

CENG314

Sayısal Optimizasyon

Ders Notları

Prof. Dr. Serdar İplikçi

Pamukkale Üniversitesi
Mühendislik Fakültesi
Bilgisayar Mühendisliği

İÇİNDEKİLER

1. GİRİŞ

1.1 Optimizasyon Kavramına Giriş

1.2 Genel Optimizasyon Problemi

1.2.1 Lineer Optimizasyon (Lineer Programlama) Problemleri

1.2.2 Lineer-olmayan Optimizasyon (Lineer-olmayan Programlama) Problemleri

1.3 Temel Optimizasyon Kavramları

1.3.1 Olurluk (Feasibility)

1.3.2 Optimallik (Optimality)

1.3.3 Konvekslik (Convexity)

2. BİR-BOYUTLU KISITSIZ OPTİMİZASYON

2.1 Bir-boyutlu Lineer-olmayan Nümerik Optimizasyon

2.1.1 Problemin Tanımı

2.2 Dolaylı Yöntemler

2.2.1 Newton-Raphson Yöntemi

2.2.2 İkiye Bölme Yöntemi

2.3 Doğrudan Yöntemler

2.3.1 Altın-Bölme Yöntemi

2.4 Bir-boyutlu Nümerik Optimizasyonun Önemi

3. ÇOK-BOYUTLU KISITSIZ OPTİMİZASYON

3.1 Problemin Tanımı

3.2 Genel Güncelleme Kuralı

3.3 Matematiksel Temeller

3.3.1 Gradyant, Hessian ve Jacobian Matrisleri

3.3.2 Taylor Teoremi ve Taylor Açılımı

3.3.3 İnış Yönü

3.4. Optimallik için Analitik Koşullar

3.4.1 Birinci-dereceden Koşullar

3.4.2 İkinci-dereceden Koşullar

3.5. Gradyant Yöntemler

3.5.1. Birinci-dereceden Yöntemler

3.5.1.1. Dik-İnış (Steepest-Descent - SD) Yöntemi

3.5.1.2. Conjugate-Gradient (Fletcher-Reeves) Yöntemi (CG)

3.5.2 İkinci-dereceden Yöntemler – Newton Yöntemi

3.5.2.1 Newton Yöntemi

3.5.2.2 Değiştirilmiş Newton Yöntemi

3.5.3 Newton-benzeri (Quasi-Newton) Yöntemler

3.5.3.1 Davidon-Fletcher-Powell (DFP) Yöntemi

3.5.3.2 Broydon-Fletcher-Goldfarb-Shanno (BFGS) Yöntemi

3.5.4 İkinci-dereceden Yaklaşık Yöntemler

3.5.4.1 Gauss-Newton (GN) Yöntemi

3.5.4.2 Levenberg-Marquardt (LM) Yöntemi

4. MODELLEME ve TAHMİN

4.1 Modelleme Kavramı

4.2 Veri Tipleri

3.2.1 Giriş-Çıkış Verisi

3.2.2 Zaman Serisi

4.3 Modelleme ve Model Seçme

4.4 Temel Modeller

4.4.1 Bir-Boyutlu Veri Modelleme

4.4.1.1. Lineer Modeller

4.4.1.1.1 Polinom Modeli

4.4.1.1.2 Radyal Tabanlı Fonksiyon (Radial Basis Function RBF) Modeli

4.4.1.2. Lineer-olmayan Modeller

4.4.1.2.1 Üstel Model

4.4.1.2.2 Hiperbolik Model

4.4.2. Çok-boyutlu Veri Modelleme

4.4.2.1. Lineer Modeller

4.4.2.1.1 Polinom Modeli

4.4.2.2. Lineer-olmayan Modeller

4.4.2.2.1 Yapay Sinir Ağları

5. YAPAY SİNİR AĞLARI

5.1. Basit Nöron Modeli

5.2 SISO-YSA Modeli

5.3 MISO-YSA Modeli

5.4 Çok-boyutlu Veri Modelleme

5.5 Uygulamalar

5.5.1 Regresyon

5.5.1.1 Süreç Modelleme ve Tahmini

5.5.1.2 Zaman Serisi Modelleme ve Tahmini

5.5.2 Sınıflandırma

5.5.2.1 Örüntü Tanıma

5.5.2.2 Karar Verme

BÖLÜM 1. GİRİŞ

1.1. Optimizasyon Kavramına Giriş

Matematiksel terimlerle ifade edilecek olursa optimizasyon, “serbest değişken(ler) üzerindeki belli kısıtları sağlayacak şekilde bir fonksiyonun aldığı değeri en küçük (veya en büyük) yapan değişken(ler)in bulunması” şeklinde tanımlanan problemin (optimizasyon problemi) çözülmesi olarak tanımlanabilir.

Gerçek dünya problemleri açısından bakıldığında, optimizasyon probleminde söz konusu olan bu fonksiyon çok farklı alanlarda karşımıza çıkabilir. Örneğin, bu fonksiyon bir yatırımdaki risk/kar oranı veya bir köprü inşaatındaki dayanıklılık/ağırlık oranı veya bir uçağın rotasının belirlenmesinde uçuş süresi/yakıt tüketimi oranı olabilir. Bu tip optimizasyon problemleri o kadar yaygındır ki hemen hemen tüm uygulama alanlarında karşılaşılabılır. Hatta, Fermat’ın ileri sürdüğü Işığın Kırılması Kanunu’nda olduğu gibi, doğanın kanunlarını açıklamakta bile kullanılabilir. Optimizasyon problemlerinin çözümünün bu kadar çok çekici olması nedeniyle, bu problemlerin çözümü için geliştirilen optimizasyon teknikleri üzerinde yüzyıllardan beri çalışılmaktadır.

Son yıllarda, yerel ve küresel iş dünyasındaki rekabetin artması, zaman, işgücü, hammadde ve benzeri kaynakların daha etkin kullanımı gerekliliğinin daha da önemli hale gelmesi, mühendislik tasarımlarındaki mükemmelliğin daha fazla istenir hale gelmesi ve benzeri itici gereklilikler optimizasyon tekniklerinin daha fazla kullanılmasını kaçınılmaz hale getirmiştir. Artık pek çok durumda, bu tekniklerin yardımı olmadan belli kararların alınması mümkün olmamaktadır. Büyük ve çok-uluslu bir şirkette örneğin çok küçük bir iyileştirme ile milyonlarca dolarlık bir kar elde edilebilmektedir. Benzer şekilde, bu teknikler sayesinde milyon mertebesinde transistöre sahip bir yonganın tasarımı mümkün hale gelebilmektedir.

Bilgisayar sistemlerindeki yazılım ve donanımsal gelişmelere paralel olarak, belli bir mertebeye kadar karmaşıklığındaki optimizasyon problemleri makul sürelerde çözülebilmektedir. Bu sayede optimizasyon teknikleri iş dünyası, bilim ve mühendislikte kullanılabilen pratik birer araç haline gelmiştir.

Bir optimizasyon probleminin çözülebilmesi için öncelikle o problemin matematiksel bir dille ifade edilebilmesi gerekir. Başka bir deyişle, optimize edilecek fonksiyon (amaç fonksiyonu), bu fonksiyonun değişkenleri (tasarım değişkenleri) ve bu değişkenler üzerindeki kısıtlar matematiksel terimlerle ifade edilebilmelidir. Matematiksel terimlerle ifade edilen bir optimizasyon problemindeki fonksiyon çok karmaşık olabilir, çok fazla sayıda değişken içerebilir ve hatta bu değişkenler üzerindeki kısıtlar da karmaşık bir biçimde olabilir. Optimizasyon probleminin karmaşıklığına göre farklı teknikler kullanılabilir.

Bir sonraki alt-bölümde optimizasyon problemlerinin en genel biçimi matematiksel bir dille verilecek ve ardından optimizasyon problemlerine ilişkin bazı örnekler verilecektir. Ayrıca, problemlerin özelliklerini ve aynı zamanda problemleri çözen teknikler arasındaki farklılıkları yansıtan kategorilere yer verilecektir.

1.2. Genel Optimizasyon Problemi

En genel halde bir optimizasyon problemi aşağıdaki gibidir:

$$\begin{aligned} \min_{\mathbf{x}} f(\mathbf{x}) & \quad \text{"optimize edilecek amaç fonksiyonu"} \\ g_i(\mathbf{x}) = 0, i \in \mathcal{E} & \quad \text{"tasarım değişkeni eşitlik kısıtları"} \\ g_i(\mathbf{x}) \geq 0, i \in \mathcal{I} & \quad \text{"tasarım değişkeni eşitsizlik kısıtları"} \end{aligned} \quad ()$$

burada $\mathbf{x} = [x_1 \ \dots \ x_n]^T \in \mathbb{R}^n$ n -boyutlu tasarım değişkenleri vektörü, $f(\mathbf{x}): \mathbb{R}^n \mapsto \mathbb{R}$ optimize edilecek amaç fonksiyonu, $g_i(\mathbf{x})$ i inci kısıt fonksiyonu, \mathcal{E} kümesi eşitlik kısıtları indeksleri kümesi ve \mathcal{I} kümesi de eşitsizlik kısıtları indeksleri kümesidir. Amaç fonksiyonu ve kısıt fonksiyonlarının durumuna göre optimizasyon problemi çeşitli isimler almaktadır.

1.2.1 Lineer Optimizasyon (Lineer Programlama) Problemleri

Bu tip problemlerde amaç fonksiyonu $f(\mathbf{x})$ ve kısıt fonksiyonları $g_i(\mathbf{x})$ 'ler tasarım değişkenlerine göre lineer birer fonksiyondurlar. Buna bir örnek olarak aşağıdaki problem verilebilir:

$$\begin{aligned} \min_{\mathbf{x}} f(\mathbf{x}) &= 2x_1 - 3x_2 - 7x_3. \\ 2x_1 + 2x_2 - 4x_3 &\geq 2 \\ x_1 - 3x_2 + 5x_3 &\leq -3 \end{aligned} \quad ()$$

Bir lineer problemde amaç fonksiyonu sabit olup kısıtlar lineer eşitlik kısıtları şeklinde olabilir. Buna bir örnek olarak bir eğri uydurma problemi verilebilir. Örneğin, (2,1), (3,6) ve (5,4) noktalarından geçen bir parabolün denklemini bulma problemini ele alalım. Bu noktalar (t_i, y_i) ile parabol de $\hat{y}(t) = x_1 + x_2 t + x_3 t^2$ ile gösterilirse aşağıdaki gibi üç bilinmeyenli üç denklem elde edilir:

$$\begin{aligned} x_1 + x_2(2) + x_3(2)^2 &= 1 \\ x_1 + x_2(3) + x_3(3)^2 &= 6. \\ x_1 + x_2(5) + x_3(5)^2 &= 4 \end{aligned} \quad ()$$

Optimizasyon problemi de aşağıdaki şekilde ifade edilir:

$$\begin{aligned} \min_{\mathbf{x}} f(\mathbf{x}) &= \text{sabit} \\ x_1 + 2x_2 + 4x_3 &= 1 \\ x_1 + 3x_2 + 9x_3 &= 6 \\ x_1 + 5x_2 + 25x_3 &= 4 \end{aligned} \quad ()$$

Bu problemde amaç fonksiyonu sabit bir fonksiyondur, kısıtlar ise aşağıdaki denklem sistemi ile ifade edilebilen lineer eşitlik kısıtlarıdır.

$$\begin{bmatrix} 1 & 2 & 4 \\ 1 & 3 & 9 \\ 1 & 5 & 25 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 6 \\ 4 \end{bmatrix}$$

Burada, çözüm $[x_1^* \ x_2^* \ x_3^*] = [-21 \ 15 \ -2]$ şeklinde olup parabolün denklemi $\hat{y}(t) = -21 + 15t - 2t^2$ şeklindedir. bu problemin özelliği, bilinmeyen sayısı kadar lineer bağımsız denklem olması ve bunun sonucunda tüm kısıtları sağlayan tek bir nokta olması ve bunun da problemin tek çözümü olmasıdır. Bu problem daha fazla sayıda nokta için genelleştirilebilir. Örneğin (t_i, b_i) şeklinde verilen N adet noktadan geçen ve $\hat{y}(t) = x_1 + x_2t + x_3t^2 + \dots + x_Nt^{N-1}$ biçimindeki eğrinin bulunması da benzer şekilde aşağıdaki denklem sistemi çözülerek bulunur:

$$\begin{bmatrix} 1 & t_1 & \dots & t_1^{N-1} \\ 1 & t_2 & \dots & t_2^{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & t_N & \dots & t_N^{N-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix}.$$

Bu denklem sisteminde bilinmeyen sayısı ile denklem sayısı birbirine eşit olduğundan eşitliğin sol tarafındaki matris kare matristir ve tersi alınarak çözüm bulunabilir. Lineer optimizasyon problemleri basit olmalarına rağmen ekonomi, ağlar, planlama gibi alanlarda sıkça karşımıza çıkmaktadır.

Şu örneği ele alalım: Varsayalım ki deri ürünleri imal eden bir firma belli bir dönemdeki kazancını maksimum yapmak istemektedir. Firma çok çeşitli siparişleri kabul etmekle birlikte bu dönemde sadece kemer, ayakkabı, çanta ve mont üretimi yapacaktır. Bu dört tip ürünün yapılması için gerekli deri malzeme miktarı, işgücü ve o üründen elde edilen kazanç aşağıdaki tabloda verilmiştir.

	Malzeme	İşgücü	Kazanç
kemer	10	2	10
ayakkabı	12	4	15
çanta	25	8	20
mont	20	12	40

Varsayalım ki firmanın 5000 birim deri malzeme ve 1500 birim işgücü mevcut olsun Hangi üründen en az kaç tane üretilmelidir ki toplam kazanç maksimum olsun? Buna ilişkin optimizasyon problemi şu şekilde ifade edilebilir: x_1, x_2, x_3 ve x_4 değişkenleri sırasıyla her bir ürün için üretim sayısı olmak üzere,

$$\begin{aligned}
\min_{\mathbf{x}} f(\mathbf{x}) &= 10x_1 + 15x_2 + 20x_3 + 40x_4 \\
10x_1 + 12x_2 + 25x_3 + 20x_4 &\leq 5000 \\
2x_1 + 12x_2 + 8x_3 + 12x_4 &\leq 1500 \\
0 &< x_1, x_2, x_3, x_4
\end{aligned}$$

0

Bu problem, deri malzemenin yanı sıra ürünlerde kullanılacak diğer malzemeler ve fazladan işçi çalıştırma veya fazla mesai gibi durumlar da hesaba katılarak daha da gerçekçi hale getirilebilir.

1.2.2 Lineer-olmayan Optimizasyon (Lineer-olmayan Programlama) Problemleri

Lineer-olmayan optimizasyon probleminde amaç fonksiyonu $f(\mathbf{x})$ ve/veya kısıt fonksiyonlarından herhangi biri ($g_i(\mathbf{x})$) tasarım değişkenlerine göre lineer-olmayan bir fonksiyondur.

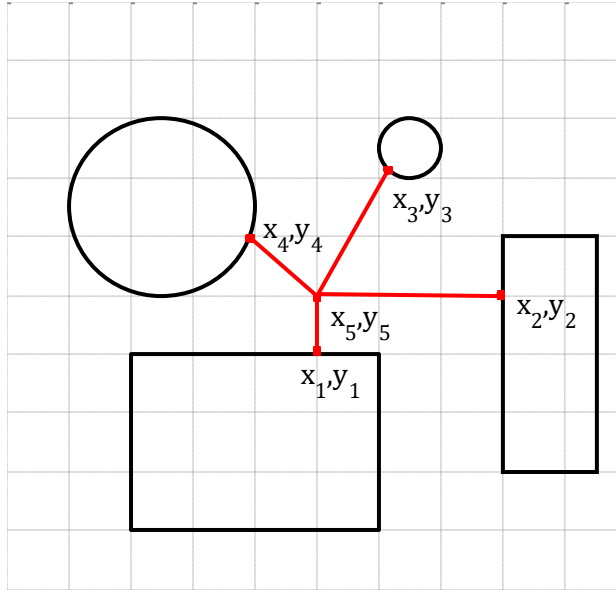
Basit bir örnekle başlamak gerekirse, (t_i, b_i) şeklinde verilen 10 adet noktaya en yakın uzaklıktan geçen daha $\hat{y}(t) = x_1 + x_2 t + x_3 t^2$ şeklinde bir parabolün bulunması problemini ele alalım. Bu problem, yukarıda verilen lineer programlama probleminden farklıdır çünkü artık denlem sayısı (10 denklem) bilinmeyen sayısından (3 bilinmeyen) fazladır Bu problem şu şekilde ifade edilebilir:

$$\begin{aligned}
\min_{\mathbf{x}} f(\mathbf{x}) &= \sum_{i=1}^{10} (b_i - \hat{y}(t_i))^2 \\
&= \sum_{i=1}^{10} (b_i - (x_1 + x_2 t_i + x_3 t_i^2))^2
\end{aligned}$$

0

Görüldüğü gibi bu lineer-olmayan problemde tasarım değişkenleri üzerinde herhangi bir kısıt olmamasına rağmen amaç fonksiyonu lineer değildir. Lineer-olmayan programlama problemleri çoğunlukla bilim ve mühendislik alanlarında karşımıza çıkar. Örneğin bir kürenin hacmi kürenin yarıçapının lineer-olmayan bir fonksiyonudur. Bir elektrik devresinde harcanan enerji, direncin lineer-olmayan bir fonksiyonudur. Bir hayvan topluluğunun nüfusu, doğum- ve ölüm-oranlarının lineer-olmayan bir fonksiyonudur. aşağıdaki gibi bir ağ problemini ele alalım:

Varsayalım ki farklı geometrik şekillere sahip dört adet binaya ortak bir dağıtım noktasından veri hattı kurulacaktır. Binaların pozisyonları aşağıdaki şekilde gösterilmiştir.



Dikdörtgen şekilli ilk binanın genişliği 4 birim, yüksekliği de 3 birim olup olup bina (2,1) noktasını sol-alt köşe kabul edecek şekilde yerleştirilmiştir. Dikdörtgen şekilli ikinci binanın genişliği 1.5 birim, yüksekliği de 4 birim olup olup bina (8,2) noktasını sol-alt köşe kabul edecek şekilde yerleştirilmiştir. Diğer iki bina ise dairesel bir şekle sahiptir ve ilkinin merkezi (6.5,7.5) noktasında olup yarıçapı 0.5 birimdir, diğer dairesel binanın yarıçapı 1.5 birim olup merkezi ise (2.5,6.5) koordinatlarındadır. Binalardan gelen kablolar (x_5, y_5) gibi tek bir ortak noktada birleştirilecek ve her bir i^{inci} binaya (x_i, y_i) noktasından bağlanacaktır. Amacımız toplam kablo uzunluğunu minimum yapmaktır. Kolaylık olması açısından birinci binanın bağlantı noktasının üst kenarda, ikinci binanın bağlantı noktasının da sol kenarda olacağını varsayıyoruz. Bu durumda bu problem için problem şu şekilde ifade edilebilir: $d_i = \sqrt{(x_5 - x_i)^2 + (y_5 - y_i)^2}$ ile i^{inci} binanın ortak bağlantı noktasına olan uzaklı olmak üzere,

$$\min_{\mathbf{x}, \mathbf{y}} f(\mathbf{x}, \mathbf{y}) = d_1 + d_2 + d_3 + d_4 = \sum_{i=1}^4 \sqrt{(x_5 - x_i)^2 + (y_5 - y_i)^2}$$

$$2 \leq x_1 \leq 6$$

$$y_1 = 4$$

$$x_2 = 8$$

$$2 \leq y_2 \leq 6$$

$$(x_3 - 6.5)^2 + (y_3 - 7.5)^2 = (0.5)^2$$

$$(x_4 - 2.5)^2 + (y_4 - 6.5)^2 = (1.5)^2$$

$$4 \leq x_5 \leq 8$$

$$4 \leq y_5 \leq 7$$

0

Gerçek dünya problemleri çoğunlukla bunun gibi lineer-olmayan optimizasyon problemi şeklinde karşımıza çıktığından bu kitapta daha çok bu tip problemler ele alınacaktır.

1.3 Temel Optimizasyon Kavramları

En genel halde bir optimizasyon problemi aşağıdaki gibidir:

$$\begin{aligned} \min_{\mathbf{x}} f(\mathbf{x}) & \quad \text{"optimize edilecek amaç fonksiyonu"} \\ g_i(\mathbf{x}) = 0, i \in \mathcal{E} & \quad \text{"tasarım değişkeni eşitlik kısıtları"} \\ g_i(\mathbf{x}) \geq 0, i \in \mathcal{I} & \quad \text{"tasarım değişkeni eşitsizlik kısıtları"} \end{aligned} \quad 0$$

burada $\mathbf{x} = [x_1 \ \dots \ x_n]^T \in \mathbb{R}^n$ n -boyutlu tasarım değişkenleri vektörü, $f(\mathbf{x}): \mathbb{R}^n \mapsto \mathbb{R}$ optimize edilecek amaç fonksiyonu, $g_i(\mathbf{x})$ i inci kısıt fonksiyonu, \mathcal{E} kümesi eşitlik kısıtları indeksleri kümesi ve \mathcal{I} kümesi de eşitsizlik kısıtları indeksleri kümesidir.

1.3.1 Olurluk (Feasibility)

Genel optimizasyon problemindeki $g_i(\mathbf{x}) = 0$ ve $g_i(\mathbf{x}) \geq 0$ şeklindeki kısıtları ele alalım. Öncelikle problemde verilen kısıtların uygun biçimde gösterilmesi gerekir. Örneğin, $x_1^2 + 5x_2 = 6 - 4x_3$ şeklinde verilen bir kısıt $g(\mathbf{x}) = x_1^2 + 5x_2 + 4x_3 - 6 = 0$ şeklinde gösterilebilirken $e^{x_2} \leq \sin x_1$ şeklinde verilen bir kısıt da $g(\mathbf{x}) = \sin x_1 - e^{x_2} \geq 0$ şeklinde gösterilebilir.

Bir optimizasyon problemindeki bütün kısıtları sağlayan noktaya optimizasyon probleminin *olurlu (feasible) noktası* denir. Olurlu noktalardan oluşan küme veya bölgeye *olurlu bölge* denir ve \mathcal{S} ile gösterilir.

Eğer $\hat{\mathbf{x}}$ gibi bir olurlu nokta, $g_i(\mathbf{x}) \geq 0$ kısıtını $g_i(\hat{\mathbf{x}}) = 0$ gibi eşitlik şeklinde sağlıyorsa o zaman bu olurlu noktada bu kısıt için *aktif*, sadece $g_i(\hat{\mathbf{x}}) > 0$ şeklinde sağlıyorsa da bu kısıt için *aktif değil* denir. İlk durum sağlanıyorsa yani kısıt aktif ise $\hat{\mathbf{x}}$ noktası *sınırdadır* denir, aksi halde *içeridedir* denir.

Tüm eşitlik kısıtları herhangi bir olurlu noktada aktiftir. Olurlu bir noktadaki aktif küme, bu noktada aktif olan tüm kısıtlardan oluşan küme olarak tanımlanır. En az bir eşitsizlik kısıtsının aktif olmadığı olurlu noktaların oluşturduğu bölgeye *sınır bölgesi*, kalanlara da *içerideki noktalar* denir.

Örnek: Bir optimizasyon probleminde aşağıdaki kısıtları ele alalım. $\mathbf{x}_a = [0 \ 0 \ 1]^T$, $\mathbf{x}_b = [1 \ 0 \ 4]^T$ ve $\mathbf{x}_c = [2 \ 1 \ 1]^T$ noktalarının olurlu olup olmadığını ve kısıtların bu noktalardaki durumunu belirleyiniz.

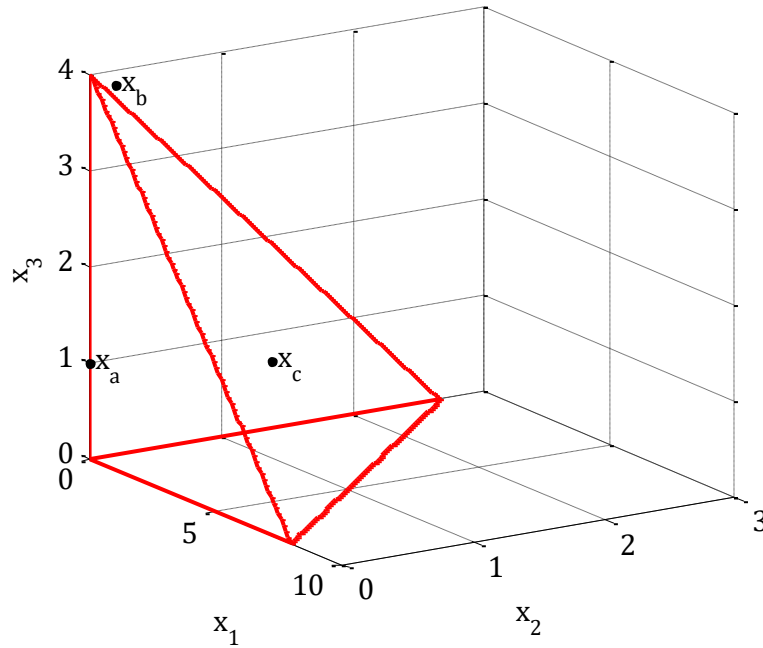
$$g_1(\mathbf{x}) = x_1 + 3x_2 + 2x_3 - 8 \leq 0$$

$$g_2(\mathbf{x}) = x_1 \geq 0$$

$$g_3(\mathbf{x}) = x_2 \geq 0$$

$$g_4(\mathbf{x}) = x_3 \geq 0$$

Kısıtların görünümü aşağıdaki şekilde gibidir.



$\mathbf{x}_a = [0 \ 0 \ 1]^T$ olurlu noktadır ve $g_2(\mathbf{x})$ ve $g_3(\mathbf{x})$ aktif, $g_1(\mathbf{x})$ ve $g_4(\mathbf{x})$ aktif değildir

$\mathbf{x}_b = [1 \ 0 \ 4]^T$ olurlu nokta değildir.

$\mathbf{x}_c = [2 \ 1 \ 1]^T$ olurlu noktadır ve hiç bir kısıt aktif olmadığından içeride bir noktadır.

1.3.2 Optimallik (Optimality)

Aşağıdaki problemi ele alalım:

$$\min_{\mathbf{x} \in \mathcal{S}} f(\mathbf{x}). \quad 0$$

Bir amaç fonksiyonunun minimizasyonu ile maksimizasyonu arasında bir fark yoktur. yukarıdaki problem bir maksimizasyon problemi olarak şu şekilde ifade edilebilir;

$$\max_{\mathbf{x} \in \mathcal{S}} -f(\mathbf{x}). \quad 0$$

Bu yüzden sadece minimizasyon problemini ele almak yeterli olacaktır. \mathcal{S} olurlu bölgesinin çoğunlukla kısıtlardan oluşan bir küme olduğu ve kısıtsız problemler için $\mathcal{S} \subset \mathbb{R}^n$ olduğu düşünülecektir.

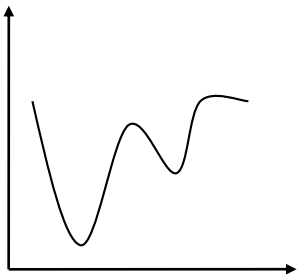
Bir optimizasyon probleminin çözümüne ilişkin en temel tanım şu şekilde yapılabilir:

$$\forall \mathbf{x} \in \mathcal{S} \text{ için } f(\mathbf{x}^*) \leq f(\mathbf{x}) \quad ()$$

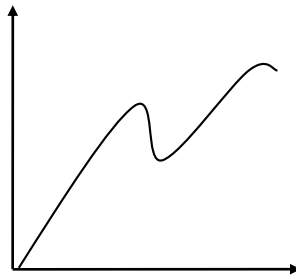
şartı sağlanıyorsa o zaman \mathbf{x}^* noktası $f(\mathbf{x})$ fonksiyonunun global minimumudur. Eğer \mathbf{x}^* noktası aşağıdaki şartı sağlıyorsa o zaman $f(\mathbf{x})$ fonksiyonunun kesin-global minimumudur:

$$\forall \mathbf{x} \in \mathcal{S} \text{ ve } \mathbf{x} \neq \mathbf{x}^* \text{ için } f(\mathbf{x}^*) < f(\mathbf{x}). \quad ()$$

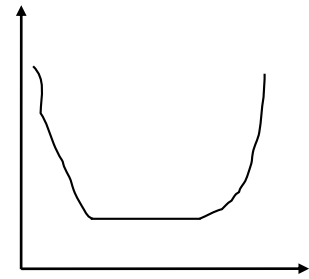
Tüm fonksiyonların sonlu global minimumları olmayabilir ve bir fonksiyonun global minimumu olsa bile bu onun kesin global minimum olduğu anlamına gelmez. Bu durum aşağıdaki şekillerde görülmektedir.



Kesin global minimum



Global minimum yok



Kesin-olmayan global minimum

Bu kitapta kullanılacak yöntemlerin Taylor serisi açılımlarını kullanması ve bu serilerin ancak belli bir nokta civarında bilgi vermesi sebebiyle global minimumu bulmak her zaman mümkün olmayabilir. dolayısıyla, global minimumlar yerine daha yerel olanlar bulunmaya çalışılacaktır.

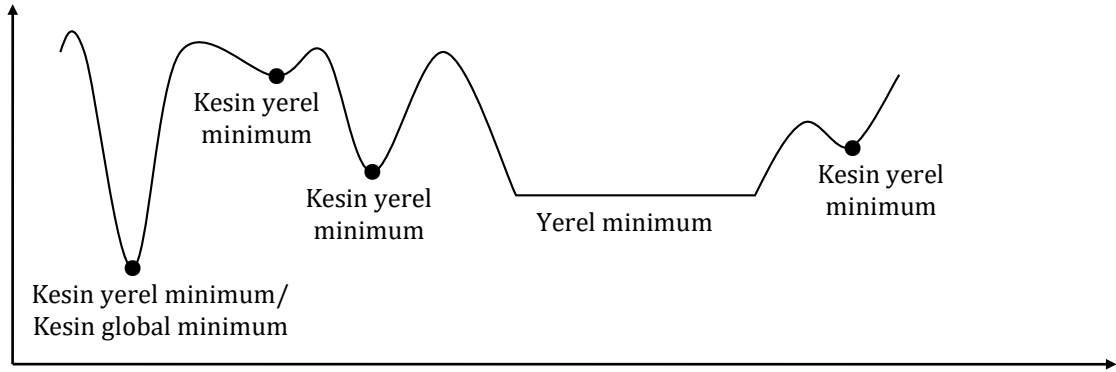
Eğer \mathbf{x}^* noktası aşağıdaki şartı sağlıyorsa o zaman $f(\mathbf{x})$ fonksiyonunun yerel minimumudur:

$$\forall \mathbf{x} \in \mathcal{S} \text{ öyle ki } \|\mathbf{x} - \mathbf{x}^*\| < \epsilon \text{ için } f(\mathbf{x}^*) \leq f(\mathbf{x}). \quad ()$$

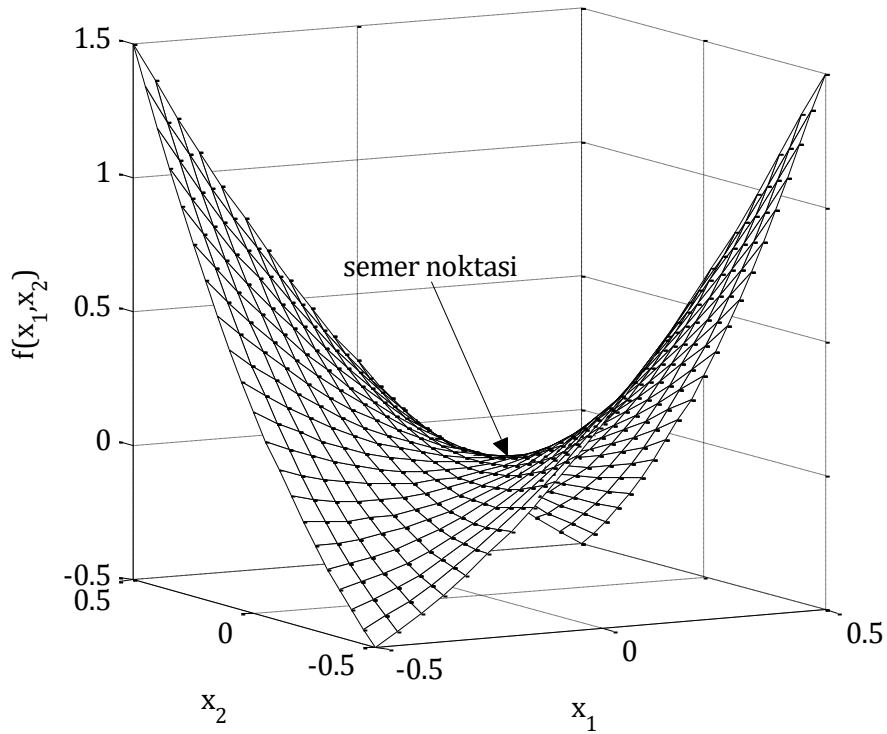
Burada ϵ değeri \mathbf{x}^* noktasına bağlı olmayan küçük bir değerdir, $\|\cdot\|$ notasyonu da vektör normudur. Eğer \mathbf{x}^* noktası aşağıdaki şartı sağlıyorsa o zaman $f(\mathbf{x})$ fonksiyonunun kesin-yerel minimumudur:

$$\forall \mathbf{x} \in \mathcal{S} \text{ öyle ki } \|\mathbf{x} - \mathbf{x}^*\| < \epsilon \text{ için } f(\mathbf{x}^*) < f(\mathbf{x}). \quad ()$$

Bu tanımlar aşağıdaki şekilde daha açık bir şekilde görülebilmektedir.



Çok-değişkenli fonksiyonlarda gözlenebilen bir başka durum da semer noktasıdır (saddle-point). Semer noktası, bir doğrultuda yerel minimum noktası iken başka bir doğrultuda yerel maksimum noktası şeklindedir. Tipik bir semer noktası $f(\mathbf{x}) = x_1^2 + x_2^2 - 4x_1x_2$ fonksiyonu için aşağıdaki şekilde görülmektedir.

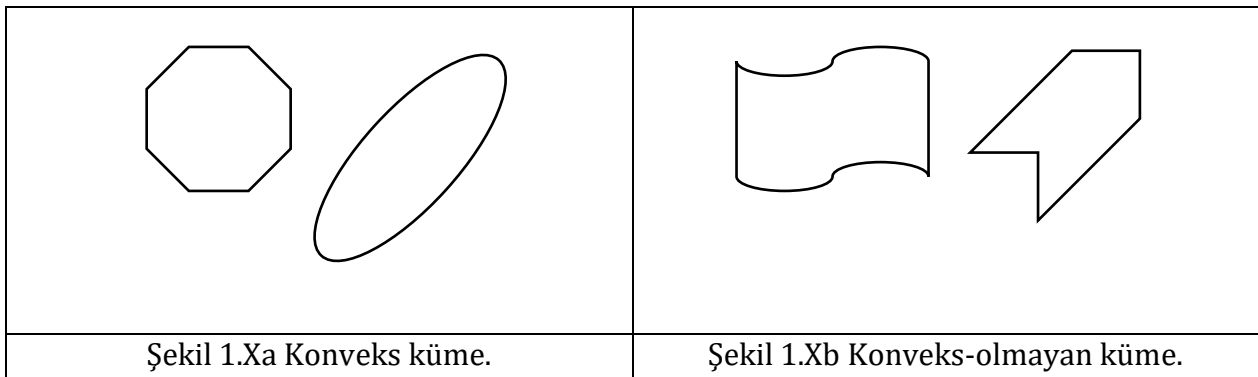


1.3.3 Konvekslik (Convexity)

Global çözümün bulunabildiği önemli bir durum vardır ki bu durumda amaç fonksiyonu konveks bir fonksiyondur ve \mathcal{S} de konveks bir kümedir. Önce konvekslik kavramına değinelim. Bir \mathcal{S} kümesi aşağıdaki şartı sağlıyorsa konvektir:

$$0 \leq \alpha \leq 1 \text{ olmak üzere } \forall \mathbf{x}, \mathbf{y} \in \mathcal{S} \text{ için } \alpha \mathbf{x} + (1 - \alpha) \mathbf{y} \in \mathcal{S}. \quad ()$$

Başka bir deyişle, \mathcal{S} bir konvek küme olmak üzere, eğer \mathbf{x} ve \mathbf{y} noktaları \mathcal{S} 'nin içinde iseler, \mathbf{x} ve \mathbf{y} noktalarını birleştiren her doğru parçası da \mathcal{S} 'nin içindedir. İki-boyutta konveks olan ve olmayan kümelere geometrik örnekler aşağıdaki şekillerde görülmektedir.

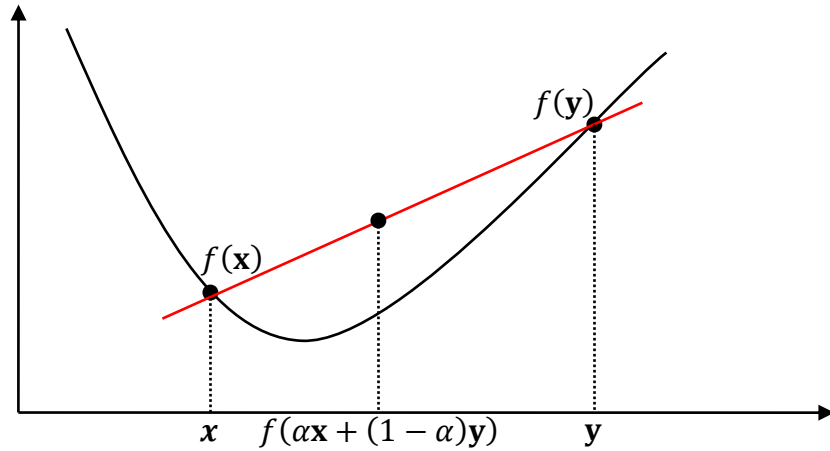


Optimizasyon problemleri açısından daha genel bir konveks küme tanımı yapılırsa, lineer kısıtlardan oluşan her kümenin konveks bir küme olduğu söylenebilir.

