} de $\{x \mid \Sigma x_i^2 = 1\}$ ile tanımlıdır. Buna göre S^0 reel eksen üzerinde (R^1 de) +1 ve -1 noktalarından oluşur. S^1, R^2 de birim çemberdir. S^2, R^3 de birim küredir. Benzer şekilde n boyutlu Δ^n simplexi R^{n+1} de $\{x \mid \Sigma x_i = 1 \text{ ve } x_i \geq 0\}$ ile tanımlıdır. Böylece Δ^0 reel doğru üzerinde +1 noktasıdır. Δ^1 düzlemde (0,1) den (1,0) 'a bir diagonal doğru parçasıdır. Δ^2 de (0,0,1), (0,1,0) ve (1,0,0) noktalarını birleştiren üçgendir. N boyutlu simplexin merkezi $a_0 = \left(\frac{1}{n+1}, \ldots, \frac{1}{n+1}\right)^T$ olup a_0 vektörü ile gösterilecektir.

2.2 Amaç Fonksiyonunun Minimize Edilmesi

x, a_0 merkezli bir S küresi üzerinde olmak üzere, c^Tx 'i minimize etmek isteyelim. $c^Tx=0$ 'ın çözümleri orijinde c vektörüne dik x vektörleri olduğundan, $c^Tx=K$ 'nın çözümleri, $c^Tx=0$ 'ın çözümlerine parelel doğru ailesinden oluşur.



Şekil 2.1:S Çemberi üzerinde c^Tx i minimize eden bir noktanın bulunması

K>0 ise yerdeğiştirme c ile aynı yönde, K<0 ise zıt yöndedir. O halde S üzerinde bir nokta olan ve c^Tx 'i minimize eden x' noktası, a_0 merkezinden çizilen c vektörü ile çemberin arakesitidir.

3. LP Problemleri ve Simplex Metodu

LP problemleri ile

$$\begin{aligned} &\text{Min } c^T x \\ &\text{Kısıtlar } Ax \geq b \\ &\quad \text{ve } x \geq 0 \end{aligned} \tag{3.1}$$

şeklindeki problemleri kastediyoruz. Burada c ve x, R^n de vektorler, A mxn lik matris, b ise R^m de bir vektördür. Amaç fonksiyonu $c^T x$, kısıtlar ise ($Ax \geq b$ ve $x \geq 0$) dir. Bu problemin duali (ikizi)

$$\begin{aligned} &\text{Max } b^T y \\ &\quad \text{Kısıtlar } A^T y \leq c \\ &\quad\quad \text{ve } y \geq 0 \end{aligned} \tag{3.2}$$

şeklinde bir maximizasyon problemdir. Buradaki y R^m de vektör olup, "dual vektör" olarak bilinirler. Bu problemlerin çözümleri birbirleri ile alakalıdır. Zayıf dualite teorisinden biliyoruz ki $b^T y \leq (Ax)^T y = x^T (A^T y) \leq x^T c = c^T x$ dir. Bu şekildeki maximizasyon problemi $b^T y$ 'yi mümkün olduğu kadar çok artırmaya uğraşırken, minimizasyon problemi de genellikle $b^T y$ 'ye eşit veya daha büyük olan $c^T x$ i küçültmeye uğraşır. Yani, dual problem artarak , esas(primal) problem de azalarak çözüme ulaşmaya çalışırlar. Sonuç olarak bir çözüm bulunabilirse bunun $b^T y = c^T x$ olduğunda oluşması gerektiği açıktır.

Simplex metodu, orijinal(primal,esas) problemin ve dual problemin her ikisini de aynı anda çözen bir LP algoritmasıdır. Simplex metodunu (3.1) şeklindeki bir probleme uygulamak için ilk önce problemi

$$\begin{aligned} &\text{Min } c^T x \\ &\quad \text{kısıtlar } Ax = b \\ &\quad\quad \text{ve } x \geq 0 \end{aligned} \tag{3.3}$$

Şeklinde eşitlikler halinde yeniden yazarız. (Bu işlem "problemi standard hale getirme" olarak bilinmektedir). Eşitsizliği eşitlik haline getirmek için küçük olan tarafa sıfır maliyetli bir slack (hayali) değişken ilave edilir. Bu durumda x, R^{n+m} de bir vektör olur. Bu halde k. $Ax_k \geq b_k$ eşitsizliğini $Ax_k - s_k = b_k$ şeklinde bir eşitliğe dönüştürmek için m tane $s_k \geq 0$ olan slack(boş) değişken kullanılır. Yeni c vektörü eski c 'ye m tane sıfırın, yine eski A matrisi ise eski A ya $-I_m$ nin ilave edilmesiyle elde edilir. İlk bakışta bu işlem , (3.1) in yeniden açık bir formulasyonu gibi görünmesine rağmen öyle değildir.

Zira, m tane slack değişken dual problemde y olarak adlandırdığımız dual değişkenlerle yakından ilgilidirler. Ayrıca, şimdi (3.3) deki Ax=b yi sağlayan bir $x^{(0)}$ temel (başlangıç) noktasını biliyoruz. Çünkü $x^{(0)}$ noktası,

$k=1,2,...,n$ için $x_k^{(0)} = 0$ ve

$k=1,2,...,m$ için $x_{k+n}^{(0)} = -b_k$

yapılarak elde edilir. Eğer $x^{(0)} \geq 0$ ise bu bir temel feasible noktadır. Çünkü (3.3) ün kısıtlarını sağlar.

Simplex metodunun 1.safhası bir temel noktayı, bir temel feasible noktaya dönüştürmek, 2. safhası ise amaç fonksiyonunun minimumuna ulaşana kadar sırasıyla amaç fonksiyonuna göre daha iyi temel feasible noktalara hareket etmektir. Sonlu sayıda temel feasible nokta olduğundan simplex metodu ya çözümü bulacaktır yada sonlu sayıdaki adımda problemin çözülemezliğini belirleyecektir. Maalesef temel feasible noktaların sayısı değişkenlerin sayısının artması halinde temel feasible noktaların (köşelerin, exterem noktaların) sayısı daha çok artacaktır. Çünkü n değişkenli m kısıtlı bir problemin temel feasible noktalarının sayısı

$$C_n^m = \frac{n!}{m!(n-m)!} \text{ ile belirlenir.}$$

Bu halde simplex metodu için optimum noktaya erişmeden önce mümkün olan bütün temel feasible noktaları tarattıracak hileli problemler oluşturmak mümkündür. Bu suretle simplex metodunun en kötü durumdaki performansı e^n olur.

Diğer yandan problemin büyüklüğünü 2 misline çıkarttığımızda çalışma süresi e^n ile çarpılacaktır. (e^n in çok hızlı büyüdüğüne dikkat!.)

En kötü durumdaki performansına rağmen simplex metodu pratikte çok güzel çalışır. Fakat birçok bilgisayar sonlu sayıda tamsayı aritmetiği kullandığından sayılarda yuvarlama yapacaktır. Bu ise dejenerasyona sebebiyet vermektedir. Bu sebeple pratikte bu kesme yada yuvarlama hataları karışıklıklara sebep olabilmektedir.

4. KARMARKAR Algoritması

Karmarkar 'ın algoritması

$$\text{Min } c^T x$$
$$\text{Kısıtlar } x \in \Omega \cap \Delta^n \qquad (4.1)$$

şeklindeki problemleri yani, kısıtların içinde n boyutlu simplexin bulunduğu ve problemin optimal çözümünün sıfır olduğu problemleri çözmektedir. Bundan başka (3.1), (3.2) veya (3.3) şeklindeki LP problemlerini çözemez. (Ancak problemin büyüklüğünü değişken sayısı ve kısıt sayısı bakımından artıran çeşitli dönüşümler yapılarak çözülebilirler). Buradaki c, $x \in R^{n+1}$ ve $\Omega = \{x | Ax = 0\}$, bir homojen denklem sisteminin çözüm uzayıdır. Δ^n ise R^{n+1} de oluşturulmuş n boyutlu simplextir. Şimdi bu (4.1) formundaki problemleri çözmek için 2 tane varsayım yapacağız.

A)Amaç fonksiyonunun minimum değeri sıfırdır.

B)Problem feasible dır ve Δ^n simplexinin a_0 merkezi bir feasible noktadır.

Açıklamalarımızın soyut olmasını önlemek için aşağıdaki açıklayıcı problemi göz önüne alalım.

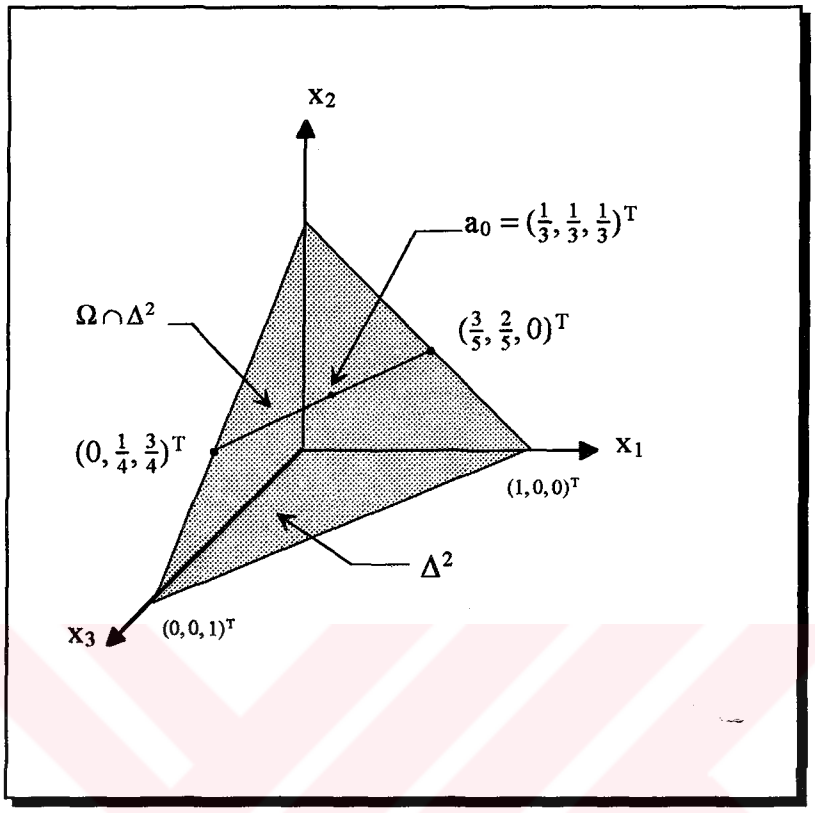
$$\text{Min } (3\ 3\ -1)x$$
$$\text{Kısıtlar } x \in \Omega \cap \Delta^2$$
$$\text{Burada } \Omega = \{x | (2, -3, 1)x = 0\} \text{ ve } x \in R^3$$

Bu problem birçok kaynakta

$$\begin{aligned} \text{Min} \quad & 3x_1 + 3x_2 - x_3 \\ \text{Kısıtlar} \quad & 2x_1 - 3x_2 + x_3 = 0 \\ & x_1 + x_2 + x_3 = 0 \\ & \text{ve} \quad x_i \geq 0 \end{aligned}$$

şeklinde de ifade edilmektedir.

Aşağıdaki Şekil 4.1 de problemin $\Omega \cap \Delta^2$ bölgesi gösterilmiştir.



Şekil 4.1 : $\Omega \cap \Delta^2$ bölgesi R^3 ün alt uzayının ve 2 boyutlu simplex Δ^2 nin ara kesitidir. Poblemimizde $\Omega = \{x \mid (2,-3,1)x = 0\}$, orijinden geçen ve Δ^2 yi $(0, \frac{1}{4}, \frac{3}{4})^T$ den $(\frac{3}{5}, \frac{2}{5}, 0)^T$ ye doğru parçası ile kesen bir düzlemdir.

Minimize edilen amaç fonksiyonu lineerdir. $\Omega \cap \Delta^2$ bölgesi bir doğru parçası olduğundan ve fonksiyon bölge üzerinde sabit olmadığından minimum bir temel feasible (köşe) noktada oluşmalıdır. $(\frac{3}{5}, \frac{2}{5}, 0)^T$ noktasında amaç fonksiyonunun değeri $(\frac{9}{5} + \frac{6}{5} - 0) = 3$. $(0, \frac{1}{4}, \frac{3}{4})^T$ noktasında ise $(0 + \frac{3}{4} - \frac{3}{4})^T = 0$ dır. Böylece problem(4.1) in (*A*) kabülü sağlanmış olur. $a_0 = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})^T$ merkez noktası için $Ax = (2 * \frac{1}{3} - 3 * \frac{1}{3} + 1 * \frac{1}{3}) = 0$ olduğundan a_0, Ω 'nın bir elemanıdır. Böylece (*B*) kabülü de sağlanmış oldu. Şu halde elimizde Δ^n simplexi ile sınırlı bölgede, bir iç noktası ve çözümünün sınır üzerinde olduğu bilinen bir problem var.

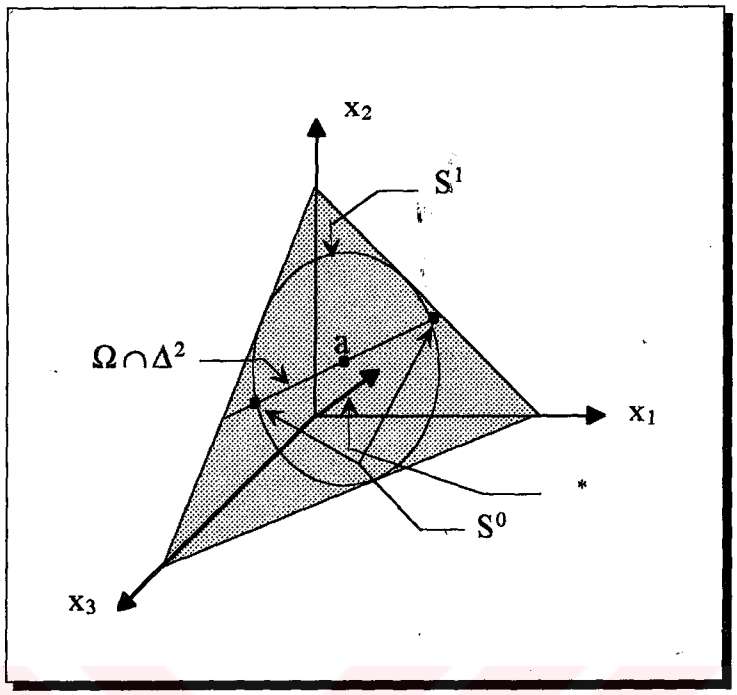
4.1 Başlangıç Adımı

Şimdi (*A*) ve (*B*) kabullerinin her ikisini de sağlayan ve daha küçük bir amaç fonksiyonu değeri veren yeni bir nokta bulmalıyız. Bu düşünce ile Karmarkar a_0 'dan hareket edeceği en iyi doğrultuyu bulmak için amaç fonksiyonunu aşağıdaki tarzda kullanmıştır.

$c \in R^{n+1}$ daha küçük boyutlu $\Omega \cap \Delta^n$ bölgesinde bir doğrultu vermediğinden c bölge üzerine ortogonal olarak projekte edilir. O zaman bu projekte edilmiş c* vektörü zıt doğrultudaki (Maximize etseydik aynı doğrultudaki) istediğimiz noktayı işaret eder. Bir iç noktadan başka bir iç noktaya geçmek algoritma için yeterlidir. Karmarkar, ayrıca problemi aşağıdaki şekilde basitleştirmiştir.

a_0 merkezli Δ^n de bir küre yaz. Bu kürenin Ω ile arakesiti daha küçük boyutlu yine bir küre olacaktır. (Çünkü, R^{n+1} in bir alt uzayıdır ve a_0 kürenin ve Ω 'nın merkezidir.) akat 2. bölümde bahsettiğimiz gibi bu halde bu minimizasyon problemi açıktır. Böylece hareket edeceğimiz noktayı buluruz. Garantili bir şekilde minimuma yaklaşmayı sağlamak ve simplexin dışına çıkmamak için Karmarkar, her adımda mümkün olduğu kadar uzağa hareket etmemiştir. Bu suretle açıkladığımız, yazılmış küreden daha küçük bir küreyi etkili bir şekilde kullanmıştır.

