OPTİMİZASYONA GİRİŞ

TOKAT GAZİOSMANPAŞA ÜNİVERSİTESİ BİLGİSAYAR MÜHENDİSLİĞİ
TOGU - CENG

Dr. Mahir Kaya TOGU-CENG

Optimizasyon NEDİR?

Kelime anlamı “Mümkün olan en iyi duruma getirmek” demektir.

İnsanlar, karşılaştıkları problemleri birer model olarak algılayarak bunları çözme yolunu seçmişlerdir. Günlük yaşamda kullanılan bu teknik, bilgisayar teknolojisinin gelişmesi ve matematik bilimi ile entegre olması sonucu bilim dünyasına yansımıştır.

Bilimin gelişmesiyle kurulan matematiksel modeller ilk başta doğrusal olup az sayıda değişkene sahipken, gerçek yaşam problemlerinin daha karmaşık yapıda olmasından dolayı doğrusal olmayan modeller kurulmuş ve bu modellerin çözümlenebilmesi için optimizasyon kavramı geliştirilmiştir.

Bir problem için belirli koşullar altında mümkün olan alternatifler içinden en iyisini seçmeyi amaçlayan optimizasyon günümüzde rekabetin artması, teknolojinin hızla gelişmesi ve kullanılan ortak kaynakların kısıtlı hale gelmesi ile ortaya çıkan karmaşık sistemlerin çözümünde klasik yöntemlerin kullanılma zorluğu nedeniyle oldukça önemli bir hale getirmiştir.

Optimizasyon, eldeki kısıtlı kaynakların en iyi şekilde kullanılmasıdır. Matematiksel olarak ise optimizasyon, bir fonksiyonunun amacına uygun olarak maksimize veya minimize edilmesi yani amaç fonksiyonunun en iyi değerini veren kısıtlardaki değişkenlerin değerinin bulunmasıdır.

Optimizasyon, en basit tanımıyla en iyi çıktıyı üretecek girdileri seçmeye yardımcı olarak sonuçları iyileştirmeyi sağlar.

Dr. Mahir Kaya TOGU-CENG

Optimizasyonda varılmak istenen sonuca **optimizasyon teknikleri** ile ulaşılmaya çalışılır. Optimizasyon teknikleri ile elde edilecek çözümün en iyi çözüm olması amaçlanır ve elde edilen bu en iyi çözüm "**Optimum Çözüm**" olarak adlandırılır.

Optimizasyonda genel olarak belirlenen **amaç**, **maksimum** kârın veya **minimum** maliyetin elde edilmesi için üretim miktarını kısıtlara bağlı olarak tespit etmektir, günümüzde ise optimizasyon çok çeşitli endüstri kesimlerinde inşaattan, tekstile, mimarlıktan, otomotive birçok alanda kendine uygulama sahası bulmaktadır.

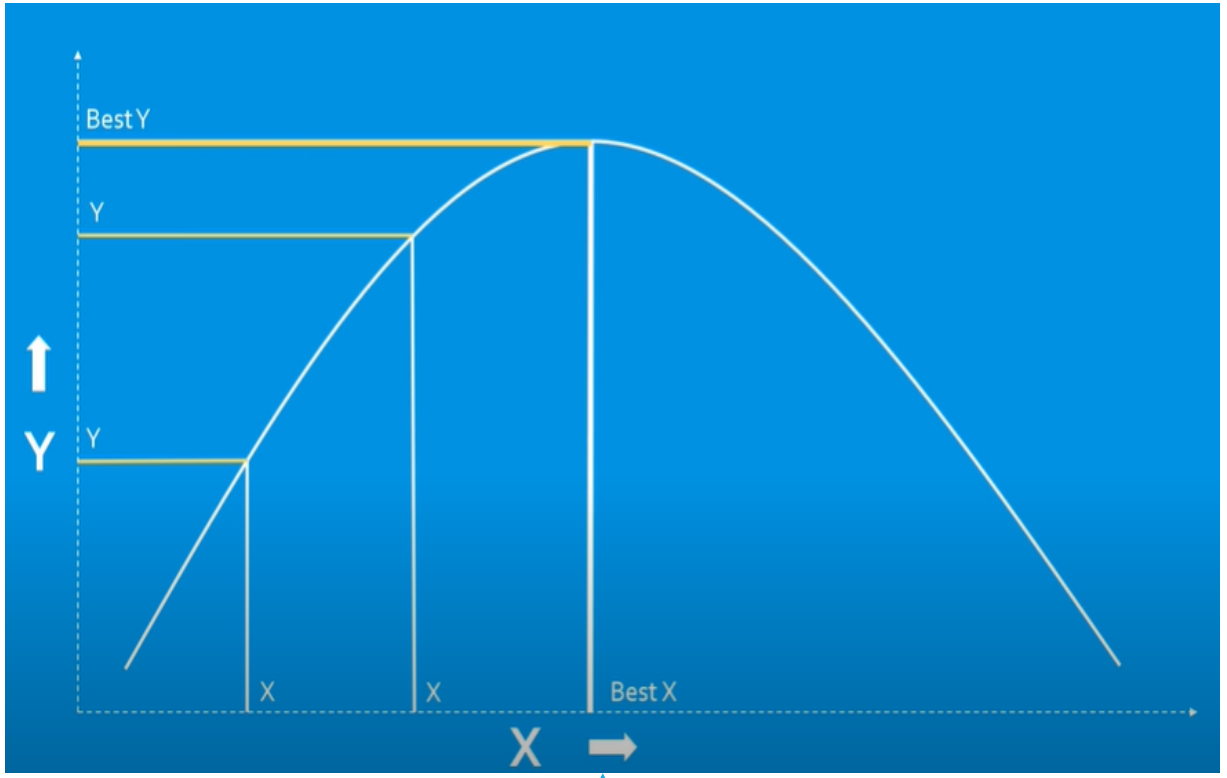
Örneğin, bir parabol verildiğinde, farklı x değerleri deneyerek en büyük y'yi elde etmeye çalıştığımızı düşünelim.

Farklı x'ler deneyerek, en sonunda buradaki x'i seçerek maksimum y değerini bulabiliriz.

Bu parabolün türevini alıp sıfıra eşitleyerek de çözülebilir.

Bu tarz basit problem için doğru çözümü görmek kolaydır. Fakat daha karmaşık problemler için, doğru çözümü hemen görmek zor olabilir, tahmin etmek ve kontrol etmek çok uzun sürebilir ve türevin sıfıra eşit olduğu değerleri bulmak zor olabilir.

Optimizasyon problemlerinin çoğunun yanıtını bulmak için **optimizasyon algoritması** adı verilen özel bir program türü kullanmamız gerekir.



- **Optimizasyon, çok çeşitli durumlara ve sorunlara uygulanabilir. Örneğin:**

Havayolu şirketleri, maliyeti minimize etmek için ekipleri ve uçakları planlar.

Yatırımcılar, riski minimize edecek, karı maksimize edecek portföyler oluşturmaya çalışır.

Üreticiler,üretim süreçlerini maksimum etkinlikle planlamaya çalışır.

Bireyler,evden işe giderken izledikleri güzergah için optimizasyon yaparlar.

Alışveriş yapılırken optimizasyon yapılır.

Mağazalar, ürün fiyatlarını ve çeşitlerini belirlerken optimizasyon yapar.

Elektronik ticaret siteleri, ürün yerleşiminde, reklam ve kampanya yönetiminde optimizasyon yapar.

Mühendisler, belirli bir maliyet için mümkün olan maksimum yükü taşıyabilecek bir köprü tasarlar.

Optimizasyon Problemi Çözme Yöntemleri

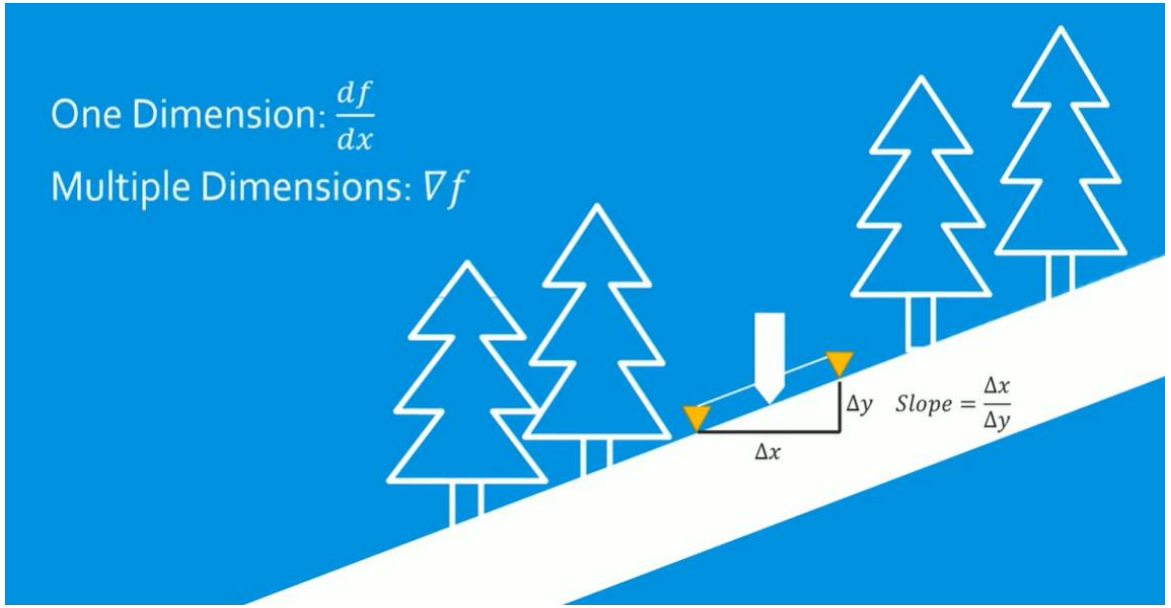
Gradyan tabanlı optimizasyon ve **gradyan içermeyen optimizasyon** olarak adlandırılan**,** optimizasyon problemlerini çözmek için iki ana yöntem vardır.