Benzer şekilde konvek fonksiyon tanımı da şu şekilde yapılabilir: Eğer bir $f(\mathbf{x})$ fonksiyonu aşağıdaki şartı sağlıyorsa, \mathcal{S} konveks kümesinde konveks bir fonksiyondur:

$$0 \leq \alpha \leq 1 \text{ olmak üzere } \forall \mathbf{x}, \mathbf{y} \in \mathcal{S} \text{ için } f(\alpha \mathbf{x} + (1 - \alpha) \mathbf{y}) \leq \alpha f(\mathbf{x}) + (1 - \alpha) f(\mathbf{y}). \quad ()$$

Bu tanıma göre, $(\mathbf{x}, f(\mathbf{x}))$ ve $(\mathbf{y}, f(\mathbf{y}))$ noktalarını birleştiren doğru parçası fonksiyonun grafiğinin üstünde kalıyorsa $f(\mathbf{x})$ konvektir.



Şekil 1.X Konveks fonksiyon.

Benzer şekilde, $f(x)$ fonksiyonu aşağıdaki şartı sağlıyorsa, \mathcal{S} kümesinde konkavdır:

$$0 \leq \alpha \leq 1 \text{ olmak üzere } \forall \mathbf{x}, \mathbf{y} \in \mathcal{S} \text{ için } f(\alpha \mathbf{x} + (1 - \alpha) \mathbf{y}) \geq \alpha f(\mathbf{x}) + (1 - \alpha) f(\mathbf{y}). \quad ()$$

Lineer fonksiyonlar hem konveks hem konkavdırlar. Eğer bir fonksiyon aşağıdaki şartı sağlıyorsa kesin-konveks'tir:

$$0 \leq \alpha \leq 1 \text{ olmak üzere } \forall \mathbf{x}, \mathbf{y} \in \mathcal{S} \text{ için } \mathbf{x} \neq \mathbf{y} \text{ olmak üzere,} \\ f(\alpha \mathbf{x} + (1 - \alpha) \mathbf{y}) < \alpha f(\mathbf{x}) + (1 - \alpha) f(\mathbf{y}). \quad ()$$

Şimdi yerel ve global çözümlere geri dönelim. Konveks programlama problemi şu şekilde tanımlanabilir:

$$\min_{\mathbf{x} \in \mathcal{S}} f(\mathbf{x}), \quad ()$$

burada \mathcal{S} konveks bir küme ve $f(\mathbf{x})$ konveks bir fonksiyondur. Konveks programlama problemi için başka bir tanımlama da şu şekilde yapılabilir:

$$\min_{\mathbf{x} \in \mathcal{S}} f(\mathbf{x}) \\ g_i(\mathbf{x}) \geq 0, i = 1, 2, \dots, l \quad ()$$

burada $f(\mathbf{x})$ konveks ve $g_i(\mathbf{x})$ 'ler konkavdır.

Teorem: Konveks Programlama Probleminin Genel Çözümü

\mathbf{x}^* bir konveks programlama probleminin yerel çözümü olsun. Bu durumda \mathbf{x}^* aynı zamanda global çözümdür. Eğer amaç fonksiyonu kesin-konveks ise \mathbf{x}^* kesin-global çözümdür.

Genellikle, optimizasyon problemlerinde fonksiyonun ve \mathcal{S} 'nin konveks olup olmadığını anlamak güçtür. Yine de bir fonksiyonun konveksliğini belirlemek için fonksiyonun türevlerinden yararlanılabilir. Örneğin, eğer bir-boyutlu bir fonksiyon olan f 'nin ikinci dereceden türevleri mevcutsa, konvekslik için alternatif bir tanım verilebilir. Böyle bir fonksiyon ancak ve ancak,

$$\forall x \in \mathcal{S} \text{ için } f''(x) \geq 0 \quad ()$$

ise konvekstir. Örneğin $f(x) = x^4$ fonksiyonu tüm reel eksen boyunca konvekstir, çünkü $\forall x \in \mathcal{S}$ için $f''(x) = 12x^2 \geq 0$. Diğer taraftan $f(x) = \sin x$ fonksiyonu ise ne konveks ne de konkavdır.

Çok-değişkenli durumda ise ikinci türevlerden oluşan Hessian matrisinin pozitif-yarı-tanımlı olması gerekir, yani $f(\mathbf{x})$ fonksiyonunun konveks olması için

$$\forall \mathbf{x} \text{ için } \mathbf{x}^T \nabla^2 f(\mathbf{x}) \mathbf{x} \geq 0 \quad ()$$

şartı sağlanmalıdır. Örnek olarak, $f(x_1, x_2) = 4x_1^2 + 12x_1x_2 + 9x_2^2$ karesel fonksiyonu konvekstir, çünkü

$$\begin{aligned} \mathbf{x}^T \nabla^2 f(\mathbf{x}) \mathbf{x} &= \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 8 & 12 \\ 12 & 18 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\ &= 8x_1^2 + 24x_1x_2 + 18x_2^2 \\ &= 2(2x_1 + 3x_2)^2 \geq 0 \end{aligned} \quad ()$$

Eğer

$$\forall \mathbf{x} \text{ için } \mathbf{x}^T \nabla^2 f(\mathbf{x}) \mathbf{x} > 0 \quad ()$$

ise $f(\mathbf{x})$ kesin-konvekstir, yani eğer $\forall \mathbf{x}$ için Hessian matrisi pozitif-tanımlı ise $f(\mathbf{x})$ kesin-konvekstir.

BÖLÜM 2. BİR-BOYUTLU KISITSIZ OPTİMİZASYON

2.1. Bir-boyutlu Lineer-olmayan Nümerik Optimizasyon

Bir-boyutlu lineer-olmayan nümerik optimizasyon aşağıda standart biçimi verilen problemdeki gibi tek bir değişkenden oluşan bir fonksiyonun en aza indirgenmesini amaçlar.

2.1.1. Problemin Tanımı

Bir-boyutlu lineer-olmayan nümerik optimizasyon probleminin standart biçimi şu şekildedir:

$$\begin{aligned} \min_x f(x) \quad & \text{"En aza indirilecek amaç fonksiyonu"} \\ x^l \leq x \leq x^u \quad & \text{"tasarım değişkeni aralığı"} \end{aligned} \quad 0$$

burada x tasarım değişkenidir. x tasarım değişkeni belirtilen aralık içerisinde öyle seçilmelidir ki $f(x)$ fonksiyonu en küçük değerini alsın. Bu problemi çözmek için eldeki araçlara bağlı olarak çeşitli yaklaşımlar vardır.

İlk yaklaşım analitik yaklaşımdır. Bu göre $f(x)$ fonksiyonunun bir yerel minimumu, $f(x)$ fonksiyonunun türevinin alınıp sıfıra eşitleyerek elde edilen denklemi çözerek $f(x)$ fonksiyonunun ekstremler noktalarını yani minimum ve maksimum noktalarını bulup, ardından da bu noktaları $f(x)$ fonksiyonunun ikinci türevinde yerine koyarak bulunan noktanın minimum ya da maksimum noktası olduğunu belirlenebilir. Bu çözüm matematiksel olarak şu şekilde ifade edilebilir:

$f'(x^*) = 0$ denklemini sağlayan herhangi bir x^* ekstremler noktası $f''(x^*) > 0$ şartını da sağlıyorsa, o zaman x^* noktası $f(x)$ fonksiyonunun bir yerel minimumudur.

Bu yaklaşımdaki ilk sorun bulunan yerel minimumun belirlenen aralıkta olmayabileceğidir. Bunu bir örnekle görebiliriz:

Örnek: Aşağıdaki kısıtsız optimizasyon problemini analitik yaklaşımla çözümlü.

$$\begin{aligned} \min_x f(x) &= x^3 + 4x^2 - 7x + 6 \\ -5 &\leq x \leq 5 \end{aligned}$$

Bu fonksiyonun birinci türevi $f'(x) = 3x^2 + 8x - 7$ ve ikinci türevi de $f''(x) = 6x + 8$ şeklindedir. Şimdi $f(x)$ fonksiyonunun ekstremler noktalarını bulalım:

$$f'(x) = 3x^2 + 8x - 7 = 0 \mapsto x_1^* = -3.3609, x_2^* = 0.6943$$

Şimdi bu noktaların tiplerini belirleyelim. $f''(x_1^*) < 0$ ve $f''(x_2^*) > 0$ olduğundan x_1^* yerel maksimum, x_2^* ise yerel minimumdur ancak x_2^* belirtilen aralık içerisinde olmadığından problemin çözümü bu yolla bulunamamaktadır.

Analitik yaklaşımdaki diğer bir sorun ise, $f(x)$ fonksiyonunun ekstremler noktalarını bulurken karşılaşılan bir sorundur: $f'(x^*) = 0$ denklemi her zaman analitik yolla çözülemeyebilir. Örneğin $f(x) = xe^{-x} + \cos x$ gibi bir fonksiyonun ekstremler noktalarını bulurken $f'(x) = e^{-x} - xe^{-x} - \sin x = 0$ denkleminin analitik olarak çözülmesi gerekir ki bu mümkün değildir.

Görüldüğü gibi analitik yaklaşımın işe yaramadığı problemlerle karşılaşmak mümkündür. Özellikle pek çok gerçek dünya problemi analitik yöntemle çözülememektedir. Bu nedenle, analitik yöntemlere alternatif olarak nümerik yöntemler geliştirilmiştir ki bu kitabın temel konusunu teşkil etmektedirler. Analitik olarak çözülemeyen bir optimizasyon problemi için literatürde çok çeşitli nümerik yöntem önerilmiştir. Bu yöntemlerden bazıları minimumu doğrudan bulurken bazıları da minimumu dolaylı olarak yani fonksiyonun türevinin sıfıra eşit olduğu noktaları bulmaya çalışır. Diğer taraftan, bazı yöntemler $f(x)$ fonksiyonunun türevini kullanırken bazıları ise bu türev bilgisine ihtiyaç duymaz. Bu yöntemlerden en yaygın kullanılanları sonraki alt-kısımlarda anlatılmıştır.

2.2. Dolaylı Yöntemler

2.2.1. Newton-Raphson Yöntemi

Bir fonksiyonun minimum noktasını bulmak için kullanılan Newton-Raphson yöntemi dolaylı bir yöntemdir. Başka bir deyişle, bu yöntem, minimum noktasını doğrudan bulmak yerine, fonksiyonun türevinin sıfıra eşit olduğu noktayı bulur ki bu noktada fonksiyon minimum ya da maksimum değerine ulaşır. Ayrıca, Newton-Raphson yöntemi minimumu bulunacak olan amaç fonksiyonu $f(x)$ 'in birinci ve ikinci türevlerine ihtiyaç duymakta olup aşağıdaki özelliklere sahiptir:

- Geometrik bir temele sahiptir
- Lineer olarak açılmış Taylor serilerini kullanır
- İteratiftir
- Karesel olarak yakınsar

Newton-Raphson yöntemi, amaç fonksiyonu $f(x)$ 'in minimum noktasının $\frac{df(x)}{dx} = f'(x) = 0$ şartını sağlaması gerektiğinden hareketle, bu şartı sağlayan noktaları bulmaya çalışır. Dolayısıyla, optimizasyon problemi, $f'(x) = 0$ şeklinde bir kök bulma problemine dönüşmüş olur. Newton-Raphson yöntemi, $f'(x) = 0$ şeklindeki problemi çözmek için birinci-dereceden Taylor açılımını kullanır. Bu açılıma göre, herhangi bir x_k değeri için

$$f'(x_k + \Delta x_k) \cong f'(x_k) + f''(x_k)\Delta x_k$$

0

Yazılabilir. Buradan, $f'(x_k + \Delta x_k) = 0$ yapıp Δx_k çekilirse,

$$\Delta x_k = -\frac{f'(x_k)}{f''(x_k)}$$

0

elde edilir. Newton-Raphson yöntemi çözüme iteratif yolla yaklaşır, yani her adımda aşağıdaki gibi bir algoritmayı kullanarak çözüme yaklaştırmaya çalışır:

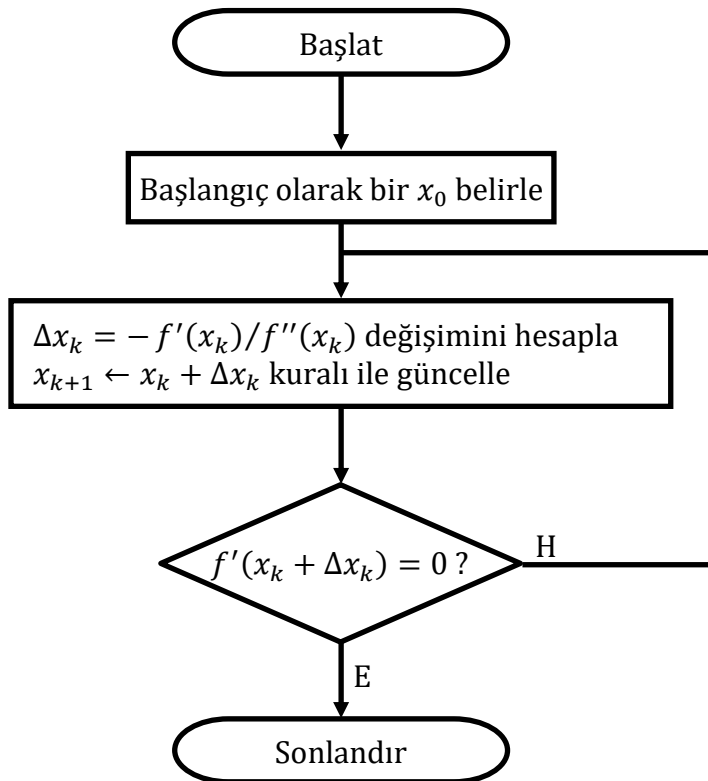
Newton-Raphson Algoritması

Adım 1 Başlangıç olarak bir x_0 belirle.

Adım 2 $\Delta x_k = -f'(x_k)/f''(x_k)$ değişimini hesapla.

Adım 3 $x_{k+1} \leftarrow x_k + \Delta x_k$ kuralı ile güncelle.

Adım 4 Eğer $f'(x_k + \Delta x_k) = 0$ şartı sağlanıyorsa algoritmayı sonlandır, sağlanmıyorsa Adım 2'ye git



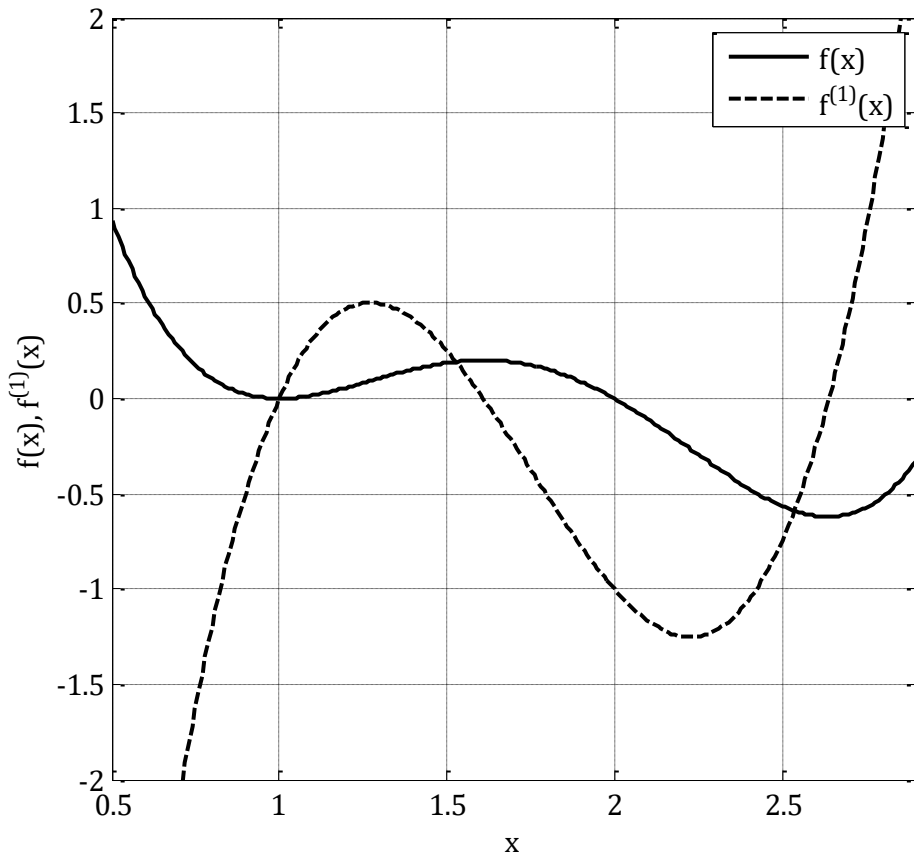
Buradaki güncellenmenin etkin olabilmesi için $f''(x_k)$ 'in sıfır olmaması gerekir. Ama $f'(x)$ fonksiyonunun düz olduğu yerlerde Δx_k değişiminin büyük, dik olduğu yerlerde de küçük olması kaçınılmazdır. Newton-Raphson yönteminin etkin olabilmesi için iterasyonların düz yerlerden yani eğimin çok küçük olduğu yerlerden kaçınması gerekir. Bu ise bu metodun en ciddi problemidir.

Örnek: Aşağıdaki kısıtsız optimizasyon problemini Newton-Raphson yöntemiyle çözünüz.

$$\min_x f(x) = (x - 1)^2(x - 2)(x - 3)$$

$$0 \leq x \leq 4$$

$f(x)$ fonksiyonu ve türevinin grafikleri aşağıdaki gibidir:



Şekil 2.1

$f(x)$ fonksiyonunun minimumunu bulmak için onun türevinin sıfıra eşit olduğu noktaları bulmak gerekir. Türevin sıfıra eşit olduğu noktalarda, $f(x)$ fonksiyonunun ya minimumu ya da maksimumu vardır. Newton-Raphson yöntemini kullanmak için önce $f(x)$ fonksiyonunun birinci ve ikinci türevlerini bulalım:

$$f'(x) = 2(x - 1)(x - 2)(x - 3) + (x - 1)^2(2x - 5)$$

$$f''(x) = 2(x-2)(x-3) + 2(x-1)(2x-5) + 2(x-1)(2x-5) + 2(x-1)^2.$$

Şimdi, $x_0 = 3$ noktasından başlayarak algoritma uygulanırsa aşağıdaki tablodaki gibi bir sonuç elde edilir:

k	x_k	$f(x_k)$	$f'(x_k)$	$f''(x_k)$	Δx_k
0	3.0000	0.0000	4.0000	16.0000	-0.2500
1	2.7500	-0.5742	0.8750	9.2500	-0.0946
2	2.6554	-0.6189	0.1040	7.0871	-0.0147
3	2.6407	-0.6197	0.0023	6.7708	-0.0003
4	2.6404	-0.6197	0.0000	6.7635	-0.0000
5	2.6404	-0.6197	0.0000	6.7635	-0.0000

İterasyona bu şekilde devam edilirse, beşinci iterasyonda $f'(x_6) = 10^{-8}$ civarında bir değer elde edilir ki bu da yakınsamanın gerçekleştiğinin bir göstergesidir. Çözümeye yakınsamanın ivmesi veya yakınsama hızı pek çok şekilde tanımlanabilir. Yakınsama hızını göstermenin bir yolu da $\frac{f'(x_k + \Delta x_k)}{f'(x_k)}$ oranını takip etmektir. Başka bir yol da sadece $f'(x)$ ifadesinin izlenmesidir. Bu örnek için $f'(x)$ değerleri izlenirse sırasıyla [4.0000 0.8750 0.1040 0.0023 0.0000] değerlerini aldığı görülecektir. Burada beşinci iterasyonda $f'(x)$ değeri neredeyse sıfırdır ve algoritma çözüme yakınsamıştır.

Newton-Raphson yönteminin bir özelliği de erişilen çözümün başlangıç noktasına bağlı olmasıdır. Algoritma $x_0 = 0.5$ noktasından yeniden başlatılırsa, aşağıdaki tabloda görüldüğü gibi 5 iterasyon sonucunda $x_6 = 1.00$ değerine ulaşılır ki bu değer de bir yerel minimumdur.

k	x_k	$f(x_k)$	$f'(x_k)$	$f''(x_k)$	Δx_k
0	0.5000	0.9375	-4.7500	16.0000	0.2969
1	0.7969	0.1094	-1.2174	8.1514	0.1493
2	0.9462	0.0063	-0.2418	5.0028	0.0483
3	0.9945	0.0001	-0.0221	4.0985	0.0054
4	0.9999	0.0000	-0.0003	4.0012	0.0001
5	1.0000	0.0000	-0.0000	4.0000	0.0000

Buradan çıkarılabilecek sonuç şudur: pek çok iteratif nümerik teknik başlangıç noktasına en yakın çözüme yakınsama eğilimindedir.

2.2.2. İkiye Bölme Yöntemi

İkiye bölme yöntemi de, Newton-Raphson yöntemi gibi dolaylı bir yöntemdir, yani amaç fonksiyonu $f(x)$ 'in türevinin sıfıra eşit olduğu noktayı bulmaya çalışır. Bu yöntemdeki nümerik teknik, bir fonksiyonun bir kökünün bir pozitif ve bir negatif değeri arasında kaldığı düşüncesine dayanmaktadır. Bulunan çözüm aslında, fonksiyonun sıfırının da içinde bulunduğu bir aralıktır. Nihai çözüm bu aralığın toleransı çok küçük tutularak bulunur. Bu bir kök bulma algoritması olduğundan, optimizasyon sırasında minimize edilecek fonksiyonun türevine ($f'(x)$) uygulanır. Böylece, amaç fonksiyonunun minimumunun bulunması, türevin sıfırının bulunmasına indirgenmiş olur. İkiye bölme yöntemini doğrudan amaç fonksiyonunun minimumunun bulunmasına uygulamak da mümkündür.

Metodu başlatmak için x_a ve x_b gibi iki başlangıç noktasına ihtiyaç vardır. $f'(x)$ 'in bu noktalardaki değerleri zıt işaretli olmalıdır. Bu durumda bu iki nokta arasında en az bir sıfırın bulunduğu varsayılır. Her bir iterasyon sırasında, sınırlarında x_a ve x_b noktalarının bulunduğu aralık ikiye bölünür öyle ki kalan kısmın uç noktaları yine zıt işaretli dir, yani kök hala kalan kısımdadır. Bu iteratif teknik aşağıdaki algoritma ile ifade edilebilir:

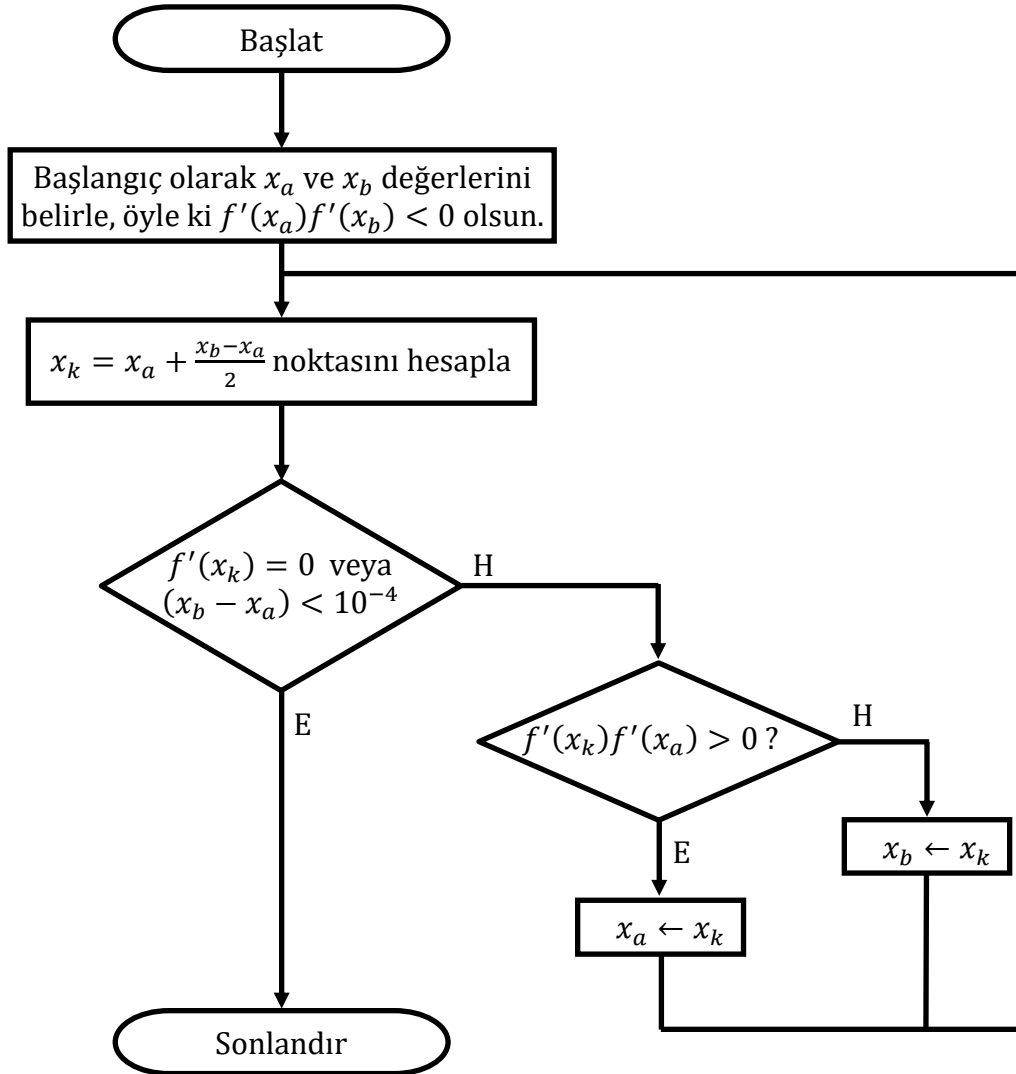
İkiye-Bölme Algoritması

Adım 1 Başlangıç olarak x_a ve x_b değerlerini belirle öyle ki $f'(x_a)f'(x_b) < 0$ olsun.

Adım 2 $x_k = x_a + \frac{x_b - x_a}{2}$ noktasını hesapla.

Adım 3 Eğer $f'(x_k) = 0$ veya $(x_b - x_a) < 10^{-4}$ şartı sağlanıyorsa algoritmayı sonlandır, sağlanmıyor ise ve eğer $f'(x_k)f'(x_a) > 0$ şartı sağlanıyor ise $x_a \leftarrow x_k$ yap, sağlanmıyor ise $x_b \leftarrow x_k$ yap.

Adım 4 Adım 2'ye git.

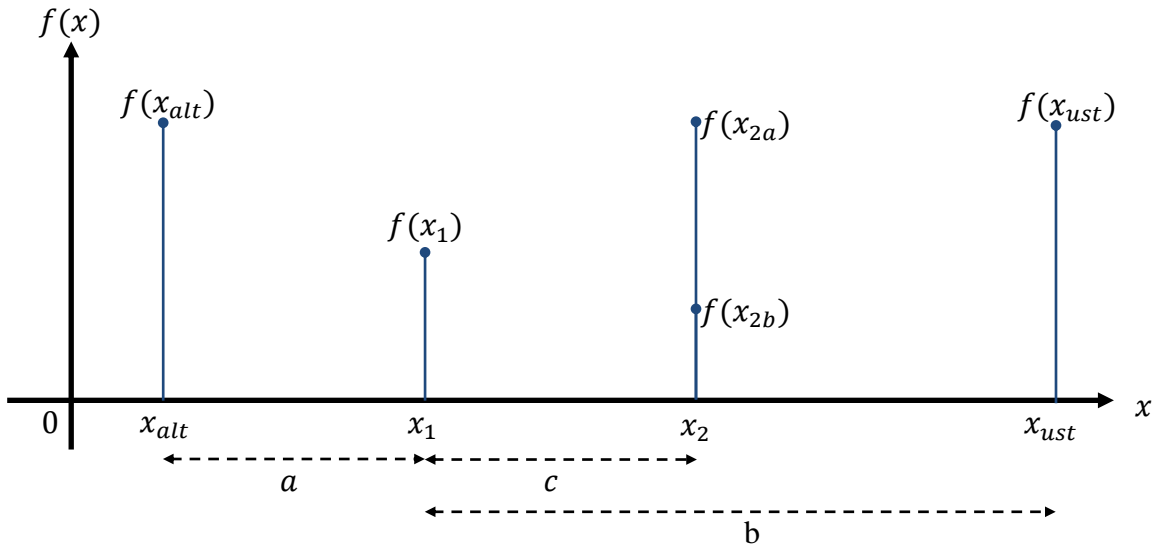


Aynı problem İkiye Bölme yöntemiyle çözüldüğünde çözüme 17 iterasyon sonucunda ulaşılmaktadır. Newton-Raphson yönteminin daha hızlı olmasının sebebi onun $f'(x)$ 'in türev bilgisini kullanıyor olmasıdır. Diğer taraftan, türev bilgisinin kullanılması algoritmaya ekstra bir yük getirmektedir. Bu iki yöntem gerçekte bir fonksiyonun sıfırını bulmaya çalıştıklarından minimizasyon problemini dolaylı olarak çözmektedir. Bundan sonra anlatılacak olan yöntem ise bu problemi doğrudan çözmektedir.

2.3. Doğrudan Yöntemler

2.3.1. Altın-Bölme Yöntemi

Altın-Bölme yöntemi, tek-modlu (unimodal) bir fonksiyonun belli bir aralıkta olduğu bilinen minimum veya maksimum noktasını bulmak için geliştirilmiş bir yöntemdir. Altın-Bölme yöntemi aralık daraltma yöntemleri içinde en cazip olanıdır. Bu yöntem aralığı uçlardan aynı oranda daraltır. Aralık uçları altın oran denilen 1.61803398 oranı kullanılarak daraltılmaktadır. Bu oran estetik ve matematikte çok önemli bir yere sahiptir. Bu yöntemin uygulaması kolaydır çünkü minimize edilecek fonksiyonun şekil ve süreklilik özelliklerinden bağımsız olarak çalışır. En önemlisi de, çözüme belli bir toleransla ulaşmak için gerekli iterasyon sayısı önceden tahmin edilebilir.



Algoritma şu şekilde işlemektedir: $f(x)$ fonksiyonunun $[x_{alt}, x_{ust}]$ aralığında olduğu bilinen minimum noktasını bulmak istiyoruz. Fonksiyonun x_{alt}, x_1 ve x_{ust} üçlü noktalarındaki değerlerinin önceden hesaplanmış olduğunu varsayalım. $[x_{alt}, x_{ust}]$ aralığında bir minimum noktası olduğundan, $f(x_1)$ değeri mutlaka $f(x_{alt})$ ve $f(x_{ust})$ değerlerinden küçük olacaktır. Aralığı daraltmak için x_2 gibi yeni bir nokta belirleyelim. x_2 noktasının $[x_{alt}, x_1]$ aralığından daha geniş olan $[x_1, x_{ust}]$ aralığında seçilmesi daha etkili olmaktadır. Eğer fonksiyonun x_2 noktasındaki değeri $f(x_{2a})$ gibiyse yani $f(x_1)$ değerinden daha büyükse o zaman yeni aralığın $[x_{alt}, x_2]$ olması gerekir. Aksi halde, fonksiyonun x_2 noktasındaki değeri $f(x_{2b})$ gibiyse yani $f(x_1)$ değerinden daha küçükse o zaman yeni aralığın $[x_1, x_{ust}]$ şeklinde olması gerekir. Her iki durumda da minimum noktasının bulunduğu aralık daraltılmış olur. x_2 gibi yeni noktanın seçimine gelince; yukarıdaki şekilden görüldüğü gibi yeni aralığın genişliği ya $a + c$ olacaktır ya da b olacaktır. Altın-Oran algoritması bu aralıkların eşit olmasını gerektirir. Aksi halde algoritmanın sonucu bulma hızı biraz şansa bırakılmış olur. $a + c = b$ olmasını garanti etmek için $x_2 = x_{alt} + x_{ust} - x_1$ şeklinde seçilmelidir. Bu durumda geriye sadece x_1 noktasının seçimi kalmaktadır. x_1 noktası öyle seçilmelidir ki, (x_{alt}, x_1, x_{ust}) üçlüsündeki oran, bu üçlünden sonra gelecek (x_{alt}, x_1, x_2) veya (x_1, x_2, x_{ust}) üçlülerinde de aynı olmalıdır ve aralık hep aynı oranda daralmalıdır. Bu nedenle, x_1 noktası x_{alt} ve x_{ust} noktalarına çok yakın

olmamalıdır. $f(x_2)$ hesaplanmadan önceki oran ile hesaplandıktan sonraki oran eşit olmalıdır. Eğer fonksiyonun x_2 noktasındaki değeri $f(x_{2a})$ gibiyse yani $f(x_1)$ değerinden daha büyükse o zaman yeni aralık $[x_{alt}, x_2]$ olur, yeni üçlü de (x_{alt}, x_1, x_2) olur ki bu durumda $\frac{c}{a} = \frac{a}{b}$ olmalıdır. Diğer taraftan, fonksiyonun x_2 noktasındaki değeri $f(x_{2b})$ gibiyse yani $f(x_1)$ değerinden daha küçükse o zaman yeni aralık $[x_1, x_{ust}]$ olur, yeni üçlü de (x_1, x_2, x_{ust}) olur ki bu durumda $\frac{c}{b-c} = \frac{a}{b}$ olmalıdır. Böylece $\frac{c}{a} = \frac{a}{b}$ ve $\frac{c}{b-c} = \frac{a}{b}$ şeklinde iki denklem elde edilir. Bu denklemlerden c 'yi yok edersek, $\left(\frac{b}{a}\right)^2 - \left(\frac{b}{a}\right) = 1$ denklemi elde edilir ki $\left(\frac{b}{a}\right)$ oranı x_1 noktasının yerini belirlemektedir. $\left(\frac{b}{a}\right) = \alpha$ dersek, bu denklemden $\alpha = \frac{1+\sqrt{5}}{2} = 1.618033988$ şeklindeki altın oran bulunur. Algoritma şu şekildedir.

Altın-Bölme Algoritması

Adım 1 Sınırların alt (x_{alt}) ve üst ($x_{üst}$) değerlerini belirle.