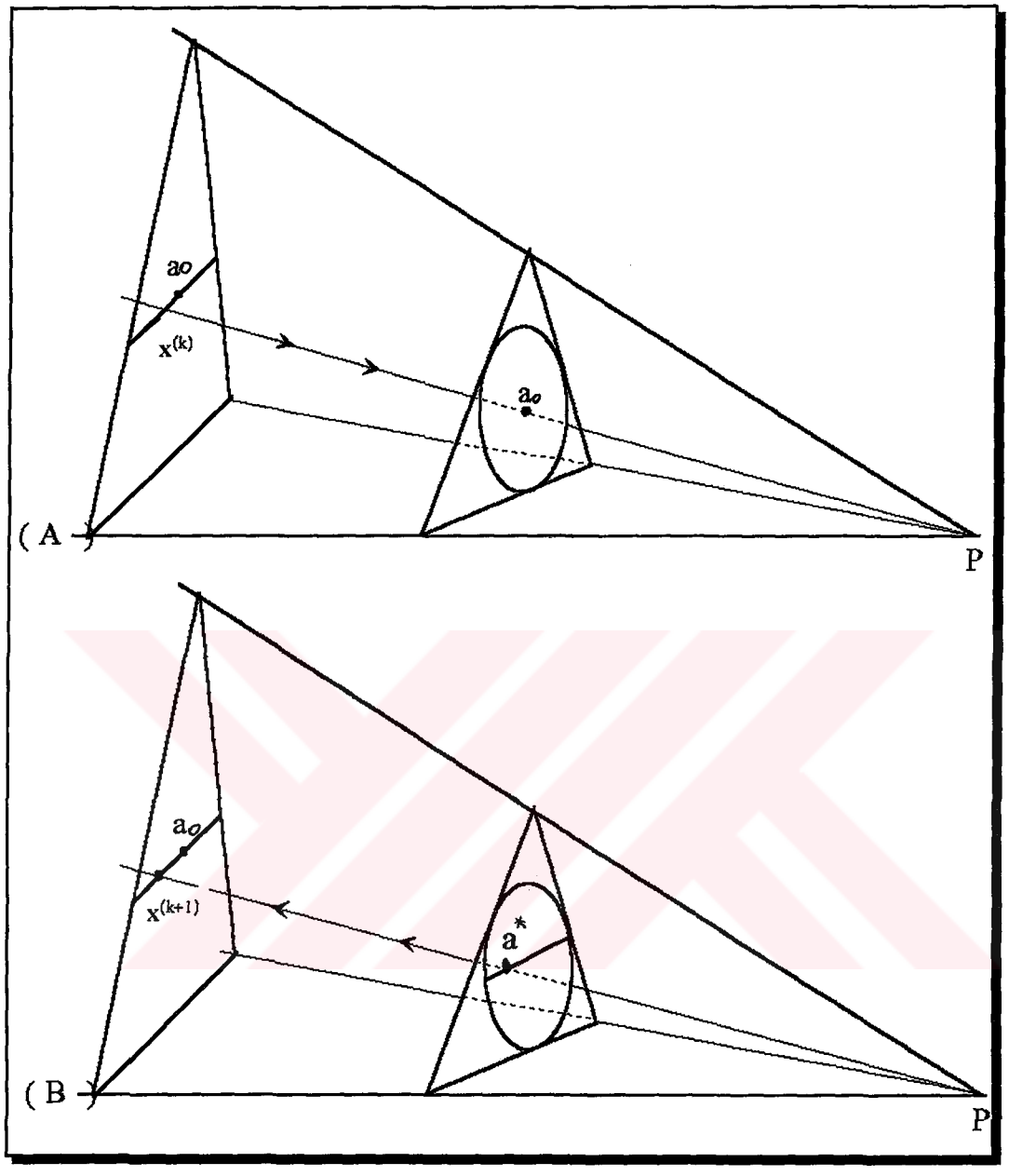
Şekil 4.2 de bu işlemin problemimize nasıl uygulandığını gösterilmiştir. Problem ve çizimimizin her ikiside 3 boyutlu olduğundan $\Omega \cap \Delta^2$ 'deki sıfır boyutlu son küre oldukça açıktır. Bununla birlikte bu son küre simplexten daha küçüktür. ve iki boyutludur, ayrıca bu durum geneldir.



Şekil 4.2:**Başlangıç Adımı:** c* vektörü $(3,3,-1)^T$ 'nin $\Omega \cap \Delta^2$ bölgesi üzerine bir ortogonal projeksiyondan elde edilir. S^1 küresi x_1 - x_2 düzleminde a_0 merkezinde çizilmiş bir küreyi göstermektedir. Bunun için Ω ile arakesiti yine a_0 merkezli daha küçük boyutlu S^0 küresidir. Karmarkar algoritması, S^0 üzerinde minimize edilen noktaya doğru yolun bir parçası olan $x^{(1)}$ noktasını sonraki nokta olarak seçmektedir.

4.2 Genel İterasyon Adımı

Başlangıç noktasına $x^{(0)}$ dersek (şöyle ki; $x^{(0)}=a_0$ ve başlangıç adımının sonucu $x^{(1)}$ dir.) o zaman $k>0$ için $x^{(k)}$ 'dan $x^{(k+1)}$ 'in oluşturulmasını tanımlamamız gerekir. Bu noktaların herbiri $\Omega \cap \Delta^2$ in içindedirler ve başlangıç adımına benzer şekilde $x^{(k)}$ dan elde edilirler. Karmarkar Δ^n 'den Δ^n 'e $x^{(k)}$ 'yı a_0 merkezine götüren ve simplexin köşelerini sabitleştiren bir projektif dönüşüm uyguluyor. akat şimdi, başlangıç adım işlemi dönüştürülmüş simplexte daha iyi nokta bulmak için uygulanabilir, ve o zaman $x^{(k+1)}$ orijinal simplexe geri dönen ters dönüşümle bulunur. Şekil 4.3 te problemimiz için böyle bir projektif dönüşümün nasıl oluşturulabileceği gösterilmiştir. Bu örnek için, $x^{(k)}$ 'ların sınır üzerindeki gerçek çözüme gitgide yaklaştığını görebiliriz .



Şekil 4.3:**Genel Adım:** Bir projektif dönüşüm. Δ^2 den Δ^2 ye bir projektif dönüşüm görmek için farklı boyutlarda ve hedeflerde 2 ayrı simplex tasavvur edelim. Şöyle ki, uygun köşeleri birleştiren doğruların hepsi bir P noktasında kesişsin ve soldaki (ilk) simplexteki $x^{(k)}$ 'nın görüntüsü ikinci simplexte a_0 'dır. (**A**) İkinci simplexteki optimizasyondan sonra a* (algoritmanın özet kısmında y olarak gösterilmiştir.) ve P çözüm noktaları birinci simplexin $x^{(k+1)}$ deki birinci simplexi kesen bir doğru belirler. (**B**) Projektif geometri ile meşgul olanlar bu şekilleri Desargues Teoreminin ispatının bir parçası olarak hatırlayacaklardır.

$x^{(k)}$ 'lar dönüşümle a_0 'a gittiklerinden, $x^{(k)}$ 'dan istenen köşe nokta olan $(0, \frac{1}{4}, \frac{3}{4})$ 'e, doğru parçası her iterasyonda uzatılır ve yeni $x^{(k+1)}$, istenen köşe noktasına biraz daha yaklaşır. Fakat asla bu sınır noktasına ulaşamaz.

Her adımda başlangıç bölgesine geri dönen ve böylece kesme hatalarını biriktirmeyen algoritma, stabledir. ve algoritma, $x^{(k)}$ tekrar feasible kalabildiği sürece devam edebilir.

4.3 Cebirsel Açıklama

Şimdi, geometrik açıklamamızı tamamlayacak işlemin çeşitli bölümleri için cebirsel formülleri verelim. $x^{(k)}$ 'yı bildiğimizi ve $x^{(k+1)}$ 'i bulmak istediğimizi farzedelim.

İlk olarak bize $x^{(k)}$ 'yı a_0 'a götüren bir $T:\Delta^n \to \Delta^n$ projektif dönüşümü gerekmektedir. Bunun için

$D = D(x^{(k)})$

$T(x) = \frac{D^{-1}x}{e^T D^{-1}x}$ olsun.

(Burada D , $x^{(k)}$ 'nın diagonal matrisi ,e ise elemanları 1 lerden oluşan n boyutlu vektördür.)

$D^{-1}x^{(k)} = e$ ve $e^T D^{-1}x^{(k)} = e^T e = n+1$ olduğundan $T(x^{(k)})=a_0$ olduğunu kolaylıkla görebiliriz. (T nin bir projektif dönüşüm olduğunu göstermek için doğruları doğrulara götürdüğünü göstermek yeterlidir.)

T(x) 'in, T(x) 'i Δ^n 'de tutan bir normalizasyonla birlikte gerçekten $D^{-1}x$ olduğuna dikkat edelim. $\Omega = \{x | Ax = 0\}$ bir affine uzay ve projektif dönüşümlerde affine uzayları muhafaza ettiklerinden, $\Omega' = T(\Omega)$ de bir affine uzaydır. ve Ω' AD 'nin bir sıfır uzayıdır. dir. (sıfır uzay, verilmiş bir dönüşümle, vektörleri sıfır vektörüne dönüştüren, vektörlerin bir alt uzayıdır.) Çünkü ancak ve ancak AD(T(x))=0 ise Ax=0 dır. (Bunu, $DD^{-1}=I_{n+1}$ olduğuna dikkat ederek hemen T(x)'in tanımında yerine yazarak görebiliriz).

B birlerden oluşan bir satırla alttan artırılmış AD matrisi olsun. (yani, B $\Omega' \cap \Delta^n$ i tanımlayacaktır. Çünkü $\Omega' \cap \Delta^n$, bileşenlerinin toplamı 1 olan bir satır vektörü ile Ω' nun birleşimidir.)

$CP = [I_n - B^T(BB^T)^{-1}B]Dc$, B 'nin sıfır uzayı üzerine C nin projeksiyonu olsun. Δ^n 'de çizilmiş en büyük kürenin yarıçapı $r = \frac{1}{\sqrt{n(n-1)}}$ olduğundan, -CP doğrultusunda a_0 'dan, r den büyük olmayan bir mesafe ilerleyerek amaç fonksiyonumuzu iyileştirebiliriz.

Karmarkar 0 ile 1 arasında bir α parametresi takdim ederek ($\alpha = \frac{1}{4}$ yada $\alpha = \frac{n-1}{3n}$ olabilir) yeni $y = a_0 - \frac{\alpha r}{\|CP\|}CP$ noktasını bulmak için amaç fonksiyonunun değerini azaltan doğrultuda a_0 'dan αr mesafesinde ilerler. (Bu parametre sayesinde yeni bulunacak noktanın feasible bölgenin dışına çıkması önlenmiş olur.) Daha sonra, $x^{(k+1)} = \frac{Dy}{e^T Dy}$ ters dönüşümü ile orijinal bölgemiz $\Omega \cap \Delta^n$ 'e dönülür.

4.4 Esas Teorem

Projektif dönüşümler problemin amaç fonksiyonu gibi lineer fonksiyonları muhafaza etmezler. Bu sebeple Karmarkar'ın esas sonucu, amaç fonksiyonunu bir potansiyel fonksiyon ile birleştirmesidir. Daha sonra her iterasyonda garantili bir miktarda potansiyel fonksiyondaki azalmanın $\frac{c^T x}{c^T a_0}$ oranındaki azalmaya eşit olduğunun ispatlanmasıdır.

Problem(4.1) in verilen amaç fonksiyonuna uygun potansiyel f(x) fonksiyonu,

$$f(x) = \sum_{i=1}^{n+1} \ln\left(\frac{c^T x}{c^T x_i}\right) \qquad (4.2)$$

olsun. Burada ln(r), r doğal sayısının doğal logaritmasıdır. x_i ise (n+1) boyutlu x noktasının i. bileşenidir. Karmarkar, bu potansiyel fonksiyonu projektif T dönüşümü altında göz önüne almıştır. Dönüştürülmüş $\Omega' \cap \Delta^n$ uzayında, αr yarıçaplı kürenin üzerine çizilmiş $(Dc)^T(T(x))$ 'i minimize eden nokta, bir sıfır değeri yada en azından $\delta > 0$ ile dönüştürülmüş potansiyel fonksiyonu azalttığını göstermiştir. Buradaki δ sabiti α ya bağlıdır. Özel olarak $\alpha = \frac{1}{4}$ ise $\delta \geq \frac{1}{8}$ dir. Daha sonra ters dönüşüm uygulanarak aşağıdaki Karmarkar teoremini elde etmiştir.

4.5 Karmarkar Teoremi

Ya $c^T x^{(k+1)} = 0$ yada $f(x^{(k+1)}) \le f(x^{(k)}) - \delta$ dır.
Burada δ, α ya bağlı bir sabittir. ve $\alpha = \frac{1}{4}$ için $\delta \ge \frac{1}{8}$ dir.

Algoritmanın k iterasyon için çalıştığını ve $c^T x^{(k)} > 0$ olduğunu farzedelim. Bu halde akla hemen çözümün ne kadar yakınına geldik? sorusu gelir. Teoremi tekrarlı bir şekilde uygulayarak $f(x^{(k)}) \le f(x^{(k-1)}) - \delta \le \ldots \le f(x^{(0)}) - k\delta$ elde ederiz. $x^{(0)} = a_0$ olduğundan buradan

$$\sum_{i=1}^{n+1} \left[\ln(c^T x^{(k)}) - \ln(x_i^{(k)}) \right] \le \sum_{i=1}^{n+1} \left[\ln(c^T a_0) - \ln(\frac{1}{n+1}) \right] - k\delta \text{ dır. Düzenlersek}$$

$$(n+1)\ln(c^T x^{(k)}) - \sum_{i=1}^{n+1} \ln(x_i^{(k)}) \le \left[(n+1)\ln(c^T a_0) + \ln(n+1) \right] - k\delta \text{ olur. Bu ise bize}$$

$$\ln(\frac{c^T x^{(k)}}{c^T a_0}) \le \ln(n+1) + \frac{1}{n+1} \sum_{i=1}^{n+1} \ln(x_i^{(k)}) - \frac{k\delta}{n+1} \qquad (4.3)$$

ifadesini verir.

$x^{(m)}$, Δ^n in içinde olduğundan $x^{(m)}$ nin bütün bileşenleri $0 < x_i^{(k)} < 1$ dir. Bundan dolayı $\sum \ln(x_i^{(k)})$ terimi negatiftir. O halde

$$\ln \frac{c^T x^{(k)}}{c^T a_0} < \ln(n+1) - \frac{k\delta}{n+1} \text{ yazılabilir.}$$

$$k = \frac{(n+1)(q + \ln(n+1))}{\delta}$$

denklemi, hesaplama ihtiyacı duyduğumuz Karmarkar algoritmasının iterasyonlarının sayısını verir. Bunun doğru olduğunu görmek için k yı eşitsizlikte yerine yazarsak $\ln \frac{c^T x^{(k)}}{c^T a_0} < -q$ olur. Her iki tarafın exponansiyelini alarak ve $q<0$ için $e^{-q} < 2^{-q}$ olduğuna dikkat ederek

$$\frac{c^T x^{(k)}}{c^T a_0} < e^{-q} < 2^{-q} \qquad \text{yazabiliriz.}$$

k için üsteki tanım $O((n+1)(q+\ln(n+1)))$ mertebesinde olan Karmarkar algoritmasının yaklaşık iterasyon sayısını verir.

4.6 Karmarkar Algoritmasının Özeti

Adım 1 :Simplexin yarıçapını $r = \frac{1}{\sqrt{n(n-1)}}$ ile hesapla.

$a_0 = (\frac{1}{n}, ..., \frac{1}{n})^T$ yi oluştur.

$0 < \alpha < 1$ olmak üzere bir α sabiti belirle. ($\alpha = \frac{1}{4}$ yada $\alpha = \frac{n-1}{3n}$ olması tavsiye edilir.)

T toleransını belirle(genellikle 10^{-6} ve 10^{-8} mertebesinde olması tercih edilir.)

İterasyon sayacı k 'yı sıfırla (k=0) ve simplexin merkezi a_0 'ı x_0 'a ($x_0=a_0$) aktar.

Adım 2 :$D_k=Diag\{x_1^{(k)},x_2^{(k)},...,x_n^{(k)}\}$ yı teşkil et.

$B = \begin{bmatrix} AD_k \\ 1 \end{bmatrix}$ matrisini hesapla.

$CP=[I_n - B^T(BB^T)^{-1}]Dc$ ile projekte edilmiş maliyet vektörünü bul.

ve CP nin uzunluğunu $\|CP\| = \sqrt{(cp_1)^2 + ... + (cp_n)^2}$ ile hesapla

Adım 3 :$y = a_0 - \frac{\alpha r}{\|CP\|} CP$ yi hesapla.

Adım 4 :Ters dönüşümle $x^{(k+1)} = \frac{D_k y}{1 D_k y}$ 'i ve buna karsılık gelen amaç fonksiyonu değerini $c^T x^{(k+1)}$ ile hesapla.

Adım 5 :$c^T x^{(k+1)} < T$ ise verilen toleransta optimum çözüme yaklaşılmıştır. Çözüm $c^T x^{(k+1)}$ amaç fonksiyonu değeri ile $x^{(k+1)}$ vektörüdür. Aksi halde $x^{(k+1)}$ 'i $x^{(k)}$ ya aktar ve Adım 2 'ye dön.

