YAŞAR ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

DOKTORA TEZİ

BLACK-SCHOLES KISMİ DİFERANSİYEL DENKLEMİNİN YAPAY SİNİR AĞLARI İLE ÇÖZÜMÜ ÜZERİNE

SAADET ESKİİZMİRLİLER

TEZ DANIŞMANI: DR. ÖĞR. ÜYESİ REFET POLAT
İKİNCİ DANIŞMAN: DR. ÖĞR. ÜYESİ KORHAN GÜNEL

MATEMATİK DOKTORA PROGRAMI

SUNUM TARİHİ: 04.10.2019

BORNOVA / İZMİR
EKİM 2019

Jüri üyeleri olarak bu tezi okuduğumuzu ve kapsam ve kalite bakımından Doktora tezi olarak uygunluğunu onaylıyoruz.

Jüri Üyeleri: **İmza:**

Dr. Öğr. Üyesi Refet POLAT (Danışman)
Yaşar Üniversitesi .....

Dr. Öğr. Üyesi Korhan GÜNEL (2.Danışman)
Adnan Menderes Üniversitesi .....

Prof. Dr. Urfat NURİYEV
Ege Üniversitesi .....

Doç. Dr. Ahmet YANTIR
Yaşar Üniversitesi .....

Dr. Öğr. Üyesi Burhan PEKTAŞ
Üsküdar Üniversitesi .....

Dr. Öğr. Üyesi Rıfat AŞLIYAN
Adnan Menderes Üniversitesi .....

Prof. Dr. Cüneyt GÜZELİŞ
Fen Bilimleri Enstitü Müdürü

ÖZ

BLACK-SCHOLES KISMİ DİFERANSİYEL DENKLEMİNİN YAPAY SİNİR AĞLARI İLE ÇÖZÜMÜ ÜZERİNE

Eskiizmirliler, Saadet
Doktora Tezi, Matematik Doktora Programı
Danışman: Dr. Öğr. Üyesi Refet POLAT
Yardımcı Danışman: Dr. Öğr. Üyesi Korhan GÜNEL
Ekim 2019

Black-Scholes modeli temettü ödemesi yapmayan Avrupa tipi opsiyonların fiyatlarını hesaplamak üzere 1973 yılında Fisher Black ve Myran Scholes tarafından geliştirilmiştir. Robert C. Merton'un yeni bir çözüm önerisi ile model literatürde Black-Scholes-Merton olarak isimlendirilmeye başlanmıştır. 1997 yılında bu çalışmaları sayesinde, Merton ve Scholes, Ekonomi alanında Nobel Ödülü almışlardır.

Yapay sinir ağları (YSA) biyolojik sinir sistemlerinden esinlenerek oluşturulan matematiksel modellerdir. Bu ağlar beyin yeteneklerine adaptif olmuştur ve bu durum makineler beyin gibi öğrenir olarak açıklanabilir. Yapay sinir ağları kullanılarak diferansiyel denklemlerin çözülmesi 1990'lı yıllarda başlamış ve son dönemlerde artmıştır.

Bu çalışmadaki amaç, Black-Scholes Probleminin YSA yöntemi ile yaklaşık çözümünü bulmak ve literatürde yer alan yaklaşık analitik çözümü ile karşılaştırmaktır.

Anahtar sözcükler: black-scholes denklemi, yapay sinir ağları, parçacık sürü optimizasyonu, sobol dizileri, halton dizileri, gradyan iniş algoritması

ABSTRACT

SOLUTIONS WITH ARTIFICIAL NEURAL NETWORKS ON BLACK-SCHOLES PARTIAL DIFFERENTIAL EQUATIONS

Eskiizmirliler, Saadet

Ph. D, Mathematics

Advisor: Asst. Prof. Dr. Refet POLAT

Co-Advisor: Asst. Prof. Dr. Korhan GÜNEL

October 2019

The Black-Scholes model was developed by Fisher Black and Myran Scholes in 1973 to calculate the prices of European options that do not pay dividend payout. After Robert C. Merton's proposition for a new solution, the model has been started to be called Black-Scholes-Merton in the literature. In 1997, Merton and Scholes received the Nobel Prize in Economics for their work. Artificial Neural Networks (ANN) are mathematical models inspired by biological nervous systems. These networks have been adaptive to brain capabilities, thus it can be concluded that the machines learn like the brain. Solving differential equations using ANN started in the 1990s and increased in recent years.

The aim of this study is to find the approximate solution of Black-Scholes Problem using ANN method and compare it to the approximate analytical solution within the literature.

Keywords: black scholes equation, artificial neural network, particle swarm optimization, sobol sequence, halton sequence, gradient descent algorithm

TEŞEKKÜR

Bu tez çalışmasının planlanmasında, araştırılmasında, yürütülmesinde ve oluşumunda ilgi ve desteğini esirgemeyen, engin bilgi ve tecrübelerinden yararlandığım, yönlendirme ve bilgilendirmeleriyle çalışmamı bilimsel temeller ışığında şekillendiren sayın hocam Dr. Refet POLAT'a ve sayın hocam Dr. Korhan GÜNEL'e, tezimin yazımı esnasında desteğini esirgemeyen çok değerli hocam Doç. Dr. Burhan PEKTAŞ'a sonsuz teşekkürlerimi sunarım.

Bana olan güvenlerini hiç kaybetmeyen, destekleriyle beni hiçbir zaman yalnız bırakmayan aileme de sonsuz teşekkür ederim.

Saadet Eskiizmirliler

İzmir, 2019

Teyzeme…

YEMİN METNİ

Doktora Tezi olarak sunduğum "BLACK-SCHOLES DENKLEMLERİNİN YAPAY SİNİR AĞLARI İLE ÇÖZÜMÜ ÜZERİNE" adlı çalışmanın, tarafımdan bilimsel ahlak ve geleneklere aykırı düşecek bir yardıma başvurmaksızın yazıldığını ve yararlandığım eserlerin bibliyografyada gösterilenlerden oluştuğunu, bunlara atıf yapılarak yararlanılmış olduğunu belirtir ve bunu onurumla doğrularım.

Saadet Eskiizmirliler

İMZA

…….

15 Ekim 2019

İÇİNDEKİLER

ŞEKİL LİSTESİ

TABLO LİSTESİ

SİMGE VE KISALTMALAR

σ	Volatilite
r	Faiz
Ω	Problemin tanım bölgesi
$\partial\Omega$	Bölgenin sınırı
BS	Black-Scholes
BSP	Black-Scholes Problemi
E	Uygulama fiyatı
PSO	Parçacık Sürü Optimizasyonu
PSO_SOBOL	Sobol Dizileri ile PSO
PSO_HALTON	Halton Dizileri ile PSO
S	Hisse Senedinin veya Malın Cari İşlem Fiyatı
T	Uygulama tarihi
V(S,T)	Opsiyon değeri
YSA	Yapay Sinir Ağı

BÖLÜM 1
GİRİŞ

Matematikte herhangi bir değişkenin bilinmeyen bir fonksiyonu ve bu fonksiyonun o değişkene göre çeşitli basamaklardan türevlerini içeren denkleme diferansiyel denklem denir. Diferansiyel denklemler bağımsız değişkenlerin sayısına ve içerdikleri türevlerin türlerine göre adi diferansiyel denklem veya kısmi diferansiyel denklem olarak sınıflandırılırlar. Adi diferansiyel denklemler tek bir bağımsız değişken içeren denklemler, kısmi diferansiyel denklemler ise birden fazla bağımsız değişken içeren denklemlerdir.