$(\Delta x)_{son}$ değerini belirle

$$\tau = 0.38197$$

$$\text{Tolerans} = \varepsilon = \frac{(\Delta x)_{son}}{x_{üst} - x_{alt}}$$

$$\text{İterasyon sayısı} = N = -2.078 \ln \varepsilon$$

$$k \leftarrow 0$$

Adım 2 Aşağıdaki değerleri hesapla:

$$x_1 \leftarrow x_{alt} + \tau(x_{üst} - x_{alt}), f_1 = f(x_1)$$

$$x_2 \leftarrow x_{üst} - \tau(x_{üst} - x_{alt}), f_2 = f(x_2)$$

Not: Bu iki nokta, uçlardan eşit uzaklıktadır

Adım 3 Eğer $k < N$ şartı sağlanıyorsa,

ve eğer $f_1 > f_2$ şartı sağlanıyorsa,

$$x_{alt} \leftarrow x_1, x_1 \leftarrow x_2, f_1 \leftarrow f_2$$

$$x_2 \leftarrow x_{üst} - \tau(x_{üst} - x_{alt}), f_2 = f(x_2)$$

$$k \leftarrow k + 1$$

Adım 3'e git,

ve eğer $f_1 < f_2$ şartı sağlanıyorsa,

$$x_{üst} \leftarrow x_2,$$

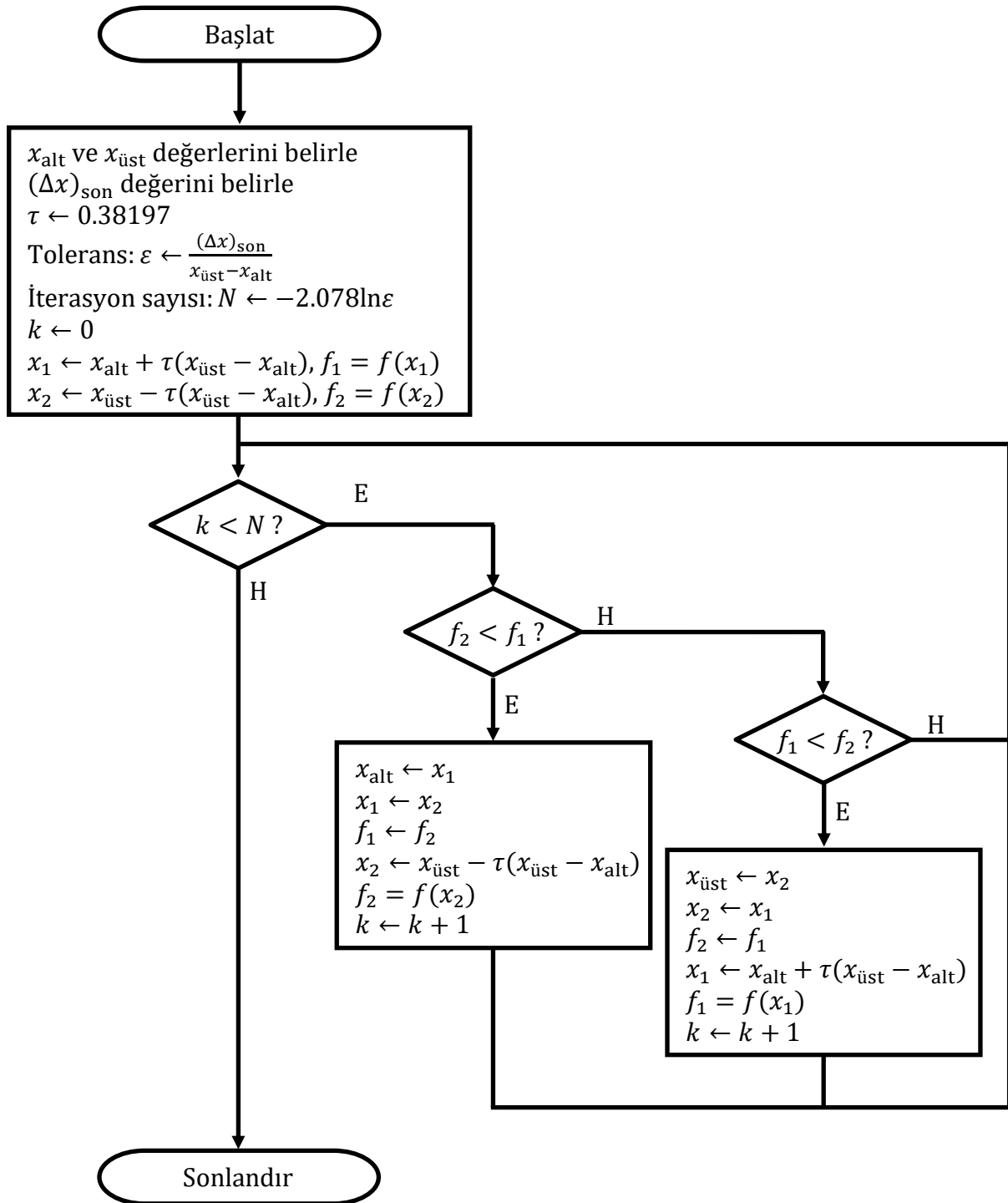
$$x_2 \leftarrow x_1, f_2 \leftarrow f_1$$

$$x_1 \leftarrow x_{alt} + \tau(x_{üst} - x_{alt}), f_1 = f(x_1)$$

$$k \leftarrow k + 1$$

Adım 3'e git,

Eğer $k < N$ şartı sağlanmıyorsa sonlandır.



Örnek: Aşağıdaki kısıtsız optimizasyon problemini Altın-Oran yöntemiyle çözünüz.

Not: $(\Delta x)_{\text{son}} = 0.0001$ alınız.

$$\min_x f(x) = (x - 1)^2(x - 2)(x - 3)$$

$$0 \leq x \leq 4$$

Bu problemde, $x_{\text{alt}} = 0$, $x_{\text{üst}} = 4$, $(\Delta x)_{\text{son}} = 0.0001$ şeklindedir. Buradan, iterasyon sayısı $N = -2.078 \ln \frac{(\Delta x)_{\text{son}}}{x_{\text{üst}} - x_{\text{alt}}} = 22$ olarak bulunur, yani en fazla 22 iterasyonda ve en fazla 0.0001 hata ile çözüme ulaşılabacaktır. Algoritma uygulandığında aşağıdaki tablodaki gibi sonuçlar elde edilir.

k	x_1	x_2	$f(x_1)$	$f(x_2)$
1	2.1115	2.4721	-0.1224	-0.5401
2	2.4721	2.6950	-0.5401	-0.6090
3	2.6099	2.6950	-0.6166	-0.6090
4	2.5573	2.6099	-0.5983	-0.6166
5	2.6099	2.6424	-0.6166	-0.6197
6	2.6300	2.6424	-0.6193	-0.6197
7	2.6377	2.6424	-0.6197	-0.6197
8	2.6406	2.6424	-0.6197	-0.6197
9	2.6395	2.6406	-0.6197	-0.6197
10	2.6402	2.6406	-0.6197	-0.6197
11	2.6399	2.6402	-0.6197	-0.6197
12	2.6402	2.6404	-0.6197	-0.6197
13	2.6403	2.6404	-0.6197	-0.6197
14	2.6404	2.6404	-0.6197	-0.6197
15	2.6404	2.6404	-0.6197	-0.6197
16	2.6404	2.6404	-0.6197	-0.6197
17	2.6404	2.6404	-0.6197	-0.6197
18	2.6404	2.6404	-0.6197	-0.6197
19	2.6404	2.6404	-0.6197	-0.6197
20	2.6404	2.6404	-0.6197	-0.6197
21	2.6404	2.6404	-0.6197	-0.6197
22	2.6404	2.6404	-0.6197	-0.6197

2.4. Bir-boyutlu Nümerik Optimizasyonun Önemi

Bir-boyutlu nümerik optimizasyon, çok-boyutlu nümerik optimizasyon probleminin çözümü esnasında adım-aralığının belirlenmesinde kullanılır. Çok-boyutlu nümerik optimizasyon probleminde genel güncelleme kuralı daha sonra da görüleceği gibi,

$$\mathbf{x}_{k+1} = \mathbf{x}_k + s\mathbf{p} \quad ()$$

şeklinde olup burada \mathbf{p} arama yönünü, s de adım-aralığını göstermektedir. Bu problemde güncelleme yapılırken önce uygun bir arama yönü belirlenir. Arama yönü belirlendikten sonra, uygun bir adım-aralığının seçimi artık bir-boyutlu bir nümerik optimizasyon problemine dönüşmüştür.

BÖLÜM 3. ÇOK-BOYUTLU KISITSIZ OPTİMİZASYON

3.1. Problemin Tanımı

Çok-boyutlu lineer-olmayan nümerik optimizasyon probleminin standart biçimi şu şekildedir:

$$\begin{aligned} \min_{x_1, x_2, \dots, x_n} f(x_1, x_2, \dots, x_n) \quad & \text{"En aza indirilecek amaç fonksiyonu"} \\ x_i^l \leq x_i \leq x_i^u \quad & \text{"tasarım değişkenleri kısıtı"} \end{aligned} \quad 0$$

burada x_1, x_2, \dots, x_n tasarım değişkenleridir. Aşağıdaki vektör notasyonu ile bu optimizasyon problemi daha sade bir şekilde yazılabilir.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad 0$$

Böylece,

$$\begin{aligned} \min_{\mathbf{x}} f(\mathbf{x}) \\ \mathbf{x}^l \leq \mathbf{x} \leq \mathbf{x}^u \end{aligned} \quad 0$$

şeklinde yazılabilir.

3.2. Genel Güncelleme Kuralı

Çok-boyutlu lineer-olmayan optimizasyon probleminini nümerik olarak çözerken tasarım değişkenlerinden oluşan \mathbf{x} vektörü her iterasyonda aşağıdaki genel güncelleme kuralı ile güncellenir:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + s\mathbf{p} \quad 0$$

burada \mathbf{p} arama yönü, s de adım-aralığıdır. Her adımda, uygun arama yönü bulunduktan sonra bir de uygun bir adım-aralığı bulunur. Adım-aralığının bulunması tipik bir bir-boyutlu optimizasyon problemidir. Arama yönünün bulunması ise bir sonraki alt-kısımda görüleceği gibi bazı matematiksel temellere dayanmaktadır.

3.3. Matematiksel Temeller

3.3.1 Gradyant, Hessian ve Jacobian Matrisleri

Bu alt-kısımda, çok değişkenli bir fonksiyonunun belli bir noktada değerinin azalması için değişkenlerin hangi yönde değiştirilmesi konusu ele alınmıştır. İlk olarak $f(x)$ gibi bir-değişkenli problemi ele alalım. Tasarım değişkenindeki değişimlere bağlı olarak bu fonksiyondaki değişimi analiz edebilmek için birinci ve ikinci türevlere ihtiyaç vardır. Bir-değişkenli bir fonksiyonun birinci türevi,

$$\frac{df(x)}{dx} = \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{\Delta x} = \lim_{\Delta x \rightarrow 0} \frac{\Delta f(x)}{\Delta x} \quad 0$$

şeklindeyken, ikinci türevi

$$\frac{d^2 f(x)}{dx^2} = \frac{d}{dx} \left(\frac{df(x)}{dx} \right) = \lim_{\Delta x \rightarrow 0} \frac{\Delta \left(\frac{df(x)}{dx} \right)}{\Delta x} \quad 0$$

şeklindeyken. Burada, $\Delta(\cdot)$ notasyonu sonlu/önemli bir değişimi gösterirken, $d(\cdot)$ ve $\delta(\cdot)$ notasyonları diferansiyel/küçük değişimleri göstermektedir.

Şimdi de, benzer şekilde, $f(x_1, x_2, \dots, x_n)$ gibi n -değişkenli bir fonksiyonu ele alalım. Artık kısmi türevler söz konusu olmaktadır. n -değişkenli bir fonksiyonun birinci mertebeden kısmi türevleri,

$$\begin{aligned} \frac{\partial f(x_1, x_2, \dots, x_n)}{\partial x_1} &= \lim_{\Delta x_1 \rightarrow 0} \frac{f(x_1 + \Delta x_1, x_2, \dots, x_n) - f(x_1, x_2, \dots, x_n)}{\Delta x_1} \\ \frac{\partial f(x_1, x_2, \dots, x_n)}{\partial x_2} &= \lim_{\Delta x_2 \rightarrow 0} \frac{f(x_1, x_2 + \Delta x_2, \dots, x_n) - f(x_1, x_2, \dots, x_n)}{\Delta x_2} \\ &\vdots \\ \frac{\partial f(x_1, x_2, \dots, x_n)}{\partial x_n} &= \lim_{\Delta x_n \rightarrow 0} \frac{f(x_1, x_2, \dots, x_n + \Delta x_n) - f(x_1, x_2, \dots, x_n)}{\Delta x_n} \end{aligned} \quad 0$$

$f(x_1, x_2, \dots, x_n)$ fonksiyonundaki değişim değişkenlerdeki değişimlerden kaynaklanır. Cebir konularından da bilindiği gibi $f(x_1, x_2, \dots, x_n)$ fonksiyonundaki değişim x_1 'deki diferansiyel değişim dx_1 , x_2 'deki diferansiyel değişim dx_2 ve bu şekilde devam ederek en sonunda x_n 'deki diferansiyel değişim dx_n 'nin bir sonucu olarak aşağıdaki gibidir:

$$df(x_1, x_2, \dots, x_n) = \frac{\partial f(x_1, x_2, \dots, x_n)}{\partial x_1} dx_1 + \frac{\partial f(x_1, x_2, \dots, x_n)}{\partial x_2} dx_2 + \dots + \frac{\partial f(x_1, x_2, \dots, x_n)}{\partial x_n} dx_n. \quad 0$$

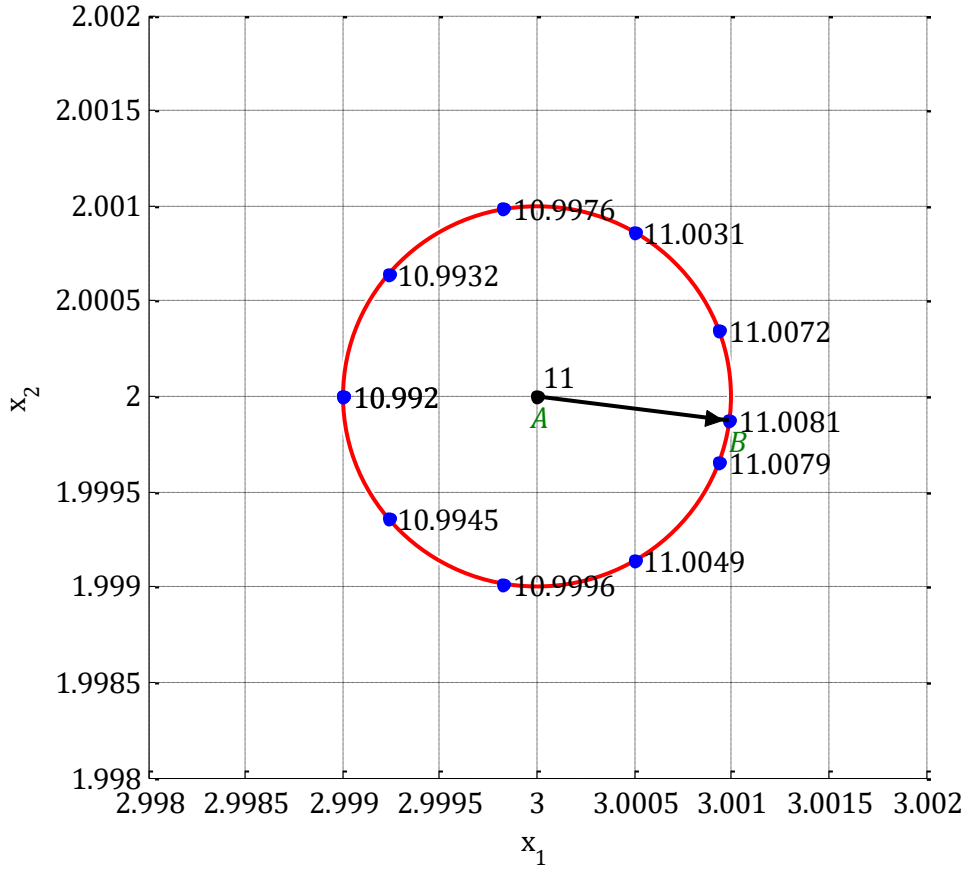
Bir-değişkenli fonksiyonun türevi o fonksiyonun belli bir noktadaki eğimi ile ilişkilidir. Çok-değişkenli bir $f(x_1, x_2, \dots, x_n)$ fonksiyonunun eğimi ise Gradyant vektörü ile gösterilir. $f(x_1, x_2, \dots, x_n)$ fonksiyonunun gradyanı şu şekildedir:

$$\nabla f(x_1, x_2, \dots, x_n) = \begin{bmatrix} \frac{\partial f(x_1, x_2, \dots, x_n)}{\partial x_1} \\ \frac{\partial f(x_1, x_2, \dots, x_n)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x_1, x_2, \dots, x_n)}{\partial x_n} \end{bmatrix} \quad 0$$

Gradyant vektörünün en önemli özelliği, herhangi bir noktadaki Gradyant vektörünün, o fonksiyonun en büyük artım yönünü göstermesidir. Bunu basit bir örnekle gösterirsek; $f(x_1, x_2) = x_1^2 - x_2^2 + x_1x_2$ şeklindeki bir fonksiyonu ele alalım. Bu fonksiyonun gradyan vektörü,

$$\nabla f(x_1, x_2) = \begin{bmatrix} 2x_1 + x_2 \\ -2x_2 + x_1 \end{bmatrix} \quad 0$$

şeklinde. Bu fonksiyon, $A = (3,2)$ noktasında $f(3,2) = 11$ değerini almaktadır. Şimdi bu fonksiyonun bu nokta civarındaki davranışına daha yakından bakalım: Merkezi A noktasında olan $\epsilon = 0.001$ yarıçaplı bir çember üzerinde bazı noktalarda bu fonksiyonun aldığı değerler aşağıdaki şekilde görülmektedir. Şekilden de görüldüğü gibi fonksiyon bu çember üzerinde en büyük değerini, gradyan vektörünün işaret ettiği noktada (B noktası) almaktadır.



Şekil 2.2

Temel amacın bir fonksiyonun en aza indirilmesi olduğu göz önüne alındığında, Gradyant vektörünün bu problemin çözümünde önemli bir rol oynadığı görülmektedir. Diğer taraftan, çok-değişkenli bir fonksiyonun Hessian matrisi de aşağıdaki gibidir:

$$\nabla^2 f(x_1, x_2, \dots, x_n) = \begin{bmatrix} \frac{\partial^2 f(x_1, x_2, \dots, x_n)}{\partial x_1^2} & \frac{\partial^2 f(x_1, x_2, \dots, x_n)}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 f(x_1, x_2, \dots, x_n)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(x_1, x_2, \dots, x_n)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x_1, x_2, \dots, x_n)}{\partial x_2^2} & \dots & \frac{\partial^2 f(x_1, x_2, \dots, x_n)}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x_1, x_2, \dots, x_n)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x_1, x_2, \dots, x_n)}{\partial x_n \partial x_2} & \dots & \frac{\partial^2 f(x_1, x_2, \dots, x_n)}{\partial x_n^2} \end{bmatrix} \quad ()$$

İleriki konularda sıkça karşılaşılabilecek olan Jacobian matrisi ise n -değişkenli N adet fonksiyon için şu şekildedir:

$$\mathbf{J}(x_1, x_2, \dots, x_n) = \begin{bmatrix} \frac{\partial f_1(x_1, x_2, \dots, x_n)}{\partial x_1} & \frac{\partial f_1(x_1, x_2, \dots, x_n)}{\partial x_2} & \dots & \frac{\partial f_1(x_1, x_2, \dots, x_n)}{\partial x_n} \\ \frac{\partial f_2(x_1, x_2, \dots, x_n)}{\partial x_1} & \frac{\partial f_2(x_1, x_2, \dots, x_n)}{\partial x_2} & \dots & \frac{\partial f_2(x_1, x_2, \dots, x_n)}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_N(x_1, x_2, \dots, x_n)}{\partial x_1} & \frac{\partial f_N(x_1, x_2, \dots, x_n)}{\partial x_2} & \dots & \frac{\partial f_N(x_1, x_2, \dots, x_n)}{\partial x_n} \end{bmatrix} \quad 0$$

3.3.2 Taylor Teoremi ve Taylor Açılımı

Belli bir x_k noktasında bu fonksiyon $f(x_k)$ değerini almaktadır. Bu x_k noktasında küçük bir Δx_k değişimi ile bu fonksiyonun azaltmasını sağlamak için fonksiyonun bu noktada nasıl davrandığını analiz etmek gerekmektedir. Bu analiz için Taylor açılımı uygun bir araçtır. Buna göre, fonksiyon bir x_k noktasında $f(x_k)$ değerini almaktayken $x_k + \Delta x_k$ noktasında hangi değeri alacağı aşağıdaki gibi Taylor açılımı ile belirlenebilir:

$$f(x_k + \Delta x_k) = f(x_k) + \left. \frac{df(x)}{dx} \right|_{x=x_k} \Delta x_k + \frac{1}{2} \left. \frac{d^2 f(x)}{dx^2} \right|_{x=x_k} (\Delta x_k)^2 + h. o. t. \quad 0$$

Bu açılımda genellikle üçüncü ve daha yüksek dereceden terimler ihmal edilir ve duruma göre ya birinci-türevli ya da hem birinci- hem de ikinci-türevli terimler kullanılır. Dolayısıyla, Taylor açılımı yaklaşık olarak

$$f(x_k + \Delta x_k) \cong f(x_k) + \left. \frac{df(x)}{dx} \right|_{x=x_k} \Delta x_k + \frac{1}{2} \left. \frac{d^2 f(x)}{dx^2} \right|_{x=x_k} (\Delta x_k)^2 \quad 0$$

şeklinde yazılabilir. Buradan $\Delta f(x_k) = f(x_k + \Delta x_k) - f(x_k)$ farkı şu şekilde bulunur:

$$\Delta f(x_k) = f(x_k + \Delta x_k) - f(x_k) \cong \left. \frac{df(x)}{dx} \right|_{x=x_k} \Delta x_k + \frac{1}{2} \left. \frac{d^2 f(x)}{dx^2} \right|_{x=x_k} (\Delta x_k)^2 \quad 0$$

Burada eşitliğin sağ tarafındaki ilk terim birinci-dereceden değişim, ikinci terim ise ikinci-dereceden/karesel değişim olarak adlandırılmaktadır.

Şimdi de çok-değişkenli bir fonksiyonun Taylor açılımını yazalım. Çok-değişkenli bir fonksiyonun $(x_1 = x_{1k}, x_2 = x_{2k}, \dots, x_n = x_{nk})$ noktasındaki değeri bilindiğinde, fonksiyonun $(x_1 = x_{1k} + \Delta x_{1k}, x_2 = x_{2k} + \Delta x_{2k}, \dots, x_n = x_{nk} + \Delta x_{nk})$ noktasındaki değerini yaklaşık olarak bulmak için Taylor serileri kullanışlı bir yöntemdir. Çok-değişkenli bir fonksiyonun Taylor açılımı şu şekildedir:

$$\begin{aligned}
& f(x_{1k} + \Delta x_{1k}, x_{2k} + \Delta x_{2k}, \dots, x_{nk} + \Delta x_{nk}) \\
&= f(x_{1k}, x_{2k}, \dots, x_{nk}) + \frac{\partial f(x_1, x_2, \dots, x_n)}{\partial x_1} \bigg|_{\substack{x_1=x_{1k} \\ x_2=x_{2k} \\ \vdots \\ x_n=x_{nk}}} \Delta x_{1k} \\
&+ \frac{\partial f(x_1, x_2, \dots, x_n)}{\partial x_2} \bigg|_{\substack{x_1=x_{1k} \\ x_2=x_{2k} \\ \vdots \\ x_n=x_{nk}}} \Delta x_{2k} + \dots + \frac{\partial f(x_1, x_2, \dots, x_n)}{\partial x_n} \bigg|_{\substack{x_1=x_{1k} \\ x_2=x_{2k} \\ \vdots \\ x_n=x_{nk}}} \Delta x_{nk} \\
&+ \frac{1}{2} \frac{\partial^2 f(x_1, x_2, \dots, x_n)}{\partial x_1^2} \bigg|_{\substack{x_1=x_{1k} \\ x_2=x_{2k} \\ \vdots \\ x_n=x_{nk}}} (\Delta x_{1k})^2 \\
&+ \frac{1}{2} \frac{\partial^2 f(x_1, x_2, \dots, x_n)}{\partial x_1 \partial x_2} \bigg|_{\substack{x_1=x_{1k} \\ x_2=x_{2k} \\ \vdots \\ x_n=x_{nk}}} \Delta x_{1k} \Delta x_{2k} + \dots \\
&+ \frac{1}{2} \frac{\partial^2 f(x_1, x_2, \dots, x_n)}{\partial x_1 \partial x_n} \bigg|_{\substack{x_1=x_{1k} \\ x_2=x_{2k} \\ \vdots \\ x_n=x_{nk}}} \Delta x_{1k} \Delta x_{nk} \\
&+ \frac{1}{2} \frac{\partial^2 f(x_1, x_2, \dots, x_n)}{\partial x_2 \partial x_1} \bigg|_{\substack{x_1=x_{1k} \\ x_2=x_{2k} \\ \vdots \\ x_n=x_{nk}}} \Delta x_{2k} \Delta x_{1k} + \dots \\
&+ \frac{1}{2} \frac{\partial^2 f(x_1, x_2, \dots, x_n)}{\partial x_2 \partial x_n} \bigg|_{\substack{x_1=x_{1k} \\ x_2=x_{2k} \\ \vdots \\ x_n=x_{nk}}} \Delta x_{2k} \Delta x_{nk} + \dots \\
&+ \frac{1}{2} \frac{\partial^2 f(x_1, x_2, \dots, x_n)}{\partial x_n \partial x_1} \bigg|_{\substack{x_1=x_{1k} \\ x_2=x_{2k} \\ \vdots \\ x_n=x_{nk}}} \Delta x_{nk} \Delta x_{1k} + \dots \\
&+ \frac{1}{2} \frac{\partial^2 f(x_1, x_2, \dots, x_n)}{\partial x_n^2} \bigg|_{\substack{x_1=x_{1k} \\ x_2=x_{2k} \\ \vdots \\ x_n=x_{nk}}} (\Delta x_{nk})^2 + h. o. t.
\end{aligned}$$

0

Bu açılım, vektör notasyonu ile şu şekilde yazılabilir:

$$f(\mathbf{x}_k + \Delta \mathbf{x}_k) = f(\mathbf{x}_k) + [\nabla f(\mathbf{x}_k)]^T \Delta \mathbf{x}_k + \frac{1}{2} [\Delta \mathbf{x}_k]^T \nabla^2 f(\mathbf{x}_k) \Delta \mathbf{x}_k + h. o. t$$

0

burada

$$\mathbf{x}_k = \begin{bmatrix} x_{1k} \\ x_{2k} \\ \vdots \\ x_{nk} \end{bmatrix}, \Delta \mathbf{x}_k = \begin{bmatrix} \Delta x_{1k} \\ \Delta x_{2k} \\ \vdots \\ \Delta x_{nk} \end{bmatrix} \quad 0$$

şeklindedir.

3.3.3 İniş Yönü

Belli bir \mathbf{x}_k noktasında çok-değişkenli bir fonksiyonun değerinin çok küçük bir $\Delta \mathbf{x}_k$ değişimi ile azalması için gerekli koşula *İniş Yönü* koşulu ve bu koşulu sağlayan değişim miktarına da *İniş Yönü* denmektedir. İniş Yönü, fonksiyon \mathbf{x}_k noktasındayken hangi yönde çok küçük bir ilerleme yapılmalı ki fonksiyonun değeri azalsın sorusuna cevap vermektedir. Şimdi iniş yönü şartını bulalım. İlerleme miktarı $\|\Delta \mathbf{x}_k\|$ çok küçük olduğundan, Taylor açılımında sadece birinci-dereceden terimler alınıp diğerleri ihmal edilebilir, yani

$$f(\mathbf{x}_k + \Delta \mathbf{x}_k) \cong f(\mathbf{x}_k) + [\nabla f(\mathbf{x}_k)]^T \Delta \mathbf{x}_k \quad 0$$

yazılabilir. \mathbf{x}_k noktasındaki $\Delta \mathbf{x}_k$ değişimi ile fonksiyonun değerinin azalması, yani $f(\mathbf{x}_k + \Delta \mathbf{x}_k) < f(\mathbf{x}_k)$ şartının sağlanması isteniyor. Taylor yaklaşıklığı kullanılırsa bu şart,

$$f(\mathbf{x}_k + \Delta \mathbf{x}_k) \cong f(\mathbf{x}_k) + [\nabla f(\mathbf{x}_k)]^T \Delta \mathbf{x}_k < f(\mathbf{x}_k) \quad 0$$

Şekline dönüşecektir ve gerekli sadeleşmeler yapıldıktan sonra aşağıdaki gibi İniş Yönü Şartı elde edilir:

$$[\nabla f(\mathbf{x}_k)]^T \Delta \mathbf{x}_k < 0. \quad 0$$

3.4. Optimallik için Analitik Koşullar

Analitik koşullar, kısıtsız problem için optimum çözümün bulunmasında kullanılacak olan gerek ve yeter koşullar. Aşağıdaki problemi ele alalım:

$$\begin{aligned} & \min_{\mathbf{x}} f(\mathbf{x}) \\ & \mathbf{x}^l \leq \mathbf{x} \leq \mathbf{x}^u \end{aligned} \quad 0$$

\mathbf{x}^* noktası bu problem için bir çözüm adayı olsun. \mathbf{x}^* noktasının bir yerel minimum olması için aşağıdaki şartı sağlaması gerekiyordu:

$$\forall \mathbf{x} \in \mathcal{S} \text{ öyle ki } \|\mathbf{x}^* - \mathbf{x}\| < \epsilon \rightarrow f(\mathbf{x}^*) \leq f(\mathbf{x}) \quad 0$$

burada \mathcal{S} notasyonu $\mathbf{x}^l \leq \mathbf{x} \leq \mathbf{x}^u$ koşulunun sağlandığı bölgeyi göstermektedir. Benzer şekilde, eğer bir \mathbf{x}^* noktası aşağıdaki şartı sağlarsa bu nokta kesin-yerel minimumdur:

$$\forall \mathbf{x} \in \mathcal{S} \text{ ve } \mathbf{x} \neq \mathbf{x}^* \text{ öyle ki } \|\mathbf{x}^* - \mathbf{x}\| < \epsilon \rightarrow f(\mathbf{x}^*) < f(\mathbf{x}) \quad ()$$

Bir fonksiyonun bir yerel minimuma sahipken hiç global minimumu olmaması mümkündür. Hatta, bir fonksiyonun ne yerel ne de global minimumu olmayabilir, her ikisi birden olabilir, birden fazla yerel minimumu olabilir. Bir noktanın optimumluğunu belirlemek için gerekli daha pratik koşullara ihtiyaç vardır. Bunları elde etmek için $f(\mathbf{x})$ fonksiyonunun birinci ve ikinci türevlerinin mevcut ve \mathbf{x}^* noktası civarında sürekli olduğunu varsayacağız.

3.4.1 Birinci-dereceden Koşullar

Varsayalım ki \mathbf{x}^* noktası $f(\mathbf{x})$ fonksiyonunun yerel minimumu olsun. Fonksiyonun \mathbf{x}^* noktası civarındaki birinci dereceden Taylor açılımı,

$$f(\mathbf{x}^* + \mathbf{p}) \cong f(\mathbf{x}^*) + [\nabla f(\mathbf{x}^*)]^T \mathbf{p} \quad ()$$

şeklindeydi ki burada \mathbf{p} herhangi bir vektördür ve bir ilerleme yönünü göstermektedir. Burada $\nabla f(\mathbf{x}^*) = \mathbf{0}$ olduğu gösterilecektir. Eğer \mathbf{x}^* bir yerel minimum ise bu noktada artık fiziksel bir *inış yönü* bulunamaz, yani mümkün olan tüm \mathbf{p} ilerleme yönleri için $[\nabla f(\mathbf{x}^*)]^T \mathbf{p} \geq 0$ olmaktadır. Bu durumda \mathbf{x}^* noktası bir yerel minimum ise,

$$\nabla f(\mathbf{x}^*) = \mathbf{0} \quad ()$$

olmaktadır ki bu koşullara *birinci-dereceden/gerek koşullar* adı verilmektedir. Bu koşulları sağlayan noktaya *durağan nokta* (stationary point) denir. Birinci-dereceden denmesinin sebebi ise koşullarda birinci dereceden türevlerin bulunmasıdır. Sadece gerek şartlar optimum noktanın bulunmasına yetmeyebilir. Gerek şart denmesinin sebebi, \mathbf{x}^* noktasının bir yerel minimum olabilmesi için sağlaması gereken şartlar olduğu içindir. Birinci dereceden koşulların sağlaması, \mathbf{x}^* noktasının bir yerel minimum olmasına yetmez, çünkü bu koşulları yerel minimumun yanı sıra yerel maksimum veya bir *semer noktası* (saddle point) da sağlayabilir. Yerel minimumlar ancak ikinci dereceden koşulların sağlanmasıyla diğerlerinden ayırt edilebilir.

3.4.2 İkinci-dereceden Koşullar

İkinci-dereceden koşullar çoğunlukla yeter koşullar olarak bilinir. Tekrar Taylor açılımını ele alalım. Bu kez ikinci dereceden yaklaşıklık kullanılacaktır:

$$f(\mathbf{x}^* + \mathbf{p}) \cong f(\mathbf{x}^*) + [\nabla f(\mathbf{x}^*)]^T \mathbf{p} + \frac{1}{2} \mathbf{p}^T \nabla^2 f(\mathbf{x}^*) \mathbf{p} \quad 0$$

Eğer \mathbf{x}^* noktası gerek şartı ($\nabla f(\mathbf{x}^*) = \mathbf{0}$) sağlıyorsa, eşitliğin sağ tarafındaki ikinci terim sıfır olur ve

$$f(\mathbf{x}^* + \mathbf{p}) \cong f(\mathbf{x}^*) + \frac{1}{2} \mathbf{p}^T \nabla^2 f(\mathbf{x}^*) \mathbf{p} \quad 0$$

haline gelir. Bu durumda, \mathbf{x}^* noktasından herhangi bir \mathbf{p} yönünde ilerlediğimizde fonksiyondaki değişim $\Delta f(\mathbf{x}^*)$

$$\Delta f(\mathbf{x}^*) \cong \frac{1}{2} \mathbf{p}^T \nabla^2 f(\mathbf{x}^*) \mathbf{p} \quad 0$$

şeklinde olmaktadır ki \mathbf{x}^* noktasının yerel minimum olması için $\Delta f(\mathbf{x}^*)$ 'ın sıfırdan büyük veya sıfıra eşit olması göz önüne alınırsa,

$$\Delta f(\mathbf{x}^*) \cong \frac{1}{2} \mathbf{p}^T \nabla^2 f(\mathbf{x}^*) \mathbf{p} \geq 0 \quad 0$$

şartı elde edilir ki bu da *ikinci-dereceden/yeter koşul* olmaktadır. Yeter koşulun sağlanması için $\nabla^2 f(\mathbf{x}^*)$ matrisinin pozitif yarı-tanımlı (positive semi-definite) olması gerekir. Benzer şekilde, bir \mathbf{x}^* noktasının kesin-yerel minimum olması için $\nabla f(\mathbf{x}^*) = \mathbf{0}$ olmalı ve $\nabla^2 f(\mathbf{x}^*)$ Hessian matrisi pozitif tanımlı olmalıdır. $\nabla^2 f(\mathbf{x}^*)$ Hessian matrisinin pozitif tanımlı olması için aşağıdaki üç seçenekten birinin sağlanması yeterlidir:

- $\forall \mathbf{p}$ için $\mathbf{p}^T \nabla^2 f(\mathbf{x}^*) \mathbf{p} > 0$ olmalı veya
- $\nabla^2 f(\mathbf{x}^*)$ matrisinin tüm özdeğerleri pozitif olmalı, veya
- $\nabla^2 f(\mathbf{x}^*)$ matrisinin kendisi de dahil olmak üzere tüm alt-kare-matrislerinin determinantları pozitif olmalı.

0

Örnek: $f(x_1, x_2) = (x_1 - 1)^2 + (x_2 - 1)^2 - x_1 x_2$ fonksiyonunun durağan noktalarını bulup tiplerini belirleyiniz.

$$\nabla f(\mathbf{x}) = \mathbf{0} \mapsto \begin{cases} \frac{\partial f}{\partial x_1} = 2(x_1 - 1) - x_2 = 0 \\ \frac{\partial f}{\partial x_2} = 2(x_2 - 1) - x_1 = 0 \end{cases} \mathbf{x}^* = \begin{bmatrix} x_1^* \\ x_2^* \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix},$$

$$\nabla^2 f(\mathbf{x}) = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} \mapsto \nabla^2 f(\mathbf{x}^*) = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$$

f/x_1^2

$\nabla^2 f(\mathbf{x}^*)$ matrisi pozitif tanımlı (özdeğerleri 1 ve 3) olduğundan \mathbf{x}^* noktası kesin-yerel minimumdur.

Örnek: $f(x_1, x_2) = \frac{1}{3}x_1^3 + \frac{1}{3}x_1^2 + 2x_1x_2 + \frac{1}{2}x_2^2 - x_2 + 9$ fonksiyonunun durağan noktalarını bulup tiplerini belirleyiniz.

$$\nabla f(\mathbf{x}) = \mathbf{0} \mapsto \begin{cases} \frac{\partial f}{\partial x_1} = x_1^2 + x_1 + 2x_2 = 0 \\ \frac{\partial f}{\partial x_2} = 2x_1 + x_2 - 1 = 0 \end{cases} \begin{cases} \mathbf{x}_a^* = \begin{bmatrix} x_{1a}^* \\ x_{2a}^* \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \\ \mathbf{x}_b^* = \begin{bmatrix} x_{1b}^* \\ x_{2b}^* \end{bmatrix} = \begin{bmatrix} 2 \\ -3 \end{bmatrix} \end{cases}$$

$$\nabla^2 f(\mathbf{x}) = \begin{bmatrix} 2x_1 + 1 & 2 \\ 2 & 1 \end{bmatrix} \mapsto \begin{cases} \nabla^2 f(\mathbf{x}_a^*) = \begin{bmatrix} 3 & 2 \\ 2 & 1 \end{bmatrix} \\ \nabla^2 f(\mathbf{x}_b^*) = \begin{bmatrix} 5 & 2 \\ 2 & 1 \end{bmatrix} \end{cases}$$

$\nabla^2 f(\mathbf{x}_a^*)$ matrisinin özdeğerlerinden biri negatif bir de pozitif (özdeğerleri 4.23 ve -0.23) olduğundan bu nokta bir semer noktasıdır. Dolayısıyla, ne minimum ne de maksimum noktasıdır. Diğer taraftan $\nabla^2 f(\mathbf{x}_b^*)$ matrisi pozitif tanımlı (özdeğerleri 5.82 ve 0.17) olduğundan \mathbf{x}_b^* noktası kesin-yerel minimumdur.

Lineer-olmayan bir Denklem Sisteminin Çözümü – Newton-Raphson Yöntemi (opsiyonel)

3.5. Gradyant Yöntemler

Gradyant yöntemler, kısıtsız optimizasyon problemini, minimumu bulunacak fonksiyonun türev bilgisini kullanarak çözmeye çalışırlar. Bunun için de, aşağıdaki gibi Taylor açılımından yararlanırlar:

$$f(\mathbf{x}_k + \Delta \mathbf{x}_k) = f(\mathbf{x}_k) + [\nabla f(\mathbf{x}_k)]^T \Delta \mathbf{x}_k + \frac{1}{2} [\Delta \mathbf{x}_k]^T \nabla^2 f(\mathbf{x}_k) \Delta \mathbf{x}_k + h. o. t. \quad ()$$

Burada, $\nabla f(\mathbf{x}_k)$ gradyant terimi birinci-dereceden, $\nabla^2 f(\mathbf{x}_k)$ terimi de ikinci-dereceden türev bilgisi içerir.