4.7 Karmarkar Algoritmasının Bilgisayar Yorumu

Aşağıdaki Program Qbasic programlama dilinde yazılmıştır. Veri olarak Daha önce bahsettiğimiz problemi kullanmaktadır. N ve M sayıları ile Datalar (kısıt matrisi ve Amaç fonksiyonu katsayıları) değiştirilerek diğer uygulamalara uyarlanabilir.

```
CLS
N = 3
M = 1
M1 = M + 1
M2 = 2 * M1
DIM A0(N), DIAG(N), X(N), A(M, N), CC(N), CP(N), B(M1, N),  B1(M1,M2),
      B2(N, M1), B3(N, N), Y(N), DY(N)
FOR C = 1 TO N
      A0(C) = 1 / N
      X(C) = A0(C)
      Y(C) = A0(C)
NEXT C
T = .00001
ALFA = .25
ITERASYON = 0
DATA 2,-3,1
FOR R = 1 TO M
      FOR C = 1 TO N
            READ A(R, C)
      NEXT C
NEXT R
DATA 3,3,-1
FOR C = 1 TO N
      READ CC(C)
```

```
NEXT C
Vilk = 0
FOR C = 1 TO N
    Vilk = Vilk + CC(C) * A0(C)
NEXT C
Vyeni = Vilk
WHILE (Vyeni / Vilk) * 100 > T
    PRINT USING "-----###.İterasyon-----"; ITERASYON
    FOR C = 1 TO N
    PRINT USING "X#= ###.##########"; C; X(C)
    NEXT C
    PRINT USING "Vyeni/Vilk=###.##########"; Vyeni / Vilk
    ITERASYON = ITERASYON + 1
    FOR C = 1 TO N
        DIAG(C) = X(C)
    NEXT C
    FOR R = 1 TO M
        FOR C = 1 TO N
            B(R, C) = A(R, C) * DIAG(C)
        NEXT C
    NEXT R
    FOR R = 1 TO M1
        FOR C = 1 TO M2
            B1(R, C) = 0
        NEXT C
    NEXT R
    FOR R = 1 TO N
        FOR C = 1 TO M1
            B2(R, C) = 0
        NEXT C
```

```
NEXT R
FOR R = 1 TO N
        FOR C = 1 TO N
            B3(R, C) = 0
        NEXT C
        CP(R)=0
        B(M1, R) = 1
NEXT R
FOR R = 1 TO M1
        FOR C = 1 TO M1
            FOR I = 1 TO N
                B1(R, C) = B1(R, C) + B(R, I) * B(C, I)
            NEXT I
      NEXT C
NEXT R
FOR I = 1 TO M1
        B1(I, I) = 1
NEXT I
FOR R = 1 TO M1
        IF B1(R, R) <> 0 THEN 550
        I = R + 1
520     IF I > M1 THEN PRINT ".....Hata.....!!!! BB^T Tekil....."
                          END
        IF B1(I, R) = 0 THEN I = I + 1: GOTO 520
        FOR C = 1 TO M2
            SWAP B1(R, C), B1(I, C)
        NEXT C
550     FOR I = R + 1 TO M1
            Z = B1(I, R) / B1(R, R)
            FOR C = 1 TO M2
```

```
                B1(I, C) = B1(I, C) - z * B1(R, C)
            NEXT C
        NEXT I
NEXT R
FOR R = M1 TO 2 STEP -1
        FOR  I = R - 1 TO 1 STEP -1
            Z = B1(I, R) / B1(R, R)
            FOR C = R TO M2
                B1(I, C) = B1(I, C) - z * B1(R, C)
            NEXT C
        NEXT I
NEXT R
FOR R = 1 TO M1
    Z = B1(R, R)
    FOR C = 1 TO M2
        B1(R, C) = B1(R, C) / z
    NEXT C
NEXT R
FOR R = 1 TO N
    FOR C = 1 TO M1
        FOR J = 1 TO M1
            B2(R, C) = B2(R, C) + B(J, R) * B1(J, C + M1)
        NEXT J
    NEXT C
NEXT R
FOR R = 1 TO N
    FOR C = 1 TO N
        FOR J = 1 TO M1
            B3(R, C) = B3(R, C) + B2(R, J) * B(J, C)
        NEXT J
```

```
        NEXT C
    NEXT R
    FOR R = 1 TO N
        B3(R, R) = B3(R, R) - 1
    NEXT R
    FOR R = 1 TO N
        FOR C = 1 TO N
            B3(R, C) = -1 * B3(R, C)
        NEXT C
    NEXT R
    FOR R = 1 TO N
        FOR C = 1 TO N
            B3(R, C) = B3(R, C) * DIAG(C)
        NEXT C
    NEXT R
    FOR R = 1 TO N
        FOR C = 1 TO N
            CP(R) = CP(R) + B3(R, C) * CC(C)
        NEXT C
    NEXT R
'Projeksiyonun boyu AA olarak hesaplanıyor...'
    AA = 0
    FOR C = 1 TO N
        AA = SQR(AA) + CP(C) * CP(C)
    NEXT C
    FOR C = 1 TO N
        Y(C) = A0(C) - (ALFA * (1 / SQR(N * (N - 1)) / AA) * CP(C)
    NEXT C
'Burada Ters dönüşüm (D*Y/1*D*Y) ile y lerden x lere geçiliyor..."
    FOR C = 1 TO N
```

```
            DY(C) = DIAG(C) * Y(C)
        NEXT C
        Vyeni = 0
        FOR C = 1 TO N
            Vyeni = Vyeni + CC(C) * X(C)
        NEXT C
    WEND
    PRINT : PRINT ".....Toleransa Erişildi.....! Vyeni/Vilk=";
    PRINT USING "###.################"; Vyeni / Vilk: PRINT
    PRINT USING "-----###.İterasyon-----"; ITERASYON
    FOR C = 1 TO N
        PRINT USING "X#= ###.##########"; C; X(C)
    NEXT C
    PRINT USING "Vyeni= ###.########## Amaç Fonmsiyonu Değeridir"; Vyeni
    PRINT "-------------------------"
    PRINT USING "Tolerans =###.#######     Vyeni/vilk=###.#####"; T;Vyeni / Vilk
    END
```

4.8 Uygulamalar

Uygulama 1: Min $z = x_2$

Kısıtlar $x_1+x_2-2x_3=0$

$x_1+x_2+x_3=1$

$x_i \geq 0$

Çözüm:Burada n=3,m=1dir.

$$r = \frac{1}{\sqrt{n(n-1)}} = \frac{1}{\sqrt{3.2}} = \frac{1}{\sqrt{6}}$$

$$a_0 = (\frac{1}{n},...,\frac{1}{n})^T = (\frac{1}{3},\frac{1}{3},\frac{1}{3})^T$$

(Gerçek optimal çözüm sıfır amaç fonksiyonu değeri ile

$x^* = (\frac{2}{3}, 0, \frac{1}{3})^T$ dir.)

$T=10^{-6}$ olsun

$\alpha = \frac{n-1}{3n} = \frac{3-1}{3.3} = \frac{2}{9}$ alalım.

k=0 ve $x_0 = a_0$ ile iterasyona başlayalım.

1.İterasyon: $D_0=Diag\{\frac{1}{3},\frac{1}{3},\frac{1}{3}\}$ dır.

$$AD_0 = \begin{bmatrix} 1 & 1 & -2 \end{bmatrix} \begin{bmatrix} \frac{1}{3} & 0 & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{3} \end{bmatrix} = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{-2}{3} \end{bmatrix}$$

$$B = \begin{bmatrix} AD_k \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{-2}{3} \\ 1 & 1 & 1 \end{bmatrix}$$

$CP=[I-B^T(BB^T)^{-1}B]Dc$ yi oluşturalım

$$CP = \left[I_3 - \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{-2}{3} \\ 1 & 1 & 1 \end{bmatrix}^T \left(\begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{-2}{3} \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{-2}{3} \\ 1 & 1 & 1 \end{bmatrix}^T \right)^{-1} \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{-2}{3} \\ 1 & 1 & 1 \end{bmatrix} \right] \begin{bmatrix} \frac{1}{3} & 0 & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{3} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$CP = \left[I_3 - \begin{bmatrix} \frac{1}{3} & 1 \\ \frac{1}{3} & 1 \\ \frac{-2}{3} & 1 \end{bmatrix} \left(\begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{-2}{3} \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{3} & 1 \\ \frac{1}{3} & 1 \\ \frac{-2}{3} & 1 \end{bmatrix} \right)^{-1} \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{-2}{3} \\ 1 & 1 & 1 \end{bmatrix} \right] \begin{bmatrix} 0 \\ \frac{1}{3} \\ 0 \end{bmatrix}$$

$$= \left[I_3 - \begin{bmatrix} \frac{1}{3} & 1 \\ \frac{1}{3} & 1 \\ \frac{-2}{3} & 1 \end{bmatrix} \begin{bmatrix} \frac{2}{3} & 0 \\ 0 & 3 \end{bmatrix}^{-1} \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{-2}{3} \\ 1 & 1 & 1 \end{bmatrix} \right] \begin{bmatrix} 0 \\ \frac{1}{3} \\ 0 \end{bmatrix}$$

$$= \left[I_3 - \begin{bmatrix} \frac{1}{3} & 1 \\ \frac{1}{3} & 1 \\ \frac{-2}{3} & 1 \end{bmatrix} \begin{bmatrix} \frac{3}{2} & 0 \\ 0 & \frac{1}{3} \end{bmatrix} \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{-2}{3} \\ 1 & 1 & 1 \end{bmatrix} \right] \begin{bmatrix} 0 \\ \frac{1}{3} \\ 0 \end{bmatrix}$$

$$= \left[I_3 - \begin{bmatrix} \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} \\ -1 & \frac{1}{3} \end{bmatrix} \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{-2}{3} \\ 1 & 1 & 1 \end{bmatrix} \right] \begin{bmatrix} 0 \\ \frac{1}{3} \\ 0 \end{bmatrix} = \left[\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \right] \begin{bmatrix} 0 \\ \frac{1}{3} \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{2} & \frac{-1}{2} & 0 \\ \frac{-1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ \frac{1}{3} \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{-1}{6} \\ \frac{1}{6} \\ 0 \end{bmatrix}, \qquad \|CP\| = \sqrt{\left(\frac{-1}{6}\right)^2 + \left(\frac{1}{6}\right)^2 + 0^2} = \frac{\sqrt{2}}{6}$$

$$y = a_0 - \frac{\alpha r}{\|CP\|} CP = \begin{bmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{bmatrix} - \left(\frac{2}{9}\right)\left(\frac{1}{\sqrt{6}}\right)\left(\frac{6}{\sqrt{2}}\right) \begin{bmatrix} \frac{-1}{6} \\ \frac{1}{6} \\ 0 \end{bmatrix} = \begin{bmatrix} 0.397484 \\ 0.269183 \\ 0.333333 \end{bmatrix} \quad \text{olur.}$$

Buradan $x_{k+1} = \frac{D_0 y}{1 D_0 y}$ ters dönüşümüyle Y lerden tekrar x koordinatlarına geçelim.

$$D_0 y = \begin{bmatrix} \frac{1}{3} & 0 & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{3} \end{bmatrix} \begin{bmatrix} 0.397484 \\ 0.269183 \\ 0.333333 \end{bmatrix} = \begin{bmatrix} 0.1324946 \\ 0.0897276 \\ 0.1111111 \end{bmatrix} \quad , \quad 1D_0 y = \frac{1}{3}$$

$$x_1 = \frac{\begin{bmatrix} 0.1324946 \\ 0.0897276 \\ 0.1111111 \end{bmatrix}}{\frac{1}{3}} = \begin{bmatrix} 0.397484 \\ 0.269183 \\ 0.333333 \end{bmatrix} = y \text{ (Daima } x_1 = y \text{ dir.)}$$

Amaç fonksiyonu değeri ise $c^T x_1 = 0.269183$ olur.

2.İterasyon: $D_1 = \text{Diag}\{0.397484, 0.269183, 0.333333\}$ dir

$$B = \begin{bmatrix} AD_1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.397484 & 0.269183 & -0.666666 \\ 1 & 1 & 1 \end{bmatrix}$$

$$CP = [I - B^T (BB^T)^{-1} B] Dc = \begin{bmatrix} -0.1324029 \\ 0.1505548 \\ -0.0181517 \end{bmatrix} \quad \text{olur.}$$

$$\|CP\| = \sqrt{(-0.132409)^2 + (0.1505548)^2 + (-0.0181517^2} = 0.2013121 \quad \text{olur.}$$

buradan,

$$y = a_0 - \frac{\alpha r}{\|CP\|} CP = \begin{bmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{bmatrix} - \left(\frac{2}{9}\right)\left(\frac{1}{\sqrt{6}}\right)(0.2013121) \begin{bmatrix} -0.1324029 \\ 0.1505548 \\ -0.0181517 \end{bmatrix} = \begin{bmatrix} 0.3930009 \\ 0.2654855 \\ 0.3415134 \end{bmatrix}$$

olur. Ters dönüşümle y uzayından tekrar x uzayına geçelim.

$$x_2 = \frac{D_1 y}{1 D_1 y} = \frac{\begin{bmatrix} 0.1562112 \\ 0.0714641 \\ 0.1138378 \end{bmatrix}}{0.3415131} = \begin{bmatrix} 0.457409 \\ 0.209258 \\ 0.333333 \end{bmatrix} \quad \text{olur.}$$

Amaç fonksiyonu değeri ise $c^T x_2=0.209258$ olur.

Bu şekilde iterasyonlara devam edilirse 36. iterasyonda istenen toleransta $x_{36}=(0.66666639, 0.00000025, 0.33333334)^T$ çözümü elde edilir. Bu çözüme karşılık gelen amaç fonksiyonu değeri ise $c^T x_{36}=0.00000025$ dır.

Uygulama 2: Min $z = -x_1-2x_2+4x_3$

Kısıtlar $x_1-x_2-2x_3 \quad 2x_5=0$

$x_1+2x_2 \quad +x_4-4x_5=0$

$x_1+x_2+x_3+x_4+x_5=1$

ve $x_i \geq 0$

Çözüm:Burada n=5,m=2dir.

$$r=\frac{1}{\sqrt{n(n-1)}}=\frac{1}{\sqrt{5.4}}=\frac{1}{\sqrt{20}}$$

$$a_0=(\frac{1}{n},\ldots,\frac{1}{n})^T=(\frac{1}{5},\frac{1}{5},\frac{1}{5},\frac{1}{5},\frac{1}{5})^T$$

(Gerçek optimal çözüm sıfır amaç fonksiyonu değeri ile $x^*=(0, 0.4, 0.4, 0, 0.2)^T$ dir.)

$T=10^{-4}$ olsun

$\alpha=\frac{n-1}{3n}=\frac{5-1}{3.5}=\frac{4}{15}$ alalım.

$k=0$ ve $x_0=a_0$ ile iterasyona başlayalım.

1.İterasyon: $D_0=\text{Diag}\{\frac{1}{5},\frac{1}{5},\frac{1}{5},\frac{1}{5},\frac{1}{5}\}$ dır.

$$AD_0=\begin{bmatrix} 1 & -1 & 2 & 0 & -2 \\ 1 & 2 & 0 & 1 & -4 \end{bmatrix}\begin{bmatrix} \frac{1}{5} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{5} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{5} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{5} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{5} \end{bmatrix}=\begin{bmatrix} \frac{1}{5} & \frac{-1}{5} & \frac{2}{5} & 0 & \frac{-2}{5} \\ \frac{1}{5} & \frac{2}{5} & 0 & \frac{1}{5} & \frac{-4}{5} \end{bmatrix}$$

$$B=\begin{bmatrix} AD_0 \\ 1 \end{bmatrix}=\begin{bmatrix} \frac{1}{5} & \frac{-1}{5} & \frac{2}{5} & 0 & \frac{-2}{5} \\ \frac{1}{5} & \frac{2}{5} & 0 & \frac{1}{5} & \frac{-4}{5} \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$CP=[I-B^T(BB^T)^{-1}B]Dc$ yi olusturalım.

$$CP=\begin{bmatrix} -0.0435088 \\ -0.0715790 \\ -0.0236257 \\ 0.1483040 \\ -0.0095906 \end{bmatrix}$$ olur.

$$\|CP\|=\sqrt{(-0.0435088)^2+(-0.071579)^2+(-0.0236257)+(0.148304)^2+(-0.0095906)^2}$$

$=0.1722231$ olur.

$$y=a_0-\frac{\alpha r}{\|CP\|}CP=\begin{bmatrix} 1/5 \\ 1/5 \\ 1/5 \\ 1/5 \\ 1/5 \end{bmatrix}-\left(\frac{2}{9}\right)\left(\frac{1}{\sqrt{20}}\right)\left(\frac{1}{0.1722}\right)\begin{bmatrix} -0.0435088 \\ -0.071579 \\ -0.0236257 \\ 0.148304 \\ -0.0095906 \end{bmatrix}=\begin{bmatrix} 0.2141224 \\ 0.2232337 \\ 0.2076686 \\ 0.1518622 \\ 0.2931130 \end{bmatrix}$$

olur.

Ters donüşümle esas koordinatlarımız olan x lere geçelim.

1. iterasyon için daima $x_1=\frac{D_0y}{1D_0y}\equiv y$ dir.

Buradan $x_1=\begin{bmatrix} 0.2141224 \\ 0.2232337 \\ 0.2076686 \\ 0.1518622 \\ 0.2931130 \end{bmatrix}$ ve buna karşılık gelen amaç fonksiyonu değeri ise

$cx_1=0.15186197$ dir.

Bu şekilde iterasyonlara devam edilirse 2. iterasyonda cx_2=0.1128728 amaç fonksiyonu değerini veren x_2=(0.2253278, 0.2421542, 0.2140314, 0.1128728, 0.2056227)^T çözümü 3. iterasyonda x_3=(0.2339081, 0.2568670, 0.2190438, 0.0826162, 0.2075646)^T çözümü ve cx_3=0.0826162 amaç fonksiyonu değeri ve nihayet 24. iterasyonda cx_{24}= 0.00005993 <T şartını sağlayan x_{24}=(0.2569092, 0.2972365, 0.2329669, 0.00005993, 0.2128274)^T çözümü elde edilir.