Günlük hayatta işlerimizi kolaylaştıran ve farkında olmadan defalarca karşılaştığımız birçok sistem diferansiyel denklemler ile modellenmektedir. Örneğin, Fizikte homojen olmayan ortamlarda enerji dağılımları (J.E. Macias-Diaz, Héctor Vargas-Rodriguez, 2018), Sosyal bilimlerde yapı tasarımında karşılaşılan bazı sorunların çözümlenmesi (Jiri Vala, Petra Jarosova, Brno, 2018), Tıpta, glukoz-insülin-karbonhidratlar dinamiği (Eduardo Ruiz Velazquez, Oscar D. Sanchez, Griselda Quiroz, Guillermo O.Pulido (2018) vb. modellenmesinde diferansiyel denklemler kullanılmaktadır.

Ekonomide görülen belirsizlikler ve buna bağlı istikrarsızlıkların doğurduğu risk, finans sektörü ile ilgili modellerde de diferansiyel denklemlerin kullanılmasına yol açmıştır. Bu riskleri azaltmak için, temettü ödemesi olmayan Avrupa tipi opsiyonların fiyatlarını hesaplamak üzere 1973 yılında Fisher Black ve Myran Scholes tarafından Black-Scholes denklemi tanımlanmıştır (F. Black, M. Scholes, 1973).

Literatürde yapay sinir ağları kullanılarak diferansiyel denklemlerin çözülmesi 1990'lı yıllarda başlamış ve son dönemlerde artmıştır. Yapılan çalışmalara örnek olarak; H. Lee ve I. Kang (1990), çalışmalarında sonlu fark eşitlikleriyle çözülebilen diferansiyel denklem çözümünü yapay nöron kullanarak minimumlaştırma algoritmalarıyla çözmüştür. L.P. Aarts ve P. Van Der Veer (2001), yüksek mertebeden kısmi türevli diferansiyel denklemlerin çözümünde ileri beslemeli ağ kullanmıştır. A. Malek ve R.S.

Beidokhti (2006), YSA'lar ile yüksek mertebeli diferansiyel denklemlerin çözümü için optimizasyon yaklaşımına dayalı hibrit metot önermiştir. K. S. McFall ve J.R. Mahan (2009), sınır değer problemleri için YSA modeli oluşturmuş ve iki ile üç boyutlu, lineer ve lineer olmayan diferansiyel denklemlerin çözümü için öneride bulunmuştur. A. Malek ve R.S. Beidokhti (2009), keyfi başlangıç ve sınır koşullarını içeren kısmi türevli diferansiyel denklem sistemlerinin çözümü için YSA'ları kullanmıştır.

M. Otadi ve M. Mosleh (2011), Riccati diferansiyel denkleminin çözümünü elde etmek için ileri beslemeli geri yayılımlı yapay sinir ağı kullanmışlardır. M.A.Z. Raja ve S. Ahmad (2014), Bratu tipteki diferansiyel denklemin çözümünde yapay sinir ağlarını kullanmıştır. Önerilen metotta yer alan yapay sinir ağı için logaritmik sigmoid, radyal tabanlı ve tanjant sigmoid fonksiyonları tercih edilmiştir. M.A.Z. Raja, S. Ahmad ve S.S. Raza (2014), ileri beslemeli yapay sinir ağı modelinin kullanılmasıyla 2-boyutlu Bratu eşitliklerini çözerek, parçaçık sürü optimizasyonu (Particle Swarm Optimization) ve ardışık kuadratik programlama (Sequential Quadratic Program) optimizasyon metotları ile optimize etmiştir. M. Kumar ve N. Yadav (2015), Bratu tipindeki diferansiyel denklemin çözümünde Çok Katmanlı Algılayıcı modeli tercih ederek optimal çözüme ulaştıklarını iddia etmişlerdir. Çalışmalar günümüzde yoğun bir şekilde devam etmektedir.

Son dönemde YSA ile BS denkleminin çözümü üzerine yapılan çalışmaları inceleyecek olursak; Xiaoquan Liu, Yi Cao, Chenghu Ma ve Liya Shen (2018) Wavelet tabanlı opsiyon fiyatlandırma modellerinde yapay sinir ağları dışındaki modeller ile kendi uyguladıkları yapay sinir ağı modelini karşılaştırmışlardır. Angela Slavova ve Nikolay Kyurkchiev (2018) Black-Scholes denkleminin klasik volatilite (dalgalanırlık) katsayısı yerine "leland correction" katsayısı kullanarak hücresel yapay sinir ağı modelini mathematica programı ile çözmüşlerdir. Heesung Yun, Sangseop Lim ve Kihwan Lee (2018) Charter seferlerinin iyileştirilmesi için Black-Scholes modeli ile yapay sinir ağları kullanarak gerçek verilerle karşılaştırmışlardır. Huisu Jang ve Jaewook Lee (2019) Generative Bayesian yapay sinir modelini kullanarak, 2003 ve 2012 yılları arasındaki (S&P 100 Amerikan Satma Opsiyon Verileri) opsiyon fiyatlandırmaları verileri ile karşılaştırma yapmışlardır. Shuaiqiang Liu, Cornelis W. Oosterlee ve Sander M. Bohte (2019) Veri-güdümlü yaklaşımın yapay sinir ağları ile bir finansal opsiyon hesaplamasını ve diğer nümerik yöntemler ile karşılaştırmasını yapmışlardır.

Bu çalışmanın 2. Bölümünde Black-Scholes denklemi ve genelleştirilmiş hali ile genel tanımlar ve teoremler, 3. Bölümünde İleri Beslemeli Yapay Sinir Ağları (YSA), YSA'nın genel yapısı, Black-Scholes Probleminin YSA ile çözümünde kullanılan optimizasyon yöntemleri ayrıntılı bir şekilde anlatılmıştır. 4. Bölümde ise literatürden farklı olarak 3. Bölümde ayrıntılı bir şekilde bahsedilen optimizasyon yöntemleri ile Black-Scholes opsiyon fiyatlandırma modelinin ve daha genel bir yapısının Yapay Sinir ağları ile çözümünde elde edilen sonuçlar yer almaktadır. Son bölümde ise sonuç ve tartışmalara yer verilmiştir.

BÖLÜM 2
BLACK-SCHOLES PROBLEMİ İLE İLGİLİ TEMEL TANIMLAR VE TEOREMLER

2.1. Vadeli İşlemler ve Opsiyonlar

Vadeli işlemlere ilişkin sözleşmeler ve opsiyonlar, bir malın sözleşmede belirlenen fiyattan, sözleşmede belirtilen tarihte alım-satımını sağlayan anlaşmalardır. Vadeli işlem sözleşmeleri, sözleşmenin taraflarına; belirli bir vadede, önceden belirlenen fiyat, nicelik ve nitelikteki malı, kıymetli madeni, finansal göstergeyi, sermaye piyasası aracını ya da yabancı para birimini bugünden üzerinde anlaşılan fiyattan alma ya da satma yükümlülüğü veren sözleşmelerdir.

Vadeli işlemlerle opsiyonlar arasındaki en büyük fark, vadeli işlemlerde malı alan ve satan taahhüt altındadır. Opsiyonlar ise iki taraf arasında yapılan ve satın alan tarafa herhangi bir ürünü bugünden belirlenen bir fiyat (kullanım fiyatı) üzerinden ileride bir vadede alma ya da satma hakkını veren bir anlaşmadır. Opsiyonu satın alan, aldığı bu hak karşılığında satıcıya prim (opsiyon fiyatı) adı verilen bir tutar ödemek zorundadır. Dolayısıyla opsiyon sözleşmesi, alıcı tarafın kullanıp kullanmamakta serbest olduğu bir hak iken, satıcıyı yükümlülük altına sokmaktadır. İki ana tip opsiyon vardır. Avrupa tipi opsiyonlar; opsiyonu alan, sözleşmeye konu olan mal veya kıymeti alma veya satma hakkını sadece vade bitim tarihinde kullanmasını sağlayan opsiyonlardır. Amerikan tipi opsiyonlar; vade sonu da dâhil olmak üzere opsiyonu alan tarafa satın alım tarihi ile vadesi arasındaki istediği herhangi bir süre içinde kullanma imkânını sağlayan opsiyonlardır. Bir başka deyişle, Amerikan tipi opsiyonlarda erken kullanım hakkı mevcuttur.