3.5.1. Birinci-dereceden Yöntemler

Birinci-dereceden yöntemler Taylor açılımında sadece birinci-dereceden türev bilgisini kullandıkları için bu ismi almışlardır. Bu yöntemleri uygulamak için fonksiyonun sadece gradyant vektörünü bilmek yeterlidir. Bu alt-kısımda, birinci-dereceden yöntemlerin başlıcaları ele alınmıştır.

3.5.1.1. Dik-İniş (Steepest-Descent - SD) Yöntemi

Cauchy tarafından 1847’de önerilmiştir. Bir fonksiyonun bir noktadaki Gradyant vektörünün, fonksiyonun o noktadaki en büyük artım yönünü gösterdiği daha önce belirtilmişti. Dik-İniş yöntemi de buradan hareketle, her adımda gradyant vektörünün ters yönünde hareket ederek fonksiyonu azaltma ilkesine dayanmaktadır. Yöntemin algoritması şu şekildedir.

Dik-İniş Algoritması

Adım 1 Bir başlangıç noktası (\mathbf{x}_0) ve maksimum iterasyon sayısı (N_{\max}) belirle.

Sonlandırma kriterleri için ε_1 , ε_2 ve ε_3 değerlerini belirle.

$k \leftarrow 0$

Adım 2 \mathbf{x}_k noktasındaki gradyant vektörünü $\nabla f(\mathbf{x}_k)$ hesapla

İlerleme yönü olarak $\mathbf{p}_k = -\nabla f(\mathbf{x}_k)$ seç.

Bir-boyutlu optimizasyon ile $f(\mathbf{x}_k + s_k \mathbf{p}_k)$ değerini minimum yapan adım-aralığını (s_k) bul.

$\mathbf{x}_{k+1} = \mathbf{x}_k + s_k \mathbf{p}_k$ kuralı ile güncellemeyi yap.

$k \leftarrow k + 1$

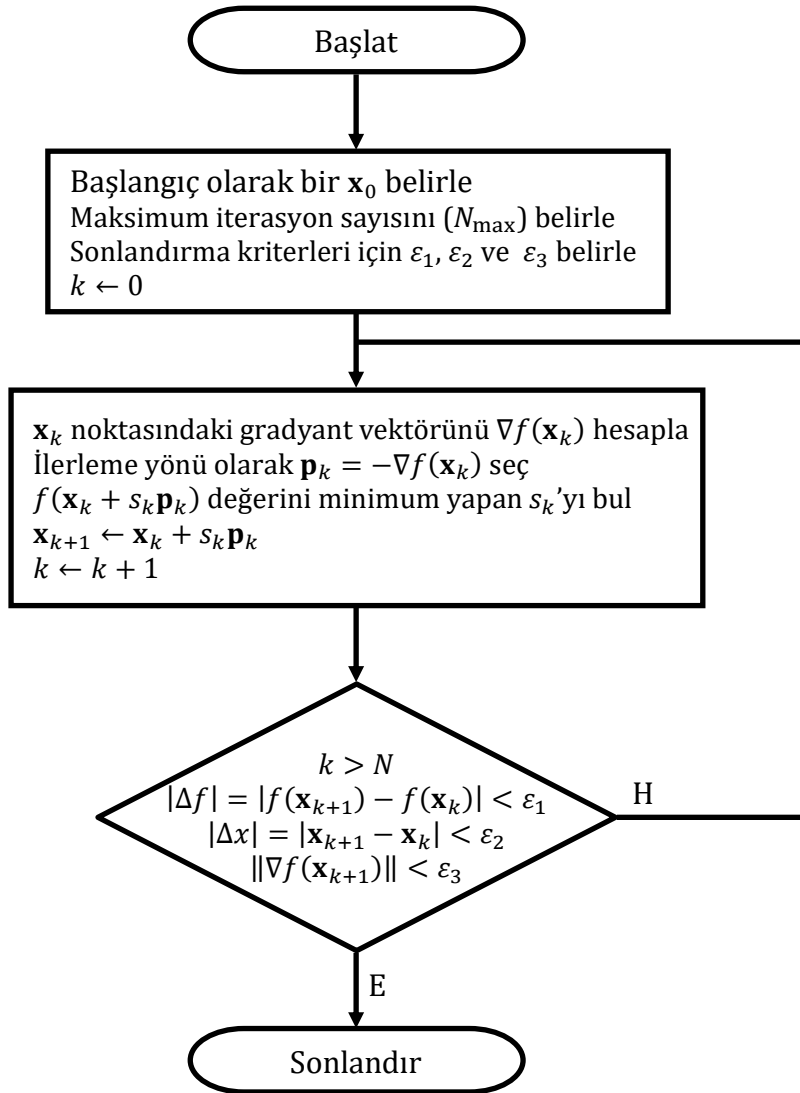
Adım 3 Aşağıdaki şartlardan herhangi biri sağlanıyorsa algoritmayı bitir, sağlanmıyorsa Adım 2’ye git.

C1: $N < k$ maksimum iterasyona ulaşıldı.

C2: $|\Delta f| = |f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k)| < \varepsilon_1$ fonksiyon değişmiyor.

C3: $|\Delta x| = |\mathbf{x}_{k+1} - \mathbf{x}_k| < \varepsilon_2$ değişkenler değişmiyor.

C4: $\|\nabla f(\mathbf{x}_{k+1})\| < \varepsilon_3$ yerel minimuma yakınsadı.



Örnek: Aşağıdaki kısıtsız optimizasyon problemini verilen aralık içerisinde Dik-İniş yöntemiyle çözünüz. Başlangıç noktası olarak $\mathbf{x}_0 = [-4.5 \ -3.5]^T$ alınız.

$$\min_{\mathbf{x}} f(x_1, x_2) = 3 + (x_1 - 1.5x_2)^2 + (x_2 - 2)^2$$

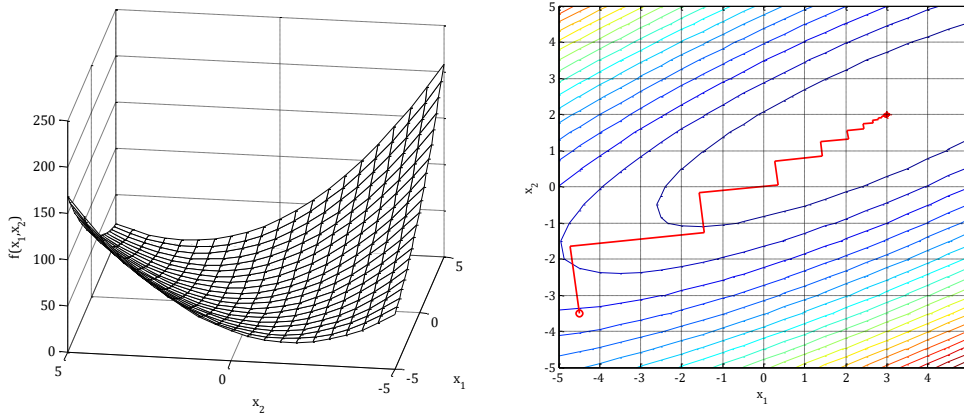
$$-5 \leq \mathbf{x} \leq 5$$

0

Aşağıdaki şekillerden soldakinde, değişkenlere bağlı olarak fonksiyonun değerinin değişimi görülmektedir. Diş-İniş yöntemi, Altın-Oran ile birlikte uygulandığında aşağıdaki tabloda görüldüğü gibi 46 adımda yerel minimuma ($\mathbf{x}^* = [3.0 \ 2.0]^T$) ulaşmaktadır.

k	x_1	x_2	s_k	$f(x_1, x_2)$
1	-4.71084	-1.637581	0.14056	21.314623
2	-1.45482	-1.268029	0.722126	13.880026
3	-1.580544	-0.160736	0.14056	9.462878
4	0.351517	0.057904	0.72122	6.841783
5	0.277115	0.715468	0.14056	5.283777
6	1.427035	0.846051	0.722233	4.35655
7	1.382629	1.237057	0.14056	3.805769
8	2.064839	1.314241	0.72122	3.479004
9	2.038561	1.546439	0.14056	3.284733
10	2.444537	1.592531	0.722126	3.169138
11	2.428867	1.730583	0.14056	3.100477
12	2.669823	1.757859	0.721393	3.059723
13	2.660537	1.83986	0.14056	3.035496
14	2.803849	1.856119	0.721953	3.021089
15	2.798319	1.904861	0.14056	3.012529
16	2.883417	1.914499	0.7215	3.007446
17	2.880137	1.943455	0.14056	3.004426
18	2.93074	1.949195	0.721953	3.002629
19	2.928787	1.966407	0.14056	3.001562
20	2.958831	1.969809	0.721393	3.000928
21	2.957673	1.980033	0.14056	3.000552
22	2.975547	1.982061	0.722126	3.000328
23	2.974856	1.988139	0.14056	3.000195
24	2.985462	1.989339	0.72122	3.000116
25	2.985053	1.992949	0.14056	3.000069
26	2.991366	1.993666	0.722233	3.000041
27	2.991122	1.995812	0.14056	3.000024
28	2.994867	1.996236	0.72122	3.000014
29	2.994722	1.99751	0.14056	3.000009
30	2.996951	1.997763	0.722126	3.000005
31	2.996865	1.998521	0.14056	3.000003
32	2.998187	1.998671	0.72122	3.000002
33	2.998136	1.999121	0.14056	3.000001
34	2.998923	1.99921	0.722233	3.000001
35	2.998893	1.999478	0.14056	3.000000
36	2.99936	1.999531	0.72122	3.000000
37	2.999342	1.99969	0.14056	3.000000
38	2.99962	1.999721	0.722233	3.000000
39	2.999609	1.999816	0.14056	3.000000
40	2.999774	1.999834	0.72122	3.000000
41	2.999768	1.99989	0.14056	3.000000
42	2.999866	1.999902	0.722126	3.000000
43	2.999862	1.999935	0.14056	3.000000
44	2.99992	1.999941	0.72122	3.000000
45	2.999918	1.999961	0.14056	3.000000
46	2.999953	1.999965	0.72178	3.000000

Aşağıdaki şekillerden sağdaki eşdeğer eğrileri ve algoritmanın ilerleyişi görülmektedir.



Şekilden görüldüğü gibi bu algoritma çözüme ilerlerken zig-zag çizmektedir, yani ilerleme yönü her adımda çok fazla değişebilmektedir ki bu da algoritmanın bir önceki ilerleme yönünü dikkate almadığını gösterir. Bu durum Diş-İniş yönteminin “hafızasız” bir yöntem olmasının bir sonucudur.

Bu örnekten de görüleceği gibi Diş-İniş yöntemi hafızasızdır, önceki ilerleme yönlerini dikkate almaz ki bu da algoritmanın yerel minimuma daha fazla adımda yakınsamasına yol açar. Bu algoritmaya alternatif olarak bir önceki yönü de dikkate alan Conjugate-Gradient yöntemi önerilmiştir.

3.5.1.2. Conjugate-Gradient (Fletcher-Reeves) Yöntemi (CG)

Dik İniş algoritmasının değiştirilmiş bir şeklidir. Arama yönü Hessian matrisine göre eşleniktir. n değişkenli bir karesel problemi n 'den daha az iterasyonda çözer.

Conjugate-Gradient Algoritması

Adım 1 Bir başlangıç noktası (\mathbf{x}_0) ve maksimum iterasyon sayısı (N_{\max}) belirle.

Sonlandırma kriterleri için ε_1 , ε_2 ve ε_3 değerlerini belirle.

$k \leftarrow 0$

Adım 2 \mathbf{x}_k noktasındaki gradyant vektörünü $\nabla f(\mathbf{x}_k)$ hesapla

Eğer $k = 0$ ise ilerleme yönü olarak $\mathbf{p}_k = -\nabla f(\mathbf{x}_k)$ seç.

Eğer $k \neq 0$ ise ilerleme yönü olarak $\mathbf{p}_k = -\nabla f(\mathbf{x}_k) + \beta \mathbf{p}_{k-1}$ seç,

burada $\beta = \frac{\nabla^T f(\mathbf{x}_k) \nabla f(\mathbf{x}_k)}{\nabla^T f(\mathbf{x}_{k-1}) \nabla f(\mathbf{x}_{k-1})}$ şeklindedir.

Bir-boyutlu optimizasyon ile $f(\mathbf{x}_k + s_k \mathbf{p}_k)$ değerini minimum yapan adım-aralığını (s_k) bul.

$\mathbf{x}_{k+1} = \mathbf{x}_k + s_k \mathbf{p}_k$ kuralı ile güncellemeyi yap.

$$k \leftarrow k + 1$$

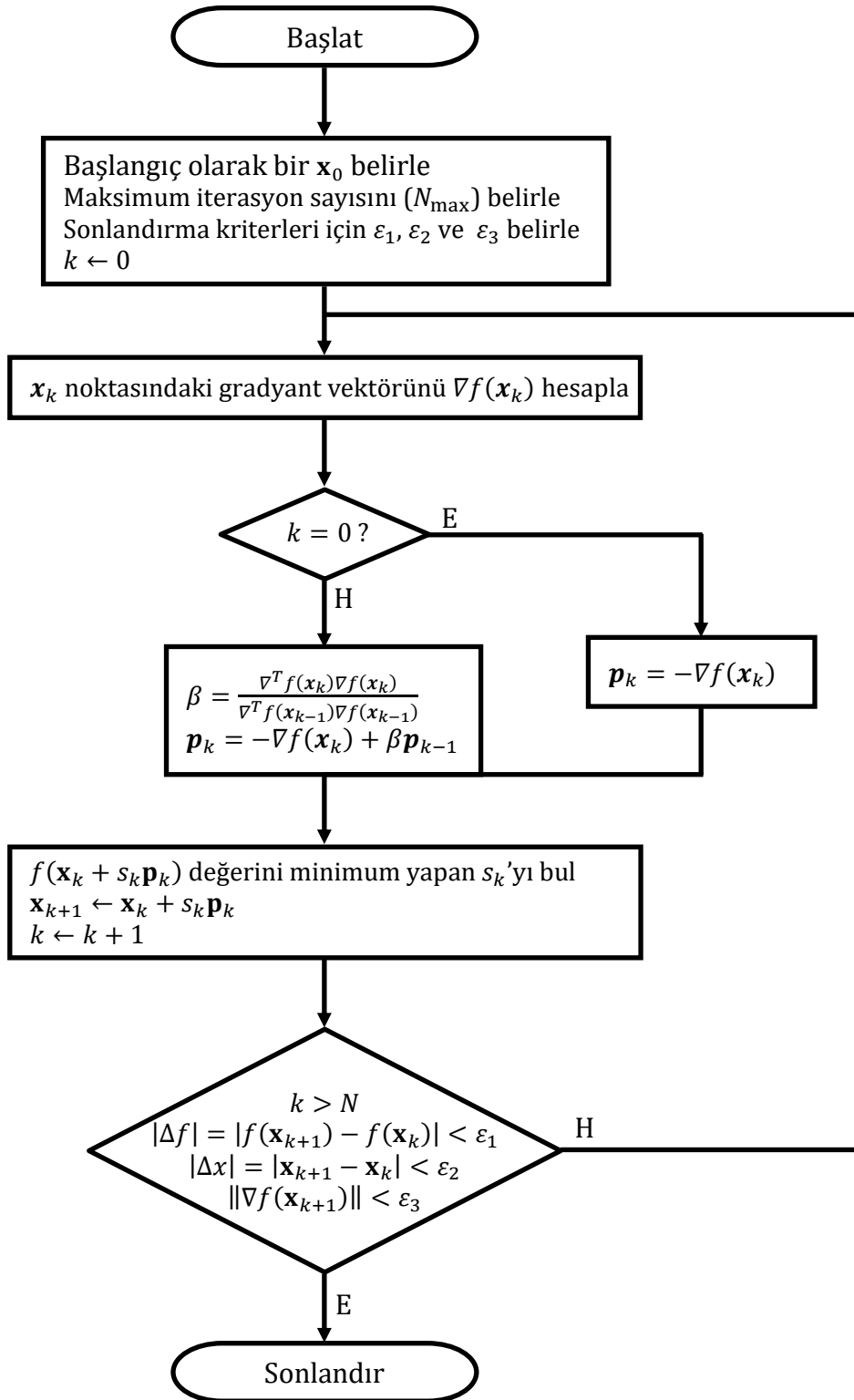
Adım 3 Aşağıdaki şartlardan herhangi biri sağlanıyorsa algoritmayı bitir, sağlanmıyorsa Adım 2'ye git.

$C1: N < k$ maksimum iterasyona ulaşıldı.

$C2: |\Delta f| = |f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k)| < \varepsilon_1$ fonksiyon değişmiyor.

$C3: |\Delta x| = |\mathbf{x}_{k+1} - \mathbf{x}_k| < \varepsilon_2$ değişkenler değişmiyor.

$C4: \|\nabla f(\mathbf{x}_{k+1})\| < \varepsilon_3$ yerel minimuma yakınsadı.



Örnek: Aşağıdaki kısıtsız optimizasyon problemini verilen aralık içerisinde Conjugate-Gradient yöntemiyle çözünüz. Başlangıç noktası olarak $\mathbf{x}_0 = [-4.5 \quad -3.5]^T$ alınız.

$$\min_{\mathbf{x}} f(x_1, x_2) = 3 + (x_1 - 1.5x_2)^2 + (x_2 - 2)^2$$

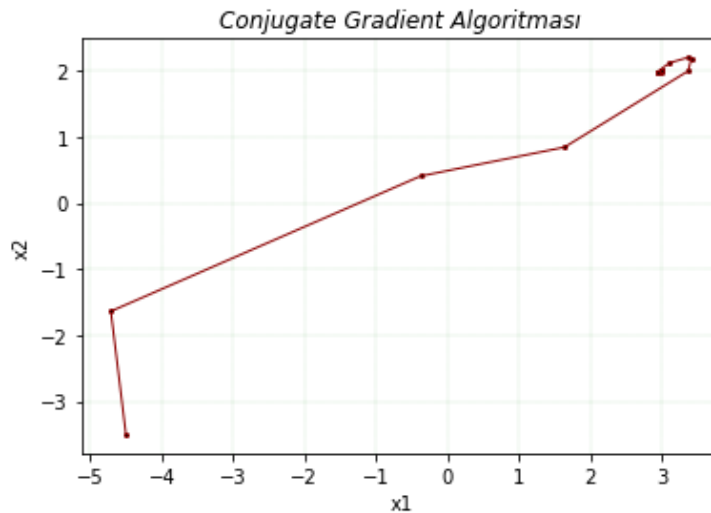
$$-5 \leq \mathbf{x} \leq 5$$

0

Conjugate-Gradient yöntemi, Altın-Oran ile birlikte uygulandığında algoritma 24 adımda yerel minimuma ($\mathbf{x}^* = [3.0 \ 2.0]^T$) ulaşmaktadır. Bu durum Conjugate-Gradient yönteminin bir önceki ilerleme yönünü dikkate almasının bir sonucudur.

k	x_1	x_2	s_k	$f(x_1, x_2)$
1	-4.710840	-1.637581	0.140560	21.314623
2	-0.376224	0.408366	0.999860	6.510970
3	1.640943	0.846300	0.717661	4.469030
4	3.371656	2.000649	0.217294	3.137406
5	3.411421	2.180206	0.095335	3.052387
6	3.369560	2.204306	0.159367	3.045723
7	3.109730	2.122144	0.793769	3.020319
8	2.943367	1.989885	0.155808	3.001821
9	2.943679	1.973890	0.095508	3.000976
10	2.951443	1.971557	0.224865	3.000844
11	2.994495	1.988829	0.706598	3.000251
12	3.008061	2.002282	0.123006	3.000027
13	3.007662	2.003716	0.098614	3.000018
14	3.006087	2.003856	0.341453	3.000015
15	2.999817	2.000847	0.520047	3.000003
16	2.998888	1.999608	0.106398	3.000000
17	2.998964	1.999477	0.106291	3.000000
18	2.999297	1.999503	0.517501	3.000000
19	3.000098	1.999952	0.343545	3.000000
20	3.000152	2.000061	0.098721	3.000000
21	3.000139	2.000073	0.123112	3.000000
22	3.000071	2.000059	0.710543	3.000000
23	2.999982	2.000000	0.224066	3.000000
24	2.999979	1.999991	0.095615	3.000000

Aşağıdaki şekilden algoritmanın ilerleyişi görülmektedir.



3.5.2 İkinci-dereceden Yöntemler – Newton Yöntemi

İkinci-dereceden yöntemler, Taylor açılımında hem birinci- hem de ikinci-dereceden türev bilgisini kullanırlar. Bu yöntemleri uygulamak için fonksiyonun gradyant vektörünün yanısıra Hessian matrisini de kullanmak gerekir. Bu alt-kısımda, ikinci-dereceden yöntemlerin başlıcaları ele alınmıştır.

3.5.2.1 Newton Yöntemi

Newton yöntemi, k . iterasyonda, bir \mathbf{x}_k noktasındayken uygun ilerleme yönü olan \mathbf{p}_k yönünü bulurken aşağıdaki gibi ikinci-dereceden Taylor yaklaşıklığını kullanır:

$$f(\mathbf{x}_k + \mathbf{p}_k) \cong f(\mathbf{x}_k) + [\nabla f(\mathbf{x}_k)]^T \mathbf{p}_k + \frac{1}{2} [\mathbf{p}_k]^T \nabla^2 f(\mathbf{x}_k) \mathbf{p}_k \quad ()$$

Bu yaklaşıklıkla $f(\mathbf{x}_k + \mathbf{p}_k)$ fonksiyonu karesel bir fonksiyonla temsil edilmektedir. $f(\mathbf{x}_k + \mathbf{p}_k)$ fonksiyonunu \mathbf{p}_k vektörüne göre optimize etmek için \mathbf{p}_k 'ya göre türevi alınıp sıfıra eşitlenirse;

$$\frac{\partial f(\mathbf{x}_k + \mathbf{p}_k)}{\partial \mathbf{p}_k} = \nabla f(\mathbf{x}_k) + \nabla^2 f(\mathbf{x}_k) \mathbf{p}_k = \mathbf{0} \quad ()$$

elde edilir. Böylece, Newton yönteminde k . iterasyondaki ilerleme yönü \mathbf{p}_k ,

$$\nabla^2 f(\mathbf{x}_k) \mathbf{p}_k = -\nabla f(\mathbf{x}_k) \quad ()$$

şeklindeki lineer denklem sisteminin çözümünden bulunur. Bu denklem sisteminin çözümünden bulunan Newton yönü her seferinde $f(\mathbf{x}_k + \mathbf{p}_k)$ fonksiyonunu minimize etmeye çalışır. Bu yaklaşım, \mathbf{x}_k noktasındaki ikinci dereceden Taylor açılımına dayanır. \mathbf{x}_k noktasındaki bu açılım lineer-olmayan $f(\mathbf{x}_k + \mathbf{p}_k)$ fonksiyonunu ne kadar iyi temsil ederse, bulunan \mathbf{p}_k yönü o kadar uygun bir yön olacaktır.

3.5.2.2 Değiştirilmiş Newton Yöntemi

Dejenere durumlar dışında Newton yöntemi karesel bir yakınsama hızına sahiptir. Eğer Newton yöntemi yakınsarsa bu yakınsama durağan noktaya olur. ancak Newton yöntemi bu haliyle nadiren kullanılır. Yöntemi daha güvenilir ve işlemsel olarak daha az karmaşık yapmak için bazı modifikasyonlar yapılmıştır. Newton yöntemi yakınsamayabilir veya

yakınsasa bile bu bir yerel minimum olmayabilir. Newton yönteminin yakınsamasını ve hatta mevcut ise bir yerel minimuma yakınsamasını garanti etmek için bazı ilave stratejiler için içine katılabilir. Bunun için bir benimsenen yaklaşım ise $\nabla^2 f(\mathbf{x}_k) \mathbf{p}_k = -\nabla f(\mathbf{x}_k)$ denkleminin çözümünden bulunan \mathbf{p}_k yönünü genel güncelleme kuralı $\mathbf{x}_{k+1} = \mathbf{x}_k + s_k \mathbf{p}_k$ içinde kullanmaktır ki burada s_k adım-aralığı $f(\mathbf{x}_{k+1}) < f(\mathbf{x}_k)$ olacak şekilde seçilir. Klasik Newton yönteminde adım-aralığı her zaman $s_k = 1$ olmaktadır ve fonksiyonun azalmasını garanti etmemektedir.

Klasik Newton yönteminde ilerleme yönü $s_k > 0$ olmak üzere $f(\mathbf{x}_k + s_k \mathbf{p}_k) < f(\mathbf{x}_k)$ olacak şekilde seçilir. Bu ancak \mathbf{p}_k 'nın bir iniş yönü (descent direction) olmasıyla mümkün olabilir, yani $[\nabla f(\mathbf{x}_k)]^T \mathbf{p}_k < 0$ olmalıdır. Bu iniş yönünün, Newton yönteminde nasıl garanti edilebileceğini bulmak için klasik Newton yöntemindeki ilerleme yönünün $\mathbf{p}_k = -[\nabla^2 f(\mathbf{x}_k)]^{-1} \nabla f(\mathbf{x}_k)$ olduğunu hatırlayalım. Eğer \mathbf{p}_k bir iniş yönü olacaksa aşağıdaki şart sağlanmalıdır:

$$[\nabla f(\mathbf{x}_k)]^T \mathbf{p}_k = -[\nabla f(\mathbf{x}_k)]^T [\nabla^2 f(\mathbf{x}_k)]^{-1} \nabla f(\mathbf{x}_k) < 0 \quad ()$$

veya başka bir ifadeyle,

$$[\nabla f(\mathbf{x}_k)]^T [\nabla^2 f(\mathbf{x}_k)]^{-1} \nabla f(\mathbf{x}_k) > 0 \quad ()$$

şartı sağlanmalıdır. Bu iniş yönü şartı ancak ve ancak $[\nabla^2 f(\mathbf{x}_k)]^{-1}$ matrisinin (veya eşdeğer olarak $\nabla^2 f(\mathbf{x}_k)$ matrisinin) pozitif-tanımlı olması ile mümkündür. $\nabla^2 f(\mathbf{x}_k)$ matrisinin pozitif-tanımlı olması şartı $[\nabla f(\mathbf{x}_k)]^T \mathbf{p}_k < 0$ şartından daha kuvvetli bir şarttır. Bunu daha iyi açıklamak için Taylor açılımına geri dönelim:

$$f(\mathbf{x}_k + \mathbf{p}_k) \cong f(\mathbf{x}_k) + [\nabla f(\mathbf{x}_k)]^T \mathbf{p}_k + \frac{1}{2} [\mathbf{p}_k]^T \nabla^2 f(\mathbf{x}_k) \mathbf{p}_k. \quad ()$$

Newton formülü bu karesel fonksiyon şeklindeki bu yaklaşıklığın \mathbf{p}_k 'ya göre türevinin alınıp sıfıra eşitlenmesinden bulunmuştu. Karesel bir fonksiyonun bir minimuma sahip olabilmesi için $\nabla^2 f(\mathbf{x}_k)$ matrisinin pozitif-tanımlı olması gerekir. Eğer $\nabla^2 f(\mathbf{x}_k)$ matrisi pozitif-tanımlı ise minimum noktası türevin sıfıra eşitlenmesinden bulunabilir. Eğer $\nabla^2 f(\mathbf{x}_k)$ matrisi pozitif-tanımlı değilse, bu karesel fonksiyonun bir minimumu olmaz.

İterasyon sırasında $\nabla^2 f(\mathbf{x}_k)$ matrisi pozitif-tanımlı olmazsa bu matrisi uygun bir pozitif-tanımlı matris ile değiştirmek en çok başvurulan yollardan biridir. Bu şekilde ilerleme yönünün iniş yönü olması garanti edilir. Bu ilerleme yönü, amaç fonksiyonunun ($f(\mathbf{x})$) karesel yaklaşıklığının minimize edilmesi yönündedir. $\nabla^2 f(\mathbf{x}_k)$ matrisi her zaman simetrik bir matristir ve simetrik matrislerin özdeğerleri her zaman reeldir.

Eğer $\nabla^2 f(\mathbf{x}_k)$ matrisi pozitif-tanımlı değilse kullanılabilecek bir yöntem *Birim Matris Ekleme* yöntemidir. Bu matrise uygun bir ekleme yapılarak $\nabla^2 f(\mathbf{x}_k) + \mu \mathbf{I}$ matrisinin pozitif-tanımlı olması sağlanır.

Böylece, Değiştirilmiş-Newton yönteminin algoritması şu şekilde verilebilir:

Değiştirilmiş-Newton Algoritması

Adım 1 Bir başlangıç noktası (\mathbf{x}_0) ve maksimum iterasyon sayısı (N_{\max}) belirle.

Sonlandırma kriterleri için ε_1 , ε_2 ve ε_3 değerlerini belirle.

$k \leftarrow 0$

Adım 2 \mathbf{x}_k noktasındaki gradyant vektörünü $\nabla f(\mathbf{x}_k)$ hesapla

\mathbf{x}_k noktasındaki Hessian matrisini $\nabla^2 f(\mathbf{x}_k)$ hesapla

Eğer Hessian matrisi pozitif-tanımlı ise ilerleme yönü olarak $\mathbf{p}_k = -[\nabla^2 f(\mathbf{x}_k)]^{-1} \nabla f(\mathbf{x}_k)$ seç.

Eğer Hessian matrisi pozitif-tanımlı değilse o zaman uygun bir matris ilavesi (\mathbf{E} veya $\mu \mathbf{I}$) ile onu pozitif-tanımlı hale getir ve ilerleme yönü olarak $\mathbf{p}_k = -[\nabla^2 f(\mathbf{x}_k) + \mu \mathbf{I}]^{-1} \nabla f(\mathbf{x}_k)$ veya $\mathbf{p}_k = -[\nabla^2 f(\mathbf{x}_k) + \mathbf{E}]^{-1} \nabla f(\mathbf{x}_k)$ seç.

Bir-boyutlu optimizasyon ile $f(\mathbf{x}_k + s_k \mathbf{p}_k)$ değerini minimum yapan adım-aralığını (s_k) bul.

$\mathbf{x}_{k+1} = \mathbf{x}_k + s_k \mathbf{p}_k$ kuralı ile güncellemeyi yap.

$k \leftarrow k + 1$

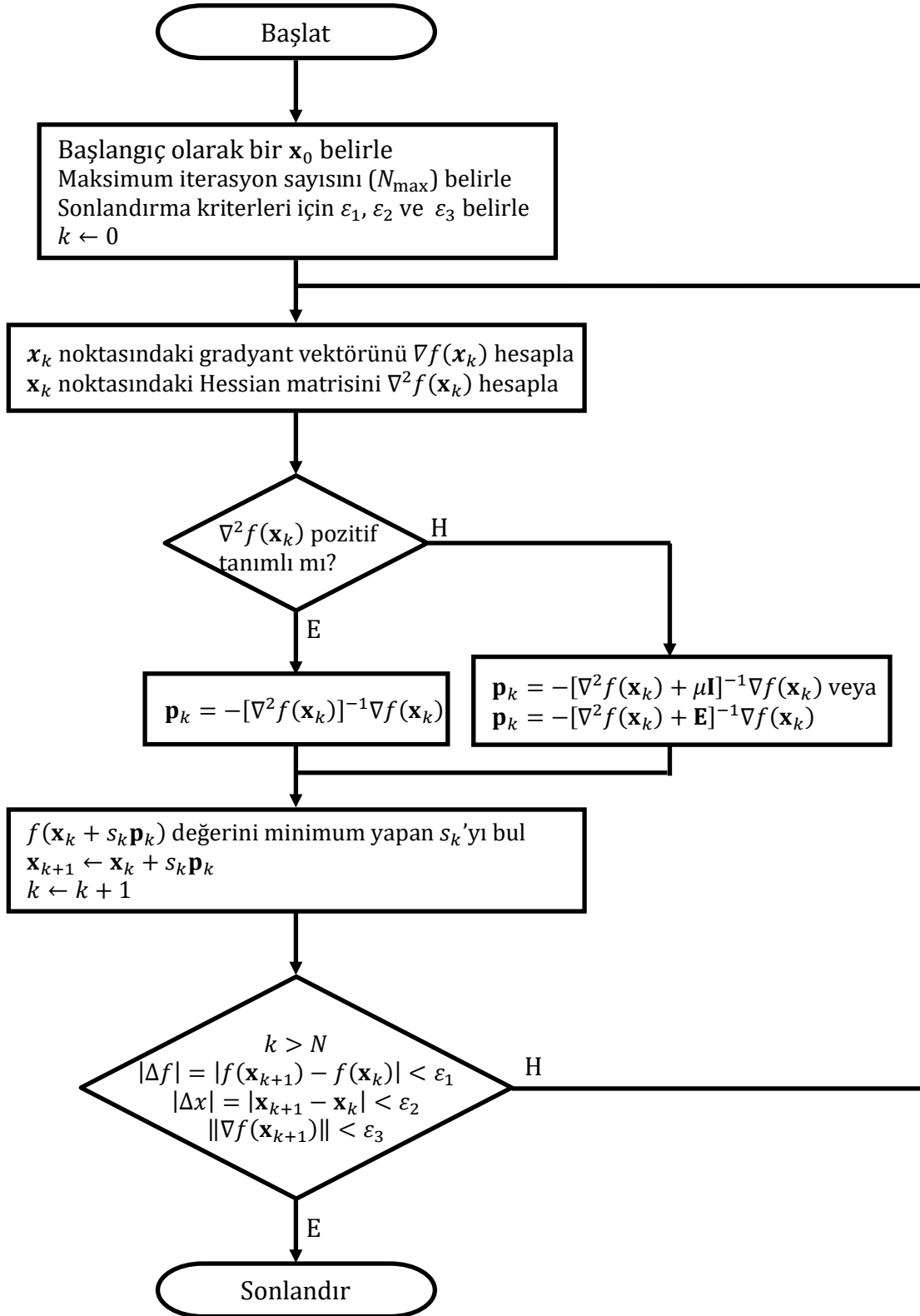
Adım 3 Aşağıdaki şartlardan herhangi biri sağlanıyorsa algoritmayı bitir, sağlanmıyorsa Adım 2'ye git.

C1: $N < k$ maksimum iterasyona ulaşıldı.

C2: $|\Delta f| = |f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k)| < \varepsilon_1$ fonksiyon değişmiyor.

C3: $|\Delta x| = |\mathbf{x}_{k+1} - \mathbf{x}_k| < \varepsilon_2$ değişkenler değişmiyor.

C4: $\|\nabla f(\mathbf{x}_{k+1})\| < \varepsilon_3$ yerel minimuma yakınsadı.



Örnek: Aşağıdaki kısıtsız optimizasyon problemini verilen aralık içerisinde Değiştirilmiş-Newton yöntemiyle çözünüz. Başlangıç noktası olarak $\mathbf{x}_0 = [0 \ 0]^T$ alınız.

$$\min_{\mathbf{x}} f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

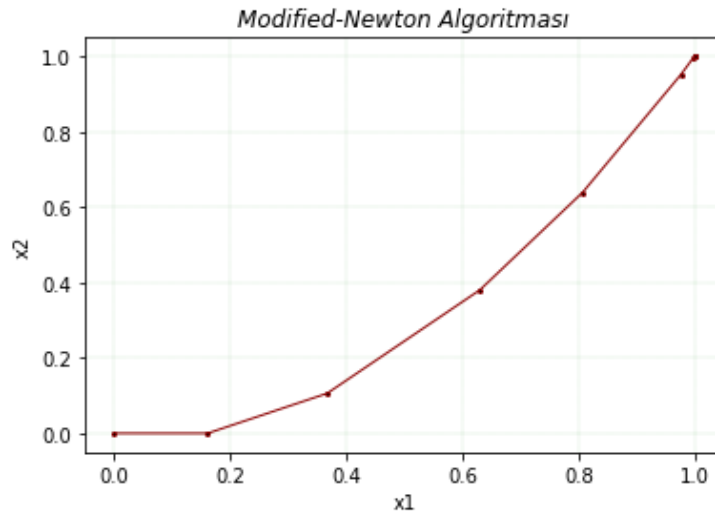
$$-3 \leq \mathbf{x} \leq 3$$

0

Aşağıdaki şekillerden soldakinde, değişkenlere bağlı olarak fonksiyonun değerinin değişimi görülmektedir. Değiştirilmiş-Newton, Altın-Oran ile birlikte uygulandığında algoritma 10 adımda yerel minimuma ($\mathbf{x}^* = [1.0 \ 1.0]^T$) ulaşmaktadır.

k	x_1	x_2	s_k	$f(x_1, x_2)$
1	0.000000	0.000000	1.000000	1.000000
2	0.161338	0.000000	0.161338	0.771110
3	0.367445	0.106206	1.525167	0.483126
4	0.630366	0.380396	2.810586	0.165411
5	0.807456	0.639366	2.104695	0.052999
6	0.976668	0.951711	3.096918	0.001015
7	0.998002	0.996228	1.311235	0.000009
8	1.000029	1.000060	0.969617	0.000000
9	1.000000	1.000000	0.999603	0.000000
10	1.000000	1.000000	1.000012	0.000000

Aşağıdaki şekilde algoritmanın ilerleyişi görülmektedir.