Uygulama 3: Min $z = x_1 + 3x_2 - 3x_3$

Kısıtlar $x_2 - x_3 = 0$

$x_1 + x_2 + x_3 = 1$

ve $x_i \geq 0$

Çözüm:Burada n=3,m=1dir.

$$r = \frac{1}{\sqrt{n(n-1)}} = \frac{1}{\sqrt{3.2}} = \frac{1}{\sqrt{6}}$$

$$a_0 = (\frac{1}{n}, ..., \frac{1}{n})^T = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})^T$$

(Gerçek optimal çözüm sıfır amaç fonksiyonu değeri ile

$x^*=(0, 0.5, 0.5)^T$ dir .)

$T=10^{-4}$ olsun

$\alpha = \frac{n-1}{3n} = \frac{3-1}{3.3} = \frac{2}{9}$ alalım.

k=0 ve $x_0 = a_0$ ile iterasyona başlayalım.

1.İterasyon: D_0=Diag$\{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\}$ dır.

$$AD_0 = \begin{bmatrix} 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} \frac{1}{3} & 0 & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{3} \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{3} & \frac{-1}{3} \end{bmatrix}$$

$$B = \begin{bmatrix} AD_0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{3} & \frac{-1}{3} \\ 1 & 1 & 1 \end{bmatrix}$$

$CP=[I-B^T(BB^T)^{-1}B]Dc$ yi olusturalım.

$$CP = \begin{bmatrix} \frac{2}{3} & \frac{-1}{3} & \frac{-1}{3} \\ \frac{-1}{3} & \frac{1}{6} & \frac{1}{6} \\ \frac{-1}{3} & \frac{1}{6} & \frac{1}{6} \end{bmatrix} \begin{bmatrix} \frac{1}{3} \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} \frac{2}{9} \\ \frac{-1}{9} \\ \frac{-1}{9} \end{bmatrix} \quad \text{olur.}$$

Buradan da $\|CP\| = \sqrt{\left(\frac{2}{9}\right)^2 + \left(\frac{-1}{9}\right)^2 + \left(\frac{-1}{9}\right)^2} = \frac{\sqrt{6}}{9}$ elde edilir.

$$y = a_0 - \frac{\alpha r}{\|CP\|} CP = \begin{bmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{bmatrix} - \left(\frac{2}{9}\right)\left(\frac{1}{\sqrt{6}}\right)\left(\frac{9}{\sqrt{6}}\right) \begin{bmatrix} \frac{2}{9} \\ \frac{-1}{9} \\ \frac{-1}{9} \end{bmatrix} = \begin{bmatrix} \frac{7}{27} \\ \frac{10}{27} \\ \frac{10}{27} \end{bmatrix} \text{ olur.}$$

Ters donüşümle esas koordinatlarımız olan x lere geçelim.

1. iterasyon için $x_1 = y$ olduğundan $x_1 = (\frac{7}{27}, \frac{10}{27}, \frac{10}{27})^T$ buna karşılık gelen amaç fonksiyonu değeri ise $cx_1 = (\frac{7}{27} + 3 * \frac{7}{27} - 3 * \frac{7}{27}) \cong 0.259259$ dır.

<u>2.İterasyon</u>: $D_1 = Diag\{\frac{7}{27}, \frac{10}{27}, \frac{10}{27}\}$ dir

$$B = \begin{bmatrix} AD_1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & \frac{10}{27} & \frac{10}{27} \\ 1 & 1 & 1 \end{bmatrix}$$

$CP = [I - B^T(BB^T)^{-1}B]Dc$ yi hesaplayalım.

$$I - B^T(BB^T)^{-1}B = \begin{bmatrix} \frac{2}{3} & \frac{-1}{3} & \frac{-1}{3} \\ \frac{-1}{3} & \frac{1}{6} & \frac{-1}{6} \\ \frac{-1}{3} & \frac{-1}{6} & \frac{1}{6} \end{bmatrix} \qquad Dc = \begin{bmatrix} \frac{7}{27} \\ \frac{10}{9} \\ \frac{-10}{9} \end{bmatrix}$$

Buradan $CP = (\frac{14}{81}, \frac{7}{81}, \frac{7}{81})^T$ ve $\|CP\| = \sqrt{\left(\frac{14}{81}\right)^2 + \left(\frac{7}{81}\right)^2 + \left(\frac{7}{81}\right)^2} = 0.2116841$ olur.

$$y = a_0 - \frac{\alpha r}{\|CP\|}CP = \begin{bmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{bmatrix} - \left(\frac{2}{9}\right)\left(\frac{1}{\sqrt{6}}\right)\left(\sqrt{\frac{294}{6561}}\right)\begin{bmatrix} \frac{14}{81} \\ \frac{7}{81} \\ \frac{7}{81} \end{bmatrix} = \begin{bmatrix} 0.2592602 \\ 0.3703704 \\ 0.3703704 \end{bmatrix}$$

Buradan ters donüşümle esas koordinatlarımız olan x lere geçelim

$$x_2 = \frac{D_1 y}{1 D_1 y} = \frac{\begin{bmatrix} 0.0672153 \\ 0.1371741 \\ 0.1371741 \end{bmatrix}}{0.3415635} = \begin{bmatrix} 0.1967872 \\ 0.4016064 \\ 0.4016064 \end{bmatrix}$$

bu çözüme karşılık gelen amaç fonksiyonu değeri ise cx_2=0.1967872 dir. Bu şekilde devam edilirse 3. iterasyonda x_3=(0.1463935, 0.4268031, 0.4268031)^T çözümü elde edilir.Nihayetinde ise 21. iterasyonun sonunda verilen toleransta x_{21}=(0.0002792, 0.49997993, 0.49992162)^T çözümü ve cx_{21}=0.0002792 amaç fonksiyonu değeri elde edilir.

5. Değiştirilmiş Karmarkar Algoritması

Daha önce de bahsettiğimiz gibi Karmarkar Algoritması LP problemlerini bazı kabuller altında çözebiliyordu. Her ne kadar problemler değişken dönüşümleri yardımıyla Karmarkar'ın istediği forma getirilebilirsede, bu işlem problemin değişken sayısını ve kısıt sayısını çok büyüteceğinden pek istenilen bir durum değildir. İşte LP problemlerini çözebilmek için aşağıda vereceğimiz Değiştirilmiş Karmarkar Algoritması (Affine-Scaling Primal Algoritması olarak ta bilinmektedir.)'nın kullanılması daha çok tercih edilmektedir. Bu sayede değişken dönüşümlerine gerek kalmayacaktır. Ayrıca bu yeni algoritmanın yorumlanması oldukça basittir ve birçok problem için oldukça iyi neticeler vermektedir. Şimdi algoritmaya geçelim.

$$\begin{aligned} &\text{Max } c^T x \\ &\text{Kısıtlar } Ax=b \\ &\qquad x\geq 0 \end{aligned} \tag{5.1}$$

şeklinde standart formda verilmiş LP problemini göz önüne alalım. Burada $x\in R^n$ ve $b \in R^m$ dir. Bir başlangıç iç çözüm vektörünün verildiğini farzedelim. Yani, $Ax_0=b$ ve $x_0>0$. Şimdi akla gelen ilk soru "x_{yeni} ile gösterilen yeni iterasyona nasıl geçeceğiz?." Yeni iterasyonun eski iterasyondan daha iyi netice vermesi gerektiği açıktır. Yeni iterasyon feasible 'liği muhafaza ederken artan doğrultuda ilerlemelidir. Yani, yeni iterasyon

$$c^T x_{yeni} \geq c^T x_0 \quad \text{(artan durum)} \tag{5.2}$$
$$Ax_{yeni} = b \quad \text{(feasible 'lik koşulu)} \tag{5.3}$$

ifadelerini sağlamalıdır. Eğer yeni iterasyon x_{yeni} , ve mevcut iterasyon x_0 , $x_{yeni} = x_0 + dx$ (burada dx adım doğrultu vektörüdür.) şeklinde iyileştirilmiş bir denklemle bağıntılı iseler o zaman aşağıdaki 2 koşulu sağlamaları gerekir.

$$c^T dx \geq 0 \quad \text{(artan durum)} \tag{5.4}$$
$$Adx=0 \quad \text{(feasible 'lik koşulu)} \tag{5.5}$$

Feasible'lik koşulu, adım doğrultu vektörü için feasible'liğin korunduğunu işaret etmektedir ve bu vektör A 'nın sıfir uzayında(null-space'inde) olmalıdır. A matrisi bütün satırlarının rankı m olan mxn lik verilmiş bir matristir. A matrisinin sıfir uzayı için bir projeksiyon operatörü

$$P = I_n - A^T(AA^T)^{-1}A \tag{5.6}$$

bağıntısı ile verilen nxn lik bir simetrik P matrisidir.

Problem maximizasyon problemi olduğundan metod, amaç fonksiyonunun gradienti doğrultusunda hareket etmelidir. (5.1) deki lineer durum için gradient basit olarak maliyet vektörü c dir. Bu doğrultu feasible'liği muhafaza edemeyebileceğinden, A 'nın sıfir uzayı üzerine projekte edilmiş gradientin doğrultusunda ilerlemelidir. Yani, dx doğrultusu dx =Pc ile verilir. Bu doğrultunun faydalı olması için (5.4) ve (5.5) denklemlerinde verilen artan durum ve feasible 'lik koşulunun her ikisini de sağlamalıdır. İkincisi yani, feasible 'liği muhafaza eden, Adx=0 olduğu kolaylıkla görülebilir. (5.4) ün sağlandığını görmek çin, $P=P^2$ olduğuna dikkat ederek,

$$c^T dx = c^T Pc = c^T P^2 c = \|Pc\|^2 \geq 0 \tag{5.7}$$

elde ederiz.

İşlemi tekrarlayarak mevcut iterasyonu polytope un içinde bir noktaya dönüştürmeliyiz. Bu merkezileştirme işlemi, bütün bileşenleri polytope'un sınırlarından bir birim mesafede bulunan çözüm vektörünün *ölçülmesi* ile yapılır. $x_0 = [x_1, x_2, ..., x_n]^T$ şeklinde feasible ve iç vektör veriliyor, bu durumda ölçülmüş x_1 vektörü

$$x_1 = D^{-1}x_0 \tag{5.8}$$

ile bulunur. Buradaki ölçülmüş vektör x_1 ve ölçülen matris D

$$x_1 = \begin{pmatrix} 1 \\ 1 \\ . \\ 1 \end{pmatrix}, \quad D = \begin{pmatrix} x_1 & 0.0 & 0.0 & 0.0 \\ 0.0 & x_2 & 0.0 & 0.0 \\ & & .... & \\ 0.0 & 0.0 & 0.0 & x_n \end{pmatrix} \tag{5.9}$$

ile tanımlıdır. Dikkat ederseniz Çözüm vektörü x daima polytope 'un içindedir.(yani, x>0) ve D matrisinin köşegen üzerindeki elemanlarının kesinlikle pozitif olduğundan tersinin alınabildiğine dikkat edin. (5.1) ile gösterilen ölçmeye tabi tutulan orijinal LP problemi

$$\begin{aligned} &\text{Min} \quad c_1^T x_1 \\ &\text{Kısıtlar} \quad A_1 x_1 = b \\ &\qquad\quad x_1 \geq 0 \end{aligned} \tag{5.10}$$

şeklinde *ölçülmüş* LP problemine dönüşür.

Burada,

$$x_1 \in R^n, \quad b \in R^m \quad \text{ve} \quad A_1 = AD, \quad c_1 = Dc \tag{5.11}$$

dır. Orijinal problem için verilmiş herhangi bir x_0 (x_0>0) başlangıç vektörü için, (5.10) ile gösterdiğimiz ölçülmüş problemin başlangıç vektörü x_1, şimdi ölçülmüş polytope 'un bütün sınırlarından bir birim mesafeye konumlanmıştır. Bu ölçülmüş LP problemi için P_1 ile verilen projeksiyon operatörü.

$$P_1 = I_n - A_1^T (A_1 A_1^T)^{-1} A_1 \tag{5.12}$$

dir. Ölçülmüş problemin artan doğrultu vektörü dx_1, ölçülmüş problemin gradienti c_1 in A 'nın sıfır uzayı üzerine projekte edilmesi ile bulunur. Bu sonuçlara göre

$$dx_1 = -P_1 c_1 = D[c - A^T (AD^2 A^T)^{-1} AD^2 c] \quad \text{olur.}$$

$$y = A^T (AD^2 A^T)^{-1} AD^2 c \tag{5.13}$$

olsun. Orijinal ölçülmemiş uzaya geri dönerek, yeni iterasyonun adım doğrultu vektörü şimdi,

$$dx = D dx_1 = -D^2 (c - A^T y) \tag{5.14}$$

ile verilir. Dikkat ederseniz ölçme işlemi açık bir şekilde gerçekleştirilmiyor, fakat işlemler dizisinin kesin olarak belirtilmesi (5.14) deki adım doğrultu vektörü dx 'in

türetilmesini sağlıyor. Simplex algoritmasında kullanılırken $c-A^Ty$ ifadesi indirgenmiş maliyet vektörünü, y de dual vektörü göstermektedir. Gerçekten (5.13) de tanımlı y değişkeni, dual değişken ve $c-A^Ty$ vektörü için bir *tahmini* göstermektedir.
(5.14) deki adım doğrultu vektörü dx ile bu doğrultuda bir adım atabiliriz ve çözüm vektörü x 'in yeni iterasyonunu elde edebiliriz. Bu ise

$$x = x_0 + \rho\alpha dx, \qquad \text{burada } \alpha > 0 \ , \quad \text{ve } 0 < \rho < 1 \tag{5.15}$$

ile verilen formülün iyileştirilmesinden(güncelleştirilmesinden) bulunur. Burada α çözüm vektörü x 'in negatif olmama kısıtlarını ihlale karşı koruyan dx doğrulutusu boyunca maximum müsade edilebilir adım boyudur. (kısaca adım boyu diyebiliriz). ρ ise yeni iterasyonu polytope 'un içinde tutan bir adım boyu çarpanıdır. α parametresini simplex algoritması için yapılan oran testine benzer bir oran testi yardımıyla buluruz.

$$\alpha \equiv \min\left\{-\frac{x_i}{dx_i} : \forall dx_i < 0. \quad 1 \le i \le n\right\} \tag{5.16}$$

dır. Görüyoruz ki değiştirilmiş algoritma basit olarak, ilk önce mevcut(geçerli) iterasyon bir ölçme işlemi yardımıyla merkezileştirilen ve daha sonra projekte edilmiş maliyet vektörü boyunca yeni iterasyona geçilen, bir adımlar dizisidir. Dualite teorisinden $c^Tx \ge b^Ty$ olduğunu ve bu eşitsizliğin ancak optimallikte eşitliğe dönüştüğünü biliyoruz. Bu sonucu kullanarak problem feasible ve dualite gapı dediğimiz $c^Tx - b^Ty$ ifadesi yeterince küçük olana kadar bu adımlar dizisine devam edilir. Şimdi aşağıda bu algoritmanın özetini veriyoruz.

5.1 Değiştirilmiş Karmarkar Algoritmasının Özeti

Standart formdaki LP problemi için bir başlangıç feasible ve kesin iç çözüm vektörü veriliyor. Yani $Ax_0=b$ ve $x_0>0$. Bu algoritma aşağıdaki şekilde çalışır.

Adım 1 :İterasyon sayacı k yı sıfırla (k=0) ve $x^{(k)}=x_0$ ile başlangıç çözüm vektörünü oluştur.

Adım 2 :İterasyon sayacını bir artır. (k=k+1) ve

$D=Diag\{x_1^{(k)},x_2^{(k)},...,x_n^{(k)}\}$ ile ölçülmüş matris D yi tanımla ve

$(AD^2A^T)^{-1}y^{(k)} = AD^2c$ ile verilen simetrik denklem sistemini çöz. Elde edilen m boyulu $y^{(k)}$ ile yine m boyutluindirgenmiş maliyet vektörü $z^{(k)}$ 'yı

$z^{(k)} = c - A^T y^{(k)}$ ile hesapla.

Adım 3 :$dx^{(k)} = -D^2 z^{(k)}$ ile adım doğrultu vektörünü hesapla ve

$x^{(k+1)} = x^{(k)} + \rho\alpha dx^{(k)}$ ile yeni yaklaşık çözümü bul.

burada, $0 < \rho < 1$, ve $\alpha = \min\left\{-\frac{x_i^{(k)}}{dx_i^{(k)}} : \forall dx_i^{(k)} < 0. \quad 1 \le i \le n\right\}$ dır.

Adım 4 :$gap = \frac{\|c^T x - b^T y\|}{1.0 + \|c^T x\|}$ ile dualite gapını hesapla.

Eğer primal ve dual problem feasible ve gap ta yeterince küçükse DUR. Aksi halde Adım 2 ye dön.