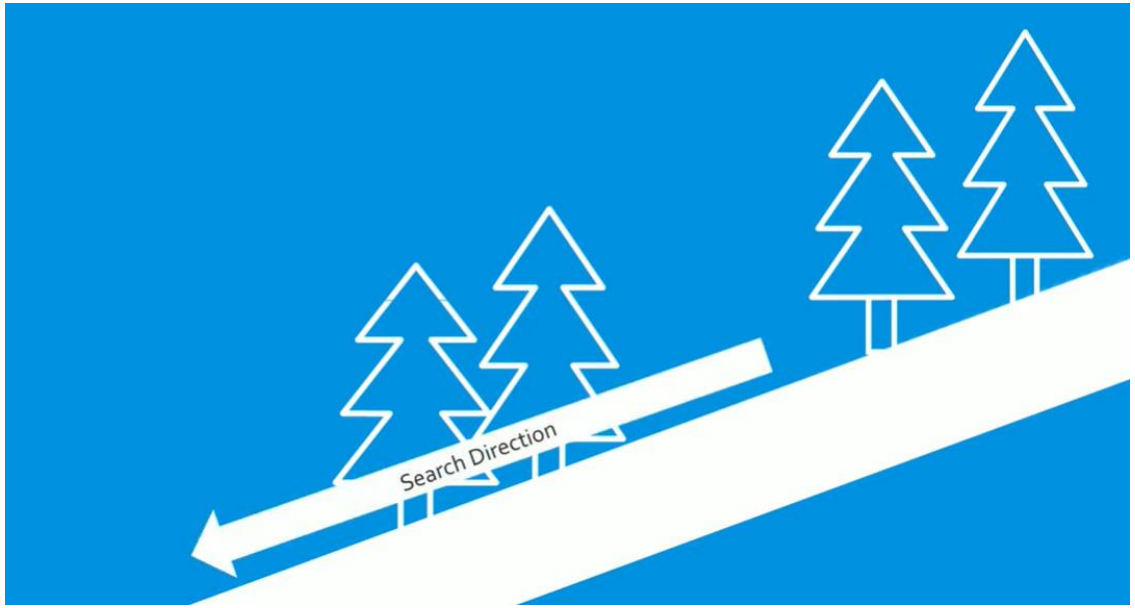
Gradyan Tabanlı Optimizasyon

Gradyan tabanlı çözücüler, bir fonksiyonun optimum değerini bulmak için türevleri kullanır.

Temel düzeyde, gradyan tabanlı algoritmaların üç ana adımı vardır:

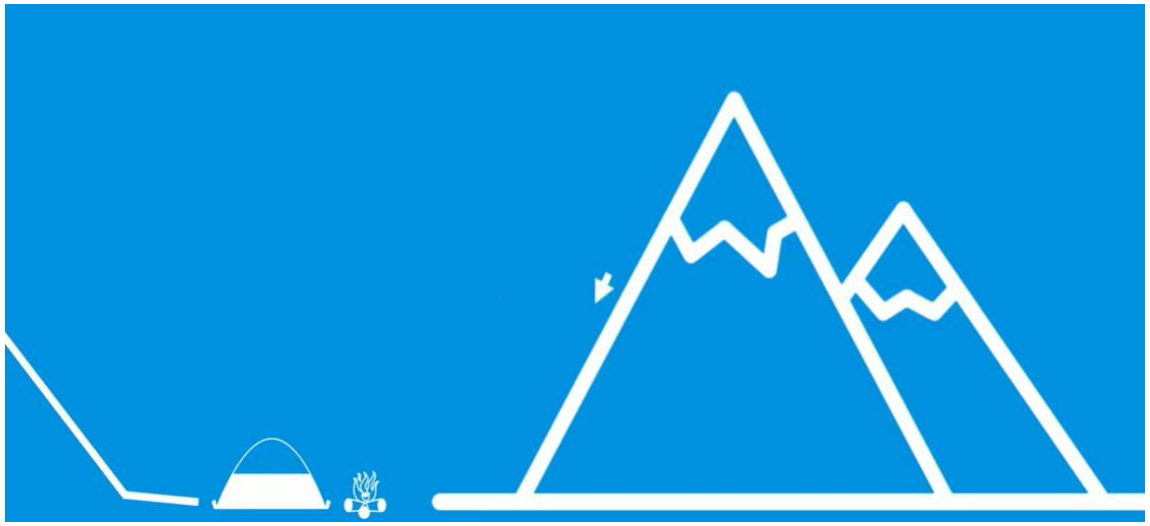
1- Arama Yönü (Search Direction): İlk adım, gidilecek yönü seçmektir. Türev mevcut konumunda alınarak eğim değerlendirir. Bir boyutta bu türev eğimdir. Birden fazla boyutta buna gradyan denir. Daha sonra bu bilgi diğer kurallarla birlikte gidilecek yönü seçmek için kullanır. Buna arama yönü denir.





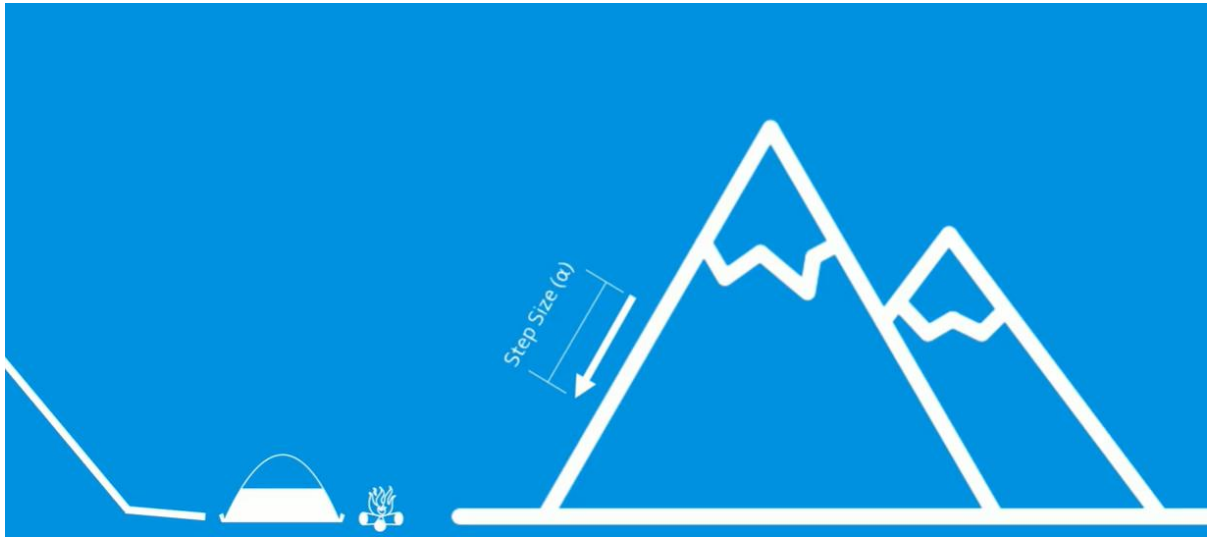
2- Adım Boyutu (Step Size): Bir sonraki adım, seçilen yönde ne kadar ilerleyeceğinize karar vermektir. Tek bir yönde çok ileri gitmek istemeyiz, aksi takdirde farklı bir dağa geri dönebilirsiniz. Çok küçük ilerleme ise hedefimize ulaşmayı geciktirecektir.