3.5.3. Newton-benzeri (Quasi-Newton) Yöntemler

Bu yöntemler gerçekte Newton yöntemi olmamakla birlikte çözüme yaklaştıkça Newton yöntemine benzediklerinden dolayı bu ismi almışlardır. Bu yöntemlere Değişken Metrik Yöntemler (Variable Metric Methods - VMM) adı da verilmektedir çünkü ilerleme yönünün bulunmasında kullanılan ve başlangıçta genellikle birim matris şeklinde seçilen bir matris (metrik) büyüklüğü her adımda güncellenir ve yerel minimuma yaklaştıkça bu metrik Hessian matrisine benzemeye başlar ve dolayısıyla da yöntem Newton yöntemine benzemeye başlar. VMM yöntemleri çözüme yaklaştıkça Newton yöntemine benzediklerinden bunlara *quasi-Newton* veya *Newton-like* yöntemler de denmektedir. CG yönteminin SD yönteminden

üstünlüğü, bir önceki iterasyondaki yönün de dikkate alınmasından kaynaklanıyordu. VMM yöntemlerinde ise geçmişte kullanılan bütün yönler için ait bilgi *metrik* adı verilen $n \times n$ 'lik bir matriste tutulmaktadır. Arama yönünün bulunmasında kullanılan bu matris her iterasyonda güncellenmektedir. Bu matris için başlangıç olarak simetrik, pozitif-tanımlı bir matris atanır. Bu genellikle birim matristir. Yöntemin yakınsaması için matrisin her iterasyonda bu özelliğini koruması gerekir.

3.5.3.1. Davidon-Fletcher-Powell (DFP) Yöntemi

Bu yöntemde, çözüme ulaşıldığında metrik Hessian matrisinin tersi olur. Algoritma şu şekildedir.

Davidon-Fletcher-Powell Algoritması

Adım 1 Bir başlangıç noktası (\mathbf{x}_0), maksimum iterasyon sayısı (N_{\max}) ve metrik'in ilk değerini \mathbf{M}_0 belirle.

Sonlandırma kriterleri için ε_1 , ε_2 ve ε_3 değerlerini belirle.

$k \leftarrow 0$

Adım 2 \mathbf{x}_k noktasındaki gradyant vektörünü $\nabla f(\mathbf{x}_k)$ hesapla

Eğer \mathbf{M}_k matrisi pozitif-tanımlı ise ilerleme yönü olarak $\mathbf{p}_k = -\mathbf{M}_k \nabla f(\mathbf{x}_k)$ seç.

Eğer \mathbf{M}_k matrisi pozitif-tanımlı değilse o zaman uygun bir matris ilavesi (\mathbf{E} veya $\mu \mathbf{I}$) ile onu pozitif-tanımlı hale getir ve ilerleme yönü olarak $\mathbf{p}_k = -[\mathbf{M}_k + \mu \mathbf{I}] \nabla f(\mathbf{x}_k)$ veya $\mathbf{p}_k = -[\mathbf{M}_k + \mathbf{E}] \nabla f(\mathbf{x}_k)$ seç.

Bir-boyutlu optimizasyon ile $f(\mathbf{x}_k + s_k \mathbf{p}_k)$ değerini minimum yapan adım-aralığını (s_k) bul.

$\mathbf{x}_{k+1} = \mathbf{x}_k + s_k \mathbf{p}_k$ kuralı ile güncellemeyi yap.

$\Delta \mathbf{x} = s_k \mathbf{p}_k$ ve $\mathbf{y} = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$ olmak üzere metriki $\mathbf{M}_{k+1} = \mathbf{M}_k + \frac{\Delta \mathbf{x} [\Delta \mathbf{x}]^T}{[\Delta \mathbf{x}]^T [\mathbf{y}]} - \frac{[\mathbf{M}_k \mathbf{y}] [\mathbf{M}_k \mathbf{y}]^T}{\mathbf{y}^T [\mathbf{M}_k \mathbf{y}]}$ şeklinde güncelle.

$k \leftarrow k + 1$

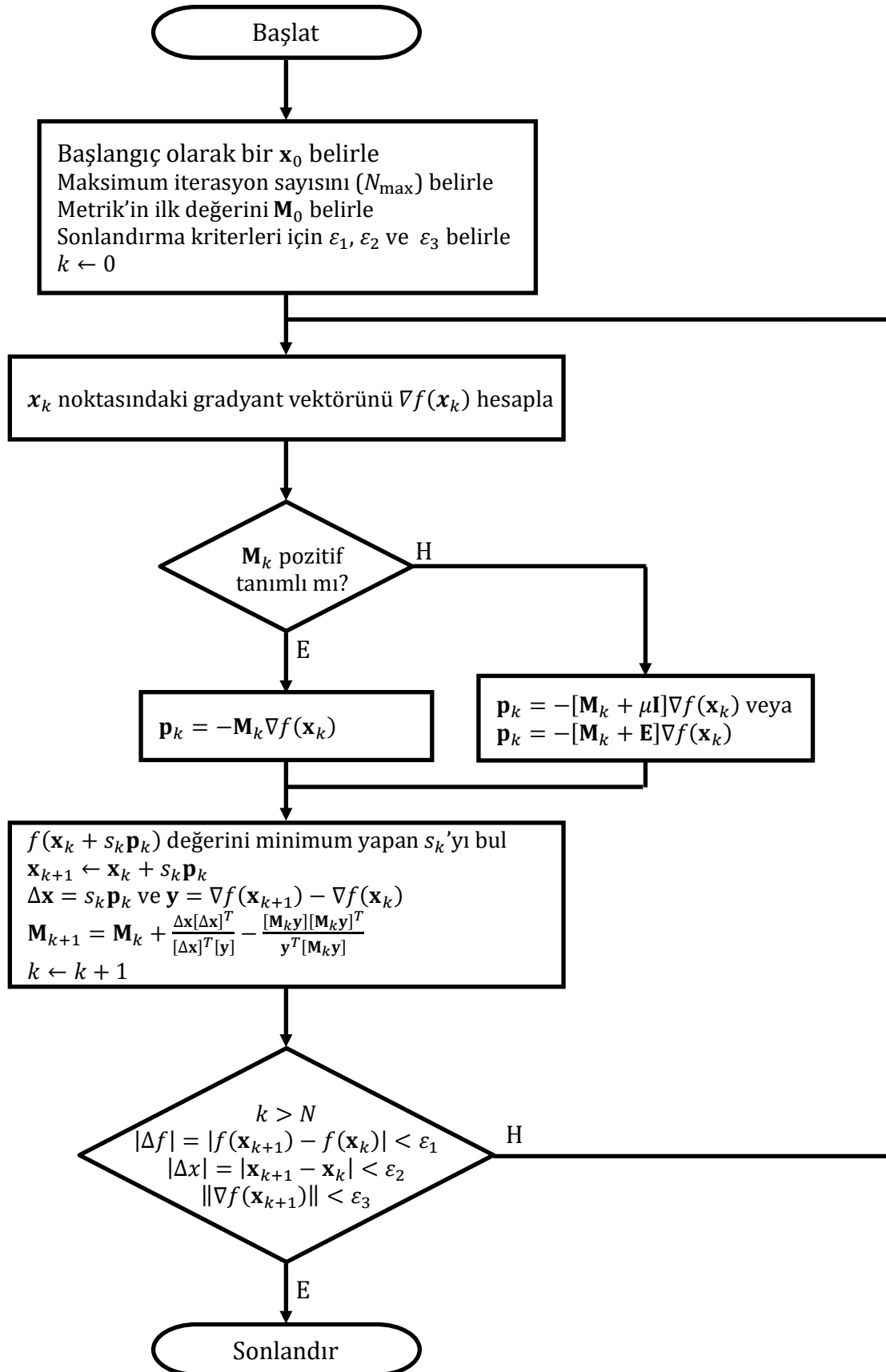
Adım 3 Aşağıdaki şartlardan herhangi biri sağlanıyorsa algoritmayı bitir, sağlanmıyorsa Adım 2'ye git.

C1: $k > N$:maksimum iterasyon sayısına ulaşıldı.

C2: $|\Delta f| = |f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k)| < \varepsilon_1$ fonksiyon değişmiyor.

C3: $|\Delta x| = |\mathbf{x}_{k+1} - \mathbf{x}_k| < \varepsilon_2$ fonksiyon değişmiyor.

C4: $\|\nabla f(\mathbf{x}_{k+1})\| < \varepsilon_3$ algoritma yerel minimuma yakınsadı.



Örnek: Aşağıdaki kısıtsız optimizasyon problemini verilen aralık içerisinde DFP yöntemiyle çözünüz. Başlangıç noktası olarak $\mathbf{x}_0 = [-1.5 \quad -1.5]^T$ ve başlangıç metriki olarak da birim matrisi alınız.

$$\min_{\mathbf{x}} f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \quad 0$$

$$-3 \leq \mathbf{x} \leq 3$$

DFP yöntemi, Altın-Oran ile birlikte uygulandığında yaklaşık 16 adımda yerel minimuma ($\mathbf{x}^* = [1.0 \quad 1.0]^T$) ulaşmaktadır. Bu noktadaki Hessian matrisi $\nabla^2 f(\mathbf{x}^*) = \begin{bmatrix} 802 & -400 \\ -400 & 200 \end{bmatrix}$ iken $[\mathbf{M}_{16}]^{-1} = \begin{bmatrix} 805 & -401 \\ -401 & 200 \end{bmatrix}$ şeklinde olup Hessian matrisine oldukça yakındır.

3.5.3.2. Broydon-Fletcher-Goldfarb-Shanno (BFGS) Yöntemi

VMM yöntemlerinin en popüler olanıdır. DPF'den farkı metriğin güncellenmesi şeklidir. DPF'de metrik Hessian matrisinin tersine yakınsarker, BFGS'de Hessian matrisinin kendisine yakınsar. BFGS daha doğrudan bir yöntemdir. Bu matris için başlangıç olarak simetrik, pozitif-tanımlı bir matris atanır. Bu genellikle birim matristir. Yöntemin yakınsaması için matrisin her iterasyonda bu özelliğini koruması gerekir. Çözüme ulaşıldığında ise bu matris Hessian matrisine eşit olur. Algoritma şu şekildedir.

Broydon-Fletcher-Goldfarb-Shanno Algoritması

Adım 1 Bir başlangıç noktası (\mathbf{x}_0), maksimum iterasyon sayısı (N_{\max}) ve metrik'in ilk değerini \mathbf{M}_0 belirle.

Sonlandırma kriterleri için ε_1 , ε_2 ve ε_3 değerlerini belirle.

$k \leftarrow 0$

Adım 2 \mathbf{x}_k noktasındaki gradyant vektörünü $\nabla f(\mathbf{x}_k)$ hesapla

Eğer \mathbf{M}_k matrisi pozitif-tanımlı ise ilerleme yönü olarak $\mathbf{p}_k = -\mathbf{M}_k^{-1} \nabla f(\mathbf{x}_k)$ seç.

Eğer \mathbf{M}_k matrisi pozitif-tanımlı değilse o zaman uygun bir matris ilavesi (\mathbf{E} veya $\mu \mathbf{I}$) ile onu pozitif-tanımlı hale getir ve ilerleme yönü olarak $\mathbf{p}_k = -[\mathbf{M}_k + \mu \mathbf{I}]^{-1} \nabla f(\mathbf{x}_k)$ veya $\mathbf{p}_k = -[\mathbf{M}_k + \mathbf{E}]^{-1} \nabla f(\mathbf{x}_k)$ seç.

Bir-boyutlu optimizasyon ile $f(\mathbf{x}_k + s_k \mathbf{p}_k)$ değerini minimum yapan adım-aralığını (s_k) bul.

$\mathbf{x}_{k+1} = \mathbf{x}_k + s_k \mathbf{p}_k$ kuralı ile güncellemeyi yap.

$\Delta \mathbf{x} = s_k \mathbf{p}_k$ ve $\mathbf{y} = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$ olmak üzere metriki $\mathbf{M}_{k+1} = \mathbf{M}_k + \frac{\mathbf{y} \mathbf{y}^T}{\mathbf{y}^T \Delta \mathbf{x}} - \frac{\mathbf{M}_k \Delta \mathbf{x} [\Delta \mathbf{x}]^T \mathbf{M}_k}{[\Delta \mathbf{x}]^T \mathbf{M}_k \Delta \mathbf{x}}$ şeklinde güncelle.

$k \leftarrow k + 1$

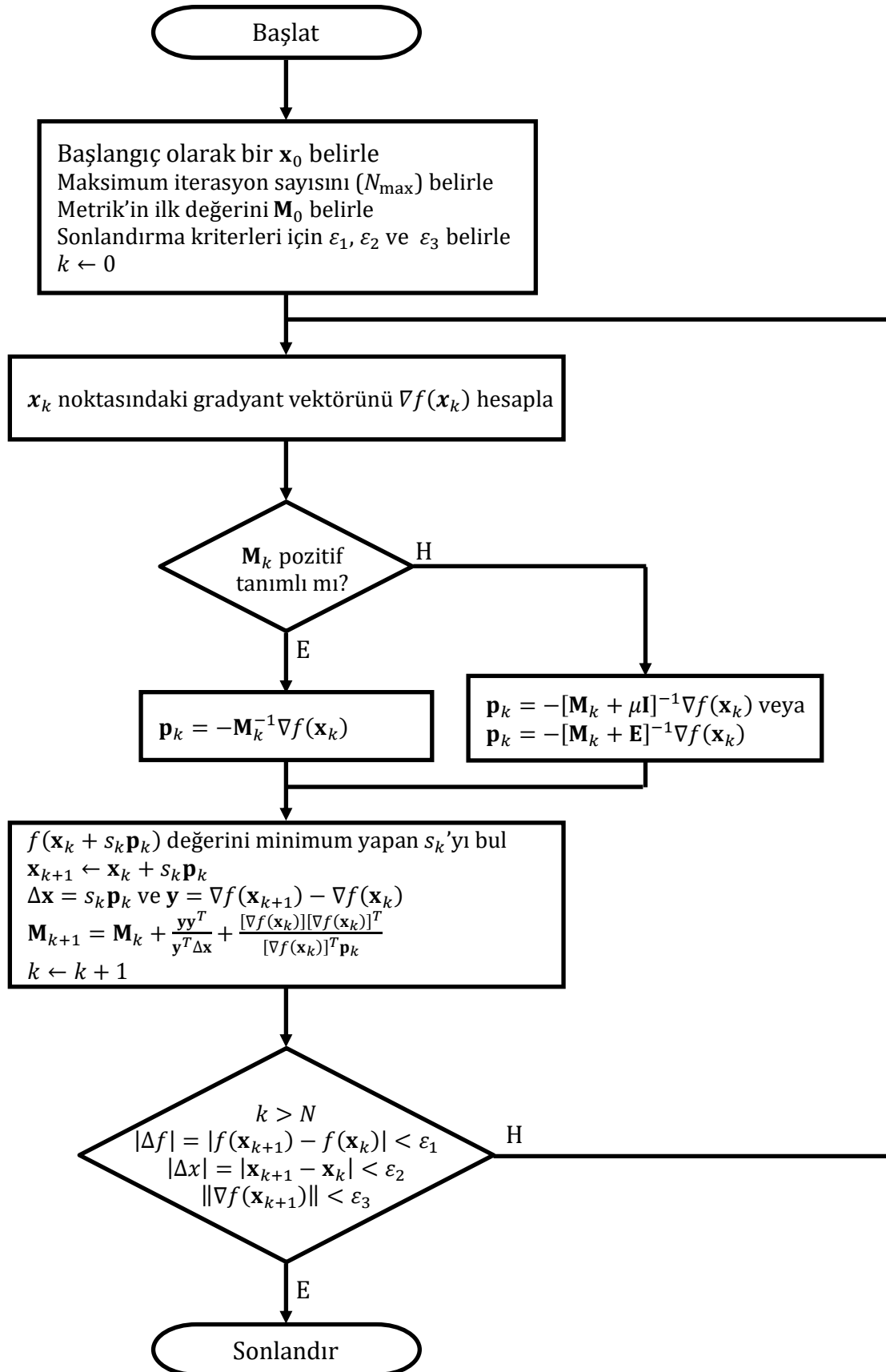
Adım 3 Aşağıdaki şartlardan herhangi biri sağlanıyorsa algoritmayı bitir, sağlanmıyorsa Adım 2'ye git.

C1: $k > N$:maksimum iterasyon sayısına ulaşıldı.

C2: $|\Delta f| = |f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k)| < \varepsilon_1$ fonksiyon değişmiyor.

C3: $|\Delta x| = |\mathbf{x}_{k+1} - \mathbf{x}_k| < \varepsilon_2$ fonksiyon değişmiyor.

C4: $\|\nabla f(\mathbf{x}_{k+1})\| < \varepsilon_3$ algoritma yerel minimuma yakınsadı.



Örnek: Aşağıdaki kısıtsız optimizasyon problemini verilen aralık içerisinde BFGS yöntemiyle çözünüz. Başlangıç noktası olarak $\mathbf{x}_0 = [-1.5 \quad -1.5]^T$ ve başlangıç metriki olarak da birim matrisi alınız.

$$\min_{\mathbf{x}} f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \quad 0$$

$$-3 \leq \mathbf{x} \leq 3$$

BFGS yöntemi, Altın-Oran ile birlikte uygulandığında yaklaşık 57 adımda yerel minimuma ($\mathbf{x}^* = [1.0 \quad 1.0]^T$) ulaşmaktadır. Bu noktadaki Hessian matrisi $\nabla^2 f(\mathbf{x}^*) = \begin{bmatrix} 802 & -400 \\ -400 & 200 \end{bmatrix}$ iken $\mathbf{M}_{57} = \begin{bmatrix} 802 & -400 \\ -400 & 200 \end{bmatrix}$ şeklinde olup Hessian matrisine oldukça yakındır.

3.5.4 İkinci-dereceden Yaklaşık Yöntemler

Bu yöntemler, sadece birinci-dereceden türev bilgisini kullanarak Hessian matrisini belli bir yaklaşıklıkla elde edip ikinci-dereceden bir yakınsama sağlamaya çalışırlar. Eğer minimize edilecek $f(\mathbf{x})$ fonksiyonu belli sayıda karelerin toplamı şeklindeyse, yani,

$$\begin{aligned} f(\mathbf{x}) &= e_1^2(\mathbf{x}) + e_2^2(\mathbf{x}) + \dots + e_N^2(\mathbf{x}) \\ &= \sum_{i=1}^N e_i^2(\mathbf{x}) \quad 0 \\ &= \mathbf{e}^T(\mathbf{x})\mathbf{e}(\mathbf{x}) \end{aligned}$$

ki burada $\mathbf{e}(\mathbf{x}) = [e_1(\mathbf{x}) \quad e_2(\mathbf{x}) \quad \dots \quad e_N(\mathbf{x})]^T$ şeklinde bir vektördür, bu durumda $f(\mathbf{x})$ fonksiyonunun gradyant vektörünün j inci elemanı,

$$\begin{aligned} [\nabla f(\mathbf{x})]_j &= \frac{\partial f(\mathbf{x})}{\partial x_j} \\ &= 2 \sum_{k=1}^N e_k(\mathbf{x}) \frac{\partial e_k(\mathbf{x})}{\partial x_j} \quad 0 \end{aligned}$$

şeklinde olmaktadır ki buna göre gradyant vektörü,

$$\nabla f(\mathbf{x}) = 2\mathbf{J}^T(\mathbf{x})\mathbf{e}(\mathbf{x}) \quad 0$$

şeklinde yazılabilir, burada $\mathbf{J}(\mathbf{x})$ matrisi aşağıdaki gibi Jacobian matrisidir:

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} \frac{\partial e_1(\mathbf{x})}{\partial x_1} & \frac{\partial e_1(\mathbf{x})}{\partial x_2} & \dots & \frac{\partial e_1(\mathbf{x})}{\partial x_n} \\ \frac{\partial e_2(\mathbf{x})}{\partial x_1} & \frac{\partial e_2(\mathbf{x})}{\partial x_2} & \dots & \frac{\partial e_2(\mathbf{x})}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_N(\mathbf{x})}{\partial x_1} & \frac{\partial e_N(\mathbf{x})}{\partial x_2} & \dots & \frac{\partial e_N(\mathbf{x})}{\partial x_n} \end{bmatrix} \quad 0$$

Benzer şekilde Hessian matrisinin $i^{\text{inci}}j^{\text{inci}}$ elemanı,

$$\begin{aligned} [\nabla^2 f(\mathbf{x})]_{ij} &= \frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j} \\ &= 2 \sum_{k=1}^N \left\{ \frac{\partial e_k(\mathbf{x})}{\partial x_i} \frac{\partial e_k(\mathbf{x})}{\partial x_j} + e_k(\mathbf{x}) \frac{\partial^2 e_k(\mathbf{x})}{\partial x_i \partial x_j} \right\} \end{aligned} \quad 0$$

şeklindedir ve Jacobian matrisi kullanılarak,

$$\nabla^2 f(\mathbf{x}) = 2\mathbf{J}^T(\mathbf{x})\mathbf{J}(\mathbf{x}) + 2\mathbf{S}(\mathbf{x}) \quad 0$$

şeklinde yazılabilir ki burada $\mathbf{S}(\mathbf{x})$ matrisinin $i^{\text{inci}}j^{\text{inci}}$ elemanı

$$[\mathbf{S}(\mathbf{x})]_{ij} = \sum_{k=1}^N e_k(\mathbf{x}) \frac{\partial^2 e_k(\mathbf{x})}{\partial x_i \partial x_j} \quad 0$$

ile verilmektedir. Eğer $\mathbf{S}(\mathbf{x})$ matrisinin elemanlarının yeterince küçük olduğu varsayılırsa, o zaman Hessian matrisi

$$\nabla^2 f(\mathbf{x}) \cong 2\mathbf{J}^T(\mathbf{x})\mathbf{J}(\mathbf{x}) \quad 0$$

şeklinde yaklaşık olarak yazılabilir. Buradan görüleceği gibi ikinci-dereceden türev bilgisi içeren Hessian matrisi, birinci-dereceden türev bilgisi içeren Jacobian matrisi yardımıyla belli bir hata ile bulunabilir.

3.5.4.1 Gauss-Newton (GN) Yöntemi

Hatırlanacağı gibi Newton yönteminde ilerleme yönü $\mathbf{p}_k = -[\nabla^2 f(\mathbf{x}_k)]^{-1} \nabla f(\mathbf{x}_k)$ şeklindeydi. Eğer bu yönü bulmak için gerekli büyüklükleri Jacobian matrisi karşılıkları kullanılırsa,

$$\begin{aligned}
\mathbf{p}_k &= -[\nabla^2 f(\mathbf{x}_k)]^{-1} \nabla f(\mathbf{x}_k) \\
&\cong -[2\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k)]^{-1} 2\mathbf{J}^T(\mathbf{x}_k)\mathbf{e}(\mathbf{x}_k) \\
&\cong -[\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k)]^{-1} \mathbf{J}^T(\mathbf{x}_k)\mathbf{e}(\mathbf{x}_k)
\end{aligned}
\tag{0}$$

şeklinde bir ilerleme yönü bulunur ki bu yönü kullanan yöntem *Gauss-Newton* yöntemi denir. Ancak bu yöntem pratikte çok fazla tercih edilmez çünkü uygulama sırasında her iterasyonda $\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k)$ matrisinin tersinin alınabiliyor olması gerekir ancak zaman zaman bu matris tekil olabilmektedir ki bu durumda bu yöntem uygulanamaz hale gelir. Bu durumu ortadan kaldırmak için Levenberg-Marquardt (LM) Yöntemi önerilmiştir.

3.5.4.2 Levenberg-Marquardt (LM) Yöntemi

Gauss-Newton yönteminde karşılaşılabilecek bir problem $\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k)$ matrisinin tersinin olmamasıdır. O yüzden, bu matrise $\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k) + \mu_k \mathbf{I}$ matrisi pozitif-tanımlı olacak şekilde bir $\mu_k \mathbf{I}$ terimi ilave edilir ki bu durumda ilerleme yönü aşağıdaki gibi Levenberg-Marquardt yönüne dönüşür:

$$\mathbf{p}_k = -[\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k) + \mu_k \mathbf{I}]^{-1} \mathbf{J}^T(\mathbf{x}_k)\mathbf{e}(\mathbf{x}_k). \tag{0}$$

Burada dikkat edilirse Levenberg-Marquardt yönündeki μ_k büyüklüğü her iterasyonda değişmektedir. μ_k büyüklüğünün ayarlanması aşağıdaki Levenberg-Marquardt Algoritması içerisinde gerçekleştirilir:

Levenberg-Marquardt Algoritması

Adım 1 Bir başlangıç noktası (\mathbf{x}_0), μ_0 başlangıç değeri ve maksimum iterasyon sayısı (N_{\max}) belirle.

μ değerinin değişimi için $\mu_{\text{scal}} > 0$, μ_{\min} ve μ_{\max} belirle.

Sonlandırma kriterleri için ε_1 , ε_2 ve ε_3 değerlerini belirle.

$k \leftarrow 0$

Adım 2 \mathbf{x}_k noktasındaki fonksiyon değerini $f(\mathbf{x}_k)$, hata vektörünü $\mathbf{e}(\mathbf{x}_k)$ ve Jacobian matrisini $\mathbf{J}(\mathbf{x}_k)$ hesapla.

Adım 3 Aday ilerleme yönü $\mathbf{z}_k = -[\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k) + \mu_k \mathbf{I}]^{-1} \mathbf{J}^T(\mathbf{x}_k)\mathbf{e}(\mathbf{x}_k)$.

Eğer $f(\mathbf{x}_k + \mathbf{z}_k) < f(\mathbf{x}_k)$ ise güncelle:

$\mathbf{p}_k \leftarrow \mathbf{z}_k$

$f(\mathbf{x}_k + s_k \mathbf{p}_k)$ değerini minimum yapan s_k 'yi bul

$\mathbf{x}_{k+1} = \mathbf{x}_k + s_k \mathbf{p}_k$

$\mu_k \leftarrow \mu_k / \mu_{\text{scal}}$

Adım 5'e git.

Adım 4 Eğer $f(\mathbf{x}_k + \mathbf{z}_k) > f(\mathbf{x}_k)$ ise

$\mu_k \leftarrow \mu_k \cdot \mu_{\text{scal}}$ yap.

Eğer $\mu_k < \mu_{\text{max}}$ ve $\mu_{\text{min}} < \mu_k$ ise Adım 3'e git, aksi halde Adım 5'e git.

Adım 5 $k \leftarrow k + 1$.

Adım 6 Aşağıdaki şartlardan herhangi biri sağlanıyorsa algoritmayı bitir, sağlanmıyorsa Adım 2'ye git.

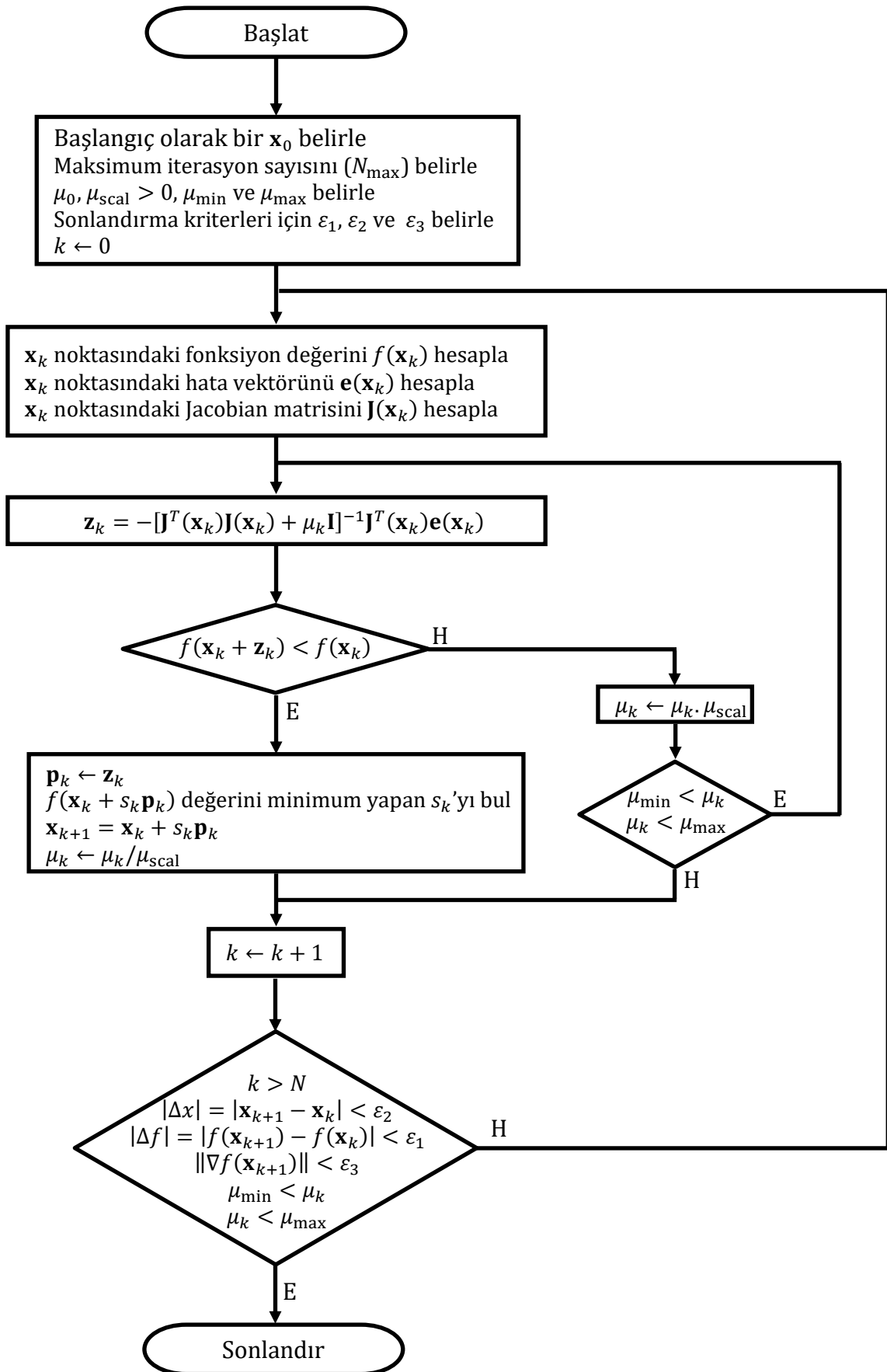
C1: $N < k$ maksimum iterasyona ulaşıldı.

C2: $|\Delta f| = |f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k)| < \varepsilon_1$ fonksiyon değişmiyor.

C3: $|\Delta x| = |\mathbf{x}_{k+1} - \mathbf{x}_k| < \varepsilon_2$ değişkenler değişmiyor.

C4: $\|\nabla f(\mathbf{x}_{k+1})\| < \varepsilon_3$ yerel minimuma yakınsadı.

C5: $\mu_{\text{min}} < \mu_k$ veya $\mu_k < \mu_{\text{max}}$ yerel minimuma yakınsadı.



Bu algoritmadan da görüleceği gibi fonksiyonu azaltan uygun bir ilerleme yönü bulunduğunda μ_k büyüklüğü azaltılarak \mathbf{p}_k ilerleme yönü Gauss-Newton (GN) yönüne dönüşür ki bu durumda yakınsama hızlanır. Diğer taraftan, fonksiyonu azaltan uygun bir ilerleme yönünün bulunamaması halinde, μ_k değeri fonksiyonu azaltan uygun bir ilerleme yönü bulunana kadar artırılır ve böylece \mathbf{p}_k ilerleme yönü Dik-İniş (SD) yönüne benzemeye başlar ki bu durumda yakınsamanın yavaşlaması pahasına da olsa fonksiyonun azalması sağlanmaya çalışılır. sonuç olarak, LM algoritması, yavaş ama güvenilir SD yönü ile hızlı ama az güvenilir GN yönü arasında uygun bir geçiş sağlar. Bu da LM algoritmasının en güçlü yanıdır.

Örnek: Aşağıdaki kısıtsız optimizasyon problemini verilen aralık içerisinde Levenberg-Marquardt yöntemiyle çözünüz. Başlangıç noktası olarak $\mathbf{x}_0 = [0 \ 0]^T$ alınız.

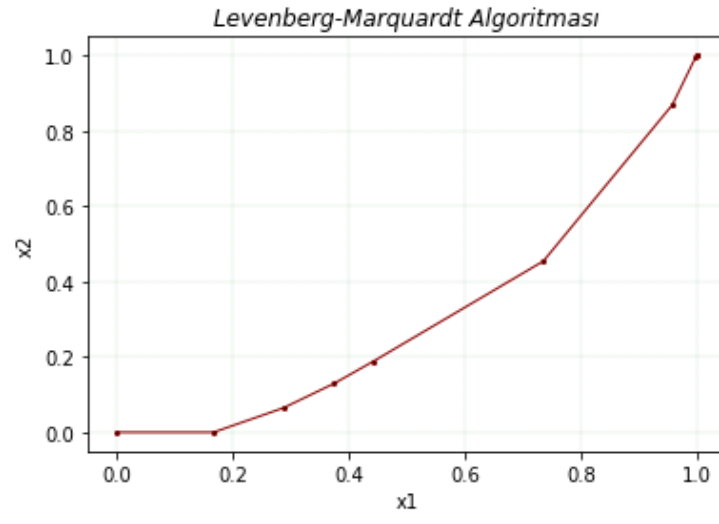
$$\min_{\mathbf{x}} f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \quad 0$$

$$-3 \leq \mathbf{x} \leq 3$$

Aşağıdaki şekillerden soldakininde, değişkenlere bağlı olarak fonksiyonun değerinin değişimi görülmektedir. Değiştirilmiş-Newton, Altın-Oran ile birlikte uygulandığında algoritma 10 adımda yerel minimuma ($\mathbf{x}^* = [1.0 \ 1.0]^T$) ulaşmaktadır.

k	x_1	x_2	s_k	$f(x_1, x_2)$
1	0.000000	0.000000	1.000000	1.000000
2	0.166667	0.000000	1.000000	0.771605
3	0.287547	0.064830	1.000000	0.539464
4	0.375147	0.129812	1.000000	0.402372
5	0.442633	0.188439	1.000000	0.316261
6	0.735808	0.454133	1.000000	0.831571
7	0.958366	0.868726	1.000000	0.249136
8	0.998584	0.995545	1.000000	0.000266
9	0.999995	0.999988	1.000000	0.000000
10	1.000000	1.000000	1.000000	0.000000

Aşağıdaki şekilde algoritmanın ilerleyişi görülmektedir.



BÖLÜM 4. MODELLEME ve TAHMİN

4.1. Modelleme Kavramı

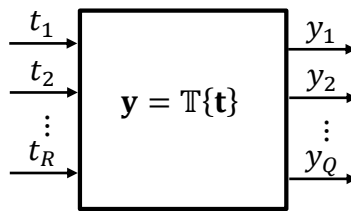
Modelleme kavramı, gerçek dünyadaki bir sürecin davranışının analitik olarak ifade edilemediği, ya da başka bir deyişle temel yasalarla ifade edilemediği durumlarda başvurulan bir yöntemdir. Burada süreç ile, biyolojik, fiziksel, kimyasal, elektriksel, mekanik, meteorolojik, sosyal, finansal vs her türlü dinamik sistem kastedilmektedir. Gerçek dünyada bu tarzda pek çok sistem bulunmaktadır. Örneğin, hava durumu analitik olarak ifade edilemeyecek kadar karmaşık dinamik bir sistemdir ve dolayısıyla modellenmesi gerekir. Benzer şekilde, borsa da içinde çok sayıda değişken barındıran karmaşık bir sistemdir. Bunlar kadar karmaşık olmasa da diğer alanlarda da karşımıza pek çok sistem çıkmaktadır ki bunların analitik olarak ifade edilmeleri mümkün değildir. O yüzden, gerekli durumlarda, bu tip süreçlerden yeterli miktarda veri toplanıp bu verilerle güvenilir bir model elde etme yoluna gidilir.

Veriler, modellenen süreçte doğasına bağlı olarak en genel halde karşımıza iki değişik biçimde çıkmaktadır:

4.2. Veri Tipleri

4.2.1 Giriş-Çıkış Verisi

Giriş-çıkış verisi, genellikle dinamik bir sistemin modellenmesinde karşımıza çıkan bir veri tipidir. En genel halde bir sistemi aşağıdaki gibi bir şekilde gösterelim.



Burada, $i = 1, \dots, R$ olmak üzere t_i 'ler sistemin giriş işaretlerini temsil etmektedir ve daha kolay bir gösterilim için tüm girişler $\mathbf{t} = [t_1 \ t_2 \ \dots \ t_R]^T$ şeklinde bir vektörle gösterilmektedir. Benzer şekilde, $i = 1, \dots, Q$ olmak üzere y_i 'ler de sistemin çıkış işaretlerini temsil etmekte olup daha kolay bir gösterilim için tüm çıkışlar $\mathbf{y} = [y_1 \ y_2 \ \dots \ y_Q]^T$ şeklinde bir vektörle gösterilmektedir. Diğer taraftan, $T\{\}$ operatörü ise sistemin giriş çıkış ilişkisini temsil etmektedir. Yani, \mathbf{t} gibi bir girişe karşı sistemin \mathbf{y} gibi bir çıkışı nasıl üreteceğini ifade etmektedir. $T\{\}$ operatörü sistemin doğasına bağlı olarak bir diferansiyel denklem veya bir fark denklemi olabilir. Belli bir fiziksel süreç için modelleme ihtiyacını ortaya çıkaran sorun

ise bu $\mathbb{T}\{\}$ operatörünün matematiksel ifadesinin bilinmemesidir. Modelleme yapabilmek için sistemin giriş ve çıkış büyüklükleri belli aralıklarla ölçülerek kaydedilir ve aşağıdaki tablodakine benzer bir veri seti elde edilir.