5.2 Değiştirilmiş Karmarkar Algoritmasının Bilgisayar Yorumu

Aşağıda değiştirilmiş algoritmanın bilgisayar yorumu QuickBasic Programlama diliyle veriyoruz. 5.3 deki uygulama bölümünde verilen örnek problem veri olarak program da kullanılmıştır. Datalar değiştirilmek suretiyle diğer problemlere adapte edilenilir. Ayrıca veriler program içinden değilde dışardan kullanıcı tarafından girilmesi istenecek şekilde (read, data) deyimleri yerine Input deyimi kullanılabilir.

```
CLS
 N = 6
 M = 3
 M2 = 2 * M
 DIM Diag(N), Xeski(N), XYENI(N), A(M, N), CC(N), RHS(M), B(M, M2),
      Z(N),Y(M), dx(N), Ad(M, N), Adc(M), Ada(M, M)
 DATA 30,5,15,75,30,11
 FOR c = 1 TO N
       READ Xeski(c)
 NEXT c
 T = .001
 Alfa = .95
 iterasyon = 0
 DATA 1,2,3,-1,0,0,2,1,-1,0,-1,0,0,0,1,0,0,-1
 FOR r = 1 TO M
      FOR c = 1 TO N
            READ A(r, c)
     NEXT c
 NEXT r
 DATA 1,1,1,0,0,0
 FOR c = 1 TO N
      READ CC(c)
```

```
        NEXT c
        DATA 10,20,4
        FOR c = 1 TO M
            READ RHS(c)
        NEXT c
DO
    iterasyon = iterasyon + 1
    FOR c = 1 TO N
        Diag(c) = Xeski(c) * Xeski(c)
    NEXT c
    FOR r = 1 TO M
        Adc(r) = 0
        FOR c = 1 TO N
            Ad(r, c) = A(r, c) * Diag(c)
            Adc(r) = Adc(r) + Ad(r, c) * CC(c)
        NEXT c
    NEXT r
    FOR r = 1 TO M
        FOR c = 1 TO M2
            B(r, c) = 0
        NEXT c
    NEXT r
    FOR r = 1 TO M
        FOR c = 1 TO M
            Top = 0
            FOR I = 1 TO N
                Top = Top + Ad(r, I) * A(c, I)
        NEXT I
        Ada(r, c) = Top
      NEXT c
```

```
        NEXT r
        FOR r = 1 TO M
            FOR c = 1 TO M
                B(r, c) = Ada(r, c)
            NEXT c
        NEXT r
'B matrisinin Tersi Gauss-Jordan Eliminasyon metodu ile hesaplanıyor.....
        FOR I = 1 TO M
            B(I, I + M) = 1
        NEXT I
        FOR r = 1 TO M
            IF B(r, r) <> 0 THEN 200
            I = r + 1
100         IF I > M THEN PRINT ".....Hata.....!!!! A(DD)AT Tekil....."
                END
            IF B(I, r) = 0 THEN I = I + 1: GOTO 100
            FOR c = 1 TO M2
                SWAP B(r, c), B(I, c)
            NEXT c
200         FOR I = r + 1 TO M
                ZZ = B(I, r) / B(r, r)
                FOR c = 1 TO M2
                    B(I, c) = B(I, c) - ZZ * B(r, c)
                NEXT c
            NEXT I
        NEXT r
        FOR r = M TO 2 STEP -1
            FOR I = r - 1 TO 1 STEP -1
                ZZ = B(I, r) / B(r, r)
                FOR c = r TO M2
```

```
                    B(I, c) = B(I, c) - ZZ * B(r, c)
                NEXT c
            NEXT I
        NEXT r
        FOR r = 1 TO M
            ZZ = B(r, r)
            FOR c = 1 TO M2
                B(r, c) = B(r, c) / ZZ
            NEXT c
        NEXT r
'Dual vektör y, hesaplanıyor.....
        FOR r = 1 TO M
            Y(r) = 0
            FOR c = 1 TO M
                Y(r) = Y(r) + B(r, c) * Adc(c)
            NEXT c
        NEXT r
'Z vektörü hesaplanıyor.....
        FOR r = 1 TO N
            Top = 0
            FOR c = 1 TO M
                Top = Top + A(c, r) * Y(c)
            NEXT c
            Z(r) = CC(r) - Top
        NEXT r
'dx Doğrultu vektörü hesaplanıyor.....
        FOR r = 1 TO N
            dx(r) = -Diag(r) * Z(r)
        NEXT r
'Oran Testi ile Eb belirleniyor.....
```

```
        FOR r = 1 TO N
            IF dx(r) < 0 THEN
                        Eb = -dx(r) / Xeski(r)
                        EXIT FOR
            END IF
        NEXT r
        FOR c = r + 1 TO N
            IF dx(c) < 0 AND ((-dx(c) / Xeski(c)) > Eb) THEN   Eb = -dx(c) / Xeski(c)
            END IF
        NEXT c
'Yeni iterasyon için x vektörü hesaplanıyor.....
        FOR r = 1 TO N
            XYENI(r) = Xeski(r) + (Alfa / Eb) * dx(r)
        NEXT r
'Gap Hesaplamaları.....
        Vdual = 0: Vesas = 0: test2 = 0
        FOR r = 1 TO N
            Vesas = Vesas + CC(r) * XYENI(r)
            test2 = test2 + XYENI(r) * Z(r)
        NEXT r
        FOR r = 1 TO M
            Vdual = Vdual + RHS(r) * Y(r)
        NEXT r
        gap = ABS(Vesas - Vdual) / (1 + ABS(Vesas))
'feasiblelik testleri.....
        PFI = 0: Top2 = 0
        FOR r = 1 TO M
            Top = 0
            FOR c = 1 TO N
                Top = Top + A(r, c) * XYENI(c)
```

```
            NEXT c
            PFI = PFI + (RHS(r) - Top) * (RHS(r) - Top)
            Top2 = Top2 + RHS(r) * RHS(r)
        NEXT r
        PFI = SQR(PFI) / (1# + SQR(Top2))
        IF PFI < .001 THEN PFIMI$ = "Feasible" ELSE PFIMI$ = "Feasible Değil"
        Top2 = 0: Dufi = 0
        FOR c = 1 TO N
            Top = 0
            FOR r = 1 TO M
                Top = Top + A(r, c) * Y(r)
            NEXT r
            Dufi = Dufi + (CC(c) - Top - Z(c)) * (CC(c) - Top - Z(c))
            Top2 = Top2 + CC(c) * CC(c)
        NEXT c
        Dufi = SQR(Dufi) / (1# + SQR(Top2))
        IF Dufi < .0001 THEN DUFIMI$ = "Feasible" ELSE DUFIMI$ = "Feasible Değil"
'Neticeler yazdırılıyor.....
        PRINT USING "    ----------###.İterasyon----------    "; iterasyon
        FOR c = 1 TO N
            PRINT USING "X#= ###.#####"; c; XYENI(c);
            IF NOT (c > M) THEN
                            PRINT USING "        Y#=###.#####"; c; Y(c)
                            ELSE PRINT
            END IF
        NEXT c
        PRINT PFIMI$; "          "; DUFIMI$
        PRINT "-----------------------------------"
        PRINT USING "Vesas=###.#####     Vdual=###.#####"; Vesas; Vdual
        IF gap > T THEN
```

```
                FOR r = 1 TO N
                    Xeski(r) = XYENI(r)
                NEXT r
        END IF
LOOP UNTIL gap < T
        CLOSE
        PRINT : PRINT ".....Toleransa Erişildi.....!"
        PRINT USING "GAP = ###.################"; gap: PRINT
        PRINT USING "----------###.İterasyon----------"; iterasyon
        FOR c = 1 TO N
            PRINT USING "X#=###.#####"; c; XYENI(c);
            IF NOT (c > M) THEN
                        PRINT USING "        Y#=###.#####"; c; Y(c)
                        ELSE PRINT
            END IF
        NEXT c
        sigdig = -(LOG(gap + 1E-08) / LOG(10#))
        PRINT PFIMI$; "          "; DUFIMI$
        PRINT "---------- Amaç Fonksiyonu Değeri--------"
        PRINT USING "Vesas=###.####          Vdual=###.####"; Vesas; Vdual
        PRINT "----------------------------------------"
        PRINT USING "Tolerans =###.############    Gap= ###.##########"; T; gap
        END
```

5.3 Uygulama

$$\text{Min } x_1 + x_2 + x_3$$

Kısıtlar

$$x_1 + 2x_2 + 3x_3 \geq 10$$
$$2x_1 + x_2 - x_3 \geq 20$$
$$x_3 \geq 4$$
$$x_1, x_2, x_3 \geq 0$$

Şeklindeki LP problemini Değiştirilmiş Karmarkar Algoritması ile çözelim.

Çözüm:İlk olarak problemi standard hale getirelim. Bu halde problem

$$\text{Min } c^T x$$

Kısıtlar $Ax=b \quad x \geq 0,$

Burada $x \in R^n$, $b \in R^m$ ve

$$A = \begin{bmatrix} 1 & 2 & 3 & -1 & 0 & 0 \\ 2 & 1 & -1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \end{bmatrix} \quad b = \begin{bmatrix} 10 \\ 20 \\ 4 \end{bmatrix} \quad c = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

olur. $x_0 = \begin{bmatrix} 30 \\ 5 \\ 15 \\ 75 \\ 30 \\ 11 \end{bmatrix}$ şeklinde bir başlangıç çözüm vektörü veriliyor.

Bu örnek için seçilen bu başlangıç çözüm vektörü birçok olasılıktan sadece bir tanesidir.

Şimdi değiştirlmiş algoritma için ilk iterasyon zincirini oluşturalım.

$$D = \begin{bmatrix} 30.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 5.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 15.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 75.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 30.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 11.0 \end{bmatrix}$$ olur.

Daha sonra AD^2A^T , $(AD^2A^T)^{-1}$ ve AD^2c matrislerini oluşturursak

$$AD^2A^T = \begin{bmatrix} 8650.0 & 1175.0 & 675.0 \\ 1175 & 4750.0 & -225.0 \\ 675 & -225.0 & 346.0 \end{bmatrix}$$ ve bu matrisin terside

$$(AD^2A^T)^{-1} = \begin{bmatrix} 0.0001478 & -0.0000518 & -0.000312 \\ -0.0000518 & 0.000235 & 0.000254 \\ -0.000312 & 0.000254 & 0.003683 \end{bmatrix}$$ dir.

ve

$$AD^2c = \begin{bmatrix} 1625.0 \\ 1600.0 \\ 225.0 \end{bmatrix}$$ olur. $(AD^2A^T)y = AD^2c$ den y dual vektör için tahmini

hesaplarsak, $$y = \begin{bmatrix} 0.0848 \\ 0.3496 \\ 0.7122 \end{bmatrix}$$ bulunur.

İndirgenmiş maliyet vektörü z yi ve adım doğrultu vektörü dx 'i hesaplarsak

$$z = c - A^Ty = \begin{bmatrix} 0.2160 \\ 0.4808 \\ 0.3830 \\ 0.0848 \\ 0.3496 \\ 0.7122 \end{bmatrix} \qquad dx = -D^2z = \begin{bmatrix} -194.3997 \\ -12.0202 \\ -86.1722 \\ -476.9716 \\ -314.6424 \\ -86.1772 \end{bmatrix}$$ olur.

Oran testini kullanarak

$$\text{Max}\left\{\frac{-194.3997}{30.0000}, \frac{-12.0202}{5.0000}, \frac{-86.1722}{15.0000}, \frac{-476.9716}{75.0000}, \frac{-314.6424}{30.0000}, \frac{-86.1772}{11.0000}\right\}$$

$$= \{6.4800\,,\ 2.4040\,,\ 5.7448\,,\ 6.3596\,,\ 10.4881\,,\ 7.8343\} = 10.4881$$

ve 0.95 adım boyu çarpanını kullanarak, yeni iterasyonu

$x = x_0 + \frac{0.95}{10.4881} dx$ den

$$x = x_0 + \left(\frac{0.95}{10.4881}\right) dx = \begin{bmatrix} 30.0 \\ 5.0 \\ 15.0 \\ 75.0 \\ 30.0 \\ 11.0 \end{bmatrix} + \left(\frac{0.95}{10.4881}\right) \begin{bmatrix} -194.3997 \\ -12.0202 \\ -86.1722 \\ -476.9716 \\ -314.6424 \\ -86.1772 \end{bmatrix} = \begin{bmatrix} 12.3915 \\ 3.9112 \\ 7.1942 \\ 31.7964 \\ 1.5000 \\ 3.1942 \end{bmatrix}$$

şeklinde elde ederiz.

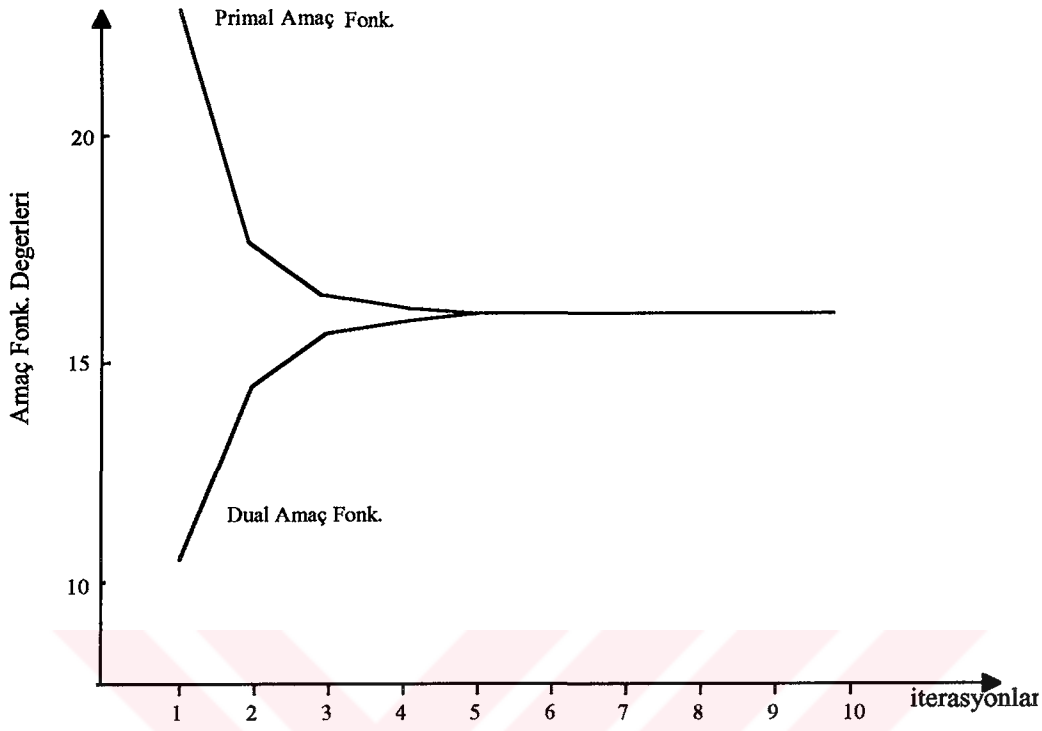
Bu yeni iterasyonla primal çözüm vektörü x ve dual tahmin y nin amaç fonksiyonları değerleri sırasıyla $J_{primal} = c^T x = 23.4968$ ve $J_{dual} = b^T y = 10.6888$ olur. Böylece birinci iterasyon tamamlanmıştır. Dual vektörler için ilk 10 iterasyon Tablo 5.1 de Primal vektörler için ilk 10 iterasyon ise Tablo 5.2 de gösterilmiştir. Daha önce belirttiğimiz gibi sadece optimallikte eşitlik sağlanan, dual maç değeri primal amaç değeri için bir alt sınır oluşturur. Bu amaç değerleri Şekil 5.1 de gösterilmiştir.

Tablo 5.1 Dual vektör ve dual amaç fonksiyonu için yapılan iterasyonun özeti

İterasyon	y_1	y_2	y_3	Amaç Fonk.
1	0,0848	0,3496	0,7122	10,6888
2	0,0493	0,4462	1,1012	14,2217
3	0,0111	0,4970	1,4617	15,8975
4	0,0007	0,4994	1,4958	15,9781
5	0,0001	0,5000	1,49996	15,9984
6	0,0000	0,5000	1,5000	16,0000
7	0,0000	0,5000	1,5000	16,0000
8	0,0000	0,5000	1,5000	16,0000
9	0,0000	0,5000	1,5000	16,0000
10	0,0000	0,5000	1,5000	16,0000

Tablo 5.2 Çözüm vektörü ve Primal(esas) Amaç fonksiyonu için yapılan iterasyonun özeti

İterasyon	x_1	x_2	x_3	x_4	x_5	x_6	Amaç Fonk.
1	12,3915	3,9112	7,1942	31,7964	1,5000	3,1942	23,4968
2	11,6317	2,1131	4,1597	18,3369	1,2167	0,1597	17,.9045
3	12,2740	0,1057	4,1249	14,8599	0,5288	0,1249	16,5045
4	11,9909	0,0856	4,0410	14,2850	0,0264	0,0410	16,1175
5	11,9971	0,0290	4,0020	14,0611	0,0210	0,0020	16,0281
6	12,0034	0,0014	4,0016	14,0112	0,0065	0,0016	16,0064
7	11,9998	0,0011	4,0005	14,0035	0,0003	0,0005	16,0014
8	12,0000	0,0003	4,0000	14,0006	0,0003	0,0000	16,0003
9	12,0000	0,0000	4,0000	14,0001	0,0000	0,0000	16,0000
10	12,0000	0,0000	4,0000	14,0000	0,0000	0,0000	16,0000



Sekil 5.1 Primal ve Dual Amaç Değerleri

6. İç-Nokta ÇALP Algoritması

Affine-scaling Primal algoritması, ilk olarak mevcut iterasyona bir ölçme işlemi yardımıyla konumlandıktan sonra kısıtlar matrisi A 'nın sıfır uzayı üzerine projekte edilmiş gradient boyunca ilerlenen bir adımlar dizisi üretiyordu. Bir ÇALP problemi düşündüğümüzde birkaç tane maliyet vektörü vardır. Bundan dolayı da genellikle projekte edilmiş gradientler birden çok doğrultuyu gösterirler. (Hemen akla Acaba hangi doğrultudan gitmeliyiz? sorusu gelir). ÇALP problemleri için iç nokta metodları bu farklı bakış açısına değinmeli ve bunlar tüm maliyet vektörlerini göz önüne alırken geçerli iterasyonun mevcut konumundan bir yenisine atlayabilecek tarzda birleştirilmelidirler. Bu bölümde bu tezin can alıcı noktasını sunuyoruz ve ÇALP problemleri için Affine-scaling primal algoritmasının bir değişiğini açıklıyoruz. Bu değişiklik, bir birleştirilmiş doğrultu boyunca alınan sonraki iterasyona ulaşmak için tek tek projekte edilmiş gradientlerin konvex kombinezonlarının elde edilmesiyle yapılmıştır.