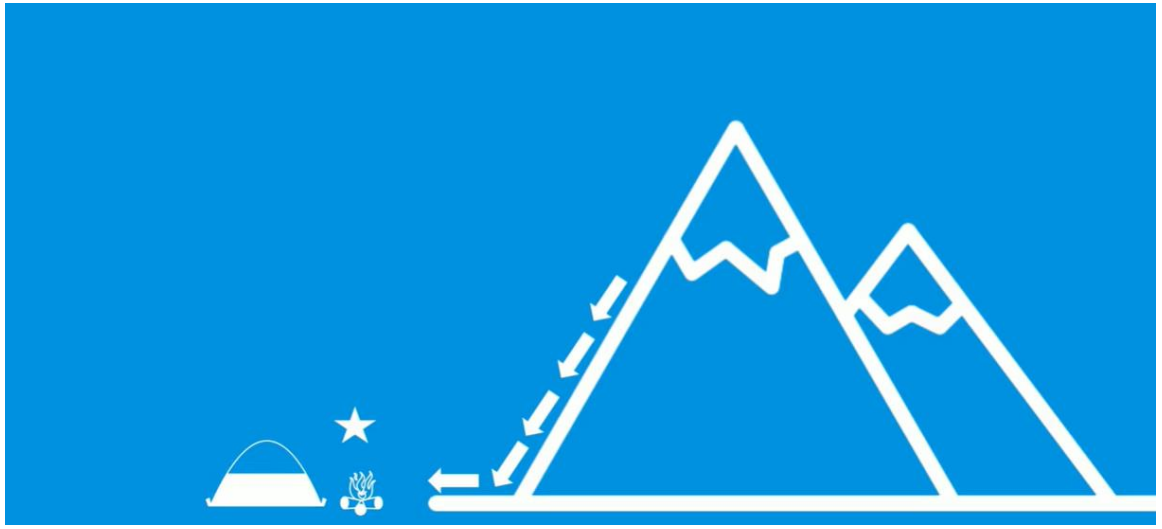
Büyük adım boyutu



Küçük adım boyutu

Ancak, hedefe doğru bir ilerleme kaydetmek için yeterince ileri gitmek istiyorsak, probleme göre uygun büyüklükte bir adım boyutu seçerek, hedefimize ulaşabiliriz.





3- Yakınsama Kontrolü (Convergence Check): Bir yön ve adım boyutu seçildikten sonra, seçilen yönde hareket edilir. Sonra dibe (minimum nokta) ulaşıp ulaşmadığını kontrol edilir. Değilse, yeni bir yön ve adım boyutu seçmek için eğim (türev, gradyan) tekrar kullanır. Bu, dağın dibine veya minimuma ulaşana kadar devam eder. Bu yakınsama (convergence) diyoruz.

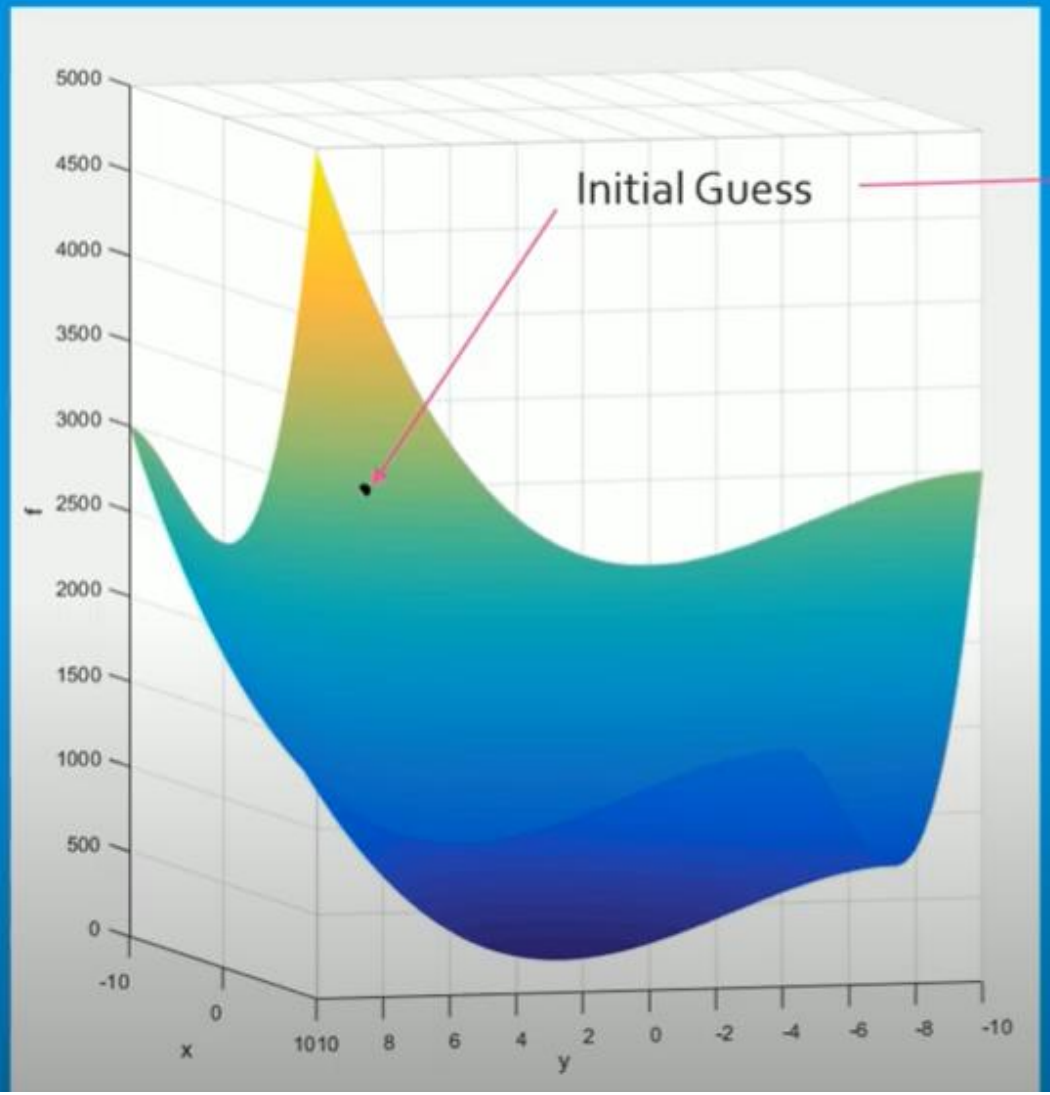
Gradyan tabanlı algoritmaların kendi güçlü ve zayıf yönleri vardır. Yaygın olarak kullanılırlar, hızlı performans gösterirler ve büyük problemlere iyi ölçeklenirler. Ancak, düzgün, sürekli fonksiyon gradyanları gerektirirler ve bu gradyanları hesaplamak hesaplama açısından pahalı olabilir. Pek çok gradyan tabanlı optimizasyon yöntemi, global bir optimumdan ziyade yerel minimumları bulmaya yatkındır, bu da tüm haritadaki en alçak nokta yerine en yakın vadinin dibini bulacakları anlamına gelir. Gradyan tabanlı optimize ediciler güçlü bir araçtır, ancak tüm optimizasyon problemlerinde olduğu gibi, hangi yöntemin doğru yöntem olduğunu bilmek deneyim ve pratik gerektirir.