2.2. Black-Scholes Problemi

Black-Scholes fiyatlama modeli, temettü ödemesi içermeyen Avrupa tipi opsiyonların fiyatlarını hesaplamak üzere 1973 yılında Fisher Black ve Myran Scholes tarafından önerilmiştir. Robert C. Merton'un gerçekleştirdiği farklı bir çözüm önerisi ile model Black-Scholes-Merton olarak isimlendirilmeye başlamıştır. 1997 yılında Merton ve

Scholes, gerçekleştirdikleri bu çalışmalar sayesinde, Ekonomi alanında Nobel Ödülü almışlardır.

Black-Scholes Problemi aşağıda yer alan geri yönlü parabolik problem ile ifade edilir.

$$\begin{cases} \dfrac{\partial V}{\partial t} + \dfrac{1}{2}\sigma^2 S^2 \dfrac{\partial^2 V}{\partial S^2} + rS\dfrac{\partial V}{\partial S} - rV = 0, & 0 < S < \infty, 0 \le t \le T \\ V(0,t) = 0, & 0 \le t \le T \\ V(S,t) \sim S, & S \to \infty, 0 \le t \le T \\ V(S,T) = \max(S-E,0), & 0 < S < \infty \end{cases}$$

Burada;

$V(S,T)$: Opsiyon değeri (option price)

S: Hisse senedinin veya malın cari işlem fiyatı (current stock price)

E: Uygulama fiyatı (expire price)

T: Uygulama tarihi (expire date)

r: Risksiz faiz oranı (risk-free rate of interest),

$V(S,T) = \max(S-E,0)$ final koşulu, uygulama zamanında opsiyon değerinin maksimumunu ifade eder. Burada $\max(S-E,0)$ final değeri $P(S) := \max(S-E,0)$ şeklinde tanımlanacaktır.

Black-Scholes Probleminde ele alınan

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0, \quad 0 < S < \infty, 0 \le t \le T \tag{2.2.1}$$

denkleminde,

Malın cari fiyatına karşılık gelen S değişkeni yerine x,

Opsiyon değerini ifade eden $V(S,t)$ fonksiyonu yerine $V(x,t)$

Volatilite (dalgalanırlık) katsayısı σ^2 yerine A katsayısı,

Risksiz faiz oranını ifade eden r katsayısı yerine sırasıyla B ve C katsayıları alınırsa, Black-Scholes probleminin genel formu aşağıdaki şekilde olur;

$$\begin{cases} \dfrac{\partial V}{\partial t} + \dfrac{1}{2}Ax^2 \dfrac{\partial^2 V}{\partial x^2} + Bx\dfrac{\partial V}{\partial x} - CV = 0, & 0 < x < \infty, 0 \le t \le T \\ V(0,t) = 0, & 0 \le t \le T \\ V(x,t) \sim x, & x \to \infty, 0 \le t \le T \\ V(x,T) = P(x), & 0 < x < \infty \end{cases} \tag{2.2.2}$$

Burada $P(x) = \max(x - E, 0)$ dir. Bu durumda (2.2.2) genel formunda A, B, C katsayıları için uygun tanımlamalar yapılarak Black-Scholes probleminin gerçek finans durumlara karşılık gelen farklı yaklaşımlarını elde etmek mümkündür.

2.3. Black-Scholes Probleminin Çözümünün Varlık ve Tekliği

Ele alınan (2.2.2) problemi geri yönlü parabolik problemdir, yani başlangıç koşulu $t = 0$ anı yerine $t = T$ final anında verilmiştir.

Burada $t = T - \mathcal{T}$ dönüşümü yapılarak $t \in (0, T)$ zaman değişkeni $\mathcal{T} \in (0, T)$ zaman değişkenine dönüştürülürse, bu problem $V(x, \mathcal{T})$ fonksiyonuna göre aşağıdaki başlangıç-değer problemine dönüşür.

$$\begin{cases} V_{\mathcal{T}} = BxV_x + \frac{1}{2}Ax^2V_{xx} - CV, \ 0 < x < \infty, \ 0 \le \mathcal{T} \le T \\ V(0, \mathcal{T}) = 0, \\ V(x, 0) = P(x) \end{cases} \tag{2.3.1}$$

Bu problemin çözümünün varlık ve tekliği, Cauchy-Kovalevskaya (Courant, R. and Hilbert, D.Methods of Mathematical Physics, Vol.1.New York: Wiley, 1989) teoreminden elde edilir.

Teorem 2.1. $P(x)$ başlangıç verisi $x = 0$ noktası civarında analitik olsun. O zaman (2.3.1) probleminin $x = 0, \ t = 0$ noktası civarında tek çözümü vardır ve bu çözüm bu nokta civarında analitiktir.

İspat: $V_{\mathcal{T}} = BxV_x + \frac{1}{2}Ax^2V_{xx} - CV, \quad 0 < x < \infty, \ 0 \le \mathcal{T} \le T$ denkleminde

$z_1 = V$, $z_2 = V_x$, $z_3 = V_{xx}$ dönüşümünü yapalım.

O zaman;

$V_{\mathcal{T}} = f(\mathcal{T}, x, z_1, z_2, z_3) = -Cz_1 + Bxz_2 + \frac{1}{2}Ax^2z_3$ denkleminin f sağ taraf fonksiyonu $(x = 0, t = 0)$ başlangıç noktasının her komşuluğunda analitiktir. Diğer taraftan $P(x) = \max\{x - E, 0\}$ başlangıç verisi $x = 0$ civarında analitik olduğundan "Cauchy-Kovalevskaya" teoremine göre ispat elde edilir. ■

BÖLÜM 3
MATERYAL VE METOTLAR

3.1. İleri Beslemeli Yapay Sinir Ağları (Feed Forward ANN)

Yapay sinir ağları (Neural Networks - NNs), insan beyninin özelliklerinden biri olan öğrenme yolu ile yeni bilgiler türetme, yeni bilgiler oluşturma gibi yetenekleri bir yardım almadan otomatik olarak gerçekleştirmek amacı ile geliştirilen, biyolojik sinir sistemlerinden esinlenlenerek oluşturulan, bilgisayar sistemleridir. Yapay sinir ağlarının başlangıcı nörobiyoloji bilimine insanların ilgi duyması ve elde ettikleri bilgileri bilgisayar bilimine uygulamaları ile başlamıştır.

Yapay sinir ağları, insanlar tarafından yapılan örnekleri kullanarak olayları öğrenebilen, çevredeki olaylara karşı nasıl tepkiler üretileceğini belirleyebilen bilgisayar sistemleridir. Biyolojik sinir ağları sinir hücrelerine sahiptir. Benzer şekilde yapay sinir ağları da yapay sinir hücrelerine sahiptir. Yapay sinir hücreleri mühendislik biliminde proses (process) elemanları olarak da isimlendirilmektedir. Her proses elemanının beş temel bileşeni vardır. Bu elemanlar;

Girdiler: Bir yapay sinir hücresine (proses elemanına) dışarıdan gelen bilgilerdir, bunlar yapay sinir ağının öğrenmesi istenen örnekleri tarafından belirlenir.

Ağırlıklar: Ağırlıklar bir yapay hücreye gelen bilginin hücre üzerindeki etkisini gösterir. Ağırlıkların büyük ya da göreceli olarak küçük olması bilginin önemli veya önemsiz olduğunu belirtmez. Ağırlıklar değişebilir veya sabit değerler alabilirler.

Toplama fonksiyonu: Bu fonksiyon, bir hücreye gelen net girdiyi hesaplar. Bunun için değişik fonksiyonlar kullanılmaktadır. En yaygın olanı ağırlıklı toplamı hesaplamaktır. Burada her gelen girdi değeri kendi ağırlığı ile çarpılarak toplanır ve ağa gelen net girdi hesaplanmış olur.