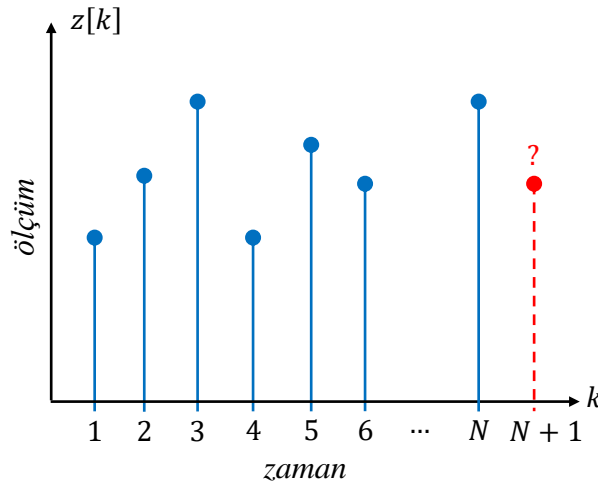
$\mathcal{D}_{\text{MISO}}$	Giriş verisi				Çıkış verisi
i	\mathbf{t}_i				y_i
	t_{i1}	t_{i2}	...	t_{iR}	
1	t_{11}	t_{12}	...	t_{1R}	y_1
2	t_{21}	t_{22}	...	t_{2R}	y_2
\vdots	\vdots				\vdots
N	t_{N1}	t_{N2}	...	t_{NR}	y_N

$\mathcal{D}_{\text{MIMO}}$	Giriş verisi				Çıkış verisi			
i	\mathbf{t}_i				\mathbf{y}_i			
	t_{i1}	t_{i2}	...	t_{iR}	y_{i1}	y_{i2}	...	y_{iQ}
1	t_{11}	t_{12}	...	t_{1R}	y_{11}	y_{12}	...	y_{1Q}
2	t_{21}	t_{22}	...	t_{2R}	y_{21}	y_{22}	...	y_{2Q}
\vdots	\vdots				\vdots			
N	t_{N1}	t_{N2}	...	t_{NR}	y_{N1}	y_{N2}	...	y_{NQ}

Burada N toplam veri sayısıdır. Bu veri seti $\mathcal{D}_{\text{MIMO}}$ ile gösterilmiştir ki burada MIMO ifadesi veri setinin çok-girişli çok-çıkışlı (Multiple-Input Multiple-Output) olduğunu göstermektedir.

4.2.2 Zaman Serisi

Zaman serisi, genellikle belli bir girişin ve çıkışın olmadığı ve sadece gözlenebilen büyüklüklerin olduğu süreçlerde karşımıza çıkmaktadır. Aşağıdaki şekli ele alalım: burada $z[k]$ büyüklüğü k gözlem anında sürecin gözlenen büyüklüğünün değeridir.



Sürecin gözlenen bu büyüklüğüne örnek olarak borsadaki bir kağıdın günlük değeri, bir barajın su seviyesinin haftalık değeri, kimyasal bir süreçteki bir elenmanın saatlik miktarı veya biyolojik bir süreçteki bir canlı bakteri sayısının saatlik değerleri vb olabilir. Süreçten eşit aralıklarla N adet gözlem yapılarak henüz gözlenemeyen $N + 1$ anındaki değerin tahmin edilmesi problemi zaman serilerinde en çok karşılaşılan problemidir. Bu zaman serisini, $k=1, \dots, N$ için $z[k] \in \mathbb{R}$ olmak üzere $\mathcal{D}_{TS} = \{z[k]\}_{k=1}^N$ şeklinde göstermek mümkündür. Burada k zaman indeksini göstermektedir. Bu zaman serisi herhangi bir büyüklüğün herhangi bir zaman aralığında yapılmış ölçümlerle elde edilmiş değerler olabilir. Bu zaman serisini modelleme problemi aşağıdaki tablodaki gibi bir giriş-çıkış veri setine dönüştürülerek bir çok-boyutlu modelleme problemi olarak çözülebilir.

\mathcal{D}_{TS}	Giriş verisi				Çıkış verisi
i	\mathbf{t}_i				y_i
	t_{i1}	t_{i2}	...	t_{iR}	
1	$z[1]$	$z[2]$...	$z[R]$	$z[R + 1]$
2	$z[2]$	$z[3]$...	$z[R + 1]$	$z[R + 2]$
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
$N - R$	$z[N - R]$	$z[N - R + 1]$...	$z[N - 1]$	$z[N]$

Buradaki R parametresi, modelde kaç tane giriş olacağını ya da başka bir deyişle zaman serisinin belli bir andaki çıkışının o andan itibaren kaç adım öncesine kadar bağlı olduğunu göstermektedir.

Sonuç olarak, her iki tipteki veriler bir giriş-çıkış veri setine dönüştürülebilmektedir. Dolayısıyla bu kısımda, verilen bir giriş-çıkış veri setini en iyi şekilde temsil edecek modelin bulunması problemi üzerinde durulacaktır.

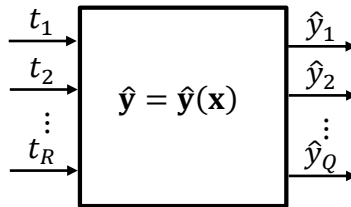
4.3. Modelleme ve Model Seçme

Aşağıdaki tabloda görüldüğü gibi, $\mathbf{t}_i \in \mathbb{R}^R$ ve $\mathbf{y}_i \in \mathbb{R}^Q$ olmak üzere $\mathcal{D}_{\text{MIMO}} = \{\mathbf{t}_i, \mathbf{y}_i\}_{i=1}^N$ şeklinde verilen bir MIMO veriyi ele alalım:

$\mathcal{D}_{\text{MISO}}$	Giriş Verisi				Çıkış Verisi				Model Çıkışı			
i	\mathbf{t}_i				\mathbf{y}_i				$\hat{\mathbf{y}}_i$			
	t_{i1}	t_{i2}	...	t_{iR}	y_{i1}	y_{i2}	...	y_{iQ}	\hat{y}_{i1}	\hat{y}_{i2}	...	\hat{y}_{iQ}
1	t_{11}	t_{12}	...	t_{1R}	y_{11}	y_{12}	...	y_{1Q}	\hat{y}_{11}	\hat{y}_{12}	...	\hat{y}_{1Q}
2	t_{21}	t_{22}	...	t_{2R}	y_{21}	y_{22}	...	y_{2Q}	\hat{y}_{21}	\hat{y}_{22}	...	\hat{y}_{2Q}
\vdots	\vdots	\vdots	\ddots	\vdots	\vdots	\vdots	\ddots	\vdots	\vdots	\vdots	\ddots	\vdots
N	t_{N1}	t_{N2}	...	t_{NR}	y_{N1}	y_{N2}	...	y_{NQ}	\hat{y}_{N1}	\hat{y}_{N2}	...	\hat{y}_{NQ}

$\mathcal{D}_{\text{MIMO}}$	Giriş Verisi				Çıkış Verisi				Model Çıkışı			
i	\mathbf{t}_i				\mathbf{y}_i				$\hat{\mathbf{y}}_i$			
	t_{i1}	t_{i2}	...	t_{iR}	y_{i1}	y_{i2}	...	y_{iQ}	\hat{y}_{i1}	\hat{y}_{i2}	...	\hat{y}_{iQ}
1	t_{11}	t_{12}	...	t_{1R}	y_{11}	y_{12}	...	y_{1Q}	\hat{y}_{11}	\hat{y}_{12}	...	\hat{y}_{1Q}
2	t_{21}	t_{22}	...	t_{2R}	y_{21}	y_{22}	...	y_{2Q}	\hat{y}_{21}	\hat{y}_{22}	...	\hat{y}_{2Q}
\vdots	\vdots	\vdots	\ddots	\vdots	\vdots	\vdots	\ddots	\vdots	\vdots	\vdots	\ddots	\vdots
N	t_{N1}	t_{N2}	...	t_{NR}	y_{N1}	y_{N2}	...	y_{NQ}	\hat{y}_{N1}	\hat{y}_{N2}	...	\hat{y}_{NQ}

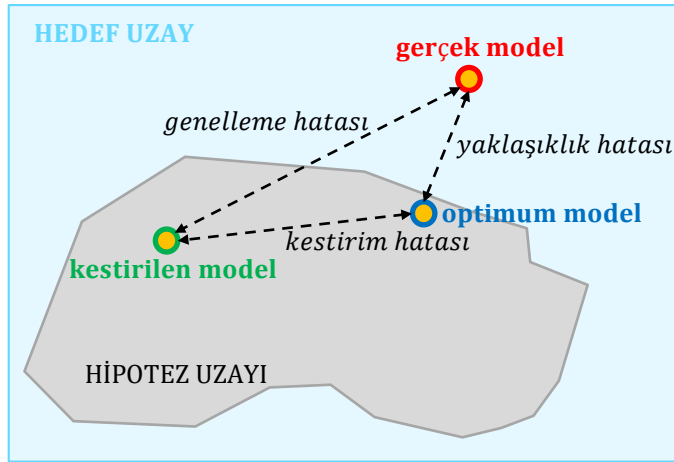
Bu veriler, $\hat{\mathbf{y}} = \hat{\mathbf{y}}(\mathbf{x})$ gibi bir yapı ile modellenenecektir. Öncelikle elde etmek istediğimiz modelin yapısına bir göz atalım: En genel halde model aşağıdaki şekildeki gibi bir gösterilebilir.



Burada, $i = 1, \dots, R$ olmak üzere t_i 'ler modelin giriş işaretlerini temsil etmektedir ve daha kolay bir gösterilim için tüm girişler $\mathbf{t} = [t_1 \ t_2 \ \dots \ t_R]^T$ şeklinde bir vektörle gösterilmektedir. Benzer şekilde, $i = 1, \dots, Q$ olmak üzere \hat{y}_i 'ler de modelin çıkış işaretlerini temsil etmekte olup daha kolay bir gösterilim için tüm çıkışlar $\hat{\mathbf{y}} = [\hat{y}_1 \ \hat{y}_2 \ \dots \ \hat{y}_Q]^T$ şeklinde bir vektörle gösterilmektedir. Diğer taraftan, $\hat{\mathbf{y}}(\cdot)$ operatörü ise modelin giriş-çıkış ilişkisini temsil etmektedir ve buradaki \mathbf{x} vektörü de modelin parametrelerini temsil eden parametre

vektörüdür. Modelleme problemi, bu giriş-çıkış verisini en iyi şekilde temsil edecek modelin giriş-çıkış ilişkisinin ($\hat{y}(\cdot)$) ve bu ilişkideki parametrelerin (\mathbf{x}) belirlenmesidir.

Verilen bir giriş-çıkış veri setini en iyi temsil eden model dendiğinde aklımıza “verileri iyi bir genelleme yaparak giriş-çıkış ilişkisini en iyi şekilde öğrenen en basit model” gelmelidir. Aslında bu durum aşağıdaki şekilde daha iyi anlaşılabilir.



Buna göre, gözlemlenen sürecin gerçek modeli hedef uzayı (target space) denilen bir uzayda bulunmaktadır. Bu sürecin zamanda belli aralıklarla gözlenerek elde edilen verilerin ışığı altında bir model (hipotez) geliştirilecektir ancak bu model sadece hipotez uzayında (hypothesis space) yer alabilir. Hipotez uzayı ise hedef uzayın bir alt-uzayıdır ve maalesef gerçek model hipotez uzayının çok dışındadır. Bunun nedeni süreçten kısıtlı sayıda veri toplanması ve model geliştirilirken yapılan bazı varsayımlar ve yaklaşıklıklardır. Neticede, elde edilebilecek en optimum model bile gerçek model olmayacak ve gerçek modelle arasında “yaklaşıklık hatası” bulunan bir model olacaktır. Hipotez uzayında bulunan modellerden optimum olanı bulmak için geliştirilmiş yöntemler (model selection methods) kullanılsa bile neticede elde edilen model (kestirilen model) optimum modelden farklı olacaktır. Kestirilen model ile optimum model arasındaki hataya da “kestirim hatası” denmektedir. En iyi modeli seçmek için kullanılacak yöntemlerle genelleme hatası en aza indirilmeye çalışılacaktır.

Yapısal Risk Minimizasyonu (Structural Risk Minimization) prensibine göre bir modelin yaptığı genelleme hatası ($R[\hat{y}]$) şu şekilde tanımlanmaktadır:

$$R[\hat{y}] \leq R_{emp}[\hat{y}] + \Omega\left(\frac{N}{h}\right) \quad 0$$

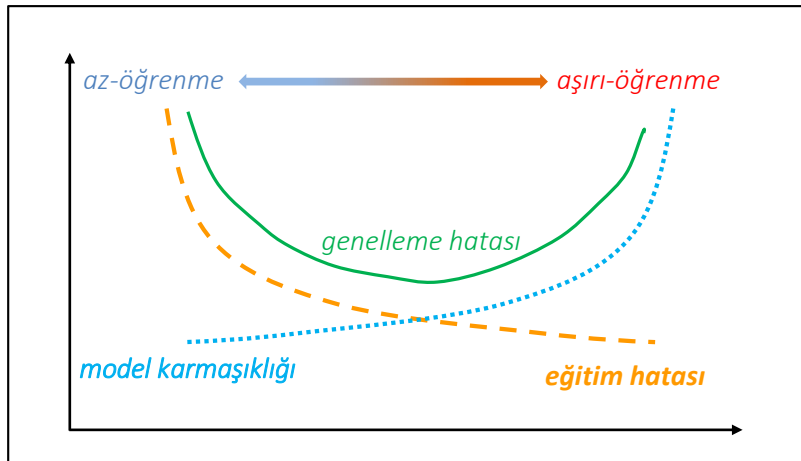
burada \hat{y} modeli temsil etmektedir ki gerçekte $\hat{y} = \hat{y}(\mathbf{x})$ gibi bir yapıdır ve buradaki \mathbf{x} vektörü modelin parametrelerini içeren optimize edilecek parametre vektörüdür. $R_{emp}[\hat{y}]$ ise

$$R_{emp}[\hat{y}] = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 \quad 0$$

şeklinde tanımlanan “Amprık Risk (Emprical Risk)” olup modelin yaptığı eğitim hatasıdır ve limit durumunda

$$\lim_{N \rightarrow \infty} R_{emp}[\hat{y}] = R[\hat{y}] \quad 0$$

olduğu varsayılmaktadır. Diğer taraftan, $\Omega\left(\frac{N}{h}\right)$ ise modelin karmaşıklığını temsil etmektedir ki buradaki N veri sayısıdır, h ise Vapnik-Chervonenkis boyutu (VC dimension) adı verilen ve modelin öğrenme kapasitesini temsil eden bir büyüklüktür. $R[\hat{y}]$ büyüklüğünü doğrudan minimize etmek mümkün değildir. Onun yerine $R_{emp}[\hat{y}] + \Omega\left(\frac{N}{h}\right)$ toplamı minimize edilmeye çalışılır ama bu toplamdaki iki terim aşağıdaki şekilde görüldüğü gibi birbiriyle çelişmektedir. Yani eldeki veriler için $R_{emp}[\hat{y}]$ büyüklüğünü en az yapacak modelin karmaşıklığı da oldukça fazla olmaktadır.



Modelin karmaşıklığı ile amprık hata arasında bir denge kurmak için en basit modelden başlanarak her bir modelin amprık hatası hesaplanır ve en az amprık hatayı veren en basit model bulunmaya çalışılır.

Diğer taraftan, $R_{emp}[\hat{y}]$ büyüklüğünün çok büyük olması zaten istenen bir durum (az-öğrenme) değilken, çok küçük olması durumu (aşırı-öğrenme) da istenen bir durum değildir. Bunu en iyi şekilde ayarlamak için kullanılacak yöntemlerin en yaygını

$$N = N_{tra} + N_{val} + N_{tst} \quad 0$$

olacak şekilde verilerin *rasgele bir şekilde* eğitim (training), doğrulama (validation) ve test olarak üç kısma ayrılmasıdır, burada N_{tra} , N_{val} ve N_{tst} sırasıyla eğitim, doğrulama ve test verilerinin sayısıdır. Bu arada, eğitim verilerinin indekslerinin bulunduğu küme TRA, doğrulama verilerinin indekslerinin bulunduğu küme VAL ve test verilerinin indekslerinin bulunduğu küme de TST ile gösterilsin.

Önce, eğitim verileri kullanılarak modelin parametreleri bulunur yani model elde edilir. Modelin parametrelerinin bulunması aslında şu şekilde tanımlanabilecek bir kısıtsız optimizasyon problemidir: $\hat{\mathbf{y}}(\mathbf{x})$ modelinin parametreleri olan \mathbf{x} vektörü öyle ayarlanmalı ki modelin her bir eğitim verisi için yaptığı hataların karelerinin toplamı minimum olsun, yani,

$$\begin{aligned} \min_{\mathbf{x}} F_{tra}(\mathbf{x}) &= \sum_{j=1}^Q \sum_{i \in \text{TRA}} \left(y_{i,j} - \hat{y}_{i,j}(\mathbf{x}) \right)^2 \\ &= \sum_{j=1}^Q \sum_{i \in \text{TRA}} e_{ij}^2(\mathbf{x}) \end{aligned} \quad 0$$

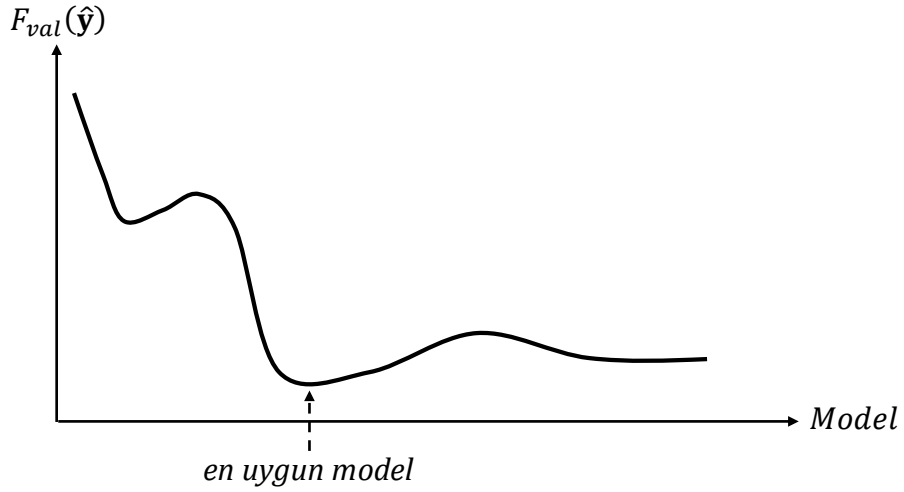
burada her bir veri ve çıkış için modelin yaptığı hata $e_{ij}(\mathbf{x}) = y_{ij} - \hat{y}_{ij}(\mathbf{x})$ şeklindedir.

Ardından, eğitim verileriyle elde edilen modelin ne kadar iyi genelleme yapabildiğini anlamak için modele daha önceden hiç kullanılmayan doğrulama verileri uygulanarak modelin doğrulama (validation) performansı bulunur. Doğrulama performansı şu şekilde tanımlanmaktadır:

$$F_{val}(\hat{\mathbf{y}}) = \frac{1}{N_{val}} \sum_{j=1}^Q \sum_{i \in \text{VAL}} \left(y_{i,j} - \hat{y}_{i,j}(\mathbf{x}) \right)^2 \quad 0$$

Doğrulama performansı modelin genelleme yeteneğinin sayısal bir göstergesidir. Birden fazla tipte model arasında bir seçim yapılacaksa o zaman her bir tipe ilişkin en iyi genelleme yapan modellerin test performansına bakılır. Eğer tek bir model tipi varsa test verilerine gerek yoktur, sadece eğitim ve doğrulama olarak ikiye ayrılması yeterlidir.

Sonuç olarak eldeki kısıtlı sayıda veriyle en optimum modeli bulmak için şu yöntem izlenebilir: Tek tip model olduğu varsayımı altında, önce veriler eğitim ve doğrulama olmak üzere ikiye ayrılır. Ardından, aynı eğitim verileriyle elde edilen her bir modelin doğrulama performansı elde edilerek aşağıdakine benzer bir grafik elde edilir:



Bu grafikten de görüldüğü gibi, “en uygun model” olarak, doğrulama hatası yeterince düşük olan en basit model seçilir.

4.4. Temel Modeller

4.4.1 Bir-Boyutlu Veri Modelleme

Eğer veri setinde $R = 1$ ve $Q = 1$ ise o zaman problem çok basit bir form alır. Aşağıdaki tabloda görüldüğü gibi, $t_i, y_i \in \mathbb{R}$ olmak üzere $\mathcal{D}_{\text{SISO}} = \{t_i, y_i\}_{i=1}^N$ şeklinde verilen bir Tek-Girişli Tek-Çıkışlı (Single-Input Single-Output-SISO) veriyi ele alalım:

$\mathcal{D}_{\text{SISO}}$	Giriş Verisi	Çıkış Verisi	Model Çıkışı	Model Hatası
i	t_i	y_i	\hat{y}_i	$e_i = y_i - \hat{y}_i$
1	t_1	y_1	\hat{y}_1	$e_1 = y_1 - \hat{y}_1$
2	t_2	y_2	\hat{y}_2	$e_2 = y_2 - \hat{y}_2$
\vdots	\vdots	\vdots	\vdots	\vdots
N	t_N	y_N	\hat{y}_N	$e_N = y_N - \hat{y}_N$

Bu veriler, $\hat{y} = \hat{y}(\mathbf{x})$ gibi bir yapı ile modellenenecektir. Bu durumda, ilk olarak model parametrelerinin ayarlanması için eğitim verileri kullanılır. $\hat{y}(\mathbf{x})$ modelinin parametreleri olan \mathbf{x} vektörü öyle ayarlanmalı ki modelin her bir eğitim verisi için yaptığı hataların karelerinin toplamı minimum olsun, yani,

$$\min_{\mathbf{x}} F_{\text{tra}}(\mathbf{x}) = \sum_{i \in \text{TRA}} e_i^2(\mathbf{x})$$

0

burada her bir veri için modelin yaptığı hata $e_i(\mathbf{x}) = y_i - \hat{y}_i(\mathbf{x})$ şeklindedir. Ardından elde edilen her bir modelin doğrulama performansı

$$F_{val}(\hat{\mathbf{y}}) = \frac{1}{N_{val}} \sum_{i \in \text{VAL}} (y_{i,j} - \hat{y}_{i,j}(\mathbf{x}))^2 \quad 0$$

ile bulunarak en iyi doğrulama performansını veren en basit model seçilir.

4.4.1.1. Lineer Modeller

Lineer modellerde $\hat{y}(\mathbf{x})$ modelinin model çıkışı model parametrelerine lineer bir ilişki ile bağlıdır. Çok farklı lineer model yapıları mevcuttur.

4.4.1.1.1 Polinom Modeli

Polinom modelinin giriş-çıkış ilişkisi $\hat{y}_i(\mathbf{x}) = x_1 + x_2 t_i + x_3 t_i^2 + \dots + x_n t_i^{n-1}$ şeklinde $(n-1)$. dereceden bir polinom şeklindedir. Model hatalarının karelerinin toplamı şeklindeki amaç fonksiyonunun durağan noktalarını bulmak için gradyant vektörü sıfıra eşitlenecektir ama bunun için öncelikle Jacobian matrisi bulunmalıdır.

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} \frac{\partial e_1(\mathbf{x})}{\partial x_1} & \frac{\partial e_1(\mathbf{x})}{\partial x_2} & \dots & \frac{\partial e_1(\mathbf{x})}{\partial x_n} \\ \frac{\partial e_2(\mathbf{x})}{\partial x_1} & \frac{\partial e_2(\mathbf{x})}{\partial x_2} & \dots & \frac{\partial e_2(\mathbf{x})}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_N(\mathbf{x})}{\partial x_1} & \frac{\partial e_N(\mathbf{x})}{\partial x_2} & \dots & \frac{\partial e_N(\mathbf{x})}{\partial x_n} \end{bmatrix} \quad 0$$

şeklindeki Jacobian matrisinin her bir $\frac{\partial e_i(\mathbf{x})}{\partial x_j}$ teriminin,

$$\begin{aligned} \frac{\partial e_i(\mathbf{x})}{\partial x_j} &= \frac{\partial (y_i - \hat{y}_i(\mathbf{x}))}{\partial x_j} \\ &= - \frac{\partial \hat{y}_i(\mathbf{x})}{\partial x_j} \\ &= - \frac{\partial (x_1 + x_2 t_i + x_3 t_i^2 + \dots + x_n t_i^{n-1})}{\partial x_j} \\ &= - t_i^{j-1} \end{aligned} \quad 0$$

olduğu bulunabilir ki böylece Jacobian matrisi

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} -1 & -t_1 & \dots & -t_1^{n-1} \\ -1 & -t_2 & \dots & -t_2^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ -1 & -t_N & \dots & -t_N^{n-1} \end{bmatrix} \quad 0$$

şeklinde yazılabilir. Gradyent vektörünün

$$\nabla f(\mathbf{x}) = 2\mathbf{J}^T(\mathbf{x})\mathbf{e}(\mathbf{x}) \quad 0$$

şeklinde olduğu bilgisiyile

$$\begin{aligned} \nabla f(\mathbf{x}) &= 2\mathbf{J}^T(\mathbf{y} - \hat{\mathbf{y}}) \\ &= 2\mathbf{J}^T\mathbf{y} - 2\mathbf{J}^T\hat{\mathbf{y}} \end{aligned} \quad 0$$

elde edilip vektör sıfıra eşitlenirse,

$$\nabla f(\mathbf{x}) = 2\mathbf{J}^T\mathbf{y} - 2\mathbf{J}^T\hat{\mathbf{y}} = \mathbf{0} \quad 0$$

ve böylece $\mathbf{J}^T\mathbf{y} = \mathbf{J}^T\hat{\mathbf{y}}$ elde edilir. $\hat{\mathbf{y}}$ vektörünü yeniden düzenleyelim:

$$\hat{\mathbf{y}} = \begin{bmatrix} \hat{y}_1(\mathbf{x}) \\ \hat{y}_2(\mathbf{x}) \\ \vdots \\ \hat{y}_N(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} x_1 + x_2 t_1 + x_3 t_1^2 + \dots + x_n t_1^{n-1} \\ x_1 + x_2 t_2 + x_3 t_2^2 + \dots + x_n t_2^{n-1} \\ \vdots \\ x_1 + x_2 t_N + x_3 t_N^2 + \dots + x_n t_N^{n-1} \end{bmatrix} = -\mathbf{J}\mathbf{x} \quad 0$$

Böylece

$$\begin{aligned} \mathbf{J}^T\mathbf{y} &= \mathbf{J}^T\hat{\mathbf{y}} \\ &= -\mathbf{J}^T\mathbf{J}\mathbf{x} \end{aligned} \quad 0$$

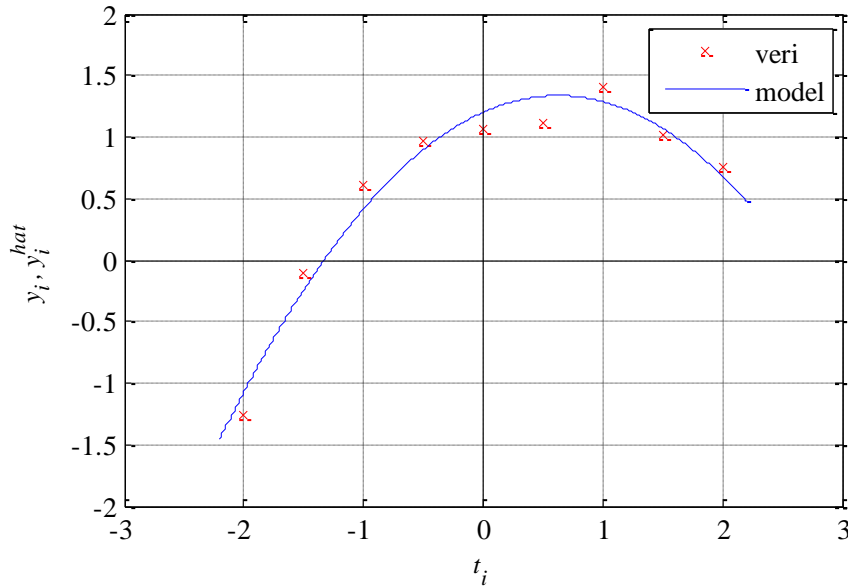
bulunur ve buradan da durağan nokta aşağıdaki gibi analitik olarak çözülebilir:

$$\mathbf{x}^* = -(\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T\mathbf{y}. \quad 0$$

Örnek: Aşağıda verilen noktalara olan uzaklıklarının karelerinin toplamı minimum olan parabolü bulunuz.

t_i	-2.0000	-1.5000	-1.0000	-0.5000	0.0000	0.5000	1.0000	1.5000	2.0000
y_i	-1.2557	-0.1074	0.6040	0.9605	1.0730	1.1060	1.4090	1.0223	0.7559

Modelin giriş-çıkış denklemi $\hat{y}_i(\mathbf{x}) = x_1 + x_2 t_i + x_3 t_i^2$ şeklinde olup çözüm noktası $\mathbf{x}^* = [1.1979 \quad 0.4397 \quad -0.3476]^T$ olarak bulunur. Bulunan parabol modeli aşağıda görüldüğü gibidir.

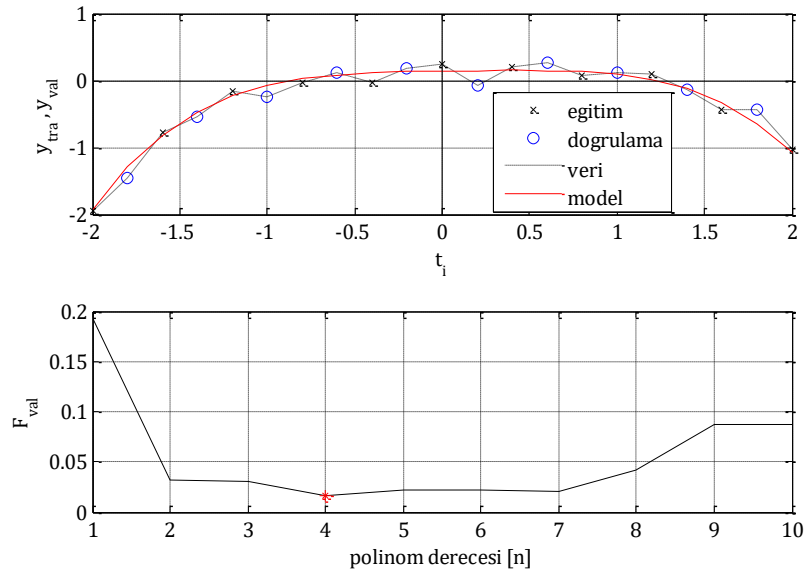


Bu örnekte model seçme (model selection) uygulanmamış sadece verilen noktalara en yakın parabol bulunmuştur. Şimdi başka bir örnekle model seçiminin nasıl yapıldığını görelim:

Örnek: Aşağıda verilen noktalara en iyi şekilde uyan polinomu bulunuz.

t_i	-2.0000	-1.8000	-1.6000	-1.4000	-1.2000	-1.0000	-0.8000	-0.6000	-0.4000	-0.2000	0.0000
y_i	-1.9499	-1.4558	-0.7719	-0.5325	-0.1693	-0.2355	-0.0295	0.1154	-0.0216	0.1753	0.2485
t_i	0.2000	0.4000	0.6000	0.8000	1.0000	1.2000	1.4000	1.6000	1.8000	2.0000	
y_i	-0.0652	0.2044	0.2604	0.0669	0.1113	0.0886	-0.1445	-0.4356	-0.4396	-1.0387	

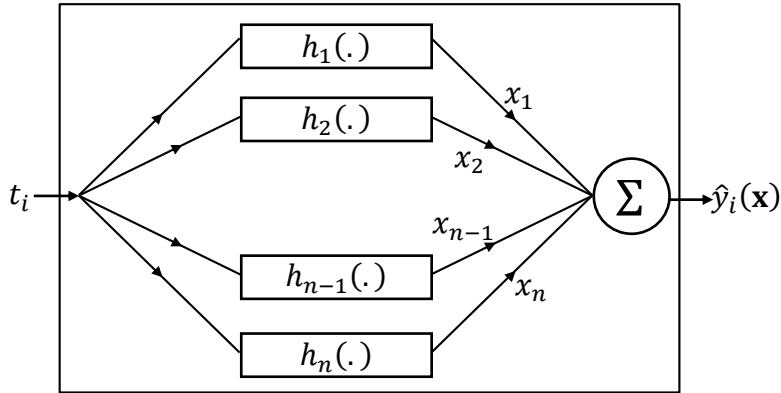
Modelin giriş-çıkış denklemi $\hat{y}_i(\mathbf{x}) = x_1 + x_2 t_i + x_3 t_i^2 + \dots + x_n t_i^{n-1}$ şeklinde $(n-1)$. dereceden bir polinomdur. Ancak polinomun derecesi belli değildir. O yüzden birinci-dereceden başlayarak belli bir dereceye kadar polinomlar teker teker denenecektir. En iyi modeli seçmek için verilerin yarısı eğitim kalan yarısı da doğrulamada kullanılacaktır. Kullanılan veriler ve bulunan parabol modellerinin doğrulama performansları aşağıda görüldüğü gibidir.



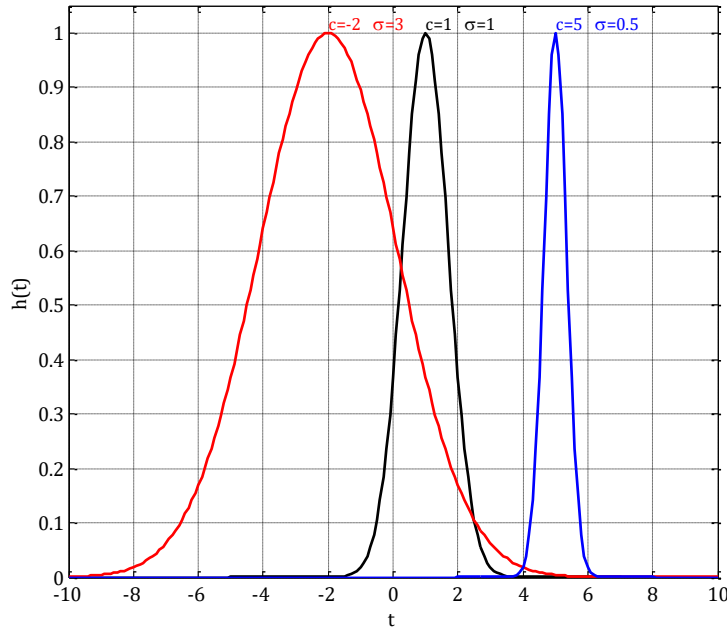
Üstteki şekilde çarpı (x) işareti ile gösterilen veriler polinomun katsayılarını belirlemede kullanılmıştır. Ardından, katsayıları bulunan polinoma yuvarlak (o) ile gösterilen doğrulama verileri uygulanmış ve polinomun doğrulama performansı $F_{val}[\hat{y}] = \frac{1}{N_{val}} \sum_{i=1}^{N_{val}} (y_i - \hat{y}_i)^2$ formülüyle hesaplanmıştır. Buna göre, alttaki şekilden de görüldüğü gibi 4^{üncü}-dereceden polinom en iyi doğrulama performansına sahiptir. O halde, bu verilere en iyi uyan polinom 4^{üncü}-dereceden olmalıdır. Bu polinomun tüm veriler için ürettiği eğri üstteki şekilde görüldüğü gibi verilere oldukça iyi uymaktadır. Gerçekte de, bu veriler, $y = 0.0866 + 0.0943t - 0.0278t^2 + 0.0288t^3 - 0.0864t^4 + \eta$ şeklinde 4^{üncü}-dereceden bir polinomdan elde edilmiştir ancak elde edilen verilere η ile gösterilen bir Gauss gürültüsü eklenmiştir.