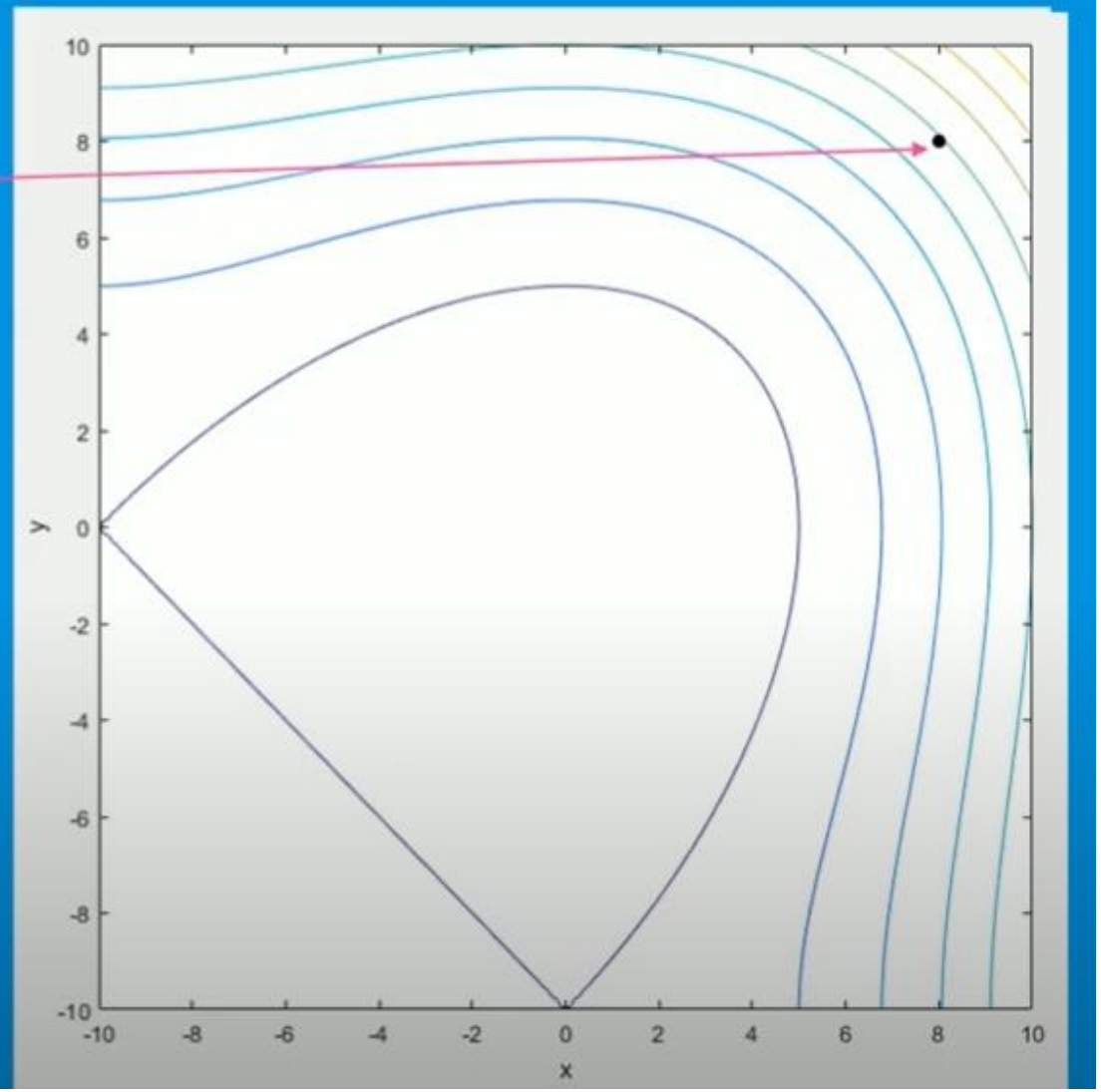
Örnek

Minimize

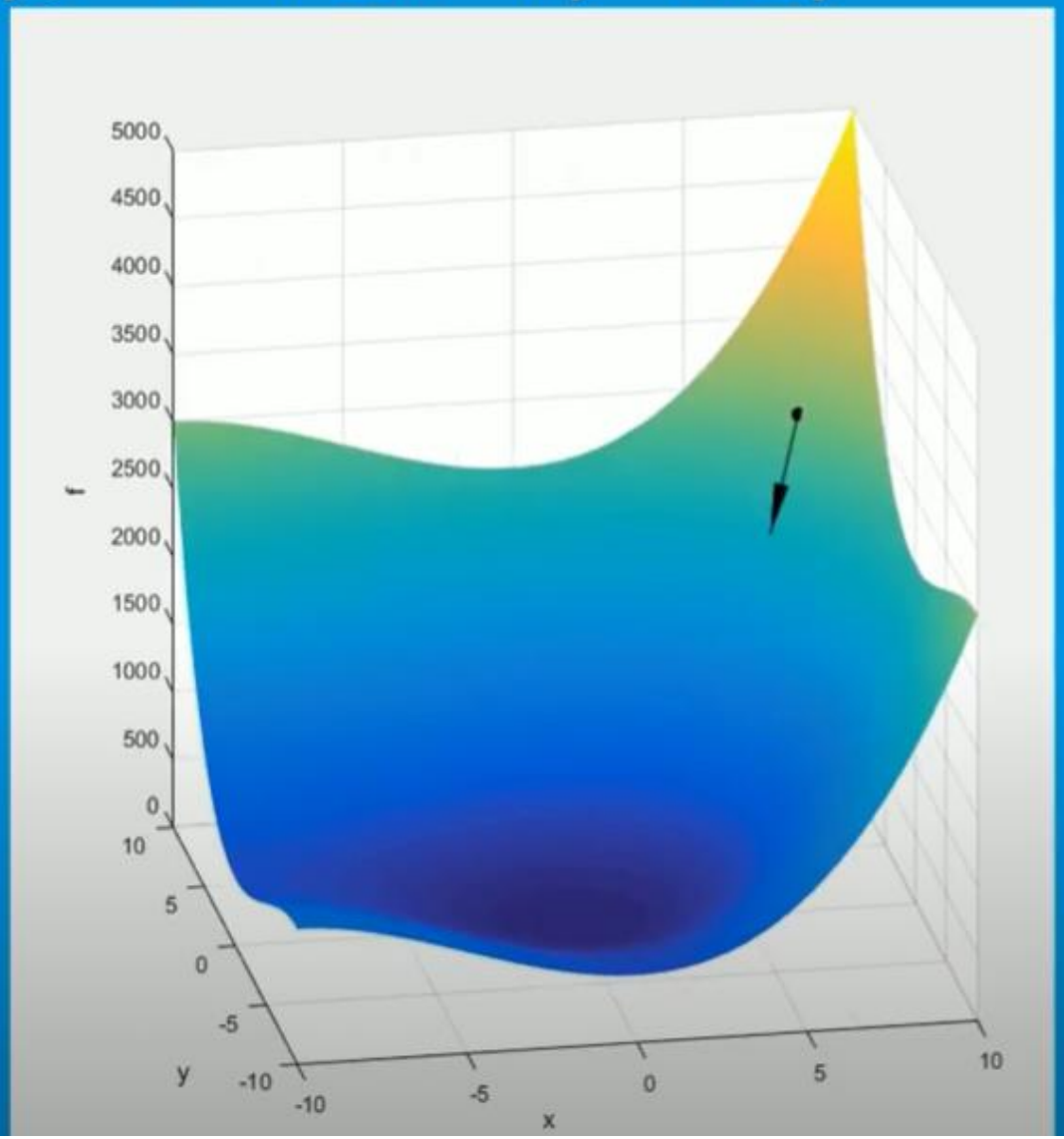
$$f(x,y) = x^3 + 15x^2 + y^3 + 15y^2$$

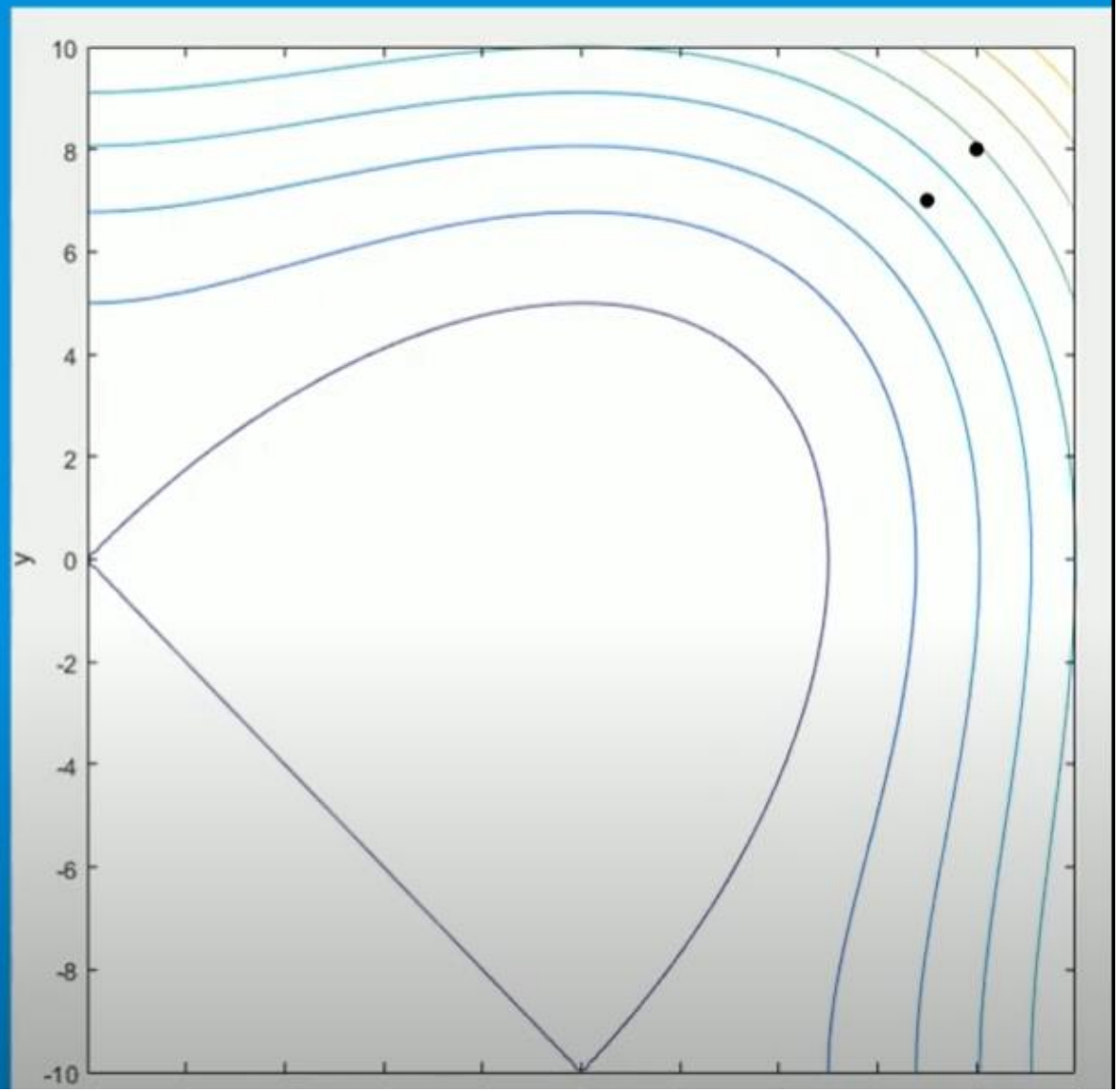
$$f(x,y) = x^3 + 15x^2 + y^3 + 15y^2$$



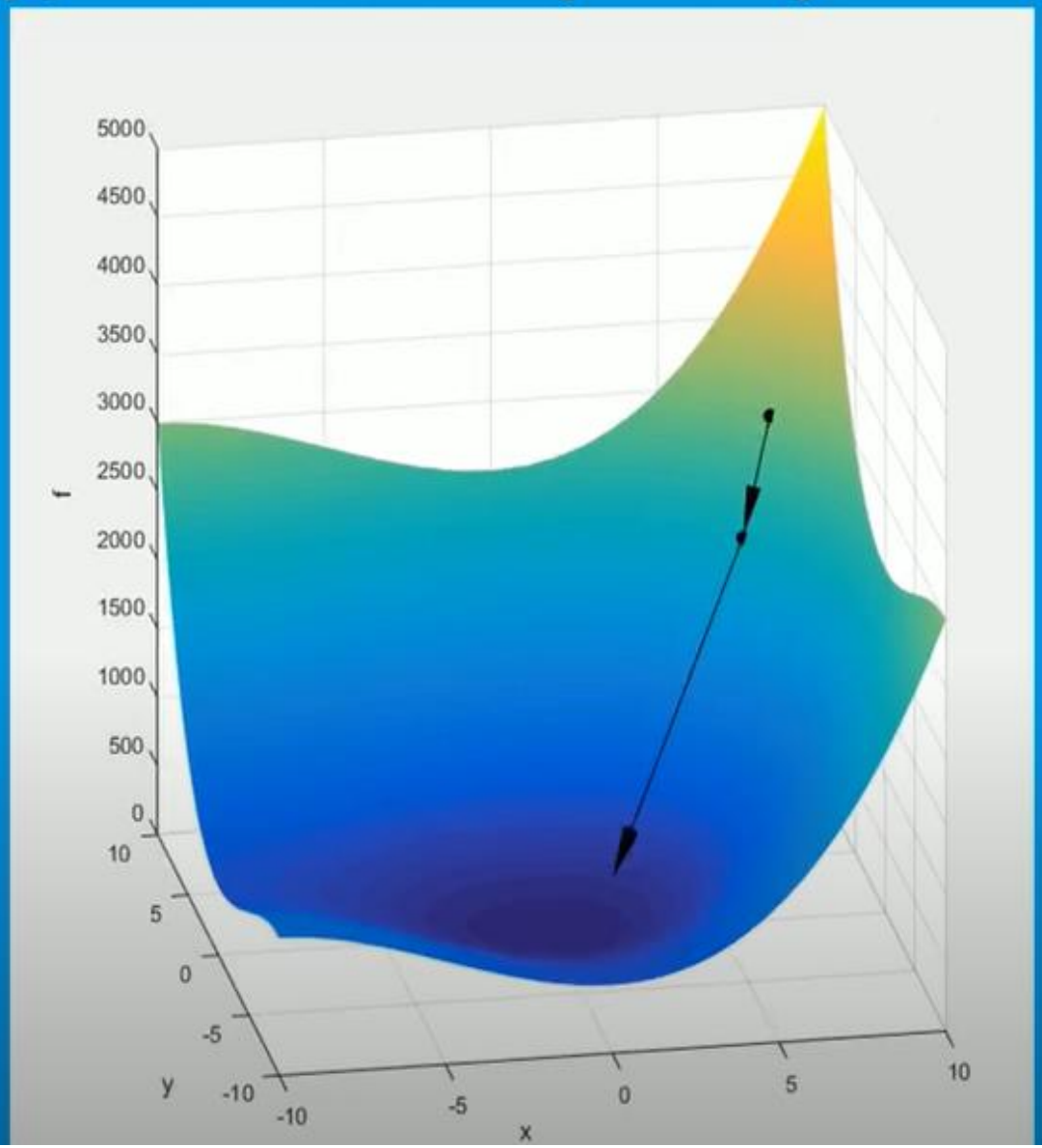


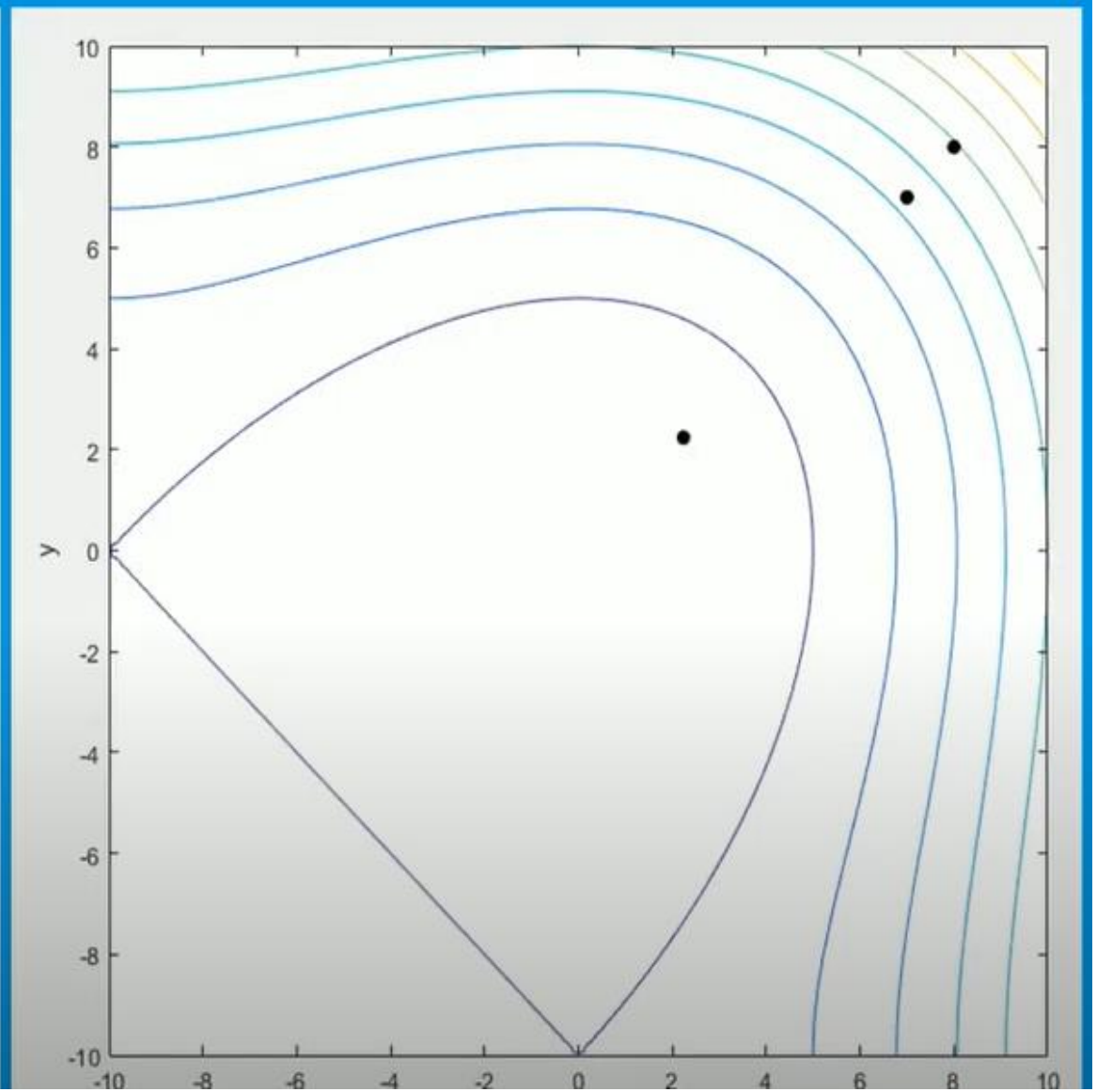