Aktivasyon Fonksiyonu: Bu fonksiyon hücreye gelen net girdiyi belirleyerek hücrenin bu girdi sonucunda üreteceği ilgili çıktıyı belirler. Aktivasyon fonksiyonu olarak çıktıyı hesaplamak için değişik fonksiyonlar kullanılmaktadır, çoğunlukla sigmoid fonksiyonu kullanılmaktadır.

Hücrenin Çıktısı: Aktivasyon fonksiyonu tarafından belirlenen değerdir, üretilen çıktı dış dünyaya veya başka bir hücreye iletilir.

YSA, insan beyni ve sinir ağlarından esinlenilerek matematiksel olarak modellenmiştir, oluşturulan matematiksel modeldeki yapay sinir hücresine "perceptron" adı verilir. Perceptron, nöron adı verilen "hesaplama birimlerinden" oluşur. Her nöron reel-değerli bazı "girdilere" sahiptir, her bir girdi buna karşılık gelen bir katsayı ile çarpılır ve bu çarpımların toplamı "bias" adı verilen bir değere eklenir. Hesaplanan bu değerler aktivasyon fonksiyonundan geçirilerek her bir nöronun "reel değerli çıktısı", ileri beslemeli olarak elde edilir. Birçok uygulamada aktivasyon fonksiyonu olarak sigmoid fonksiyonu kullanılır ancak farklı fonksiyonlar da kullanılabilir.

Öğrenme süreci optimal katsayıların (synaptic weights) bulunmasıdır. Eğer bu süreç bilinen çıktı ile hesaplanan çıktıların hatasının minimizasyonuna dayanıyorsa, buna öğretmenli öğrenme (supervised learning), aksi durumda ise yani bilinen çıktı yoksa buna öğretmensiz öğrenme (unsupervised learning) denir.

"çoklu-girdili-tek çıktılı" çok katmanlı ileri beslemeli ağlarda geri besleme döngüsü ve sonraki tabaka için nöronların çıktısı yoktur. Bu tür ağlar üç katmandan oluşur.

YSA yöntemi ile ele alınan problemlerin sayısal çözümlerinin elde edilmesi aşağıdaki teoremlere dayanır.

Teorem 3.1. $f: [0,1]^n \to R$ keyfi, sürekli fonksiyonu için $g: R \to R$ ve

$S_i: [0,1] \to [0,1]$, $(i = 0,1,\dots,2n)$ fonksiyonları ve $0 \leq V_j \leq 1$ $(j = 0,1,\dots,2n)$ sabitleri vardır, öyleki;

$f(x_1, x_2, \dots, x_n) = \sum_{i=0}^{2n} g\left[\sum_{j=1}^{n} V_j S_i(x_j) + b_i\right]$ dir. (Shekari Beidokhti, R.ve Malek, A.,2007)

Teorem 3.2. $f: [0,1]^n \to R$ keyfi, sürekli fonksiyonu ve $S: R \to R$ sınırlı fonksiyonu ve $\varepsilon > 0$ için, H doğal sayısı ve b_i, V_i, w_{ij} $(i = 1,2,\dots,H;\ J = 1,2,\dots,n)$ reel sabitleri vardır. Öyleki;

$\left|f(x_1, x_2, \dots, x_n) - \sum_{i=1}^{H} V_i S\left[\left(\sum_{j=1}^{n} w_{ij} x_j\right) + b_i\right]\right| < \varepsilon$ eşitsizliği sağlanır.

(Shekari Beidokhti, R.ve Malek, A.,2007)

3.2. Yapay Sinir Ağı Yönteminin Genel Yapısı

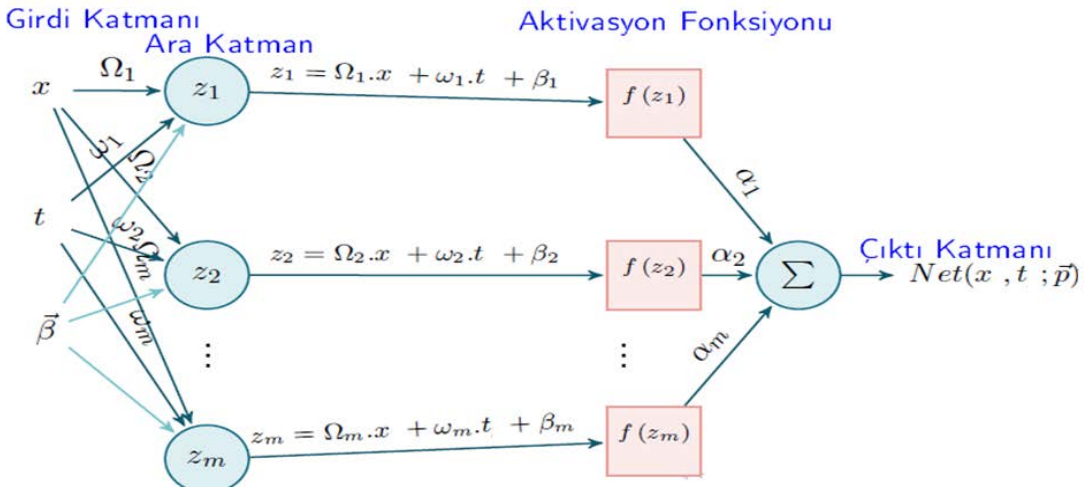
YSA yöntemi ile bir problemin yaklaşık (sayısal) çözümünün nasıl elde edilebileceğini aşağıda ele alınan başlangıç, sınır-değer problemi üzerinden açıklanabilir.

$$\begin{cases} \dfrac{\partial u}{\partial t} = F\left(t, x_1, \dots, x_n, \dfrac{\partial u}{\partial x_i}\right), x = (x_1, x_2, \dots, x_n) \in \Omega\,, t \geq t_0 > 0 \\ u(x, t_0) = I(x), \qquad x \in \Omega \\ B(x, t, u) = 0, \qquad x \in \partial\Omega\,, t \geq t_0 > 0 \end{cases}$$

Burada F, I ve B sırasıyla kaynak (source), başlangıç ve sınır fonksiyonlarıdır ve bilinen değerlerdir. $u(x_1, x_2, \dots, x_n, t)$ ise aranan çözüm fonksiyonudur. YSA yöntemi ile bu problemin yaklaşık çözümünün bulunması temelde iki aşamadan oluşur. Birinci aşama, çözüm fonksiyonunun problemde verilen $u = I$ başlangıç ve $B(u) = 0$ sınır koşullarını sağlayan yaklaşık çözüm fonksiyonu yani deneme fonksiyonunu (trial function) tanımlama kısmıdır. Deneme fonksiyonunun yapısı aşağıdaki şekildedir.

$$Y_t(x, t, \vec{p}) = Y_I(x, t, I) + (t - t_0) Y_B(x, t, B) Net(x, t; \vec{p}).$$

Burada $Y_I(x, t, I)$ fonksiyonu yaklaşık çözümün başlangıç koşulunu sağlayan kısmı, $Y_B(x, t, B)$ ise sınır koşulunu sağlayan kısımdır. $Net(x, t; \vec{p})$ ise YSA çıktı fonksiyonudur ve Teorem 3.1 ve Teorem 3.2'ye göre tanımlanır. Black-Scholes Probleminin çözümü için tezde önerilen YSA topolojisi Şekil 3.1'de verilmiştir.



Şekil 3.1. Black-Scholes Problemi için İleri Beslemeli YSA'nın şematik gösterimi

Şekil 3.1 ile verilen topolojide yapay sinir ağı çıktısı $\vec{p} = \vec{p}(\alpha, \omega, \Omega, \beta)$ içerisinde $(\alpha, \omega, \Omega, \beta)$ bilinmeyen parametrelerini bulunduran parametre vektörüdür. $\vec{p}$ vektörünün bilinmeyen bileşenlerinin değerleri, Y_t yaklaşık çözümünü u çözümüne yaklaştıracak şekilde belirlenebilir.