Önceki bölümlerde

$$\begin{aligned} &\text{Max } C^T x \\ &\text{Kısıtlar } Ax=b \\ &\qquad x\geq 0 \quad (\text{Burada } x\in R^n, \quad b\in R^m \quad \text{dir.}) \end{aligned} \tag{6.1}$$

ile verilmiş standart formdaki tek amaçlı LP problemini göz önüne almıştık. Bu bölümde q tane lineer amaç fonksiyonu olan bir ÇALP problemini göz önüne alıyoruz. Böyle bir problem

$$\begin{aligned} &\text{Max } c^T x \\ &\text{Kısıtlar } Ax=b \\ &\qquad x\geq 0 \end{aligned} \tag{6.2}$$

(Burada C sütünları q tane amaç fonksiyonu olan nxq boyutlu amaç matrisidir.)
Karar vericinin (Decision Maker=DM) bütün sonuçlar bölgesi (aralık) nde tanımlı bir u(x) fayda fonksiyonuna sahip olduğunu farzedelim. Bu vektör değerli optimizasyon problemi

$$\text{Max} \quad u(x)$$

Kısıtlar $Ax=b$ (6.3)

$x \geq 0$

şeklinde yazılabilir ve bu problem için optimal çözüm fayda fonksiyonunun en büyük değerini veren x çözüm vektörüdür. Bu halde kavramsal olarak bir fayda fonksiyonu bilindiğinde algoritma bir feasible noktadan başlar ve fayda fonksiyonunun gradienti boyunca geçerli iterasyonu daha yüksek fayda değerli yeni bir iterasyona taşıyan bir arama doğrultusu oluşturur. Mevcut bir fayda fonksiyonu olmadığında algoritma, fayda fonksiyonunun gradientinin yerini alacak bir vekil bulmalıdır. Bu gradient bilgisi karar verici ile beraber hareket edilerek elde edilir. Karar vericiye, lokal gradient için bir geçici ölçü ile desteklenen ve bizi mevcut iterasyondan bir yenisine götürecek birleştirilmiş adım düzenlemesine müsade eden adım doğrultuları için bir seçim sunuluyor. Daha sonra, aday adım doğrultularının nasıl oluşturulacağı ve ilgili tercihlere nasıl karar verileceği (değer biçileceği) ve onların (aday adım doğrultularının) bir tek adım doğrultusuna nasıl birleştirileceği sorusu ile ilgileniyoruz. Eğer Amaç matrisi C, q tane sütuna sahipse sonraki iterasyona varırken(adımlarken) takip edeceği q tane doğrultusu olacaktır. Bu adım doğrultuları Affine -scaling primal algoritmasında yapılan bu doğrultuyu bulma işlemi amaçların herbiri için bulunur. Kısıtlar matrisi A 'nın sıfır uzayındaki i. amacı projekte ederek ve bir maximizasyon problemini göz önüne aldığımızı hatırlayarak,

$$dx_i = -D^2(c - A^T y_i) \tag{6.4}$$

ile verilmiş dx_i adım doğrultu vektörüne sahibiz ve burada y_i dual vektör için tahmin olup

$$(AD^2A^T)y_i = AD^2c \tag{6.5}$$

çözülerek bulunur.

α_i adım boylarını bulmak için daha belirttiğimiz oran testi kullanılarak,

$$x_i = x_0 + \rho\alpha dx_i \qquad \alpha > 0,\ 0 < \rho < 1,\ 1 \leq i \leq q \tag{6.6}$$

ile verilen q tane yeni iterasyon kümesini bulmak için bu q tane adım doğrultu vektörünü kullanırız. Burada x_0 geçerli iterasyon, ρ da adım boyu çarpanıdır. Bu yeni iterasyonların herbirindeki q tane amaç fonksiyonunun herbirinin değeri bir v_i amaç değerleri vektöründe toplanır. Burada $v_i = C^T x_i$ dir. Karar verici geçerli iterasyon x_0 dan

başlayarak q tane adım doğrultu vektörünün herbiri boyunca adımlayabilir ve bir x_i iterasyonunda ilgili değer vektörü v_i ile belirtilebilir. Açık Görünen bu son noktalardan sonra karar verici doğrultuların herbiri için ilgili tercihlere değer biçmeyi sorguladı. q doğrultu için bu ilgili tercihler daha sonra, sonraki adıma geçmek için kullanılır. Gerçekte bu tezin bu bölümünde bahsettiğimiz algoritma, sonraki adım boyunca alınacak bir tek birleştirilmiş adım doğrultu vektörü olan dx 'i veren tek tek herbir adım doğrultu vektörlerinin bir konvex kombinezonu gibi bu ilgili tercihlerin kullanımına dayanır. Bu birleştirilmiş adım doğrultu vektörü

$$dx = \sum_{i=1}^{q} \lambda_i(\alpha_i dx_i) \tag{6.7}$$

ile bulunur. Burada bütün $1 \le i \le q$ ler için $\sum_{i=1}^{q} \lambda_i = 1$, $\lambda_i \ge 0$ dır. ve $\{\lambda_i\}$ ağırlıklarının kümesi, q tane tek tek herbir adım doğrultu vektörünün ilgili tercihinin açıklar. İlk başta q doğrultunun ilgili tercihleri mevcuttur. Bu birleştirilmiş adım doğrultu vektörünü kullanmak ve sonraki iterasyonu türetmek için

$$x = x_0 + \rho dx \tag{6.8}$$

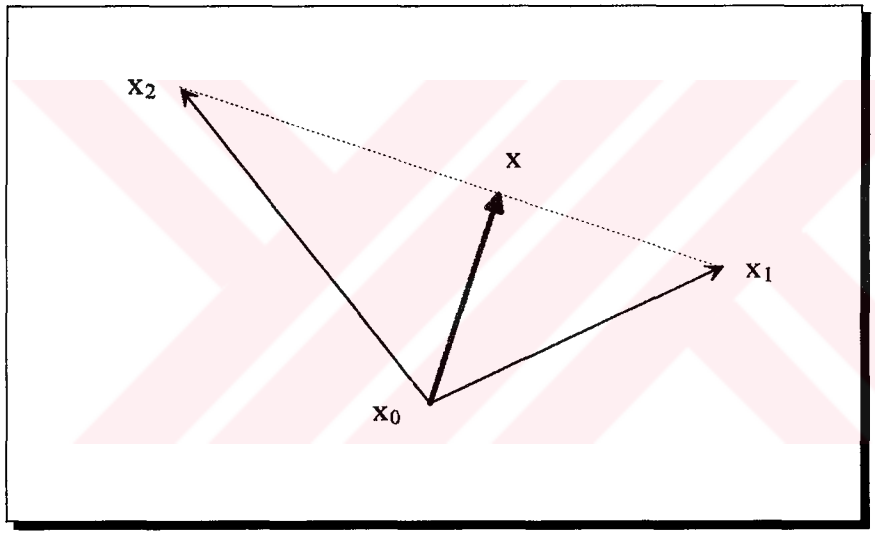
ifadesi ile işleme devam ederiz. Daha önce herhangi bir ÇALP algoritmasının bir faydalı özelliğinin test edilebilir olması olduğunu belirtmiştik. Bununla demek istediğimiz eğer algoritma, teklif ettiği işlemi çalıştırırsa ve çeşitli adım doğrultuları için karar vericiden tercih alınması yerine bir fayda fonksiyonu kullanırsa, algortima doğru optimal sonuca varmalıdır. Bizim durumumuzda, eğer bir u(x) fayda fonksiyonu biliniyorsa ve q tane yeni iterasyon, $\{x_i\}$ ile verilirse, lokal olarak $\{\lambda_i\}$ ölçme katsayıları

$$\lambda_i = \frac{u(x_i)}{\sum_{i=1}^{q} u(x_i)} \quad , \quad 1 \le i \le q \qquad \text{ile türetilebilir.}$$

Bir fayda fonksiyonunun olmaması durumunda (ki genelde böyledir) $\{\lambda_i\}$ağılıklarına değer biçilmesi AHP(Analytic Hiyerachy Proscess) nin kullandığı gibi q tane değer vektörü $\{v_i\}$ lerin doğrudan karşılaştırılmaları ile gerçekleştirilebilir. AHP ile birleştirilmiş iyi-bilinen detaylara girmeksizin, değer biçme söyle soruların sorulmasını icap ettirir:"

i. doğrultudaki adımlamanın sonundaki bir v_i değer vektörü göz önüne alınarak, bir v_j değer vektörü ile neticelenen j. doğrultudaki adımlamayla karşılaştırıldığında, hangi doğrultunun tercih edileceği ve bununla ilgili tercih nedir?." Bu soru için kabul edilen cevap ij. elaman olarak qxq boyutlu karşılaştırma matrisine girilir, ki bu matrise ait esas özvektör daha sonra, bileşenleri (6.7) de gösterilmiş konvex kombinezonundan oluşturulmuş vektör olarak kullanılır.

Bu durum, iki amaçlı bir problem için Şekil 6.1 de gösterilmiştir. q tane adım doğrultu vektörünü sonraki iterasyon için bir tek adım doğrultu vektörüne nasıl birleştirileceği gösterilmiştir.



Şekil 6.1 Tek tek olan adım doğrultu vektörlerinin konvex kombinezonu

Bu yeni iterasyonun feasible kaldığını görmek istersek,

$$Ax = A(x_0 + dx) = A\left[x_0 + \rho \sum_{i=1}^{q} \lambda_i(\alpha_i dx_i) \right]$$

$$= Ax_0 + \rho \sum_{i=1}^{q} \lambda_i(\alpha_i A dx_i) = b \quad \text{dır.}$$

Bütün i ler için $Ax_0=b$ ve $Adx_i=0$ olduğuna dikkat edelim.

λ_i ağırlıklar kümesi, q tane tek tek adım doğrultu vektörünü birtek birleştirilmiş doğrulutuya birleştirmek için kullanılır. Tabii ki bu işlem, birtek amaç vektörü elde etmek için C amaç matrisinin sütunlarına bu ağırlıkların kümesinin uygulanmasına denktir. Aşağıda bu işlem gerçekleştiriliyor.

i. adım doğrultu vektörü dx_i

$$dx_i = D^2\left[c_i - A^T(AD^2A^T)^{-1}AD^2c_i\right] \text{ den bulunur.}$$

Burada c_i amaç matrisi C'nin i. sütunudur. Eğer birleştirilmiş adım doğrultu vektörü (6.7) ile verilmiş bir konvex kombinezon yardımıyla bulunursa, o zaman q boyutlu vektörü gösteren α nın bileşenleri sıraysıyla $\{\alpha_i\}$ maximal adımlarıdır. Bu halde birleştirilmiş adım doğrultu vektörü

$dx_i = [dx_1, ..., dx_q]\begin{bmatrix} \lambda_1 & & \\ & \ldots\ldots & \\ & & \lambda_q \end{bmatrix}\alpha$ olarak yazılabilir ve herbir amaç fonksiyonu için adım doğrultu vektörleri dx_i lerin denklemini kullanarak

$$dx = D^2\left[I_n - A^T(AD^2A^T)^{-1}AD^2\right][c_1 ... c_q]\begin{bmatrix} \lambda_1 & & \\ & \ldots\ldots & \\ & & \lambda_q \end{bmatrix}\alpha$$

$$= D^2\left[I_n - A^T(AD^2A^T)^{-1}AD^2\right]C\begin{bmatrix} \lambda_1 & & \\ & \ldots\ldots & \\ & & \lambda_q \end{bmatrix}\alpha$$

$$= D^2\left[I_n - A^T(AD^2A^T)^{-1}AD^2\right]\ \bar{c}\alpha$$

Burada sütun vektörü $\bar{c} = C\begin{bmatrix} \lambda_1 & & \\ & \ldots\ldots & \\ & & \lambda_q \end{bmatrix}$ C amaç matrisinin sütunlarının bir konvex kombinezonudur. Daha önce algoritmanın tek tek adım doğrultu vektörlerinin konvex kombinezonuna dayandığını belirtmiştik, şimdi bu noktaya dönelim. Bir LP probleminin çözümü, amaç vektörlerine veya özel çözüm metoduna bakılmaksızın daima kısıtlar polytope 'unun sınırı üzerinde biryerde olacaktır. Bu bölümde sunduğumuz

algoritmayla son işimiz, işlemi sona erdirebileceğimiz bir çözüm için sınırdaki bütün adayları sürekli olarak incelemek olacaktır.

Konvex kombinezon (6.7) ile verilmiş bir dx adım doğrultusu temin eder ki böylece yeni iterasyon x 'e

$$x = x_0 + \rho dx \tag{6.8}$$

ile ulaşırız. Burada ρ , $0 < \rho < 1$ olan adım boyu çarpanıdır. Bununla beraber, eğer adım boyu çarpanının $\rho = 1$ olmasına müsade edersek, mevcut iterasyondan max müsade edilebilir adım atarak ilerleriz ve böylece yeni iterasyon artık polytope'un içinde değildir. Fakat kısıtlar polytope unun sınırındadır. Bu sınır noktasını x_b ile göstererek

$$x_b = x_0 + dx \tag{6.9}$$

dır ve bunun birleştirilmiş değer vektörü $V_b = C^T x_b$ ile verilir. Şuna dikkat edin: mevcut iterasyon x den x_b sınır noktasına atlamak için

$$dx_b = x_b - x \tag{6.10}$$

ile verilen dx_b adım doğrultusunu kullanıyoruz.

Bu yeni doğrultu dx_b nin önemi, her iterasyonda (tabii ki 1. den sonra) karar vericiye q tane tek tek adım doğrultu vektörü ile beraber bu doğrultunun da sunulabilmesidir. q+1 tane değer vektörü arasından tercihe değer biçmek

$$dx = \sum_{i=1}^{q} \lambda_i (\alpha_i dx_i) + \lambda_b dx_b \tag{6.11}$$

ile verilen birtek adım doğrultu vektörü dx ile olur.
Burada şimdi,

$$\sum_{i=1}^{q} \lambda_i + \lambda_b = 1 , \qquad \text{bütün} 1 \le i \le q \quad \text{için} \qquad \lambda_i , \lambda_b \ge 0 \tag{6.12}$$

şeklinde yeni bir doğrultuya sahibiz. Bu yeni doğrultu ile (6.8) de verildiği gibi yeni iterasyonu elde edebiliriz. Sonraki iterasyona devam etmeden önce yeni sınır noktası için bu adayı göz önüne alabiliriz. Bu aday (6.11) deki adım doğrultu vektörü ile bir tam adım boyu çarpanı kullanılarak hesaplanır ve

$$x_{yeni} = x_0 + dx \tag{6.13}$$

ile verilir ve bunun birlerştirilmiş değer vektörü $v_{yeni} = C^T x_{yeni}$ den heasplanır. x_b ve x_{yeni} sınır noktalarının her ikiside polytope 'un sınırı üzerindedir, ve basit bir tercih sorusu sorularak hangi noktanın karar verici tarafından daha çok tercih edildiğine kara verilir. Eğer yeni aday nokta tercih edilirse eski sınır noktası x_b yi yeni nokta x_{yeni} ile yerdeğiştiririz. Dikkat ederseniz mevcut iterasyon herbir iterasyonda değişirken, sınır noktasının her iterasyonda değişmesi gerekli değildir. Biraz önce açıklanan işlem, (6.10) ile sürekli olarak yenilenen dx_b doğrultu vektörü yardımıyla o ana kadar bulunan en iyi sınır noktasını tutmamıza müsade eder ve ne zaman böyle istersek bu yeni noktaya direk bağlantı yapabiliriz.

6. 1 İç-nokta ÇALP Algoritmasının Özeti

Standart formdaki LP problemi için bir başlangıç feasible ve kesin iç çözüm vektörü veriliyor. Yani $Ax_0=b$ ve $x_0>0$. Bu algoritma aşağıdaki şekilde çalışır.

Adım 1 :İterasyon sayacı k yı sıfırla (k=0) ve $x^{(k)}=x_0$ ile başlangıç çözüm vektörünü oluştur.