$$f(x,y) = x^3 + 15x^2 + y^3 + 15y^2$$





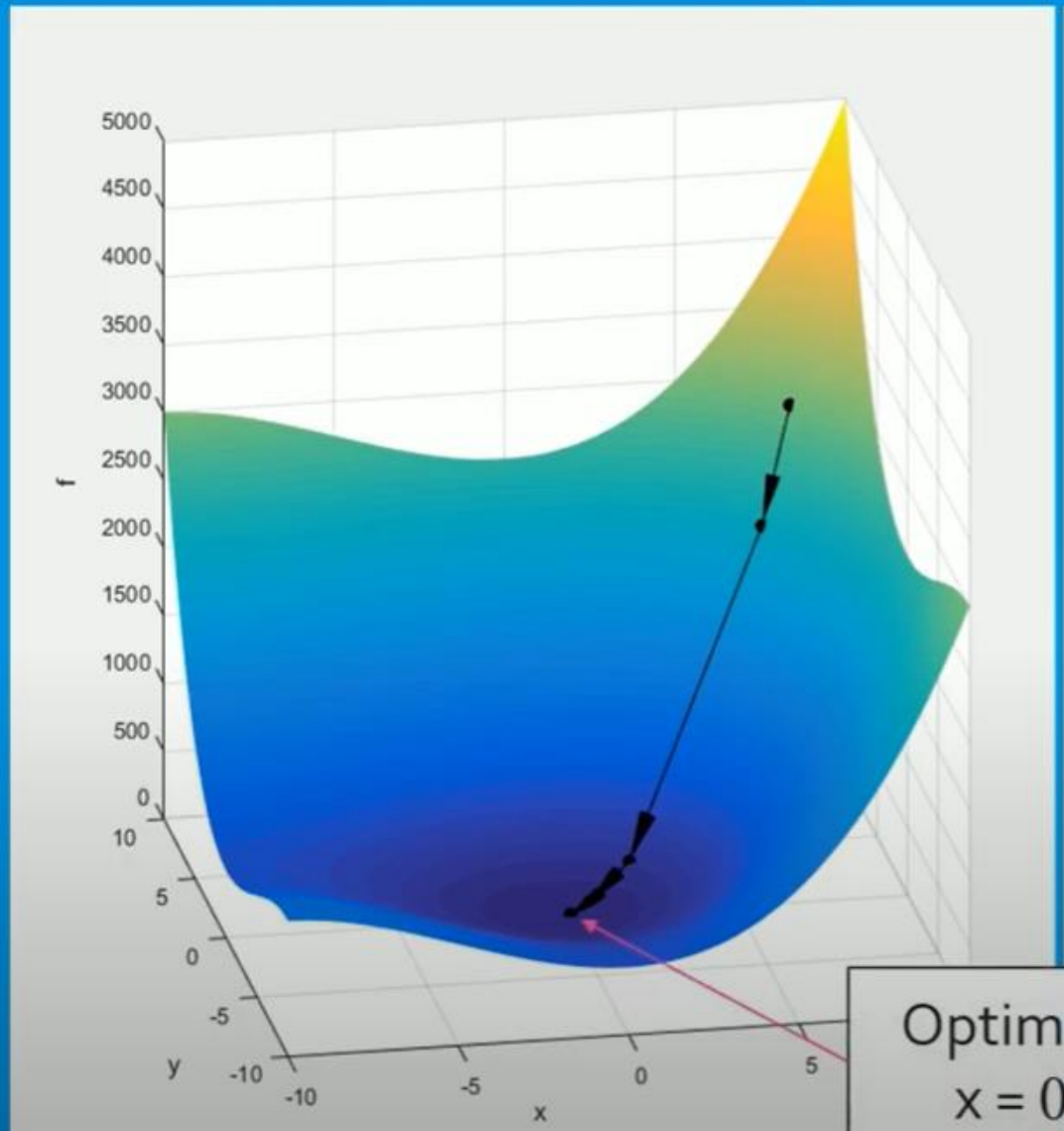
$$f(x,y) = x^3 + 15x^2 + y^3 + 15y^2$$

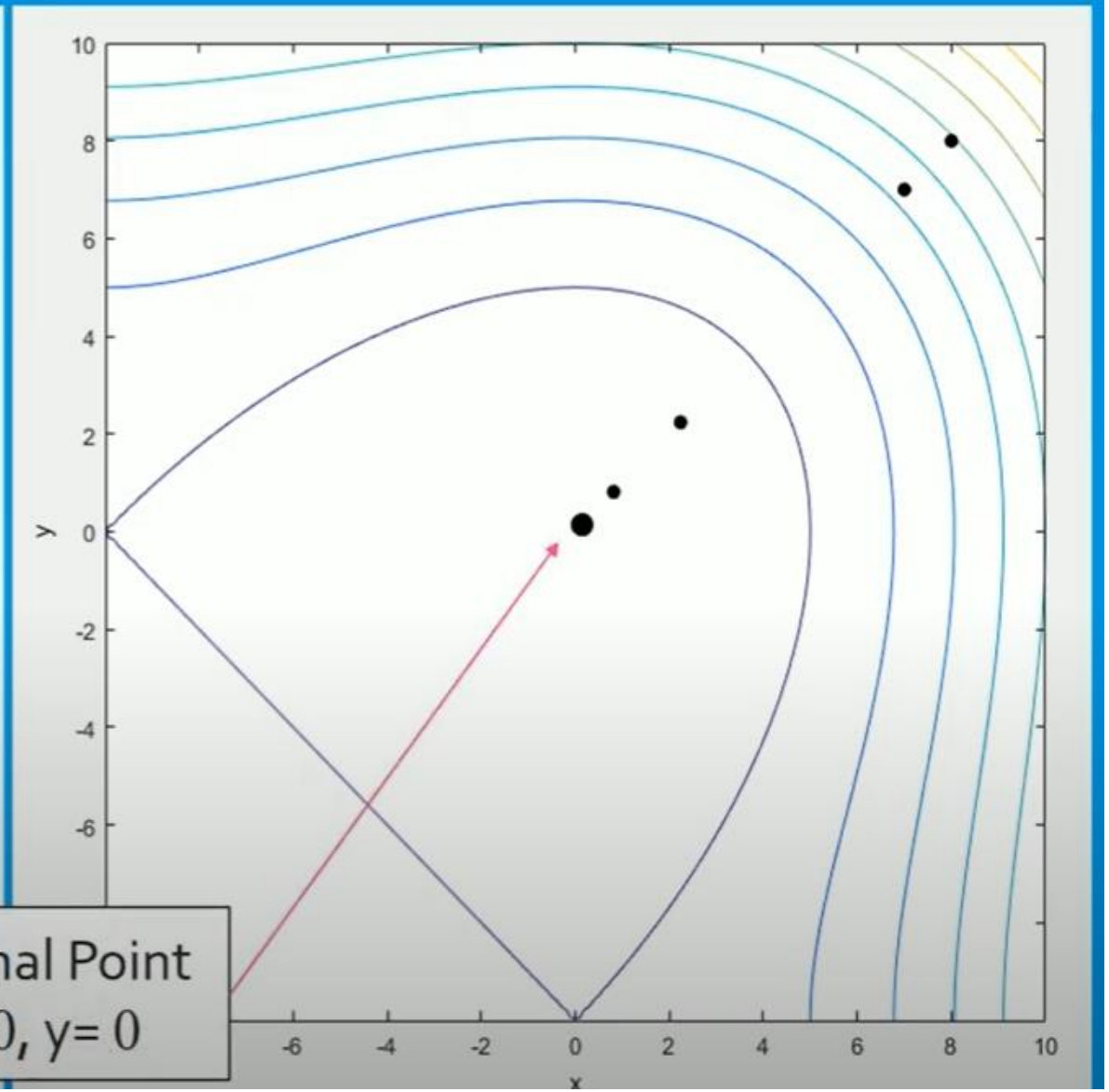




$$f(x,y) = x^3 + 15x^2 + y^3 + 15y^2$$

Dr. Mahir Kaya TOGU-CENG



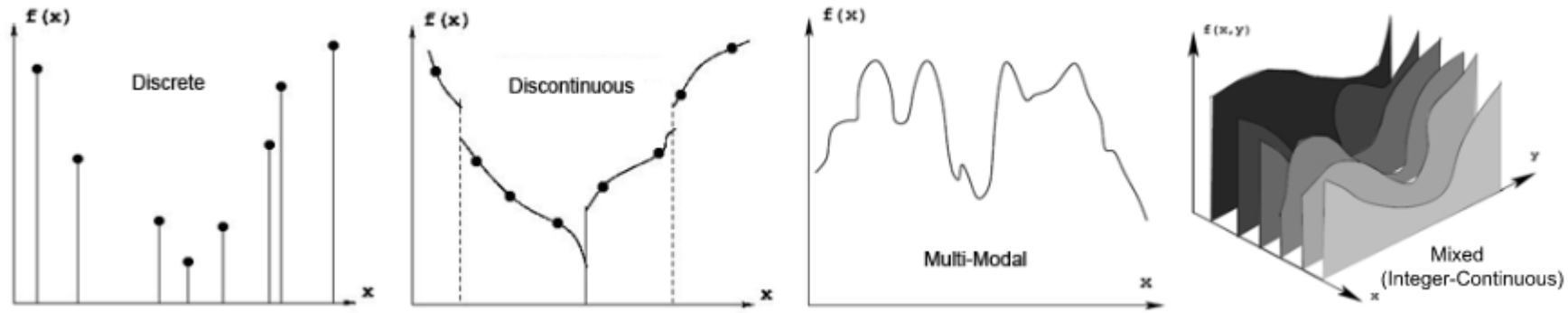


Optimal Point
x = 0, y= 0

Gradyan İçermeyen Algoritmalar

Gradyan tabanlı optimize ediciler, yüksek boyutlu, doğrusal olmayan kısıtlı, dışbükey problemler için yerel minimumları bulmada etkilidir; ancak, aşağıdaki sebeplerden dolayı uygulanması zor veya mümkün olmayabilir.