YSA yönteminin ikinci aşaması ise bu $\vec{p}$ bilinmeyeninin belirlenmesi aşamasıdır. Bunun için bazı optimizasyon araçları kullanılır. Bu aşamada, tanımlanan bir hata (penalty, cost) fonksiyonunun minimum değeri aşağıda tanımlanan minimum problemi üzerinden elde edilmeye çalışılır.

$$\vec{p}^{*}=\min E(x,t,\vec{p}),$$

$$E(x,t,\vec{p}) = \left\{ \int_{t_0}^{t_{max}} \int_{-\Omega} \left(\frac{\partial Y_t}{\partial t} - F\left(t, x_1, \dots, x_n, \frac{\partial Y_t}{\partial x_i}\right)\right)^2 dxdt \right\}^{1/2}$$

Bu formüldeki minimizasyon süreci, bu sinir ağındaki öğrenme prosedürü olarak algılanabilir. Bu süreçte bilinmeyen parametrelerin optimal değeri $\vec{p}^{*}$ elde edilir.

YSA yönteminin yukarıda özetlediğimiz iki aşamasını aşağıdaki parabolik ve adi diferansiyel denklemler için verilen başlangıç-değer problemleri üzerinde inceleyelim.

Örnek 3.1: Adi Diferansiyel Denklemler için başlangıç-değer problemi

$$\begin{cases} \dfrac{dy}{dt} = f(t,y), \;\; t \in [t_0, t_{max}] \\ y(t_0) = Y_0 \end{cases}$$

Problemi verilsin. Bu durumda yaklaşık çözüm aşağıdaki şekilde tanımlanır.

$$Y_t(t,\vec{p}) = Y_0 + (t - t_0) Net(t,\vec{p})$$

$Y_t(t,\vec{p})$ yaklaşık çözümünün başlangıç koşulunu sağladığı açıktır. O halde aşağıdaki minimizasyon problemi tanımlanır.

$$min\, E(\vec{p}) = min \left\{ \sum_{j=1}^{n} \left| \frac{dY_T(t_j,\vec{p})}{dt} - f\big(t_j, Y_T(t_j,\vec{p})\big) \right|^2 \right\} \rightarrow E(\vec{p}^{*})$$

Burada $\{t_j\}_{j=1}^{n}$, $[t_0, t_{max}]$ aralığındaki belirlenen kollokasyon noktalarıdır.

Örnek 3.2: Parabolik denklem için başlangıç-değer problemi

$$\begin{cases} u_t = ku_{xx} + F(x,t), & x \in (0,1), \qquad t \in (0,T) \\ u(x,0) = g(x), & x \in (0,1) \\ u(0,t) = 0, \qquad u(1,t) = 0, & t \in (0,T) \end{cases}$$

problemi verilmiş olsun. Uyum koşullarından $g(0) = g(1) = 0$ elde edilir. Bu problemin $u = u(x,t)$ çözümüne karşılık gelen deneme fonksiyonu aşağıdaki şekilde tanımlanır.

$$Y_T(x,t;\vec{p}) = g(x) + xt(x-1)Net(x,t;\vec{p})$$

$Y_T(x,t;\vec{p})$ deneme fonksiyonunun başlangıç ve sınır koşullarını sağladığı açıktır. Burada $\vec{p}$ bilinmeyen vektörünün bulunması için aşağıdaki minimum probleminin çözülmesi gereklidir.

$$\vec{p}^{*}=\min E(x,t,\vec{p}),$$

$$E(\vec{p}) = \left\{ \int_0^T \int_0^1 \left| \frac{\partial Y_T}{\partial t} - k\frac{\partial^2 Y_T}{\partial x^2} - F(x,t) \right|^2 dxdt \right\}^{1/2}$$

3.3. Black-Scholes Probleminin İleri Beslemeli Yapay Sinir Ağları ile Sayısal Çözümü

Black-Scholes Probleminin ileri beslemeli YSA yöntemi ile sayısal çözümü aşağıdaki şekilde elde edilir.

$$\begin{cases} \dfrac{\partial V}{\partial t} + \dfrac{1}{2}Ax^2\dfrac{\partial^2 V}{\partial x^2} + Bx\dfrac{\partial V}{\partial x} - CV = 0, & 0 < x < \infty, 0 \le t \le T \\ V(0,t) = 0, & 0 \le t \le T \\ V(x,t) \sim x, & x \to \infty, 0 \le t \le T \\ V(x,T) = P(x), & 0 < x < \infty \end{cases}$$

Burada $P(x) = \max(x - E, 0)$ dir. Ele alınan problemin YSA ile çözümünde temel nokta, problemin yaklaşık çözümünü ifade eden deneme (trial) fonksiyonu tanımlamaktır;

$$\varphi(x,t) = \frac{t}{T}P(x) + x(T-t)Net(x,t;\vec{p})$$

Burada;

- $\varphi(x,t)$ final ve sınır koşullarını sağlayan yaklaşık çözüm fonksiyonudur $\varphi(0,t)=0,\ \varphi(x,T)=P(x)$
- $Net(x,t;\vec{p})$, YSA ile elde edilen çıktı fonksiyonudur ve

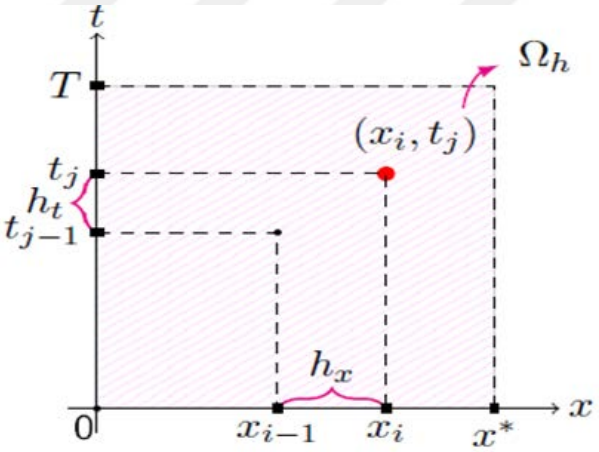
  $Net(x,t;\vec{p})=\sum_{k=1}^{H}\alpha_k f(z_k);\ z_k=w_k t+\Omega_k x+b_k$ şeklinde hesaplanır.
- $f(z)=1/(1+e^{-z})$ fonksiyonuna aktivasyon fonksiyonu denir.
- m, YSA'nın ara katmanında bulunan nöron sayısını belirtmek üzere, $\vec{\alpha},\vec{w},\vec{\Omega},\vec{b}\in\mathbb{R}^m$ için $\vec{p}=(\vec{\alpha},\vec{w},\vec{\Omega},\vec{b})$ bilinmeyen parametreler vektörüdür.

Black-Scholes Probleminin YSA ile çözümü için öncelikle Ω_H şebekesini oluşturalım.

Şebeke adım uzunlukları $h_x=x^*/N,\ h_t=T/M$ olmak üzere,

$$\Omega_H=\left[(x_i,t_j)\epsilon\Omega\middle|x_i=i.h_x,t_j=j.h_t,i=\overline{0,N},\ j=\overline{0,M}\right]$$

Ω_H de tanımlı "φ" deneme fonksiyonuna Black-Scholes probleminin YSA ile yaklaşık çözümü denir.



Şekil 3.2. Black-Scholes Problemi için sayısal çözüm şebekesinin oluşturulması

$$\varphi(x_i,t_j)=\frac{t_j}{T}P(x_i)+x_i(T-t_j)Net(x_i,t_j;\vec{p}),\quad (x_i,t_j)\epsilon\Omega_H$$

Burada Net fonksiyonu $k=1,2,\dots,m$ için $z_k=w_k t_j+\Omega_{ki}x_i+b_k$ olacak şekilde

$$Net(x_i,t_j;\vec{p})=\sum_{k=1}^{m}\alpha_k f(z_k)$$

yapısındadır.