Adım 2 :İterasyon sayacını bir artır. (k=k+1) ve

D=Diag$\{x_1^{(k)},x_2^{(k)},...,x_n^{(k)}\}$ ile ölçülmüş matris D yi tanımla ve

$$(AD^2A^T)y_i^{(k)} = AD^2c_i \quad , \qquad 1 \le i \le q$$

şeklindeki q tane simetrik denklem sistemini çöz. Burada c_i m boyutlu $y_i(k)$ vektörü için nxq boyutlu amaç matrisi C nin i. sütunudur.

ve $dx_i^{(k)} = -D^2\left[c_i - A^T y_i^{(k)}\right]$ $\quad 1 \le i \le q$ ile q tane dx_i adım doğrultu vektörünü hesapla.

Adım 3: $x_i^{(k+1)} = x^{(k)} + \rho\alpha_i dx_i^{(k)}$, $\quad 1 \le i \le q$ bağıntısı ile q tane $\{x_i\}$ yeni iterasyonu bul. Burada ρ , $0<\rho<1$ olan adım boyu çarpanıdır ve $\{\alpha_i\}$ler oran testi yardımıyla bulunur. Daha sonra

$v_i = C^T x_i^{(k+1)}$, $\quad 1 \le i \le q$ ile $\{v_i\}$ değer vektörlerini hesapla.

Adım 4: Tek tek olan bütün değer vektörlerini karşılaştır ve bir karşılaştırma matrisi ve AHP yardımıyla ilgili tercihlerini belirle. Eğer k=1 ise elimizde sadece q tane değer vektörü ve sonuçlanan q boyutlu öncelik vektörü λ, vardır. k>1 ise elimizde q tane değer vektörüne ilaveten x_{son} sınır noktası ve bunun birleştirilmiş değer vektörü v_{son} vardır. Bu durumda q+1 tane vektörü karşılaştırırız ve sonuçlanan öncelik vektörü q+1 boyutlu olacaktır.

$$dx = \begin{cases} k=1 \quad \text{için} \quad \sum_{i=1}^{q} \lambda_i(\alpha_i dx_i) \quad \text{burada} \quad \sum_{i=1}^{q} \lambda_i = 1 \\ k>1 \quad \text{için} \quad \sum_{i=1}^{q} \lambda_i(\alpha_i dx_i) + \lambda_b dx_b \quad \text{burada} \quad \lambda_b + \sum_{i=1}^{q} \lambda_i = 1 \end{cases}$$

ve burada $\lambda_i \geq 0$ olup λ öncelik vektörünün bileşenleridir.

ile verilen birleştirilmiş adım doğrultu vektörü dx i oluştur.

Adım 5: $x^{(k+1)} = x^{(k)} + \rho dx^{(k)}$ ile verilen iyileştirlmiş(güncelleştirilmiş) denklem yardımıyla yeni (sonraki) iterasyonu bul. Burada $0<\rho<1$ olup daha önce kullanılan aynı adım boyu çarpanıdır. Tam bir adım atarak yani $\rho=1$ alarak dx doğrultusu boyunca yeni sınır noktası x_{yeni} yi bul.

$x_{yeni} = x^{(k)} + dx^{(k)}$

sınır değer vektörü v_{yeni} yi

$v_{yeni} = C^T x_{yeni}$ ile hesapla.

Eğer k=1 ise: $x_b = x_{yeni}$, $v_b = v_{yeni}$
aksi halde bütün k>1 ler için v_{yeni} değer vektörü v_b ye tercih edilirse

$x_b = x_{yeni}$, $v_b = v_{yeni}$ yap.

ve $dx_b = x_b - x^{(k+1)}$ i hesapla.
Eğer herhangi bir sona erme (durdurma) koşulu ile karşılaşırsa $x=x_b$ yap ve DUR aksi halde Adım 2 ye dön.

Açıklamalar

1) Affine-scaling primal (değiştirilmiş Karmarkar) algoritmasının normal durdurma (sona erdirme sonlandırma) durumları, primal ve dual feasible' lik ve küçük bir dualite gapıdır. Bu ÇALP algoritması için sona erdirme koşulu primal feasible'liği ve bazı ilave koşulları içermelidir. Mesela,

iterasyon sayacı üzerinde bir üst sınır dahil edilebilir. Optimal çözümün aranmasını sona erdirmek için bir diğer olasılık mevcut en iyi sınır noktasındaki gelişim (iyileşme, değişim) çok küçük olduğunda oluşur.

2) 5. adımda v_{yeni} ye bağlı v_b nin ilgili tercihi, iki vektörün basit bir ikili karşılaştırılması ile elde edilir. Tercihin nümerik olarak çalışması gerekli değildir. Hepimizin aradığı bayağı sıralama gibidir.

3) Üste anahatları çizilmiş işlem basit olarak 2 adımlıktır. Bunlardan birincisi , tanımlı bir adım boyu çarpanıyla birleştirlmiş adım doğrultu vektörü boyunca adımlamaktır. Diğeri ise bu doğrultuda tam bir adım atarak yeni bir sınır noktası adayı için kısıtlar polytope'unun sınırı üzerinde derinlemesine çalışmaktır (sondaj yapmaktır.) Eğer bu sondajlama işlemi mevcut olana tercih edilen bir sınır noktası tanımlarsa, yeni sınır noktasını muhafaza eder eskisini ekarte ederiz. Bu, işlem boyunca karşılaşılan en iyi sınır noktasını tutmamıza ve istediğimiz zaman yada sona erdirme koşuluyla karşılaşıldığında böyle bir sınır noktasında çözüm aramamızı bitirmemize olanak verir.

6.2 Değiştirilmiş Karmarkar Algoritmasının Bilgisayar Yorumu

Aşağıda değiştirilmiş algoritmanın bilgisayar yorumu QuickBasic Programlama diliyle veriyoruz. 6.3 deki uygulama bölümünde verilen örnek problem veri olarak program da kullanılmıştır. Datalar değiştirilmek suretiyle diğer problemlere adapte edilenilir. Ayrıca veriler program içinden değilde dışardan kullanıcı tarafından girilmesi istenecek şekilde (*read, data*) deyimleri yerine *Input* deyimi kullanılabilir.

```
CLS
N = 6
M = 4
Q = 3
M2 = 2 * M
Q2 = Q + 1
DIM Diag(N), Xeski(N), Xyeniler(N, Q), Xson(N), A(M, N), CC(N, Q),
    RHS(M), B(M, M2), Z(N, Q), Y(M, Q), Dxler(N, Q), AD(M, N),
    Adc(M, Q), Ada(M, M), Alfa(Q), Lamda(Q2), Dx(N), Fayda(Q2), Dxb(N),
    Vb(Q), Xb(N), Xyeni(N), Vyeni(Q), Vson(Q)
DEF Fnu (x1, x2) = x1 * x2
DATA 1,2,7,11,2,1
FOR c = 1 TO N
    READ Xeski(c)
NEXT c
T = .00000001#
Ro = .5
iterasyon = 0
DATA 1,1,1,0,0,0, 1,2,0,1,0,0, 8,1,0,0,-1,0, 1,5,0,0,0,-1
FOR r = 1 TO M
        FOR c = 1 TO N
            READ A(r, c)
```

```
            NEXT c
        NEXT r
        DATA 1,0,0,0,0,0, 0,1,0,0,0,0, 3,4,0,0,0,0
        FOR i = 1 TO Q
            FOR c = 1 TO N
                READ CC(c, i)
            NEXT c
        NEXT i
        DATA 10,16,8,10
        FOR c = 1 TO M
            READ RHS(c)
        NEXT c
DO
        iterasyon = iterasyon + 1
        FOR c = 1 TO N
            Diag(c) = Xeski(c) * Xeski(c)
        NEXT c
        FOR r = 1 TO M
            FOR c = 1 TO N
                AD(r, c) = A(r, c) * Diag(c)
            NEXT c
        NEXT r
        FOR i = 1 TO Q
            Adc(1, i) = 0
            FOR r = 1 TO M
                FOR c = 1 TO N
                    Adc(r, i) = Adc(r, i) + AD(r, c) * CC(c, i)
                NEXT c
            NEXT r
        NEXT i
```

```
        FOR r = 1 TO M
                FOR c = 1 TO M2
                        B(r, c) = 0
                NEXT c
        NEXT r
        FOR r = 1 TO M
                FOR c = 1 TO M
                        Top = 0
                        FOR i = 1 TO N
                                Top = Top + AD(r, i) * A(c, i)
                        NEXT i
                        Ada(r, c) = Top
                NEXT c
        NEXT r
        FOR r = 1 TO M
                FOR c = 1 TO M
                        B(r, c) = Ada(r, c)
                NEXT c
        NEXT r
'Gauss-Jordan Eliminasyon Metodu ile B matrisinin tersi hesaplanıyor.....
        FOR i = 1 TO M
                B(i, i + M) = 1
        NEXT i
        FOR r = 1 TO M
                IF B(r, r) <> 0 THEN 200
                i = r + 1
100             IF i > M THEN PRINT ".....Hata.....!!!! A(DD)AT Tekil....."
                                END
                IF B(i, r) = 0 THEN i = i + 1: GOTO 100
                FOR c = 1 TO M2
```

```
                SWAP B(r, c), B(i, c)
            NEXT c
200         FOR i = r + 1 TO M
                ZZ = B(i, r) / B(r, r)
                    FOR c = 1 TO M2
                        B(i, c) = B(i, c) - ZZ * B(r, c)
                    NEXT c
            NEXT i
        NEXT r
        FOR r = M TO 2 STEP -1
            FOR i = r - 1 TO 1 STEP -1
                ZZ = B(i, r) / B(r, r)
                FOR c = r TO M2
                    B(i, c) = B(i, c) - ZZ * B(r, c)
                NEXT c
            NEXT i
        NEXT r
        FOR r = 1 TO M
            ZZ = B(r, r)
            FOR c = 1 TO M2
                B(r, c) = B(r, c) / ZZ
            NEXT c
        NEXT r
        FOR r = 1 TO M
            FOR c = 1 TO M
                B(r, c) = B(r, c + M)
            NEXT c
        NEXT r
'q tane Dual vektör y, hesaplanıyor.....
        FOR i = 1 TO Q
```

```
            Y(1, i) = 0
            FOR r = 1 TO M
                FOR c = 1 TO M
                    Y(r, i) = Y(r, i) + B(r, c) * Adc(c, i)
                NEXT c
            NEXT r
        NEXT i
'q tane Z vektörü hesaplanıyor.....
        FOR i = 1 TO Q
            FOR r = 1 TO N
                Top = 0
             FOR c = 1 TO M
                    Top = Top + A(c, r) * Y(c, i)
             NEXT c
             Z(r, i) = CC(r, i) - Top
            NEXT r
        NEXT i
'q tane dx Doğrultu vektörü hesaplanıyor.....
        FOR i = 1 TO Q
            Dxler(1, i) = 0
            FOR r = 1 TO N
                Dxler(r, i) = Diag(r) * Z(r, i)
            NEXT r
        NEXT i
'Oran Testi ile q boyutlu Alfa dizisi belirleniyor.....
        FOR i = 1 TO Q
            FOR r = 1 TO N
                IF Dxler(r, i) < 0 THEN
                                Alfa(i) = -Xeski(r) / Dxler(r, i)
                                EXIT FOR
```

```
                END IF
            NEXT r
            FOR c = r + 1 TO N
                IF Dxler(c, i) < 0 AND ((-Xeski(c) / Dxler(c, i)) < Alfa(i)) THEN
                                        Alfa(i) = -Xeski(c) / Dxler(c, i)
                END IF
            NEXT c
        NEXT i
'Yeni iterasyon için q tane x vektörü hesaplanıyor.....
        FOR i = 1 TO Q
            FOR r = 1 TO N
                Xyeniler(r, i) = Xeski(r) + Ro * Alfa(i) * Dxler(r, i)
            NEXT r
        NEXT i
        FOR i = 1 TO Q
            Fayda(i) = Fnu(Xyeniler(1, i), Xyeniler(2, i))
        NEXT i
'lamda ağırlıkları hesaplanıyor.....
        Faydatop = 0
        FOR i = 1 TO Q
            Fayda(i) = Fnu(Xyeniler(1, i), Xyeniler(2, i))
            Faydatop = Faydatop + Fayda(i)
        NEXT i
        IF iterasyon > 1 THEN
                Fayda(Q + 1) = Fnu(Xson(1), Xson(2))
                Faydatop = Faydatop + Fnu(Xson(1), Xson(2))
                Lamda(Q + 1) = Fayda(Q + 1) / faydatop
        END IF
        FOR i = 1 TO Q
            Lamda(i) = Fayda(i) / faydatop
```

```
        NEXT i
'birleştirilmiş adım doğrultu vektörü dx hesaplanıyor.....
        FOR i = 1 TO Q
                FOR r = 1 TO N
                        Dxler(r, i) = Alfa(i) * Lamda(i) * Dxler(r, i)
                NEXT r
        NEXT i
        FOR c = 1 TO N
                Dx(c) = 0
                FOR i = 1 TO Q
                        Dx(c) = Dx(c) + Dxler(c, i)
                NEXT i
                IF iterasyon > 1 THEN Dx(c) = Dx(c) + Dxb(c) * Lamda(Q + 1)
        NEXT c
'yeni iterasyon xyeni ve sınır noktası xson hesaplanıyor.....
        FOR c = 1 TO N
                Xyeni(c) = Xeski(c) + Ro * Dx(c)
                Xb(c) = Xeski(c) + Dx(c)
        NEXT c
'yeni iterasyon xyeni nin ve sınır noktası xson daki Amaç fonk değerleri hesaplanıyor....
        FOR i = 1 TO Q
                Vyeni(i) = 0
                Vb(i) = 0
                FOR c = 1 TO N
                        Vyeni(i) = Vyeni(i) + CC(c, i) * Xyeni(c)
                        Vb(i) = Vb(i) + CC(c, i) * Xb(c)
                NEXT c
        NEXT i
'son bulunan xb ve vb xson ve vson altında saklanıyor.
        IF iterasyon = 1 THEN
```

```
                    FOR c = 1 TO N
                        Xson(c) = Xb(c)
                    NEXT c
                    FOR i = 1 TO Q
                        Vson(i) = Vb(i)
                    NEXT i
        END IF
'Vb ve Vson arasında tercih yapılıyor....
        IF iterasyon > 1 THEN
                    IF Fnu(Xb(1), Xb(2)) > Fnu(Xson(1), Xson(2)) THEN
                        FOR c = 1 TO N
                            Xson(c) = Xb(c)
                        NEXT c
                        FOR i = 1 TO Q
                            Vson(i) = Vb(i)
                        NEXT i
                    END IF
        END IF
        FOR c = 1 TO N
            Dxb(c) = Xson(c) - Xyeni(c)
            Xeski(c) = Xyeni(c)
        NEXT c
'feasiblelik testleri.....
        pfi = 0: Toprhs = 0
        FOR r = 1 TO M
            Top = 0
            FOR c = 1 TO N
                Top = Top + A(r, c) * Xyeni(c)
            NEXT c
            pfi = pfi + (RHS(r) - Top) * (RHS(r) - Top)
```

```
            Toprhs = Toprhs + RHS(r) * RHS(r)
        NEXT r
        pfi = SQR(pfi) / (1# + SQR(Toprhs))
'Neticeler yazdırılıyor.....
        PRINT USING "----------###.İterasyon----------"; iterasyon
        FOR c = 1 TO N
            PRINT USING "X#=##.#####"; c; Xson(c)
        NEXT c
        PRINT "-------------"
        FOR i = 1 TO Q
            PRINT USING "V#=###.#####"; i; Vson(i)
        NEXT i
LOOP UNTIL pfi < T
        PRINT : PRINT ".....Toleransa Erişildi.....!"
        PRINT USING "primal infesible=###.################"; pfi: PRINT
        PRINT USING "----------###.İterasyon----------"; iterasyon
        FOR c = 1 TO N
            PRINT USING "X#=###.#####"; c; Xson(c);
        NEXT c
        PRINT "----------- Amaç Fonksiyonu Değerleri--------"
        FOR i = 1 TO Q
            PRINT USING "V=###.####"; i; Vson(i)
        NEXT i
        PRINT "-------------"
        PRINT USING "Tolerans =###.##### Primal Infeasiblity= ###.######"; T; pfi
        END
```

6.3 Uygulama

$$\text{Max } v_1=x_1$$
$$v_2=x_2$$
$$v_3=3x_1+4x_2$$

Kısıtlar

$$x_1 + x_2 \le 10$$
$$x_1 + 2x_2 \le 16$$
$$8x_1 + x_2 \ge 8$$
$$x_1 + 5x_2 \ge 10$$
$$x_1 \ge 0$$
$$x_2 \ge 0$$

İlk olarak problemi aşağıdaki formda standard hale getirelim.

$$\max\{v = C^T x\}$$

Kısıtlar:

$$Ax=b$$
$$x \ge 0$$

Bu halde

$$A=\begin{bmatrix} 1.0 & 1.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 2.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ 8.0 & 1.0 & 0.0 & 0.0 & -1.0 & 0.0 \\ 1.0 & 50 & 0.0 & 0.0 & 0.0 & -1.0 \end{bmatrix}, \quad b=\begin{bmatrix} 10.0 \\ 16.0 \\ 8.0 \\ 10.0 \end{bmatrix}$$

$$C=\begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 3.0 & 4.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}, \quad v_0=\begin{bmatrix} 1.0 \\ 2.0 \\ 11.0 \end{bmatrix}$$

Buradaki $v_0, x_0 = \begin{bmatrix} 1.0 & 2.0 & 7.0 & 11.0 & 2.0 & 1.0 \end{bmatrix}^T$ ile verilen bir başlangıç iç feasible noktaya karşılık gelenamaç fonksiyonunun başlangıç değeridir.