4.4.1.1.2 Radyal Tabanlı Fonksiyon (Radial Basis Function RBF) Modeli

RBF modelinin giriş-çıkış ilişkisi, $\hat{y}_i(\mathbf{x}) = x_1 h_1(t_i) + x_2 h_2(t_i) + \dots + x_n h_n(t_i)$ şeklinde n adet RBF fonksiyonun lineer kombinasyonu şeklindedir. Bu aşağıdaki şekille daha iyi görülebilir:



Burada, RBF fonksiyonu olarak aşağıdaki Gaussian fonksiyon kullanılabilir:



$$h_j(t) = \exp\left(-\frac{(t - c_j)^2}{\sigma_j^2}\right)$$

0

burada c_j ve σ_j RBF fonksiyonun sırasıyla merkez ve genişlik parametreleridir. Durağan nokta yine $\mathbf{x}^* = -(\mathbf{J}^T \mathbf{J})^{-1} \mathbf{J}^T \mathbf{y}$ eşitliğinden çözülür ancak bu kez \mathbf{J} matrisinin elemanları aşağıdaki gibi bulunur:

$$\begin{aligned}
\frac{\partial e_i(\mathbf{x})}{\partial x_j} &= \frac{\partial (y_i - \hat{y}_i(\mathbf{x}))}{\partial x_j} \\
&= -\frac{\partial \hat{y}_i(\mathbf{x})}{\partial x_j} \\
&= -\frac{\partial (x_1 h_1(t_i) + x_2 h_2(t_i) + \dots + x_n h_n(t_i))}{\partial x_j} \\
&= -h_j(t_i) \\
&= -\exp\left(-\frac{(t_i - c_j)^2}{\sigma_j^2}\right)
\end{aligned}$$

Böylece Jacobian matrisi şu şekildedir:

$$\mathbf{J}(\mathbf{x}) = -\begin{bmatrix} h_1(t_1) & h_2(t_1) & \dots & h_n(t_1) \\ h_1(t_2) & h_2(t_2) & \dots & h_n(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ h_1(t_N) & h_2(t_N) & \dots & h_n(t_N) \end{bmatrix}$$

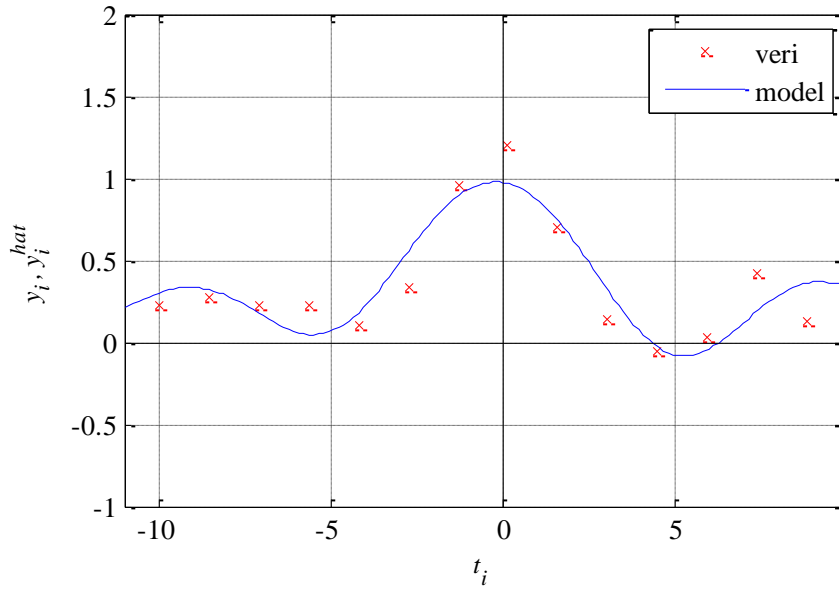
Örnek: Aşağıda verilen noktalara olan uzaklıklarının karelerinin toplamı minimum olan 6 düğümlü bir RBF modelini bulunuz.

t_i	-10.0000	-8.5500	-7.1000	-5.6500	-4.2000	-2.7500	-1.3000	0.1500	1.6000	3.0500	4.5000	5.9500	7.4000	8.8500
y_i	0.2297	0.2762	0.2254	0.2239	0.0991	0.3313	0.9591	1.2017	0.6974	0.1354	-0.0524	0.0257	0.4170	0.1296

Modelin giriş-çıkış denklemi $\hat{y}_i(\mathbf{x}) = x_1 h_1(t_i) + x_2 h_2(t_i) + \dots + x_6 h_6(t_i)$ şeklindedir. Kolaylık olması açısından merkezler (c_j 'ler) tüm t_i aralığında eşit olarak dağıtılmıştır, bunun sonucunda merkez noktaları $c_1 = -8.3333, c_2 = -5.0000, c_3 = -1.6667, c_4 = 1.6667, c_5 = 5.0000$ ve $c_6 = 8.3333$ şeklinde bulunmuştur. Genişlik parametreleri (σ_j 'ler) de tüm düğümler için aynı seçilmiş ve tüm t_i aralığının düğüm sayısına bölünmesiyle $\sigma_j = 3.333, j = 1, \dots, 6$ elde edilmiştir. Böylece çözüm noktası

$$\mathbf{x}^* = [0.4343 \quad -0.3724 \quad 0.7284 \quad 0.6393 \quad -0.5092 \quad 0.5014]^T$$

olarak bulunur. Bulunan RBF modeli aşağıda görüldüğü gibidir.

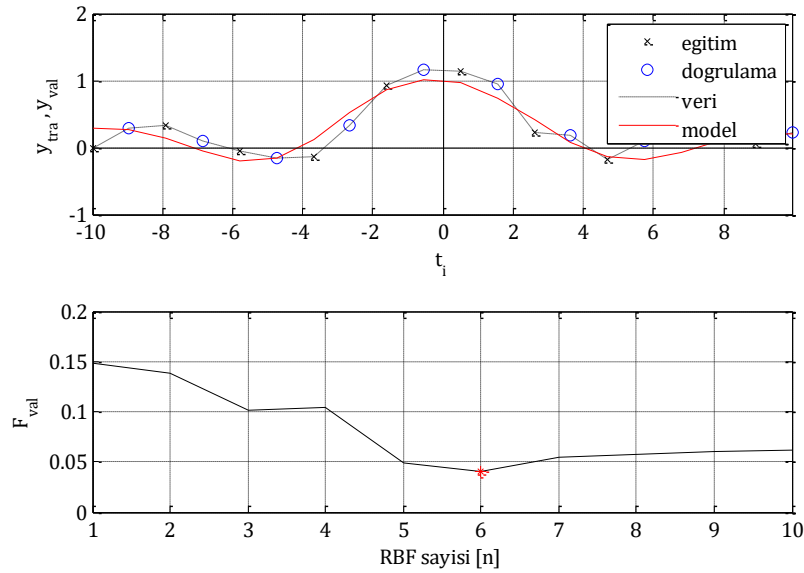


Bu örnekte model seçme (model selection) uygulanmamış sadece verilen noktalara en yakın 6 düğümlü RBF modeli bulunmuştur. Şimdi başka bir örnekle model seçiminin nasıl yapıldığını görelim:

Örnek: Aşağıda verilen noktalara en iyi şekilde uyan RBF modelini bulunuz.

t_i	-10.0000	-8.9500	-7.9000	-6.8500	-5.8000	-4.7500	-3.7000	-2.6500	-1.6000	-0.5500
y_i	-0.0116	0.2860	0.3363	0.0980	-0.0604	-0.1570	-0.1363	0.3304	0.9160	1.1664
t_i	0.5000	1.5500	2.6000	3.6500	4.7000	5.7500	6.8000	7.8500	8.9000	9.9500
y_i	1.1409	0.9474	0.2325	0.1845	-0.1749	0.0926	0.1228	0.3232	0.0579	0.2179

Modelin giriş-çıkış denklemi $\hat{y}_i(\mathbf{x}) = x_1 h_1(t_i) + x_2 h_2(t_i) + \dots + x_n h_n(t_i)$ şeklinde n adet RBF fonksiyonun lineer kombinasyonudur. Ancak n belli değildir. O yüzden birden başlayarak belli bir rakama kadar teker teker denenecektir. En iyi modeli seçmek için verilerin yarısı eğitim kalan yarısı da doğrulamada kullanılacaktır. Kullanılan veriler ve bulunan RBF modellerinin doğrulama performansları aşağıda görüldüğü gibidir.



Üstteki şekilde çarpı (x) işareti ile gösterilen veriler RBF modelinin katsayılarını belirlemede kullanılmıştır. Ardından, katsayıları bulunan modele yuvarlak (o) ile gösterilen doğrulama verileri uygulanmış ve modelin doğrulama performansı $F_{val}[\hat{y}] = \frac{1}{N_{val}} \sum_{i=1}^{N_{val}} (y_i - \hat{y}_i)^2$ formülüyle hesaplanmıştır. Buna göre, alttaki şekilden de görüldüğü gibi 6 RBF içeren model en iyi doğrulama performansına sahiptir. O halde, bu verilere en iyi uyan model 6 RBF'den oluşmalıdır. Bu modelin tüm veriler için ürettiği eğri üstteki şekilde görüldüğü gibi verilere oldukça iyi uymaktadır. Gerçekte de, bu veriler, $y = \frac{\sin t}{t} + \eta$ şeklinde bir sinc fonksiyonundan elde edilmiştir ancak elde edilen verilere η ile gösterilen bir Gauss gürültüsü eklenmiştir.

4.4.1.2. Lineer-olmayan Modeller

Lineer-olmayan modellerde $\hat{y}(\mathbf{x})$ modelinin model çıkışı ile model parametreleri arasında lineer-olmayan bir ilişki vardır. Bunun sonucu olarak da \mathbf{J} matrisinin bazı elemanları sabit skaler büyüklük olmayıp değişken hale gelir ve dolayısıyla durağan noktanın bulunması için iteratif bir yaklaşım gerekir. Literatürde çok sayıda lineer-olmayan model yapısı bulunabilir.

4.4.1.2.1 Üstel Model

Eğer giriş-çıkış verisinin üstel bir doğaya sahip olduğu önceden biliniyorsa,

$$\hat{y}_i(\mathbf{x}) = x_1 e^{x_2 t_i}$$

0

şeklinde verilen model kullanılabilir. Üstel modelde sadece iki parametre mevcuttur. \mathbf{J} matrisine ilişkin terimler şu şekildedir:

$$\begin{aligned}\frac{\partial e_i(\mathbf{x})}{\partial x_1} &= \frac{\partial (y_i - \hat{y}_i(\mathbf{x}))}{\partial x_1} & \frac{\partial e_i(\mathbf{x})}{\partial x_2} &= \frac{\partial (y_i - \hat{y}_i(\mathbf{x}))}{\partial x_2} \\ &= -\frac{\partial \hat{y}_i(\mathbf{x})}{\partial x_1} & &= -\frac{\partial \hat{y}_i(\mathbf{x})}{\partial x_2} \\ &= -\frac{\partial (x_1 e^{x_2 t_i})}{\partial x_1} & \text{ve} & \\ &= -e^{x_2 t_i} & &= -x_1 t_i e^{x_2 t_i}\end{aligned}\quad 0$$

şeklindedir. Buradan Jacobian matrisi

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} \frac{\partial e_1(\mathbf{x})}{\partial x_1} & \frac{\partial e_1(\mathbf{x})}{\partial x_2} \\ \frac{\partial e_2(\mathbf{x})}{\partial x_1} & \frac{\partial e_2(\mathbf{x})}{\partial x_2} \\ \vdots & \vdots \\ \frac{\partial e_N(\mathbf{x})}{\partial x_1} & \frac{\partial e_N(\mathbf{x})}{\partial x_2} \end{bmatrix} = \begin{bmatrix} -e^{x_2 t_1} & -x_1 t_1 e^{x_2 t_1} \\ -e^{x_2 t_2} & -x_1 t_2 e^{x_2 t_2} \\ \vdots & \vdots \\ -e^{x_2 t_N} & -x_1 t_N e^{x_2 t_N} \end{bmatrix}\quad 0$$

gibi yazılabilir. \mathbf{J} matrisi kullanılarak LM algoritmasıyla çözüm bulunabilir. Eğer Değiştirilmiş-Newton yöntemi kullanılacaksa, Hessian matrisi gerekir ama bu durumda işlemsel karmaşıklık artar.

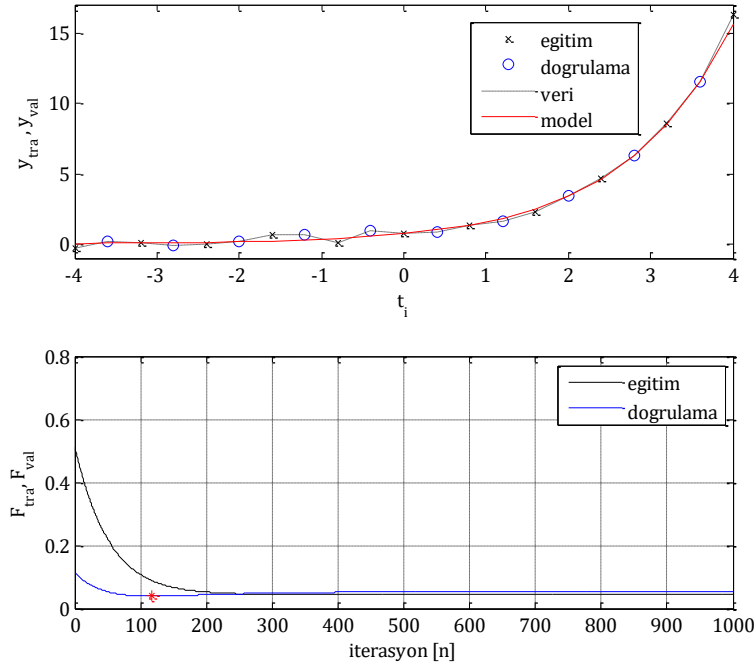
Örnek olarak, t_i 'ler -4 ile $+4$ arasında 0.4 aralıklarla alınan girişler olmak üzere

$$y_i(\mathbf{x}) = 0.6294e^{0.8116t_i} + \eta \quad 0$$

şeklindeki bir sistemden elde edilen gürültülü ölçümleri ele alalım, burada η Gauss gürültüsünü temsil etmektedir. Aşağıdaki şekilde görüldüğü gibi, alınan ölçümlerin yarısı eğitim kalan yarısı da doğrulamada kullanılarak rasgele bir $[x_1 \ x_2]$ noktasında başlanmış ve bu nokta her adımda

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k) + 10^{-8}\mathbf{I}]^{-1}\mathbf{J}^T(\mathbf{x}_k)\mathbf{e}(\mathbf{x}_k). \quad 0$$

şeklindeki LM algoritmasıyla güncellenerek eğitim ve doğrulama performansları kaydedilerek en iyi doğrulama performansını veren çözüm $[0.6711 \ 0.7970]$ olarak bulunmuştur.

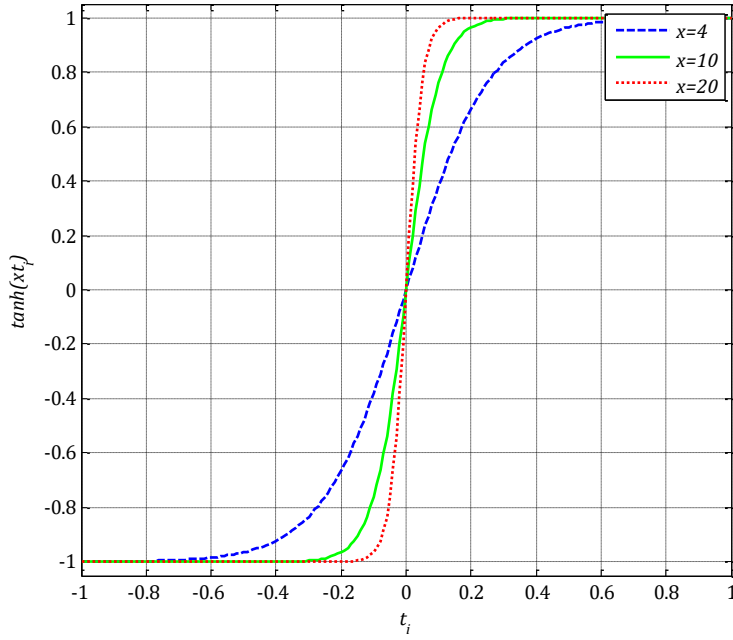


4.4.1.2.2 Hiperbolik Model

Hiperbolik modeller hiperbolik fonksiyonlardan oluşur. Örnek olarak

$$\tanh(xt_i) = \frac{e^{xt_i} - e^{-xt_i}}{e^{xt_i} + e^{-xt_i}} \quad 0$$

şeklindeki hiperbolik tanjant fonksiyonunu ele alalım, buradaki x parametresi aşağıdaki şekilde görüldüğü gibi fonksiyonun yayılımını belirlemektedir.



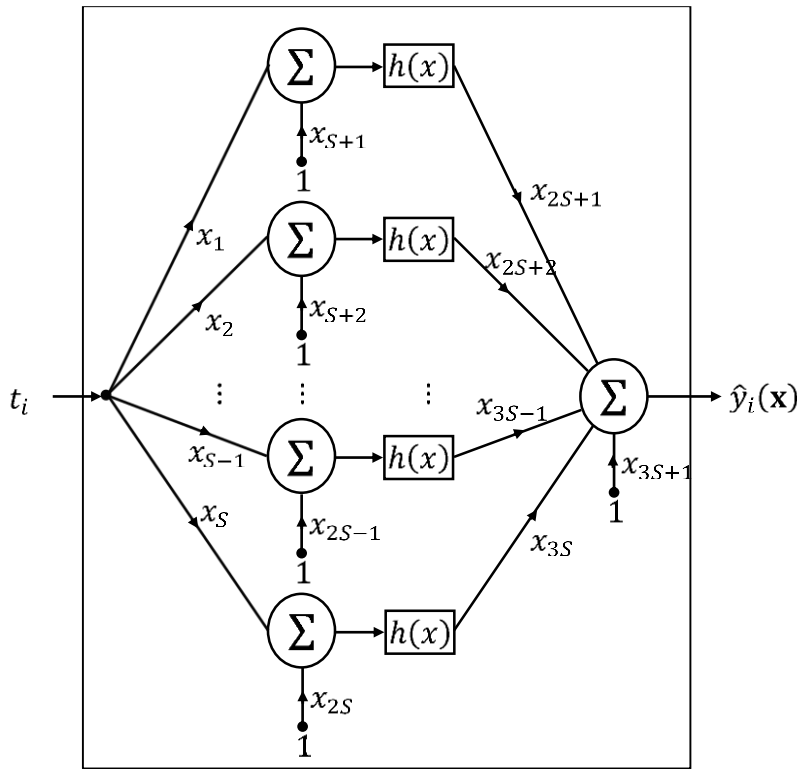
Bu tip fonksiyonların bir avantajı da türevinin kolaylıkla alınabiliyor olmasıdır:

$$\frac{d \tanh(xt_i)}{dx} = t_i(1 - \tanh^2(xt_i)) \quad 0$$

Bunun gibi S adet hiperbolik fonksiyonlardan oluşan hiperbolik modelin giriş-çıkış ilişkisi şu şekildedir:

$$\hat{y}_i(\mathbf{x}) = x_{2S+1} \tanh(x_1 t_i + x_{S+1}) + \dots + x_{3S} \tanh(x_S t_i + x_{2S}) + x_{3S+1} \quad 0$$

Bu ilişki aşağıdaki şekilde daha iyi görülebilir:



Hiperbolik modelde $3S + 1$ adet parametre mevcuttur. Bu parametrelerin bir kısmı $\hat{y}_i(\mathbf{x}) = x_{2S+1} \tanh(x_1 t_i + x_{S+1}) + \dots + x_{3S} \tanh(x_S t_i + x_{2S}) + x_{3S+1}$ şeklindeki giriş-çıkış ilişkisinde lineer olarak yer alırken kalan kısmı da lineer-olmayan bir şekilde yer almaktadır. Bu da Jacobiandeki terimlerin bulunmasındaki kolaylığı belirler. \mathbf{J} matrisine ilişkin terimler şu şekildedir:

$$\begin{aligned}
 \frac{\partial e_i(\mathbf{x})}{\partial x_j} &= \frac{\partial (y_i - \hat{y}_i(\mathbf{x}))}{\partial x_j} \\
 &= -\frac{\partial \hat{y}_i(\mathbf{x})}{\partial x_j} \quad j = 1, \dots, S \quad 0 \\
 &= -x_{j+2S} t_i (1 - \tanh^2(x_j t_i + x_{S+j}))
 \end{aligned}$$

ve

$$\begin{aligned}
 \frac{\partial e_i(\mathbf{x})}{\partial x_j} &= \frac{\partial (y_i - \hat{y}_i(\mathbf{x}))}{\partial x_j} \\
 &= -\frac{\partial \hat{y}_i(\mathbf{x})}{\partial x_j} \quad j = S + 1, \dots, 2S \quad 0 \\
 &= -x_{j+S} (1 - \tanh^2(x_{j-S} t_i + x_j))
 \end{aligned}$$

ve

$$\begin{aligned}
\frac{\partial e_i(\mathbf{x})}{\partial x_j} &= \frac{\partial (y_i - \hat{y}_i(\mathbf{x}))}{\partial x_j} \\
&= - \frac{\partial \hat{y}_i(\mathbf{x})}{\partial x_j} \quad j = 2S + 1, \dots, 3S \\
&= - \tanh(x_{j-2S}t_i + x_{j-S})
\end{aligned} \quad 0$$

ve

$$\begin{aligned}
\frac{\partial e_i(\mathbf{x})}{\partial x_j} &= \frac{\partial (y_i - \hat{y}_i(\mathbf{x}))}{\partial x_j} \\
&= - \frac{\partial \hat{y}_i(\mathbf{x})}{\partial x_j} \quad j = 3S + 1 \\
&= -1
\end{aligned} \quad 0$$

şeklinde. \mathbf{J} matrisi kullanılarak LM algoritmasıyla çözüm bulunabilir. Eğer Değiştirilmiş-Newton yöntemi kullanılacaksa, Hessian matrisi gerekir ama bu durumda işlemsel karmaşıklık artar.

4.4.2. Çok-boyutlu Veri Modelleme

Aşağıdaki tabloda görüldüğü gibi, $\mathbf{t}_i \in \mathbb{R}^R$ ve $y_i \in \mathbb{R}$ olmak üzere $\mathcal{D}_{\text{MISO}} = \{\mathbf{t}_i, y_i\}_{i=1}^N$ şeklinde verilen bir Çok-Girişli Tek-Çıkışlı (Multi-Input Single-Output-MISO) veriyi ele alalım:

$\mathcal{D}_{\text{MISO}}$	Giriş Verisi				Çıkış Verisi	Model Çıkışı	Model Hatası
i	\mathbf{t}_i				y_i	\hat{y}_i	$e_i = y_i - \hat{y}_i$
	t_{i1}	t_{i2}	...	t_{iR}			
1	t_{11}	t_{12}	...	t_{1R}	y_1	\hat{y}_1	e_1
2	t_{21}	t_{22}	...	t_{2R}	y_2	\hat{y}_2	e_2
\vdots	\vdots	\vdots	\ddots	\vdots	\vdots	\vdots	\vdots
N	t_{N1}	t_{N2}	...	t_{NR}	y_N	\hat{y}_N	e_N

Bu veriler, $\hat{y} = \hat{y}(\mathbf{x})$ gibi bir yapı ile modellenenecektir. Bu durumda, Çok-Girişli Tek-Çıkışlı (Multi-Input Single-Output-MISO) modelleme probleminin tanımı şu şekilde bir optimizasyon problemi ile yapılabilir: $\hat{y}(\mathbf{x})$ modelinin parametreleri olan \mathbf{x} vektörü öyle ayarlanmalı ki modelin her bir veri için yaptığı hataların karelerinin toplamı minimum olsun, yani,

$$\min_{\mathbf{x}} f(\mathbf{x}) = e_1^2(\mathbf{x}) + e_2^2(\mathbf{x}) + \dots + e_N^2(\mathbf{x}) \quad 0$$

burada her bir veri için modelin yaptığı hata $e_i(\mathbf{x}) = y_i - \hat{y}_i(\mathbf{x})$ şeklindedir. Eğer istenen çıkış vektörü $\mathbf{y} = [y_1 \ y_2 \ \dots \ y_N]^T$ ve modelin ürettiği çıkış vektörü $\hat{\mathbf{y}}(\mathbf{x}) =$

$[\hat{y}_1(\mathbf{x}) \ \hat{y}_2(\mathbf{x}) \ \dots \ \hat{y}_N(\mathbf{x})]^T$ olmak üzere hata vektörü $\mathbf{e}(\mathbf{x}) = \mathbf{y} - \hat{\mathbf{y}}(\mathbf{x})$ şeklinde yazılırsa, problem aşağıdaki hale gelir:

$$\min_{\mathbf{x}} f(\mathbf{x}) = \mathbf{e}^T(\mathbf{x})\mathbf{e}(\mathbf{x}) \quad 0$$

4.4.2.1 Lineer Modeller

4.4.2.1.1 Polinom Modeli

İki-Girişli Tek-Çıkışlı bir polinom modelinin giriş-çıkış ilişkisi $\hat{y}_i(\mathbf{x}) = x_0 + x_1 t_{i1} + x_2 t_{i1}^2 + \dots + x_n t_{i1}^n + x_{n+1} t_{i2} + x_{n+2} t_{i2}^2 + \dots + x_{2n} t_{i2}^n$ şeklinde n -dereceden bir polinom şeklindedir. Model hatalarının karelerinin toplamı şeklindeki amaç fonksiyonunun durağan noktalarını bulmak için gradyant vektörü sıfıra eşitlenecektir ama bunun için öncelikle Jacobian matrisi bulunmalıdır.

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} \frac{\partial e_1(\mathbf{x})}{\partial x_0} & \frac{\partial e_1(\mathbf{x})}{\partial x_1} & \frac{\partial e_1(\mathbf{x})}{\partial x_2} & \dots & \frac{\partial e_1(\mathbf{x})}{\partial x_n} & \frac{\partial e_1(\mathbf{x})}{\partial x_{n+1}} & \frac{\partial e_1(\mathbf{x})}{\partial x_{n+2}} & \dots & \frac{\partial e_1(\mathbf{x})}{\partial x_{2n}} \\ \frac{\partial e_2(\mathbf{x})}{\partial x_0} & \frac{\partial e_2(\mathbf{x})}{\partial x_1} & \frac{\partial e_2(\mathbf{x})}{\partial x_2} & \dots & \frac{\partial e_2(\mathbf{x})}{\partial x_n} & \frac{\partial e_2(\mathbf{x})}{\partial x_{n+1}} & \frac{\partial e_2(\mathbf{x})}{\partial x_{n+2}} & \dots & \frac{\partial e_2(\mathbf{x})}{\partial x_{2n}} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_N(\mathbf{x})}{\partial x_0} & \frac{\partial e_N(\mathbf{x})}{\partial x_1} & \frac{\partial e_N(\mathbf{x})}{\partial x_2} & \dots & \frac{\partial e_N(\mathbf{x})}{\partial x_n} & \frac{\partial e_N(\mathbf{x})}{\partial x_{n+1}} & \frac{\partial e_N(\mathbf{x})}{\partial x_{n+2}} & \dots & \frac{\partial e_N(\mathbf{x})}{\partial x_{2n}} \end{bmatrix} \quad 0$$

şeklindeki Jacobian matrisinin her bir $\frac{\partial e_i(\mathbf{x})}{\partial x_j}$ terimi, $j = 0$ için

$$\begin{aligned} \frac{\partial e_i(\mathbf{x})}{\partial x_j} &= \frac{\partial (y_i - \hat{y}_i(\mathbf{x}))}{\partial x_j} \\ &= - \frac{\partial \hat{y}_i(\mathbf{x})}{\partial x_j} \\ &= - \frac{\partial (x_0 + x_1 + x_2 t_{i1} + x_3 t_{i1}^2 + \dots + x_n t_{i1}^{n-1} + x_{n+1} + x_{n+2} t_{i2} + x_{n+3} t_{i2}^2 + \dots + x_{2n} t_{i2}^{n-1})}{\partial x_j} \\ &= -1 \\ j &= 1, \dots, n \text{ için} \end{aligned} \quad \left(\begin{array}{l} \\ \\ \end{array} \right)$$

$$\begin{aligned}
\frac{\partial e_i(\mathbf{x})}{\partial x_j} &= \frac{\partial(y_i - \hat{y}_i(\mathbf{x}))}{\partial x_j} \\
&= -\frac{\partial \hat{y}_i(\mathbf{x})}{\partial x_j} \\
&= -\frac{\partial(x_0 + x_1 + x_2 t_{i1} + x_3 t_{i1}^2 + \dots + x_n t_{i1}^{n-1} + x_{n+1} + x_{n+2} t_{i2} + x_{n+3} t_{i2}^2 + \dots + x_{2n} t_{i2}^{n-1})}{\partial x_j} \\
&= -t_{1i}^{j-1}
\end{aligned}$$

ve $j = n + 1, \dots, 2n$ için

$$\begin{aligned}
\frac{\partial e_i(\mathbf{x})}{\partial x_j} &= \frac{\partial(y_i - \hat{y}_i(\mathbf{x}))}{\partial x_j} \\
&= -\frac{\partial \hat{y}_i(\mathbf{x})}{\partial x_j} \\
&= -\frac{\partial(x_0 + x_1 + x_2 t_{i1} + x_3 t_{i1}^2 + \dots + x_n t_{i1}^{n-1} + x_{n+1} + x_{n+2} t_{i2} + x_{n+3} t_{i2}^2 + \dots + x_{2n} t_{i2}^{n-1})}{\partial x_j} \\
&= -t_{2i}^{j-1}
\end{aligned}$$

,olduğu bulunabilir ki böylece $N \times 2n + 1$ Jacobian matrisi

$$\mathbf{J}(\mathbf{x}) = - \begin{bmatrix} 1 & t_{1,1} & \dots & t_{1,1}^{n-1} & t_{1,2} & \dots & t_{1,2}^{n-1} \\ 1 & t_{2,1} & \dots & t_{2,1}^{n-1} & t_{2,2} & \dots & t_{2,2}^{n-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 1 & t_{N,1} & \dots & t_{N,1}^{n-1} & t_{N,2} & \dots & t_{N,2}^{n-1} \end{bmatrix} \quad 0$$

şeklinde yazılabilir. Gradyent vektörünün

$$\nabla f(\mathbf{x}) = 2\mathbf{J}^T(\mathbf{x})\mathbf{e}(\mathbf{x}) \quad 0$$

şeklinde olduğu bilgisiyle

$$\begin{aligned}
\nabla f(\mathbf{x}) &= 2\mathbf{J}^T(\mathbf{y} - \hat{\mathbf{y}}) \\
&= 2\mathbf{J}^T\mathbf{y} - 2\mathbf{J}^T\hat{\mathbf{y}}
\end{aligned} \quad 0$$

elde edilip vektör sıfıra eşitlenirse,

$$\nabla f(\mathbf{x}) = 2\mathbf{J}^T\mathbf{y} - 2\mathbf{J}^T\hat{\mathbf{y}} = \mathbf{0} \quad 0$$

ve böylece $\mathbf{J}^T\mathbf{y} = \mathbf{J}^T\hat{\mathbf{y}}$ elde edilir. $\hat{\mathbf{y}}$ vektörünü yeniden düzenleyelim:

$$\begin{aligned}
\hat{\mathbf{y}} &= \begin{bmatrix} \hat{y}_1(\mathbf{x}) \\ \hat{y}_2(\mathbf{x}) \\ \vdots \\ \hat{y}_N(\mathbf{x}) \end{bmatrix} \\
&= \begin{bmatrix} x_0 + x_1 + x_2 t_{11} + x_3 t_{11}^2 + \cdots + x_n t_{11}^{n-1} + x_{n+1} + x_{n+2} t_{12} + x_{n+3} t_{12}^2 + \cdots + x_{2n} t_{12}^{n-1} \\ x_0 + x_1 + x_2 t_{21} + x_3 t_{21}^2 + \cdots + x_n t_{21}^{n-1} + x_{n+1} + x_{n+2} t_{22} + x_{n+3} t_{22}^2 + \cdots + x_{2n} t_{22}^{n-1} \\ \vdots \\ x_0 + x_1 + x_2 t_{N1} + x_3 t_{N1}^2 + \cdots + x_n t_{N1}^{n-1} + x_{n+1} + x_{n+2} t_{N2} + x_{n+3} t_{N2}^2 + \cdots + x_{2n} t_{N2}^{n-1} \end{bmatrix} \\
&= -\mathbf{J}\mathbf{x}
\end{aligned}
\tag{0}$$

Böylece,

$$\begin{aligned}
\mathbf{J}^T \mathbf{y} &= \mathbf{J}^T \hat{\mathbf{y}} \\
&= -\mathbf{J}^T \mathbf{J} \mathbf{x}
\end{aligned}
\tag{0}$$

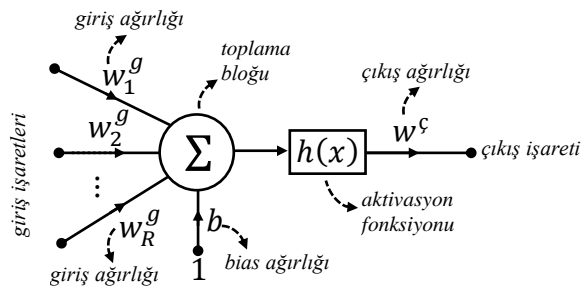
bulunur ve buradan da durağan nokta aşağıdaki gibi analitik olarak çözülebilir:

$$\mathbf{x}^* = -(\mathbf{J}^T \mathbf{J})^{-1} \mathbf{J}^T \mathbf{y}.
\tag{0}$$

BÖLÜM 5. YAPAY SİNİR AĞLARI

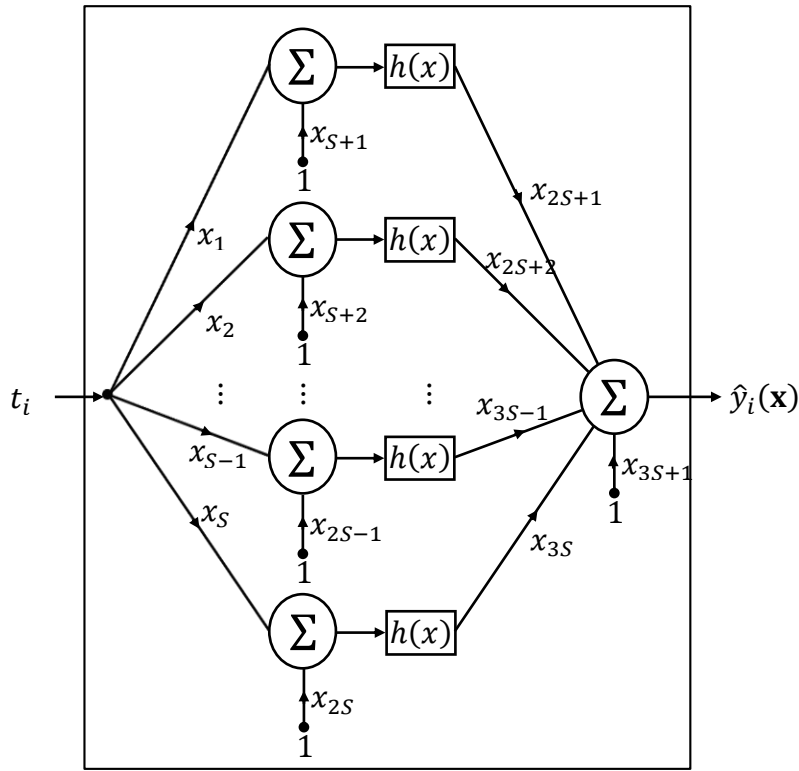
5.1. Basit Nöron Modeli

Yapay Sinir Ağları (YSA), insan beyninin işleyişinden esinlenilerek geliştirilmiş olan ve onun çok basit bir modelini içeren matematiksel yapılardır. Bu yapılarda kullanılan en temel eleman aşağıdaki şekildeki gibi basit bir nöron modelidir. Bu basit nöron modelinde giriş ağırlıkları (w_i^g), bir çıkış ağırlığı (w^c), bir bias ağırlığı (b), bir aktivasyon fonksiyonu ($h(x)$) ve bir toplama bloğu bulunmaktadır. Nörona gelen giriş işaretleri, giriş ağırlıkları ile çarpılarak toplama bloğunda bias ağırlığı ile toplanıp aktivasyon fonksiyonundan geçirilerek çıkış ağırlığı ile çarpılır ve böylece çıkış işareti elde edilir.



5.2. SISO-YSA Modeli

Bu basit nöron modeli kullanılarak, belli sayıda nöronun bir araya gelmesinden oluşan yapıya *Yapay Sinir Ağı* denmektedir. Bu model aynı zamanda *İleri-Beslemeli Yapay Sinir Ağı (Feed-forward Artificial Neural Network)* olarak da adlandırılabilir. Bir-girişli Bir-çıkışlı bir veriyi modellemek için kullanılacak bir SISO-YSA yapısı şekilde görüldüğü gibidir.