Sayısal çözüm için aşağıdaki değer (cost) fonksiyoneli tanımlanır.

$$E(\vec{p}) = \frac{1}{2}\sum_{i=0}^{N}\sum_{j=1}^{M}\{e_{ij}\}^2, \qquad e_{ij} = \frac{\partial\varphi}{\partial t_j} + \frac{1}{2}A^2{x_i}^2\frac{\partial^2\varphi}{\partial {x_i}^2} + Bx_i\frac{\partial\varphi}{\partial x_i} - C\varphi(x_i, t_j)$$

$\overrightarrow{p_k} = (\overrightarrow{\alpha_k}, \overrightarrow{w_k}, \overrightarrow{\Omega_k}, \overrightarrow{b_k})$ bilinmeyen vektörü, bu değer (cost) fonksiyoneli üzerinden, ele alınan problem bir optimizasyon problemine dönüştürülmesi ile belirlenir.

Burada e_{ij} içindeki hesaplamalar aşağıdaki gibidir:

$$\frac{\partial\varphi}{\partial t} = max\{x - E, 0\} - x.Net(x, t; \vec{p}) + x(T - t)\frac{\partial Net(x, t; \vec{p})}{\partial t}$$

$$\frac{\partial Net(x, t; \vec{p})}{\partial t} = \sum_{k=1}^{m}\alpha_k \mathrm{w}_k f(z_k)\big(1 - f(z_k)\big)$$

$$\frac{\partial\varphi}{\partial x} = \begin{cases} (T - t)Net(x, t; \vec{p}) + x(T - t)\dfrac{\partial Net(x, t; \vec{p})}{\partial x}, x \le E \\ t + (T - t)Net(x, t; \vec{p}) + x(T - t)\dfrac{\partial Net(x, t; \vec{p})}{\partial x}, x > E \end{cases}$$

$$\frac{\partial^2\varphi}{\partial x^2} = 2(T - t)\frac{\partial Net(x, t; \vec{p})}{\partial x} + x(T - t)\frac{\partial^2 Net(x, t; \vec{p})}{\partial x^2}$$

$$\frac{\partial Net(x, t; \vec{p})}{\partial x} = \sum_{k=1}^{m}\alpha_k\Omega_k f(z_k)\big(1 - f(z_k)\big)$$

$$\frac{\partial^2 Net(x, t; \vec{p})}{\partial x^2} = \sum_{k=1}^{m}\alpha_k{\Omega^2}_k f(z_k)\big(1 - f(z_k)\big)\big(1 - 2f(z_k)\big)$$

Black-Scholes probleminin yaklaşık çözümünü bulma problemi aşağıdaki minimum problemine dönüştürülür.

$$\exists\vec{p}^*, E(\vec{p}^*) = minE(\vec{p})$$

Bu minimizasyon probleminin çözümü için literatürde farklı yaklaşımlar mevcuttur. Bu çalışmada türev içermeyen en bilinen optimizasyon yöntemlerinden olan "Parçacık Sürü Optimizasyonu (PSO)" yöntemi kullanılmıştır.

3.4. Parçacık Sürü Optimizasyonu

Parçacık Sürü Optimizasyonu (PSO) bir sürü temelli sezgisel optimizasyon yöntemidir. 1995 yılında Dr. Russel C. Eberhart ve Dr. James Kennedy tarafından geliştirilmiştir. PSO sürülerin, özellikle kuş sürülerinin davranışlarından esinlenilerek geliştirilen bir yöntemdir. Sürü içerisinde besine yakın olan kuşların konumu diğer kuşların konumlarını belirlemek için bir temel teşkil eder.

Bu algoritmada kuş, balık ve hayvan sürülerinin bilgi paylaşımı yaklaşımıyla çevrelerine adapte olabilme, zengin yiyecek kaynaklarını bulabilme ve avcılardan kaçabilme yeteneklerinden yararlanılmıştır.

PSO, genetik algoritmalar gibi evrimsel hesaplama algoritmaları ile benzerlik gösterir. Sistem rastgele çözümlerden oluşan bir popülasyon ile başlatılır ve en iyi çözüm için nesilleri güncelleyerek arama yapar.

Genetik algoritmaların tersine PSO'da çaprazlama (cross over) ve mutasyon gibi evrimsel operatörler bulunmaz. PSO'da parçacık (particle) denilen potansiyel çözümler, mevcut en iyi çözümleri takip ederek problem uzayında gezinirler.

Algoritma temel olarak aşağıdaki basamaklardan oluşur;

Adım 1. Rasgele üretilen başlangıç pozisyonları ve hızları ile başlangıç sürüsü oluşturulur.

Adım 2. Sürü içerisindeki tüm parçacıkların uygunluk değerleri hesaplanır.

Adım 3. Her bir parçacık için mevcut jenerasyondan yerel en iyi (pbest) bulunur. Sürü içerisinde en iyilerin sayısı parçacık sayısı kadardır.

Adım 4. Mevcut jenerasyondaki yerel en iyiler içerisinden küresel en iyi (gbest) seçilir.

Adım 5. Pozisyon ve hızlar aşağıdaki gibi yenilenir.

Adım 6. Durdurma kriteri sağlanıncaya kadar 2-5 adımları tekrar edilir.

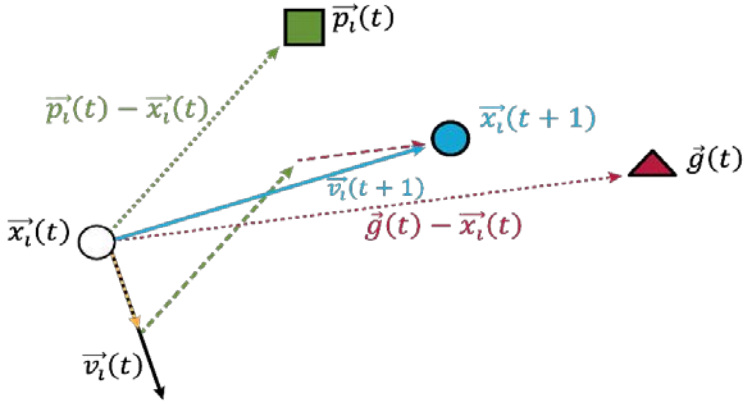
PSO'nun Matematiksel Modeli

$\overrightarrow{v_i}(t+1) = w.\overrightarrow{v_i}(t)$ Atalet terimi (inertia term)

$+r_1 c_1(\overrightarrow{p_i}(t) - \overrightarrow{x_i}(t))$ Bilişsel Bileşen (cognitive componenet)

$+r_2 c_2(\vec{g}(t) - \overrightarrow{x_i}(t))$ Sosyal Bileşen (social component)

$\overrightarrow{x_i}(t+1) = \overrightarrow{x_i}(t) + \overrightarrow{v_i}(t+1)$



Şekil 3.3. PSO'nun matematiksel modeli

3.5. Sobol ve Halton Dizileri

PSO algoritmasında yakınsaklık, parçacıkların başlangıç konumlarına bağlıdır. PSO yönteminin yakınsaklığını iyileştirmek için başlangıç pozisyonları için düzgün dağılım uygulamak yerine alternatif başlangıç pozisyonları uygulamak önerilebilir. Sobol ve Halton dizileri bu tür yaklaşımlar için en yaygın kullanılan yaklaşık rastlansal (quasi-random) dağılım dizileridir. Sobol dizisi adını ünlü Rus matematikçisi Sobol'dan almıştır. Bu diziler, birim aralığın art arda daha homojen bölümlerini oluşturmak için ikilik sayı tabanını kullanır ve ardından her bir boyut için koordinatları yeniden sıralar.