- türevlenemeyen fonksiyonlar
- karışık değişkenler (ayrık, sürekli, permütasyon)
- büyük boyutluluk
- çoklu yerel minimumlar
- çoklu amaçlar



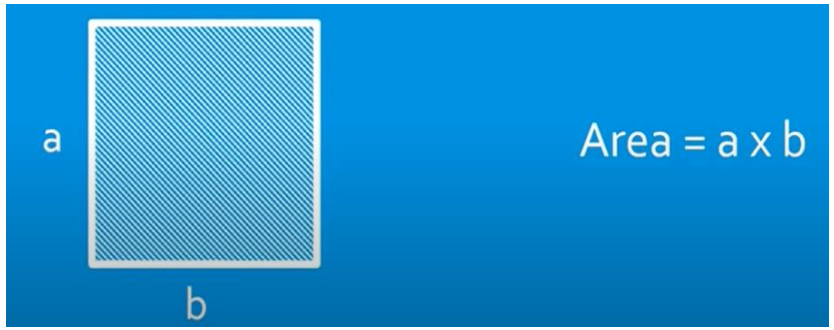
Gradyan tabanlı optimizasyon algoritmaları için sorunlu olabilecek çeşitli fonksiyon türlerini gösteren grafikler

- Gradyan tabanlı ve gradyan içermeyen optimizasyon yöntemleri arasındaki temel fark, gradyan içermeyen yöntemlerin türev gerektirmemesidir.
- Bu, gradyan içermeyen optimizasyon yöntemlerinin türevlerinin elde edilemediği veya elde edilmesinin zor olduğu optimizasyon problemlerinde kullanılabilecekleri anlamına gelir. Bu, gradyan içermeyen algoritmaları, uygulanabilecekleri problem türlerinde çok esnek hale getirir. Gradyan içermeyen algoritmaların en büyük dezavantajı, genellikle gradyan tabanlı algoritmalardan çok daha yavaş olmalarıdır.
- Gradyan içermeyen yöntemlerin temel gücü, gradyan tabanlı yöntemler kullanarak çözülmesi zor olan sorunları çözme becerileridir. Dahası, birçoğu global optimize ediciler olarak tasarlanmıştır ve bu nedenle global optimum aranırken çok sayıda yerel optimayı bulabilir.
- Çok çeşitli gradyan içermeyen algoritmalar ve bu algoritmaların birçok varyasyonu vardır. En yaygın olanlardan bazılarına genetik algoritmalar, parçacık sürü optimizasyon algoritması … örnek verilebilir.

Optimizasyon Probleminin Standart Yapısı

Amaç Fonksiyonu, optimize etmeye çalıştığınız değerdir.

Örneğin, mümkün olduğu kadar büyük bir dörtgen yapmaya çalıştığımızı düşünelim. Burada alan amaç fonksiyonu olacaktır.

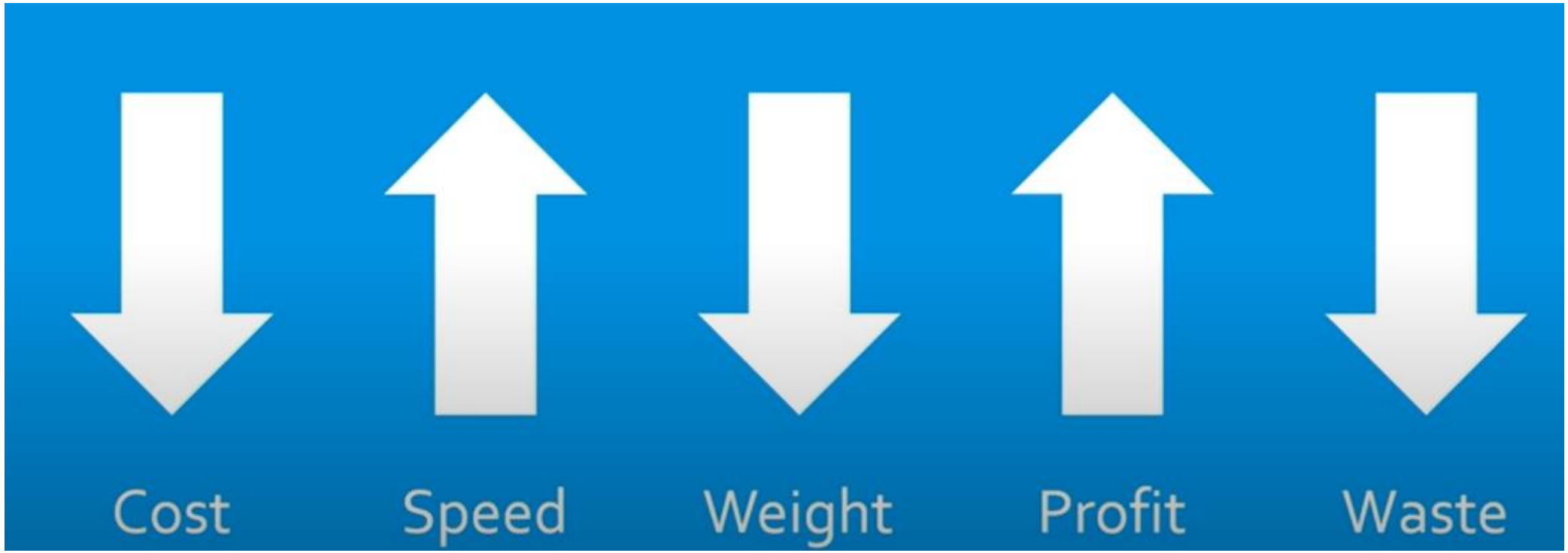


Optimizasyonun ana hedeflerinden biri, amaç değerini iyileştirmeye çalışmaktır; bu, onu minimize etmek, maksimize etmek veya belirli bir değere getirmeye çalışmak anlamına gelebilir.

Amaç fonksiyon değerine bakmak, bir optimizasyonun ne kadar iyi çalıştığını anlamanın en yaygın yollarından biridir.

Optimizasyon sorunları genellikle en aza indirgeme f (x) biçiminde yazılır. Burada f, amaç fonksiyonudur.

$$\underset{x}{\text{minimize}}\ f(x)$$



Diğer amaç fonksiyon örnekleri, maliyeti en aza indirmek, hızı en üst düzeye çıkarmak, ağırlığı en aza indirmek, karı en üst düzeye çıkarmak veya israfı en aza indirmek olabilir. Seçilen belirli amaç fonksiyonu, çözülecek probleme ve onu çözmedeki hedeflerinize bağlıdır.

- Karar değişkenleri, probleminizin girdileridir.
- Optimize edicini algoritmamızın amaç fonksiyon değerini iyileştirmek için karar değişkenlerini değiştirmesine izin verilir.
- Daha önceki kare örneğinde, karar değişkenleri kenar uzunlukları olacaktır. Bu değişkenlere tasarım değişkenleri de denir.
- Daha önce belirtildiği gibi, optimizasyon sorunları genellikle f(x) fonksiyonunu minimize etme biçiminde yazılır.

$$\underset{x}{minimize}\ f(x)$$

- Burada x, bir veya daha fazla karar değişkenini temsil eder. Genel olarak, karar değişkenleri ne kadar fazlaysa, optimizasyon probleminin çözülmesi o kadar zor hale gelir.

$$\underset{x}{minimize}\ f(x_1, x_2, x_3, x_4, x_5 \ldots)$$

- ÖZETLE;
- Amaç fonksiyonu, optimizasyon probleminde optimize etmeye çalıştığımız değerdir. Amaç fonksiyonu ya minimize ya da maksimize edilir
- Karar değişkenleri, optimizasyon algoritmasının seçmesine veya değiştirmesine izin verilen değerlerdir.