Karar vericinin $u(x)=v_1(x)v_2(x)=x_1 x_2$ ile verilen bir fayda fonksiyonu olduğunu kabul edelim. Bu problemin doğru optimal çözümü $x^*=[5.0\ \ 5.0]^T$ noktası verilir. Bizi bu noktadan ileri götüren iterasyon işleminin gelişimini anahatları ile belirteceğiz.

$$dx_1 = \begin{bmatrix} 0.0621 \\ -0.0171 \\ -0.0450 \\ -0.0279 \\ 0.4801 \\ -0.0234 \end{bmatrix} \qquad dx_2 = \begin{bmatrix} -0.0171 \\ 0.0438 \\ -0.0267 \\ -0.0706 \\ -0.0931 \\ 0.2021 \end{bmatrix} \qquad dx_3 = \begin{bmatrix} 0.1180 \\ 0.1240 \\ -0.2420 \\ -0.3661 \\ 1.0679 \\ 0.7382 \end{bmatrix}$$

bu adım doğrultu vektörleri ile daha önce açıkladığımız oran testini kullanarak bu doğrultular boyuncaα adım boylarınıbulmaya geçelim.

Örneğin 1. iterasyon için αadım boyu

$$\alpha = \min\left\{\frac{2.0}{0.0171}, \frac{7.0}{0.0450}, \frac{11.0}{0.0279}, \frac{1.0}{0.0234}\right\} = 42.7350$$ olarak bulunabilir.

Daha sonra yeni iterasyonalar

$$x = x_0 + \rho\alpha dx (\text{burada} 0<\rho<1 \text{dir.})$$

ile verilen güncelleştirilmiş(iyileştirlmiş) denklemden bulunurlar. Adım boyu çarpanı ρ yu 0.5 alarak aşağıda gösterilen 3 yeni iterasyona ulaşılır.

$$x_1 = \begin{bmatrix} 2.3266 \\ 1.6347 \\ 6.0387 \\ 10.4040 \\ 12.2474 \\ 0.5000 \end{bmatrix}, \quad x_2 = \begin{bmatrix} 0.8161 \\ 2.4711 \\ 6.7128 \\ 10.2416 \\ 1.0000 \\ 3.1718 \end{bmatrix}, \quad x_3 = \begin{bmatrix} 2.7062 \\ 3.7938 \\ 3.5000 \\ 5.7062 \\ 17.4433 \\ 11.6752 \end{bmatrix}$$

Dikkat ederseniz bu 3 yeni iterasyon,tek tek üç amacın herbirini ayrı ayrı maximize eden sonraki hareketi göstermektedir. Bütün amaçları göz önüne alarak yeni iterasyon bu üç iterasyonun kombinezonu olmalıdır. Bu üç yeni iterasyonda,amaçvektörlerini değerleri hesaplanabilir ve tercihin doğrudan karşılaştırılması için karar vericiye sunulabilir. Bu halde neticelenen öncelikler 3 tane adım doğrultu vektörünü sonraki yeni iterasyonun takip edeceği bir tek adım doğrultu vektörine birleştirmek için kullanılır. Bilinen bir optimal sonuca karşı burada bahsettiğimiz yaklaşımı test etmek istediğimizden bir fayda fonksiyonu kullanmak istiyoruz. Bu fayda fonksiyonu $u(x)=x_1 x_2$ ile verildiğinden bu 3 yeni iterasyondaki fayda değerleri

$u(x_1)=3.8032 \quad u(x_2)=2.0167 \quad u(x_3)=10.2868$ olur.

Üç adım doğrultu vektörünün bir kombinezonu için ilgili ağırlıklar olarak bu değerler kullanılarak, ağırlıklandırılmış (normalize edilmiş) λ vektörü

$$\lambda = [0.2364 \quad 0.1254 \quad 0.6382]^T$$

elde edilir ki bununla dabirleştirilmiş doğrultu vektörü dx i ve yeni birleştirlmiş iterasyon x i aşağıdaki şekilde elde ederiz

$$dx = \begin{bmatrix} 1.3795 \\ 1.1175 \\ -2.4970 \\ -3.6146 \\ 12.1535 \\ 6.9671 \end{bmatrix} \quad x = \begin{bmatrix} 2.3795 \\ 3.1175 \\ 4.5030 \\ 7.3854 \\ 14.1535 \\ 7.9671 \end{bmatrix} \quad x_b = \begin{bmatrix} 4.8672 \\ 5.1328 \\ 0.0000 \\ 0.8672 \\ 36.0704 \\ 20.5312 \end{bmatrix}$$

yeni iterasyon daha önce kullandığımız yenileştirilmiş(güncelleştirlmiş) ve ρ adım boyu 0.5 alınarak elde edilir. x_b sınır noktası da aynı bağıntı kulanılarak elde edilir. Fakat bu sefer $\rho=1.0$ alınır. Bu iterasyonlar birazdan bahsedeceğimiz hariç, aynı öncekiler gibi devam eder. Şimdi sonraki iterasyon ulaşan ve birleştirilmiş adım doğrultu vektörü üreten x_b yi de kullanıyoruz.

Tablo 6.1 de 3 başlangıç noktası için ilk 10 iterasyonun özeti sunuluyor. Sadece çözüm vektörünün ilk 2 bileşeni gösterilmiştir. Diğer slack ve yapma değişkenler gösterilmemiştir. Tablodaki son satır mevcut en iyi sınır(son) noktasını ve bunun birleştirilmiş değer vektörünü göstermektedir.

Tablo 6.1 3 başlangıç noktasından çözümün izlenmesi(ρ=0.5)

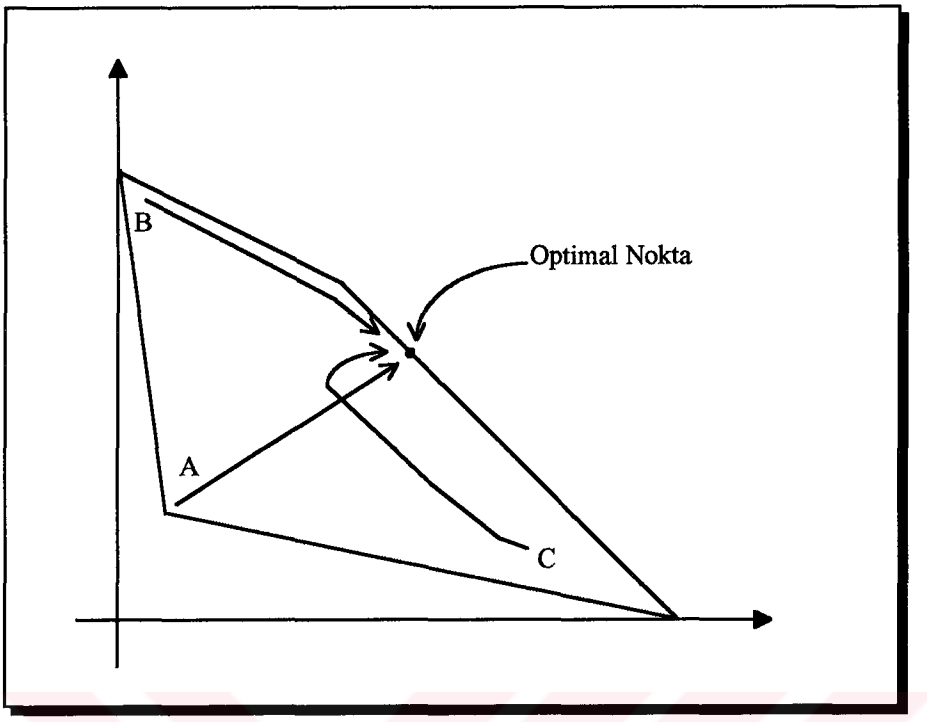
	A Noktası			B Noktası			C Noktası		
k	x_1	x_2	u	x_1	x_2	u	x_1	x_2	u
0	1,0000	2,0000	2,0000	0,5000	7,5000	3,7500	7,0000	1,5000	10,500
1	2,3795	3,1175	7,4181	2,0416	6,6980	13,6746	6,1225	2,4743	15,149
2	4,1683	4,4086	18,3765	3,4271	6,0258	20,6508	4,8457	4,1523	20,120
3	4,6976	4,6716	21,9455	3,9025	5,7978	22,6261	4,4959	4,8822	21,949
4	4,8688	4,8167	24,4515	4,1617	5,6749	23,6168	4,4602	5,2171	23,269
5	4,9959	4,8294	24,1271	4,3241	5,5834	24,1431	4,5274	5,2946	23,971
6	5,1527	4,7439	24,4439	4,4928	5,4526	24,4973	4,6546	4,4011	24,401
7	5,2230	4,7098	24,5994	4,8171	5,1498	24,8069	4,9230	5,0154	24,690
8	5,2611	4,6933	24,6923	5,0295	4,8850	24,8769	5,1336	4,8258	24,774
9	5,2843	4,6841	24,7521	5,1794	4,8046	24,8853	5,2055	4,7663	24,811
10	5,2993	4,6784	24,7922	5,2091	4,7795	24,8968	5,2340	4,7461	24,841
Son	4,9948	5,0052	25,0000	4,7361	5,2639	24,9304	4,8296	5,1704	24,991

Bu problemin doğru optimal çözümünü tekrar hatırlarsakx_1=x_2=5 noktası optimal çözüm u*=25.0 de fayda fonksiyonu değeri idi. Tablo daki A noktası daha önce tanımlanmiş x_0 başlangıç noktasını göstermektedir. B ve C noktaları ise aşağıda gösterilen iki ilave başlangıç şartına karşılık gelmektedir.

$$\text{B noktası} : x_0 = \begin{bmatrix} 0.5 & 7.5 & 2.0 & 0.5 & 3.5 & 28.0 \end{bmatrix}^T$$

$$\text{C noktası} : x_0 = \begin{bmatrix} 7.0 & 1.5 & 1.5 & 6.0 & 49.5 & 4.5 \end{bmatrix}^T$$

Tablo da özetlenen 3 çözüm Şekil 6.2 de gösterilmiştir.



Şekil 6.2: İç çözümün izlenmesi

Şimdiye kadar sunulan sonuçların hepsi adım boyu çarpanı olarak 0.5 kullanılarak elde edildi. Şimdi ρ=0.9 şeklinde daha büyük bir adım attığımızda çözümün ilerlemesini inceliyoruz. ρ=0.9 ile bu algoritmayı kullanarak ve daha önce tanımladığımız A noktasından başlayarak daha öncekiler gibi aynı adımları tekrarlıyoruz. Böylece elde ettiğimiz yeni sonuçları Tablo 6.2 de özetliyoruz. Bu sayade yeni çözüm, ρ=0.5 kullanılarak elde edilen çözümün hemen yanında sunuluyor.

Tablo 6.2 Adım boyunun çözüme etkisi

	ρ=0.5			ρ=0.9		
k	x_1	x_2	u	x_1	x_2	u
0	1,0000	2,0000	2,0000	1,0000	2,0000	2,0000
1	2,3795	3,1175	7,4181	3,7255	4,4282	16,4969
2	4,1683	4,4086	18,3765	4,7970	4,9140	23,5727
3	4,6976	4,6716	21,9455	4,8604	5,0142	24,3708
4	4,8688	4,8167	23,4515	4,6871	5,2437	24,5779
5	4,9959	4,8294	24,1271	4,7861	5,1746	24,7663
6	5,1527	4,7439	24,4439	4,6735	5,3019	24,7782
7	5,2230	4,7098	24,5994	4,6867	5,2976	24,8282
8	5,2611	4,6933	24,6923	4,6958	5,2940	24,8593
9	5,2843	4,6842	24,7521	4,7013	5,2929	24,8789
10	5,2993	4,6784	24,7922	4,7049	5,2905	24,8916
Son	4,9948	5,0052	25,0000	4,9958	5,0042	25,0000

Adım boyunun seçimi kesinlikle önemlidir. Çok küçük bir adım çözümün ilerlemesini yavaşlatırkençok büyük adımlar çözümü çabucak optimal çözümün hemen yakınına götürecektir. Fakat bir doğrultudan diğerine geçerken daha çok zaman tüketir. Çözüm işleminin ilerlemesine bağlı olarak adım boyu çarpanını değiştirecek bir metod un gelecek çalışmalar için çok kıymetli olacağı görünüyor.

7. Özet ve İddialar

Bu bölümde ÇALP problemleri için Affine -scaling primal algoritmasını kullanan yeni bir algoritma sunuldu. Yaklaşım, tek tek maliyet vektörlerini projekte edilmesine, projekte edilmiş gradientlerin konvex kombinezonuna ulaşmak için kısıtlar polytope'unun sınırlarının incelenmesine ve ilerlemenin feasible bölgenin dışı üzerinde olması yerine içi yardımıyla yapılmasına bağlıdır. İçi yardımıyla yapılan ilerleme, problem büyüklüğüne daha az hassas olduğundan yapılan arama işlemi otomatikman bir potansiyel kazanca sahiptir. Çünkü köşelerin sayısı çözüm işleminde herhangi bir rol oynamaz. Bu alanda daha sonra yapılacak çalışmalar, algortimanın çeşitli safhalarında adım boylarının seçimine karar veren herbir amacın adım doğrultu vektörlerinin bir tek adım doğrultu vektörine birleştiren ilave metodlar geliştiren yöntemler üzerinde yoğunlaşmalıdır. Durdurma (sona erdirme) işlemi geliştirilmeli ve yakınsaklık özellikleri incelenmelidir.

KAYNAKÇA

1. T. M. Cavalier and A.L. Soyster, Some Computational experience and a modification ofKarmarkar'salgorithm. *12th Int. Symp. Mathematical Programming, Massachussets Institute of Technology,* August (1985)
2. T. M. Cavalier and Kenneth C. Shall, Implementing an affine scaling algorithm for linear programming. *Computers & Operations Research* **14** 341-347. (1987)
3. R.J. Vanderbei, M.S. Meketon and B.A. Freedman, A modification of Karmarkar's linear programming Algorithm. *Algorithmica* **1** 395-407 (1986)
4. M.J. Todd and B.P. Burrell, An extension of Karmarkar's algorithm for linear programming using dual variables. *Algorithmica* **1** 409-424 (1986)
5. M.H. Dodani and A.J.G. Babu, Karmarkar's projective method for linear programming:A computational appraisal, *Computers & Industrial Engineering* **16** 189-206 (1989)
6. D.M.Gay, A variant of Karmarkar's linear programming algorithm for problems in standard form, *Mathematical Programmig* **37** 81-90 (1987)
7. I.J.Lustig, Feasibility issues in primal-dual interior-point method for linear programming, *Mathematical Programmig* **49** 145-162 (1991)
8. E.R.Barnes, A variation on Karmarkar's algortihm for solving linear programming problems, *Mathematical Programmig* **36** 174-182 (1986)
9. I.Adler, M.G.C. Resende, G. Veiga and N. Karmarkar, An implemantion of Karmarkar's algorithm for linear programming, *Mathematical Programmig* **44** 297-335 (1989)
10. R.J. Vanderbei, Affine-scaling for linear programming with free variables, *Mathematical Programmig* **43** 31-44 (1989)
11. M.C.Ferris and A.B.Philpott, On the performance of Karmarkar's algortihm, *Operation Research Society* **39** 257-270 (1988)
12. A. Arbel, A multiobjective interior primal-dual linear programming algorithm, *Computers & Operations* Research **21** 433-445 (1994)

13. A. Arbel, An İnterior multiobjective linear programmingalgorithm, *Computers & Operations Research* **20** 723-735 (1993)
14. Andrew M. Rockett and John C. Stevenson, Karmarkar's Algorithm. *Byte* v 12n 10Sep 1987 11p between p 146-160
15. M.S.Bazaraa, J.J.Jarvis, H.D.Sheralil, Linear programming and network flows. *John Wiley*, New York (1990)
16. A. Arbel, Exploring interior-point linear programming:Algorithms and Software. *MIT Press*, Massachussets (1993)
17. F.S. Hillier and G.J.Liebermann, Introduction to Operations Research. 5th Ed. *McGraw-Hill*, New-York (1990)

ÖZGEÇMİŞ

1970 yılında Kemaliye 'de doğdum. İlkokul ve ortaokuldan sonra liseyi Kadıköy Mehmed Bayazıd Lisesi 'nde bitirdim. Aynı yıl üniversite sınavlarına girip Yıldız Üniversitesi Matematik Mühendisliği Bölümünü kazandım. 1991 yılında Aynı bölümü 3. sırada bitirdikten sonra Yüksek lisans sınavlarına girip aynı bölümde Yüksek Lisans okumaya hak kazandım. Bir yıl ingilizce yabancı dilinde hazırlık okudum. Halen yüksek lisans eğitimime devam etmekteyim. Aynı zamanda bahsettiğim bölümde Araştırma Görevlisi olarak çalışmaktayım. Uygulamalı Matematik, Linear programlama ve bilgisayar konuları ilgi alanlarımdır.