Sobol dizisinin matematiksel formülü şu şekilde oluşturulur;

Bir Sobol dizisindeki noktaların j. bileşenini oluşturmak için, $\mathbb{Z}_2$ alanında bir dereceli s_j ilkel polinomunu seçmeniz gerekir.

$$x^{s_j} + a_{1,j}x^{s_j-1} + a_{2,j}x^{s_j-2} + \ldots + a_{s_j-1,j}\,x + 1.$$

Buradaki $a_{1,j},\ a_{2,j}, \ldots, a_{s_j-1,j}$ katsayıları 0 yada 1'dir. Pozitif tamsayılardan bir $\{m_{1,j},\ m_{2,j} \ldots\}$ dizisi tanımlanır.

$$m_{k,j} := 2a_{1,j}m_{k-1,j} \oplus 2^2 a_{2,j}m_{k-2,j} \oplus \ldots \oplus 2^{s_j-1}a_{s_j-1}, m_{k-s_j+1,j} \oplus 2^{s_j}m_{k-s_j}, j \oplus m_{k-s_j,j}$$

Burada $\oplus$, XOR işlemcisidir. Her $m_{k,j}$, $1 \le k \le s_j$ olmak üzere tek ve 2^k'dan küçük olmak üzere $m_{1,j},\ m_{2,j}$ başlangıç değerleri serbestçe seçilebilir.

Sözde yön numaraları aşağıdaki gibi tanımlanır:

$v_{k,j} := \frac{m_{k,j}}{2^k}$.

Sobol dizisindeki $x_{i,j}$, i. nokta ve j. bileşen

$x_{i,j} := i_1 v_{1,j} \oplus i_2 v_{2,j} \oplus \dots$,

$i = (\dots i_3, i_2, i_1)_2$ ikili (binary) yazıldığında i_k sağdan k. rakamdır.

Örneğin; $s_j = 3, a_{1,j} = 0$ ve $a_{2,j} = 1$ iken ilkel polinomumuz $x^3 + x + 1$ olur. $m_{1,j} = 1, m_{2,j} = 3, m_{3,j} = 7$ den başlayarak yukarıdaki formülden $m_{4,j} = 5$,

$m_{5,j} = 7$ bulunur. Buradan da yön numaraları bulunur.

$v_{1,j} = (0.1)_2$, $v_{2,j} = (0.11)_2$, $v_{3,j} = (0.111)_2$, $v_{4,j} = (0.0101)_2$, $v_{5,j} = (0.00111)_2, \dots$

$x_{i,j} := i_1 v_{1,j} \oplus i_2 v_{2,j} \oplus \dots$, formülünden ilk birkaç noktanın j. bileşenleri aşağıdaki gibidir.

$0 = (0)_2$, $x_{0,j} = 0,$

$1 = (1)_2$, $x_{1,j} = (0.1)_2 = 0.5,$

$2 = (10)_2$, $x_{2,j} = (0.11)_2 = 0.75,$

$3 = (11)_2$, $x_{3,j} = (0.1)_2 \oplus (0.11)_2 = (0.01)_2 = 0.25,$

$4 = (100)_2$, $x_{4,j} = (0.111)_2 = 0.875,$

$5 = (101)_2$, $x_{5,j} = (0.1)_2 \oplus (0.111)_2 = (0.011)_2 = 0.375$.

Sobol dizisinin oluşturulma algoritması aşağıdaki adımlardan oluşur.

$s_i \in [0,1]$

Adım 1: i. sayının gray kodu hesaplanır

$$h_i \leftarrow i \oplus \left\lceil \frac{i}{2} \right\rceil$$

Adım 2: hesaplanan gray kodu (h_i) ikilik tabanında gösterilir

$$h_i \rightarrow \sum_{j=0}^{l} a_j 2^j ; \text{// } l = \lfloor log_r\, i \rfloor$$

Adım 3: h_i'nin rakamları ile ilişkili yön numaralarının XOR'u toplanır

$$u_i \leftarrow a_1 v_1 \oplus \dots \oplus a_l v_l$$

Adım 4: u_i'nin önüne bir yarıçap noktası koyulur ve ondalık basamağa dönüştürülür

$$s_i \leftarrow \sum_{j=0}^{l} a_j 2^{-j-1} \leftarrow u_i;$$

Adım 5: Koşul sağlanıyorsa algoritma sonlandırılır. Koşul sağlanmıyorsa Adım 1'e gidilerek süreç tekrarlanır ve bir sonraki ardışık s_i değeri hesaplanır.

Halton dizileri, ilk defa 1964 yılında J. Halton tarafından kullanılmış olup, farklı türden olasılık yoğunluk fonksiyonlarına sahip Van Dar Corput dizilerine benzerliği ile tanınır. Bu algoritma ile Monte Carlo simülasyonları gibi nümerik metotlar için uzayda noktalar üretilir. Bu diziler deterministik olmasına rağmen biraz tutarsızdır, birçok amaç için rastgele gibi görünür.

Halton dizisi Van der Corput dizisinin bir boyuttan n-boyuta genişletilmiş hali olarak kabul edilir. Halton görünüşte rassal dizi denklemi ile oluşturulur.

$$x_i = \left(\Phi_{p_1}(i), \Phi_{p_2}(i), \dots, \Phi_{p_n}(i)\right)$$

Burada n problem boyutu, p problem boyutu kadar asal sayı tabanlarını, i dizinin eleman sayısını, her bir $\Phi_{p_n}(i)$ o boyuttaki görünüşte rassal diziyi ifade eder.

Halton dizisinin kod algoritması aşağıdaki gibidir.

Girdiler: indeks i; taban b

$$f \leftarrow 1$$

$$r \leftarrow 0$$

$i > 0$ iken

$$f \leftarrow f/b$$

$$r \leftarrow r + f * (i \bmod b)$$

$$i \leftarrow \lfloor i/b \rfloor$$

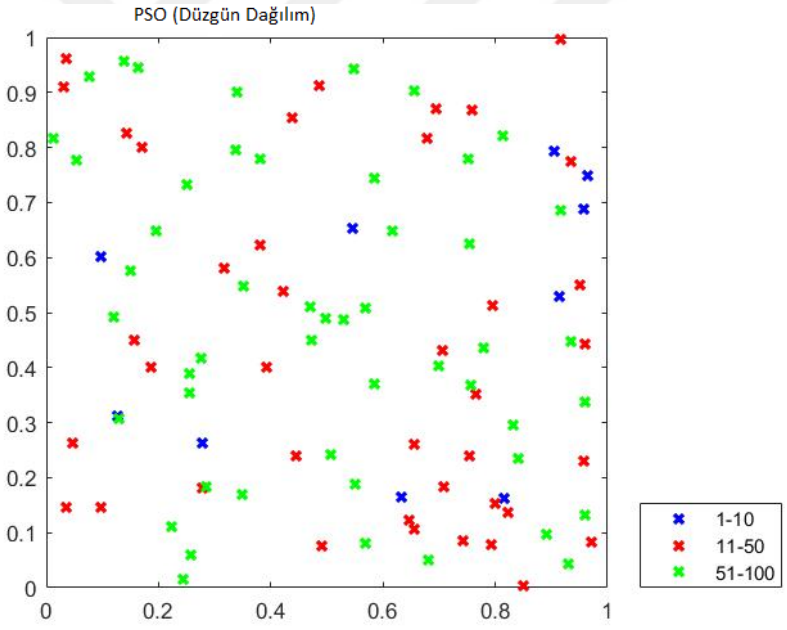
r'yi **döndür**

PSO'da parçacıkların başlangıç popülasyonlarının Sobol ve Halton dizileri yardımıyla oluşturulması sayesinde araştırma uzayında düzgün dağılımlardan daha homojen dağılımların elde edilmesi sağlanmıştır.