- ÖRNEK:

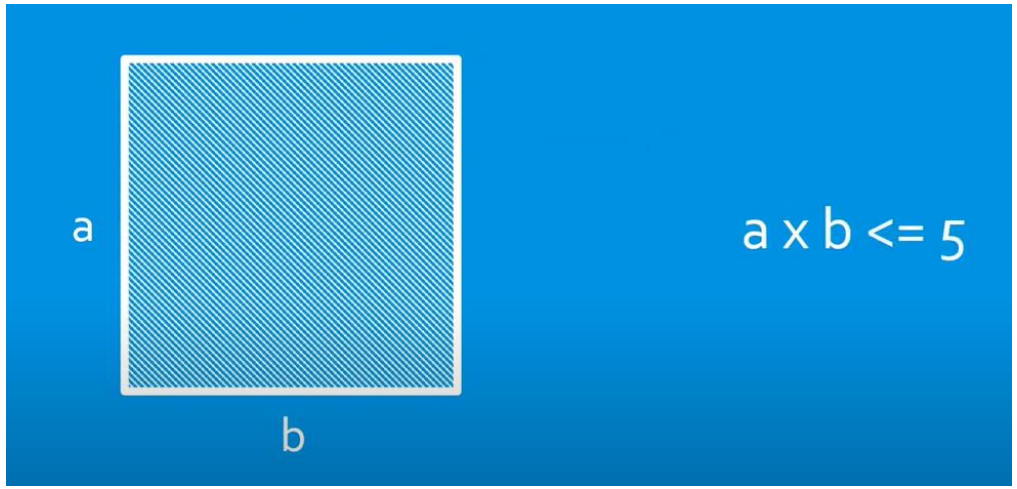
a

b

Soldaki dörtgenin çevresi 24 birimdir. Alanın maksimum olmasını sağlayan a ve b değerleri nedir?

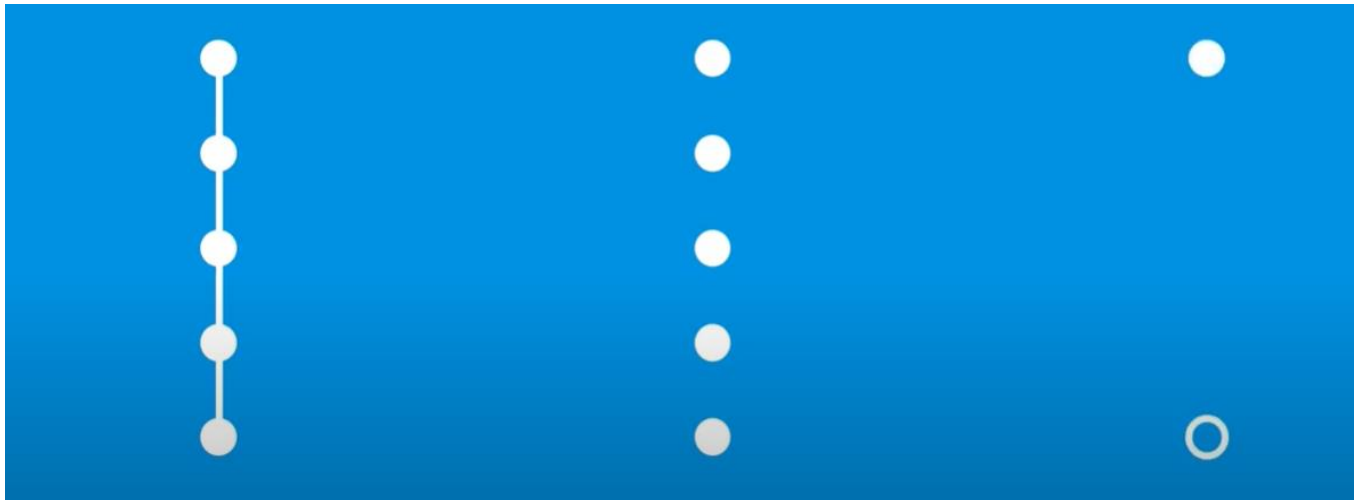
Amaç fonksiyonu: f(x) = a x b
Amaç : maximize f(x)
Karar değişkenleri: a,b

- Kısıtlamalar (constraints), optimize edicinin nereye gidemeyeceğini veya başarılı bir çözüm için karşılanması gereken ek koşulları tanımlar.
- Örneğin, bir karenin boyutunu optimize ederken, iki kenarın birlikte çarpılmasıyla 5'ten az uzunluk sınırlaması ekleyebiliriz. Bu bir eşitsizlik kısıtlamasıdır. Veya a+b = 10 olacak şekilde bir eşitlik kısıtlaması da ekleyebiliriz.



Optimizasyon Problemlerinin Sınıflandırması (Karar değişken türüne göre**)**

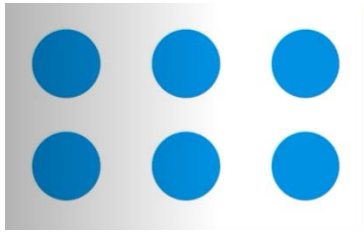
Karar değişkenlerinin türüne göre optimizasyonda dikkat edilmesi gereken önemli bir ayrım, sürekli ve ayrık, ikili veya tamsayı değişkenler arasındaki farktır.



Sürekli Ayrık İkili

Birçok değişken boyut, ağırlık, hız gibi sürekli değerler aralığı olarak gösterilebilir. Örneğin, bir araba 50 km/h, 100 km/h veya bunların arasında herhangi bir değere sahip olabilir.

Bununla birlikte, bir tahtada delinmesi gereken deliklerin sayısı gibi problemler ayrık değişkenler ile temsil edilebilir.



Bir anahtarın açık veya kapalı olması gibi bir problem 1 veya 0 ikili değerleriyle temsil edilir.

Bir soruna dahil olan değişken türleri, kullanılabilecek optimizasyon yöntemlerini etkiler. Genel olarak, ikili veya tamsayı değişkenleri içeren optimizasyon problemleri, sürekli olmayan türevler ürettikleri için sürekli problemlerden çok daha zordur.

- Karar değişken sayısına göre
  - Tek değişkenli optimizasyon problemleri
  - Çok değişkenli optimizasyon problemleri

- Stokastik ve determistik optimizasyon
  - Stokastik optimizasyon (stochastic optimization)algoritmaları, rastgele değişken üretir ve kullanır.
  - Rastgele değişkenler, amaç fonksiyonunda veya kısıtlarda olabilir.
  - Stokastik algoritmalarda aynı başlangıç şartları için farklı sonuçlar elde edilebilir.

  - Deterministik optimizasyon algoritmalarında, problem kesin bir şekilde tanımlanır ve rastgele değer bulunmaz.
  - Deterministik algoritmalarda aynı başlangıç şartları için her zaman aynı sonuç elde edilir.

Optimizasyon algoritmalarının özellikleri

- Optimizasyon algoritmaları **iteratif çalışır.**
- Değişkenlerin **tahmin edilen en iyi değerleri** ile başlanır ve **her iterasyonda yeni değerler üretilir.**
- Algoritma **istenen sonuç** elde edilince **sonlanır.**
- Bazı algoritmalar **amaç fonksiyonuna** göre, bazıları **kısıtlara göre sonlanabilir.**
- Bazı algoritmalar amaç fonksiyonu dışında **başka bir fonksiyonun değerine göre sonlanabilir.**
- İyi bir algoritmanın özellikleri:
  - Robustness:Başlangıç değişkenlerinin tüm değerleri ve çeşitli problemler için **iyi sonuç vermelidir.**
  - Efficiency:Çok fazla **süreye** ve **alana** ihtiyaç duymamalıdır.
  - Accuracy:**İstenen doğrulukta** sonucu elde etmelidir.

Python Örnek

- Yüzey alanı maksimum 36 birim olabilecek bir küpün maksimum hacim ve alabileceği değerleri bulunuz?

```
import numpy as np
from scipy.optimize import minimize

def hacimHesapla(x):
    a = x[0]
    b = x[1]
    c = x[2]
    hacim = a * b * c
    return hacim

def yuzeyAlaniHesapla(x):
    a = x[0]
    b = x[1]
    c = x[2]
    yuzeyAlani = 2 * (a*b + a*c + b*c)
    return yuzeyAlani

def amacFonksiyonu(x):
    return -hacimHesapla(x)

def kisitlar(x):
    return 36 - yuzeyAlaniHesapla(x)

cons = ({"type":"ineq","fun":kisitlar})     #problem kısıtları tanımlanıyor

kd_a = 1
kd_b = 1                                    #karar değişkenleri
kd_c = 1

xo = np.array([kd_a,kd_b,kd_c])

cozum = minimize(amacFonksiyonu,xo,constraints=cons,options={'disp':True})

xOpt = cozum.x
hacimOpt = -cozum.fun

yuzeyAlaniOpt = yuzeyAlaniHesapla(xOpt)

print("a :" + str(xOpt[0]))
print("b :" + str(xOpt[1]))
print("c :" + str(xOpt[2]))
print("Hacim :" + str(hacimOpt))
print("Yuzey Alanı :" + str(yuzeyAlaniOpt))
```

```
Optimization terminated successfully.    (Exit mode 0)
        Current function value: -14.696938456696023
        Iterations: 6
        Function evaluations: 26
        Gradient evaluations: 5
a :2.4494897249166683
b :2.4494897249166705
c :2.449489778515688
Hacim :14.696938456696023
Yuzey Alanı :35.999999999995026
```