Bu SISO-YSA modelinin giriş-çıkış ilişkisi ise şu şekildedir:

$$\hat{y}_i(\mathbf{x}) = x_{2S+1}h(x_1t_i + x_{S+1}) + x_{2S+2}h(x_2t_i + x_{S+2}) + \cdots + x_{3S}h(x_St_i + x_{2S}) + x_{3S+1} \quad 0$$

şeklinde S adet düğümden oluşan modelde $h(x)$ fonksiyonu olarak aşağıdaki gibi hiperbolik-tanjant-sigmoid fonksiyon kullanılabilir:

$$h(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad 0$$

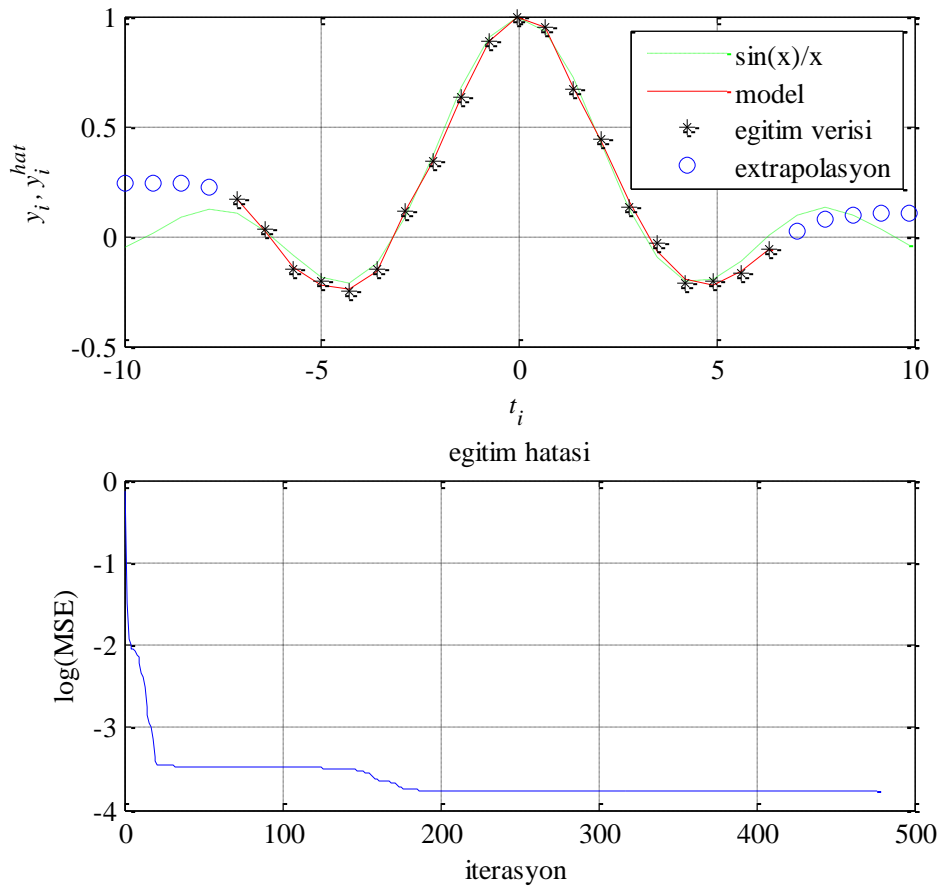
Aktivasyon fonksiyonu olarak hiperbolik-tanjant-sigmoid kullanılmasının en önemli nedeni, bu fonksiyonun türevinin aşağıdaki gibi basitçe elde edilebilmesidir:

$$\frac{dh(x)}{dx} = 1 - \left(\frac{e^x - e^{-x}}{e^x + e^{-x}} \right)^2 = 1 - (h(x))^2 \quad 0$$

Bu model için \mathbf{J} matrisine ilişkin terimler şu şekildedir:

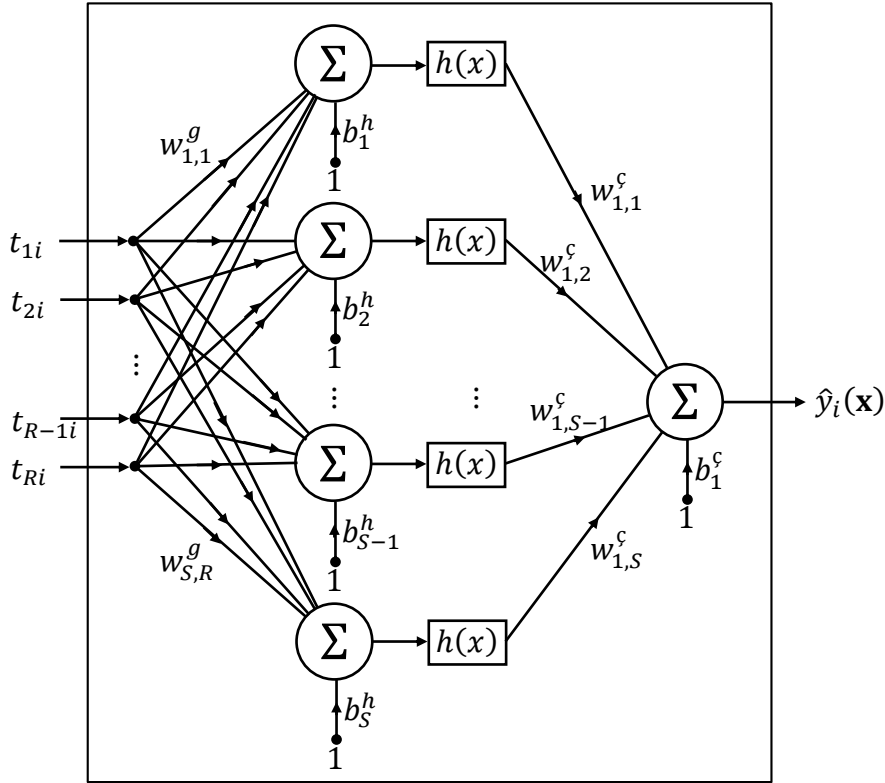
$$\begin{aligned}
\frac{\partial e_i(\mathbf{x})}{\partial x_j} &= \frac{\partial (y_i - \hat{y}_i(\mathbf{x}))}{\partial x_j} \\
&= - \frac{\partial \hat{y}_i(\mathbf{x})}{\partial x_j} \\
&= - \frac{\partial (x_{2S+1}h(x_1t_i + x_{S+1}) + \dots + x_{3S}h(x_St_i + x_{2S}) + x_{3S+1})}{\partial x_j} \\
&= - \begin{cases} x_{j+2S}t_i (1 - [h(x_jt_i + x_{S+j})]^2) & j = 1, \dots, S \\ x_{j+S} (1 - [h(x_{j-S}t_i + x_j)]^2) & j = S+1, \dots, 2S \\ h(x_{j-2S}t_i + x_{j-S}) & j = 2S+1, \dots, 3S \\ 1 & j = 3S+1 \end{cases}
\end{aligned}$$

Örnek: $\frac{\sin(x)}{x}$ fonksiyonundan elde edilmiş verilerle eğitilmiş 6-nöronlu bir SISO-YSA modelinin çıktısı aşağıdadır:



5.3 MISO-YSA Modeli

Belli sayıda çok-girişli nöron modelinin bir araya gelmesiyle elde edilen MISO-YSA yapısı şekilde görüldüğü gibidir.



Bu YSA yapısının giriş-çıkış ilişkisi,

$$\begin{aligned} \hat{y}_i(\mathbf{x}) = & w_{1,1}^c h(w_{1,1}^g t_{1i} + w_{1,2}^g t_{2i} + \dots + w_{1,R}^g t_{Ri} + b_1^h) \\ & + w_{1,2}^c h(w_{2,1}^g t_{1i} + w_{2,2}^g t_{2i} + \dots + w_{2,R}^g t_{Ri} + b_2^h) \\ & + \dots \\ & + w_{1,S}^c h(w_{S,1}^g t_{1i} + w_{S,2}^g t_{2i} + \dots + w_{S,R}^g t_{Ri} + b_S^h) \\ & + b_1^c \end{aligned}$$

0

Bu giriş-çıkış ilişkisi daha modüler bir şekilde ifade edilirse,

$$\hat{y}_i(\mathbf{x}) = \mathbf{W}^c \mathbf{h}(\mathbf{W}^g \mathbf{t}_i + \mathbf{b}^h) + b_1^c$$

0

şeklinde bir ifade yazılabilir, buradaki $\mathbf{h}(\cdot)$ büyüklüğü her bir düğümün çıkışındaki aktivasyon fonksiyonlarından oluşan bir vektördür, matrissel büyüklükler de aşağıdaki gibidir:

$$\mathbf{W}^g = \begin{bmatrix} \mathbf{w}_1^g \\ \mathbf{w}_2^g \\ \vdots \\ \mathbf{w}_S^g \end{bmatrix} = \begin{bmatrix} w_{1,1}^g & w_{1,2}^g & \cdots & w_{1,R}^g \\ w_{2,1}^g & w_{2,2}^g & \cdots & w_{2,R}^g \\ \vdots & \vdots & \ddots & \vdots \\ w_{S,1}^g & w_{S,2}^g & \cdots & w_{S,R}^g \end{bmatrix}, \mathbf{b}^h = \begin{bmatrix} b_1^h \\ b_2^h \\ \vdots \\ b_S^h \end{bmatrix}, \mathbf{W}^\zeta = [w_{1,1}^\zeta \quad w_{1,2}^\zeta \quad \cdots \quad w_{1,S}^\zeta]. \quad 0$$

Bu yapıda modelde aktivasyon fonksiyonu olarak hiperbolik-tanjant-sigmoid fonksiyon kullanılabilir. MISO-YSA yapısının ayarlanabilir parametrelerini tüm ağırlık ve bias terimleri oluşturmaktadır. Dolayısıyla bir güncelleme yaparken, tüm ayarlanabilir parametrelerin \mathbf{x} parametre vektörüne aktarılması gerekir. Önce bu aktarmayı yapıp ardından bu model için \mathbf{J} matrisine ilişkin terimleri bulalım. Tüm ayarlanabilir parametrelerin \mathbf{x} parametre vektörüne aktarılması aşağıdaki gibi belli bir düzene göre yapılmalıdır:

$$\mathbf{x} = [x_j] = \begin{bmatrix} w_{1,1}^g & j = 1 \\ \vdots & \vdots \\ w_{S,1}^g & j = S \\ w_{1,2}^g & j = S + 1 \\ \vdots & \vdots \\ w_{S,2}^g & j = 2S \\ \vdots & \vdots \\ w_{1,R}^g & j = (R - 1)S + 1 \\ \vdots & \vdots \\ w_{S,R}^g & j = SR \\ b_1^h & j = SR + 1 \\ \vdots & \vdots \\ b_S^h & j = SR + S \\ w_{1,1}^\zeta & j = S(R + 1) + 1 \\ \vdots & \vdots \\ w_{1,S}^\zeta & j = S(R + 2) \\ b_1^\zeta & j = S(R + 2) + 1 \end{bmatrix} \quad 0$$

Şimdi MISO-YSA modeli için \mathbf{J} matrisine ilişkin terimleri bulalım.

$$\begin{aligned}
\frac{\partial e_i(\mathbf{x})}{\partial x_j} &= \frac{\partial (y_i - \hat{y}_i(\mathbf{x}))}{\partial x_j} \\
&= - \frac{\partial \hat{y}_i(\mathbf{x})}{\partial x_j} \\
&= - \begin{cases} w_{1,k}^c t_{m,i} \left(1 - [h(\mathbf{w}_k^g \mathbf{t}_i + b_k^h)]^2 \right) & j = 1, \dots, SR \\ w_{1,(j-SR)}^c \left(1 - [h(\mathbf{w}_{j-SR}^g \mathbf{t}_i + b_{j-SR}^h)]^2 \right) & k = \text{mod}(j-1, S) + 1 \\ h(\mathbf{w}_{j-(R+1)S}^g \mathbf{t}_i + b_{j-(R+1)S}^h) & m = \text{fix}((j-1)/S) + 1 \\ 1 & j = SR + 1, \dots, SR + S \\ & j = S(R+1) + 1, \dots, S(R+2) \\ & j = S(R+2) + 1 \end{cases} \quad ()
\end{aligned}$$

buradaki mod fonksiyonu standart mod alma fonksiyonudur, fix fonksiyonu da argümanının sadece tamsayı kısmını döndüren bir fonksiyondur.

YSA için Levenberg-Marquardt Algoritması

- ◆ Öğrenilecek veri kümesini al
- ◆ Algoritmaya ilişkin parametre değerlerini başlat
 - nöron sayısı
 - maksimum iterasyon sayısı (maxite)
 - minimum MSE (MinTraErr)
 - $\|s_k \mathbf{p}_k\|$ güncelleme vektörünün minimum normu (NORMG)
 - $\mu_{\min}, \mu_{\max}, \mu_{scal} = 10$
- ◆ YSA'ya ilişkin ağırlık ve bias matrislerini sıfır civarında rasgele başlat
- ◆ iterasyon sayacını $k=1$ yap
- ◆ Veri kümesinin her bir giriş değeri \mathbf{t}_i 'ne karşı YSA'nın cevabı $\hat{y}_i = \hat{y}(\mathbf{x}_k, \mathbf{t}_i)$ 'nı bul
- ◆ Hata vektörü $\mathbf{e}(\mathbf{x}_k)$ ve amaç fonksiyonunun değeri $f(\mathbf{x}_k)$ bul
- ◆ loop1=1; $\mu_k = 1$
- ◆ while loop1
 - $k=k+1; \mu_{k+1} = \mu_k$
 - YSA matrislerini \mathbf{x}_k vektörüne aktar
 - Jacobian Matrisi $\mathbf{J}(\mathbf{x}_k)$ 'i hesapla
 - loop2=1;
 - while loop2
 - $\mathbf{p}_k = -[\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k) + \mu_k \mathbf{I}]^{-1} \mathbf{J}^T(\mathbf{x}_k)\mathbf{e}(\mathbf{x}_k)$ % aday yön
 - $\mathbf{z}_k = \mathbf{x}_k + \mathbf{p}_k$
 - \mathbf{z}_k vektörünü YSA matrislerine aktar
 - Veri kümesinin her bir giriş değeri \mathbf{t}_i 'ne karşı YSA cevabı $\hat{y}(\mathbf{z}_k, \mathbf{t}_i)$ 'nı bul.
 - Hata vektörü $\mathbf{e}(\mathbf{z}_k)$ ve amaç fonksiyonunun değeri $f(\mathbf{z}_k)$ bul
 - if $f(\mathbf{z}_k) < f(\mathbf{x}_k)$
 - $f(\mathbf{x}_k + s\mathbf{p}_k)$ fonksiyonunu s 'e göre minimum yapan s_k 'yı bul (opsiyonel)
 - güncelle
 - $\mathbf{x}_{k+1} = \mathbf{x}_k + s_k \mathbf{p}_k$
 - \mathbf{x}_{k+1} vektörünü YSA matrislerine aktar
 - $\mu_k \leftarrow \frac{\mu_k}{\mu_{scal}}$
 - loop2=0
 - else
 - $\mu_k \leftarrow \mu_k \cdot \mu_{scal}$
 - end
 - if $(\mu_k < \mu_{\min}) | (\mu_{\max} < \mu_k)$
 - loop1=0
 - loop2=0
 - end
 - end
 - if $(f(\mathbf{x}_{k+1}) < \text{MinTraErr}) | (\|s_k \mathbf{p}_k\| < \text{NORMG}) | (\text{maxite} < k)$
 - loop1=0;
 - end
 - end
 - end

5.4. Çok-boyutlu Veri Modelleme

Aşağıdaki tabloda görüldüğü gibi, $\mathbf{t}_i \in \mathbb{R}^R$ ve $y_i \in \mathbb{R}$ olmak üzere $\mathcal{D}_{\text{MISO}} = \{\mathbf{t}_i, y_i\}_{i=1}^N$ şeklinde verilen bir Çok-Girişli Tek-Çıkışlı (Multi-Input Single-Output-MISO) veriyi ele alalım:

$\mathcal{D}_{\text{MISO}}$	Giriş Verisi				Çıkış Verisi	Model Çıkışı	Model Hatası
i	\mathbf{t}_i				y_i	\hat{y}_i	$e_i = y_i - \hat{y}_i$
	t_{i1}	t_{i2}	...	t_{iR}			
1	t_{11}	t_{12}	...	t_{1R}	y_1	\hat{y}_1	e_1
2	t_{21}	t_{22}	...	t_{2R}	y_2	\hat{y}_2	e_2
\vdots	\vdots	\vdots	\ddots	\vdots	\vdots	\vdots	\vdots
N	t_{N1}	t_{N2}	...	t_{NR}	y_N	\hat{y}_N	e_N

Bu veriler, $\hat{y} = \hat{y}(\mathbf{x})$ gibi bir yapı ile modellenenecektir. Bu durumda, Çok-Girişli Tek-Çıkışlı (Multi-Input Single-Output-MISO) modelleme probleminin tanımı şu şekildeki bir optimizasyon problemi ile yapılabilir: $\hat{y}(\mathbf{x})$ modelinin parametreleri olan \mathbf{x} vektörü öyle ayarlanmalı ki modelin her bir veri için yaptığı hataların karelerinin toplamı minimum olsun, yani,

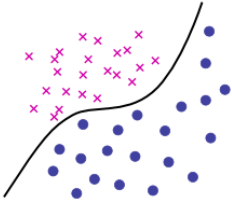
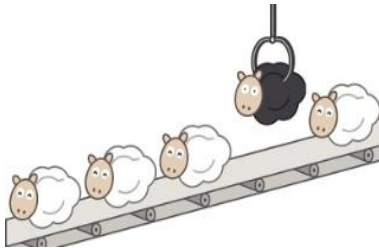
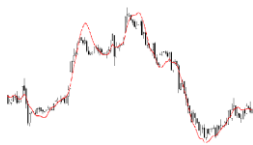
$$\min_{\mathbf{x}} f(\mathbf{x}) = e_1^2(\mathbf{x}) + e_2^2(\mathbf{x}) + \dots + e_N^2(\mathbf{x}) \quad (1)$$

burada her bir veri için modelin yaptığı hata $e_i(\mathbf{x}) = y_i - \hat{y}_i(\mathbf{x})$ şeklindedir. Eğer istenen çıkış vektörü $\mathbf{y} = [y_1 \ y_2 \ \dots \ y_N]^T$ ve modelin ürettiği çıkış vektörü $\hat{\mathbf{y}}(\mathbf{x}) = [\hat{y}_1(\mathbf{x}) \ \hat{y}_2(\mathbf{x}) \ \dots \ \hat{y}_N(\mathbf{x})]^T$ olmak üzere hata vektörü $\mathbf{e}(\mathbf{x}) = \mathbf{y} - \hat{\mathbf{y}}(\mathbf{x})$ şeklinde yazılırsa, problem aşağıdaki hale gelir:

$$\min_{\mathbf{x}} f(\mathbf{x}) = \mathbf{e}^T(\mathbf{x})\mathbf{e}(\mathbf{x}) \quad (2)$$

5.5 Uygulamalar

Uygulamalar sınıflandırma ve regresyon problemi olarak karşımıza çıkmaktadır. Uygulama alanları şu şekildedir:

Sınıflandırma 	Örüntü Tanıma (Pattern Recognition) {2→2, 5→5, 4→8, 0→0, 2→2, 7→7, 5→5, 1→1, 3→3, 0→0, 3→3, 9→9, 6→6, 2→2, 8→8, 2→2, 0→0, 6→6, 4→6, 1→1, 1→1, 7→7, 8→8, 5→5, 0→0, 4→4, 7→7, 6→6, 0→0, 2→2, 5→5, 3→3, 1→1, 5→5, 6→6, 7→7, 5→5, 8→8, 1→1, 9→9, 3→3, 6→6, 8→8, 0→0, 9→9, 3→3, 0→0, 3→3, 7→7, 4→4, 4→4, 7→7, 8→8, 0→0, 4→4, 1→1, 3→3, 7→7, 6→6, 4→4, 7→7, 2→2, 7→7, 2→2, 5→5, 2→2, 0→0, 9→9, 8→8, 9→9, 8→8, 1→1, 6→6, 4→4, 8→8, 5→5, 8→8, 0→0, 6→6, 7→7, 4→4, 5→5, 8→8, 4→4, 3→3, 1→1, 5→5, 1→1, 9→9, 9→9, 2→2, 4→4, 7→7, 3→3, 1→1, 9→9, 2→2, 9→9, 6→6}]]	El Yazısı Tanıma, Ses Tanıma, Yüz Tanıma, Parmak İzi Tanıma, İris Tanıma, Plaka Tanıma, Dudak Okuma, El Hareketleri Tanıma, Yürüyüş Tanıma, Üretim Hatası Belirleme, Sözdizimsel Örüntü Tanıma, DNA Dizilerinin Sınıflandırılması, Doğal Dil İşleme
	Karar Verme (Decision Making) 	Banka Kredisi Başvurusu Değerlendirme, Gen Mikrodizilimleri, Spam Mail Teşhisi, Uydu Fotoğrafı Yorumlama, Öğrenci Başarı Durumu Değerlendirme, Hava Tahmini, Davranış analizi, Kişilik Tanıma, EMG, EEG Sinyal Analizi, Beyin Aktiviteleri Analizi, Tıbbi Teşhis, Arama Motorları, Uyarlamalı Web Siteleri
Regresyon 	Modelleme	Zaman Serisi Modelleme, Süreç Modelleme
	Tahmin	Zaman Serisi Tahmini, Süreç Tahmini
	Kontrol	Model Tabanlı Kontrol

5.5.1 Regresyon

Aşağıdaki tabloda görüldüğü gibi, $\mathbf{t}_i \in \mathbb{R}^R$ ve $y_i \in \mathbb{R}$ olmak üzere $\mathcal{D}_{\text{MISO}} = \{\mathbf{t}_i, y_i\}_{i=1}^N$ şeklinde verilen bir Çok-Girişli Tek-Çıkışlı (Multi-Input Single-Output-MISO) veriyi ele alalım:

$\mathcal{D}_{\text{MISO}}$	Giriş Verisi				Çıkış Verisi	Model Çıkışı	Model Hatası
i	\mathbf{t}_i				y_i	\hat{y}_i	$e_i = y_i - \hat{y}_i$
	t_{i1}	t_{i2}	...	t_{iR}			

1	t_{11}	t_{12}	...	t_{1R}	y_1	\hat{y}_1	e_1
2	t_{21}	t_{22}	...	t_{2R}	y_2	\hat{y}_2	e_2
\vdots	\vdots	\vdots	\ddots	\vdots	\vdots	\vdots	\vdots
N	t_{N1}	t_{N2}	...	t_{NR}	y_N	\hat{y}_N	e_N

Bu verilerin, $\hat{y} = \hat{y}(\mathbf{x})$ gibi bir yapı ile modellenmesi problemi tipik bir regresyon problemidir. Elde edilen regresyon modeli sayesinde giriş verisi ile çıkış verisi arasındaki ilişkiyi en iyi şekilde temsil etmeye çalışılır. Bu bir YSA ile yapıldıktan sonra YSA'ya uygulanan herhangi bir \mathbf{t}_i giriş test vektörüne karşı YSA

$$\hat{y}_i = \mathbf{W}^c \mathbf{h}(\mathbf{W}^g \mathbf{t}_i + \mathbf{b}^h) + b_1^c \quad ()$$

şeklinde çıkış üretecektir.

5.5.1.1 Süreç Modelleme ve Tahmini

Doğada bulunan veya sentetik bir şekilde oluşturulmuş bir süreçten $\mathbf{t}_i \in \mathbb{R}^R$ ve $y_i \in \mathbb{R}$ olmak üzere $\mathcal{D}_{\text{MISO}} = \{\mathbf{t}_i, y_i\}_{i=1}^N$ şeklinde elde edilen bir Çok-Girişli Tek-Çıkışlı (Multi-Input Single-Output-MISO) verisinin YSA ile modellenmesi ve herhangi bir test girişine karşı YSA'nın bir tahmin değeri üretmesi mümkündür.

5.5.1.2 Zaman Serisi Modelleme ve Tahmini

$n = 1, \dots, N$ için $z[n] \in \mathbb{R}$ olmak üzere $\mathcal{D}_{\text{TS}} = \{z[n]\}_{n=1}^N$ şeklinde verilen bir zaman serisini ele alalım. Burada n zaman indeksini göstermektedir. Bu zaman serisi herhangi bir büyüklüğün herhangi bir zaman aralığında yapılmış ölçümlerle elde edilmiş değerler olabilir. Örneğin bu değerler bir barajın su seviyesinin günlük eğreri veya biyolojik bir süreçteki bir canlı bakteri sayısının saatlik değerleri veya bir şirketin borsadaki günlük kapanış değerleri olabilir. Bu zaman serisini modelleme problemi aşağıdaki tablodaki gibi bir giriş-çıkış veri setine dönüştürülerek bir çok-boyutlu modelleme problemi olarak çözülebilir.

\mathcal{D}_{TS}	Giriş verisi				Çıkış verisi
i	\mathbf{t}_i				y_i
	t_{i1}	t_{i2}	...	t_{iR}	
1	$z[1]$	$z[2]$...	$z[R]$	$z[R + 1]$
2	$z[2]$	$z[3]$...	$z[R + 1]$	$z[R + 2]$
\vdots	\vdots	\vdots	\ddots	\vdots	\vdots
$N - R$	$z[N - R]$	$z[N - R + 1]$...	$z[N - 1]$	$z[N]$

Buradaki R parametresi, modelde kaç tane giriş olacağını ya da başka bir deyişle zaman serisinin belli bir andaki çıkışının o andan itibaren kaç adım öncesine kadar bağlı olduğunu göstermektedir.

\mathbf{t}_i giriş vektörü olmak üzere giriş-çıkış ilişkisi

$$\hat{y}_i = \mathbf{W}^c \mathbf{h}(\mathbf{W}^g \mathbf{t}_i + \mathbf{b}^h) + b_1^c \quad 0$$

şeklinde olan bir YSA, bu çok-boyutlu veri kümesi ile eğitilip uygun \mathbf{W}^c , \mathbf{W}^g , \mathbf{b}^h ve b_1^c parametreleri bulunduktan sonra zaman serisinin bir sonraki değeri olan $z[n + 1]$ değerini tahmin etmek için YSA'nın girişine

$$\mathbf{t}_{i+1} = \begin{bmatrix} z[N - R + 1] \\ z[N - R + 2] \\ \vdots \\ z[N] \end{bmatrix} \quad 0$$

şeklinde bir giriş vektörü uygulanır ve YSA'nın çıkışı

$$\hat{y}_{i+1} = \mathbf{W}^c \mathbf{h}(\mathbf{W}^g \mathbf{t}_{i+1} + \mathbf{b}^h) + b_1^c \quad 0$$

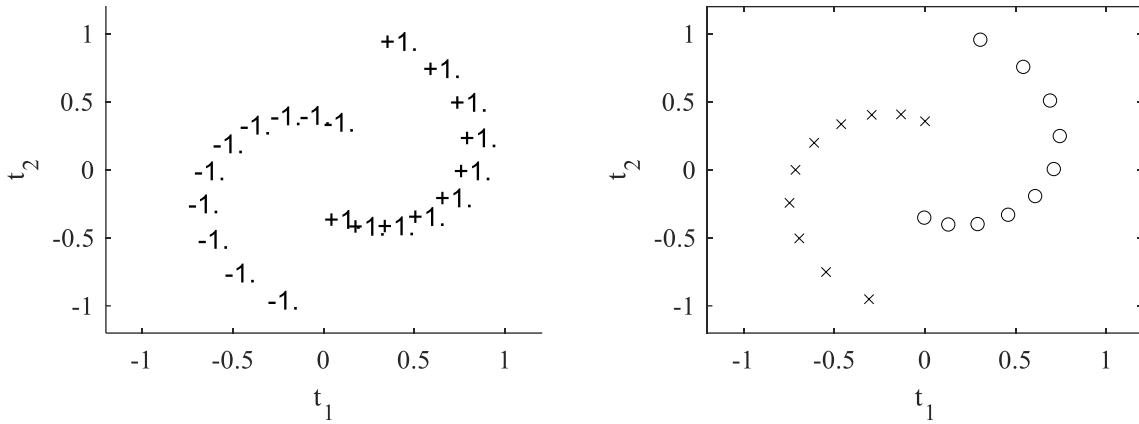
ilişkisiyle elde edilir. Elde edilen bu değer tahmini $z[n + 1]$ değeridir.

5.5.2 Sınıflandırma

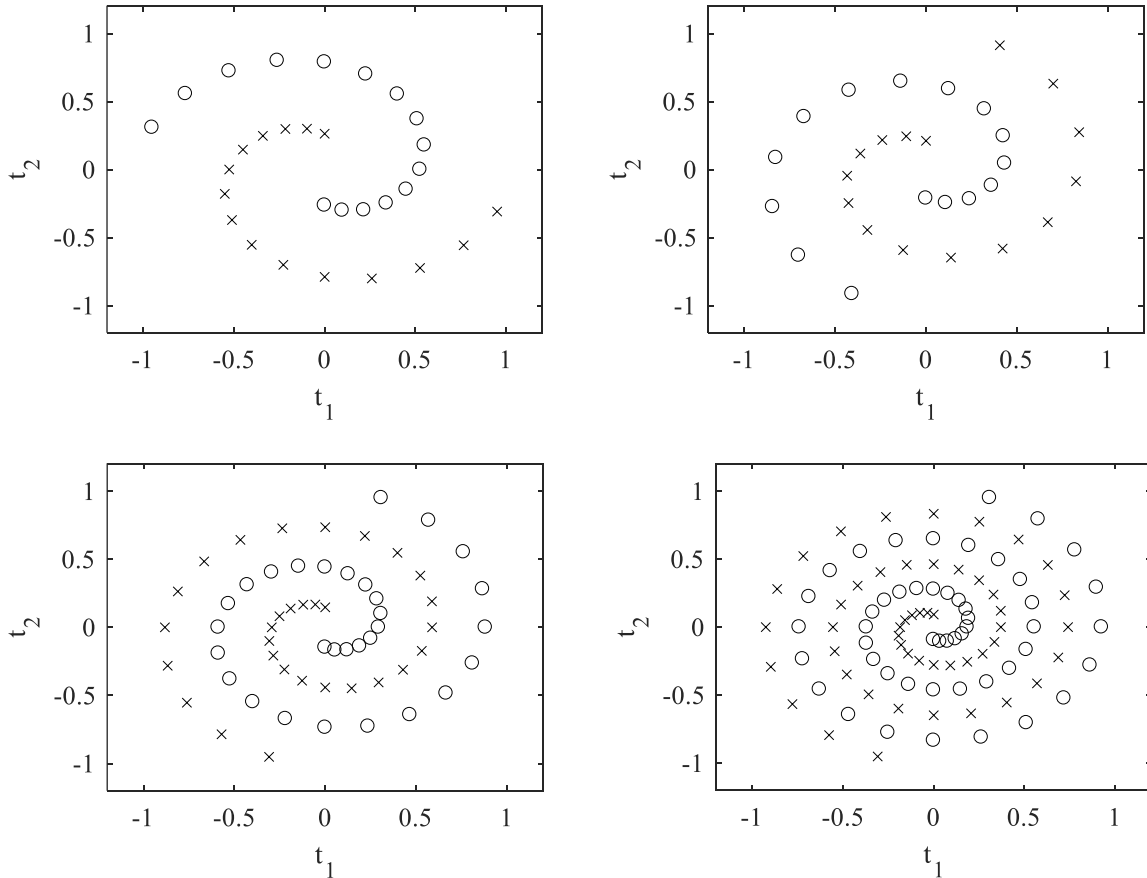
Aşağıdaki tabloda görüldüğü gibi, $\mathbf{t}_i \in \mathbb{R}^R$ ve $y_i \in \{-1 \quad +1\}$ olmak üzere $\mathcal{D}_{\text{MISO}} = \{\mathbf{t}_i, y_i\}_{i=1}^N$ şeklinde verilen bir Çok-Girişli Tek-Çıkışlı (Multi-Input Single-Output-MISO) veriyi ele alalım:

$\mathcal{D}_{\text{MISO}}$	Giriş Verisi				Çıkış Verisi	Model Çıkışı	Model Hatası
i	\mathbf{t}_i				y_i	\hat{y}_i	$e_i = y_i - \hat{y}_i$
	t_{i1}	t_{i2}	...	t_{iR}			
1	t_{11}	t_{12}	...	t_{1R}	y_1	\hat{y}_1	e_1
2	t_{21}	t_{22}	...	t_{2R}	y_2	\hat{y}_2	e_2
\vdots	\vdots	\vdots	\ddots	\vdots	\vdots	\vdots	\vdots
N	t_{N1}	t_{N2}	...	t_{NR}	y_N	\hat{y}_N	e_N

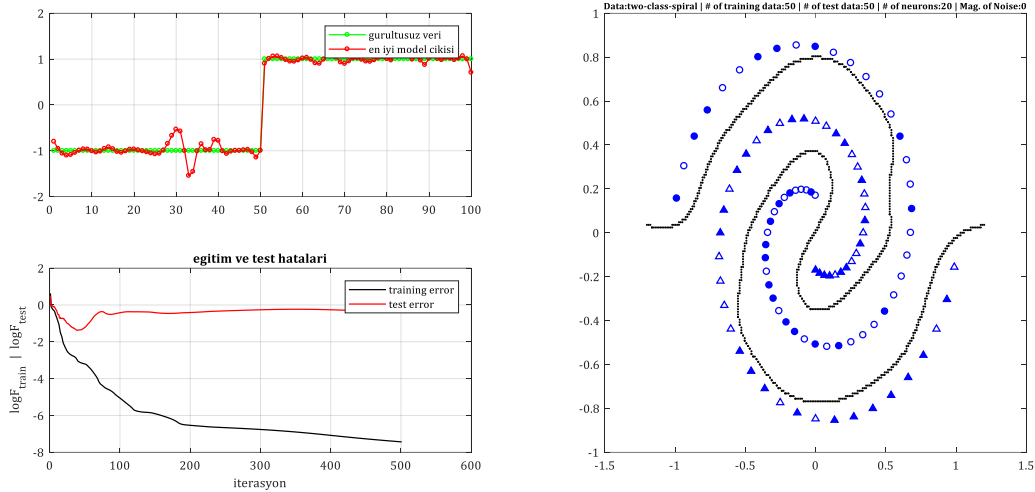
Bu verilerin, $\hat{y} = \hat{y}(\mathbf{x})$ gibi bir yapı ile modellenmesi problemi tipik bir sınıflandırma problemidir. Aşağıda tipik bir sınıflandırma problemi görülmektedir.



Burada problem, bu iki-sınıflı veriyi uygun bir çizgi ile birbirinden ayırmaktır. Kolaylık olması açısından giriş uzayının boyutu $R = 2$ olarak alınırsa, giriş verileri şekilde görülene benzer bir şekilde olacaktır. Burada içi boş yuvarlaklar (o) +1 sınıfı verileri, çarpılar (x) da -1 sınıfı verileri temsil etsin. Sağdaki şekilden de görüleceği gibi bu iki sınıfı birbirinden ayıran çok sayıda “Karar Çizgisi (KÇ)” mevcuttur. Bu tip problemler çok basit olabileceği gibi çok karmaşık da olabilir. Aşağıda farklı zorluklarda problemler görülmektedir.



Daha kolay görülebilmesi açısından bu problemlerde giriş uzayının boyutu $R = 2$ alınmıştır. Ama gerçek hayatta bu boyut çok daha yüksek değerlere çıkabilmektedir. Yine de, YSA ile bu problemi çözmek mümkündür.



Karmaşıklık Matrisi

Elde edilen sınıflandırma modelinin başarımını ölçmek için kullanılan bir metriktir. Aşağıdaki şekilde gösterilmektedir.

		TAHMİN	
		Positive	Negative
GERÇEK	Positive	TP	FP
	Negative	FN	TN

Burada,

TP (True Positive): Positive olarak sınıflanan ve gerçekten de Positive olan değerlerin sayısıdır.

FP (False Positive): Positive olarak sınıflanan fakat gerçekte Negative olan değerlerin sayısıdır.

TN (True Negative): Negative olarak sınıflanan ve gerçekten de Negative olan değerlerin sayısıdır.

FN (False Negative): Negative olarak sınıflanan fakat gerçekte Positive olan değerlerin sayısıdır.

Güvenilir doğruluk sonuçları elde etmek için karmaşıklık matrisi içindeki bu değerler kullanılarak *Accuracy*, *Precision*, *Recall* ve *F1score* olmak üzere dört farklı metrik kullanılmaktadır. Bu metrikler şu şekildedir:

Accuracy:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Tüm sınıfların (Positive and Negative) içerisinde, tam olarak kaç tanesinin doğru tahmin edildiğini gösterir. *Accuracy*, modelin doğru tahmin etme olasılığı gibi ele alınabilir. Doğru tahminlerin sayısının toplam tahmin sayısına bölünmesiyle hesaplanır.

Precision:

$$Precision = \frac{TP}{TP + FP}$$

Positive etiketli verilerin hangi oranda doğru tahmin edildiğini göstermek için kullanılır. *Precision*, doğru tahmin edilen Positive veri sayısının, tüm Positive veri sayısına oranı ile hesaplanır. Spam email tespiti gibi, yanlış sınıflandırılan bir Positive verinin (*FP*) maliyetinin yüksek olduğu yerlerde oldukça önemlidir. Böyle durumlarda, yüksek değerli *Precision* değerine sahip modeller tercih edilmelidir.

Recall:

$$Recall = \frac{TP}{TP + FN}$$

Positive olarak tahmin edilen verilerin hangi oranda gerçekten Positive olduğunu göstermek için kullanılır. *Recall*, doğru tahmin edilen Positive veri sayısının, Positive olarak tahmin edilen veri sayısına oranı ile hesaplanır. Kötü huylu kanser tespiti gibi, yanlış sınıflandırılan bir Negative verinin (*FN*) maliyetinin yüksek olduğu yerlerde oldukça önemlidir. Böyle durumlarda, yüksek değerli *Precision* değerine sahip modeller tercih edilmelidir.

F1score:

$$F1score = \frac{2 * Recall * Precision}{Recall + Precision}$$

Düşük *Precision* ve yüksek *Recall* değerlerine sahip bir model ile tam tersi değerlere sahip bir model arasında seçim yapmak zordur. Böyle durumlarda, *F1score* kullanılır. *F1score* metriği. *Recall* ve *Precision* metriklerini birlikte değerlendirir. Ekstrem değerleri daha fazla cezalandırmak için *Aritmetik Ortalama* yerine, *Harmonik Ortalama*'yı kullanır. *Precision* ve *Recall* metriklerini tek bir metriğe dönüştürür, yanlış Positive (*FP*) ve yanlış Negative (*FN*)

değerlerini dengelemek için onları ağırlıklandırır. Yüksek değerli *F1score* değerine sahip modeller tercih edilmelidir.

5.5.2.1 Örüntü Tanıma

5.5.2.2 Karar Verme

REFERANSLAR

1. Applied Optimization with MatLab Programming, P. Venkataraman, Wiley, 2009.
2. Numerical Optimization, Nocedal, Jorge, Wright, S., Springer Series in Operations Research and Financial Engineering, 1999.
3. Linear and Nonlinear Programming, Nash and Sofer, 1996.
4. Nonlinear Programming , by D.P. Bertsekas, Athena Scientific, 2000.