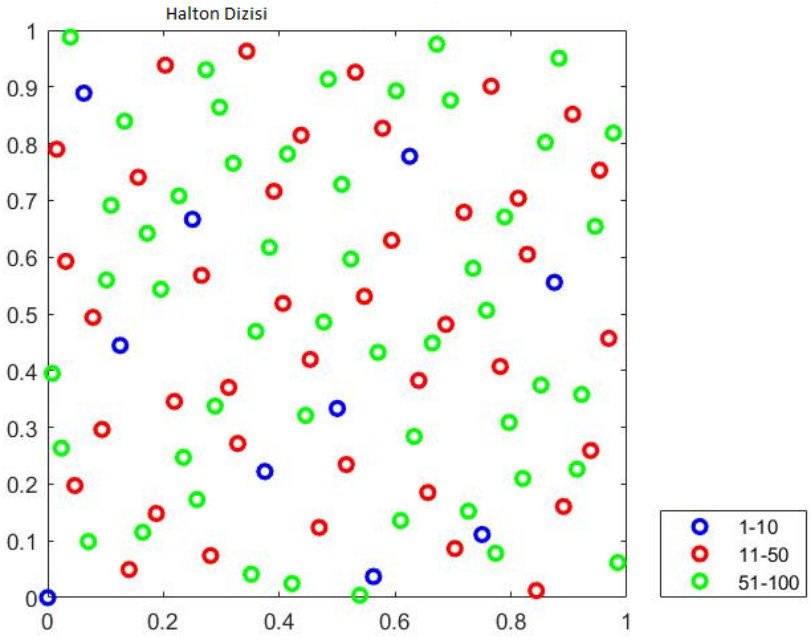
PSO'nun sayısal uygulamalarında Sobol ve Halton veri setinin oluşturulması için MATLAB programının kütüphanesinde hazır bulunan "sobolset" ve "haltonset" sınıf fonksiyonları kullanılmıştır. Yaygın kullanım şekilleri,

P=sobolset(N) veya P=haltonset(N)

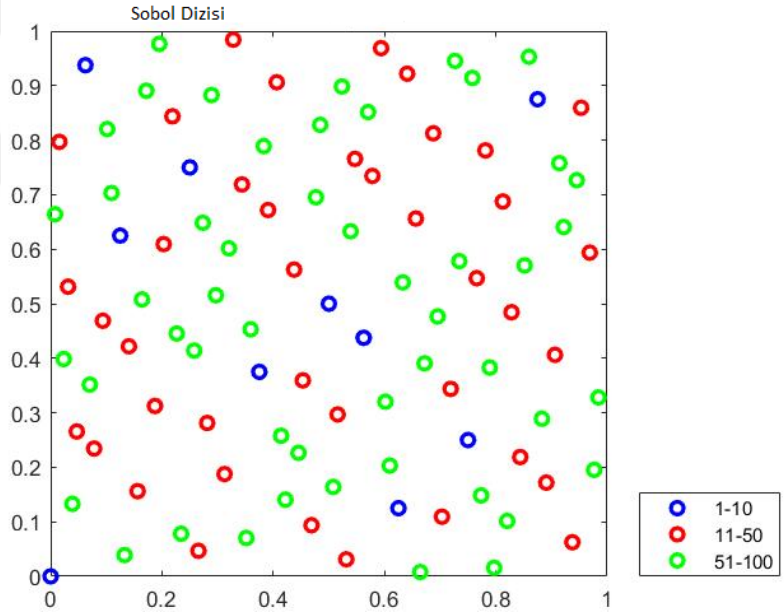
şeklindedir. Burada "N" değeri üretilecek veri setinin boyutunu ifade eden parametredir. Örneğin "P=sobolset(3)" komutu ile 3-boyutlu veri kümesi oluşturulmuş olur. Bu sınıf fonksiyonları için daha geniş bilgi "mathworks" şirketinin ilgili resmi erişim sayfasından (https://www.mathworks.com/help/matlab) veya MATLAB programının yardım menüsünden elde edilebilir.



Şekil 3.4. Klasik PSO'da 2-boyutlu uzayda üretilen ilk 100 nokta



Şekil 3.5. Halton dizileri ile 2-boyutlu uzayda üretilen ilk 100 nokta

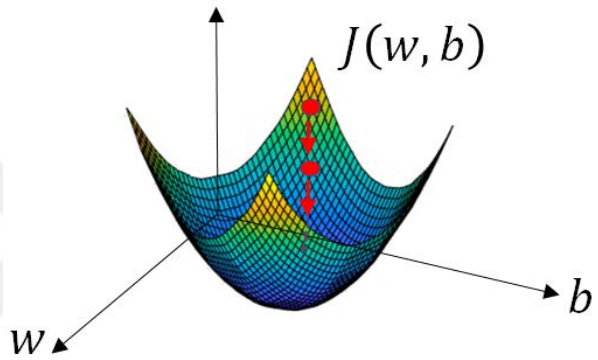


Şekil 3.6. Sobol dizileri ile 2-boyutlu uzayda üretilen ilk 100 nokta

Şekil 3.4, Şekil 3.5 ve Şekil 3.6 dan görüldüğü gibi, Sobol ve Halton ile üretilen noktalar arama uzayında, düzgün dağılımla üretilen noktalara göre daha homojen bir yapıda dağılmaktadır.

3.6. Gradyan İnişi (Gradient Descent) Optimizasyon Yöntemi

Gradyan İnişi (Gradient Descent), özellikle bilgisayar bilimlerinin de gelişmesiyle son yıllarda makine öğrenmesi (machine learning) ve derin öğrenmede (deep learning) kullanılan en popüler optimizasyon stratejilerinden biridir. Bir dışbükey fonksiyonun minimum değerini bulmayı temel alan bir optimizasyon algoritmasıdır ve parametrelerini tekrar tekrar düzenleyerek optimal sonuca ilerlemeyi hedef alır. Basitçe, bir maliyet fonksiyonunun en küçük değerine ulaşmak için, parametrelerini (katsayılar) bulmaya çalışır. Örneğin maliyet fonksiyonunun iki parametreli $J(w,b)$ fonksiyonu olduğunu kabul edelim.



Şekil 3.7. Gradyan inişi modelinin şematik gösterimi

Bu durumda maliyet fonksiyonunu minimize etmek için optimal 'w' ve 'b' değerleri bulunulmaya çalışılır. Bunun için başlangıç olarak keyfi bir $J(w,b)$ noktası alınır, bu nokta Gradyan İniş Metodu için başlangıç noktası olacaktır. Bilindiği gibi bir yüzey üzerinde yüzeyin en hızlı azaldığı yön "Gradyan vektörün" negatif yönüdür. Bu yönde seçilecek belli bir adım uzunluğunda ilerleyerek, ardışık olarak $J(w,b)$ maliyet fonksiyonunun yeni değerleri bulunulmaya çalışılır.

Yöntemin her adımında, yeni bir gradyan değeri hesaplanarak bulunan değer bir adım uzunluğu ile çarpılır ve bir önceki maliyet değerinden çıkartılarak maliyet fonksiyonunun yeni değeri elde edilir. Böylelikle aranan parametrelerin optimal değerleri elde edilir. Yöntemin algoritmasının genel yapısı, $J(w,b)$ maliyet fonksiyonu baz alınırsa, şu şekildedir:

Adım 1: Parametre değerlerinin (w_0, b_0) başlangıç değerleri seçilir ve bu noktada maliyet fonksiyonunun $J(w_0, b_0)$ değeri hesaplanır.

Adım 2: Seçilen başlangıç noktasında $\nabla J(w_0, b_0)$ gradyan değeri hesaplanır.

Adım 3: İniş adım parametresi olarak θ_0 değeri seçilir.

Adım 4: Bir sonraki parametre değeri

$$(w_1, b_1) = (w_0, b_0) - \theta_0 \nabla J(w_0, b_0)$$

formülü ile hesaplanır.

Adım 5: Durdurma koşulunun yani

$$J(w_1, b_1) < \varepsilon$$

eşitsizliğinin sağlanıp sağlanmadığı kontrol edilir. Burada ε, önceden belirlenmiş durdurma parametresidir.

Adım 6: Koşul sağlanıyorsa algoritma sonlandırılır. Koşul sağlanmıyorsa Adım 2'ye gidilerek süreç tekrarlanır ve bir sonraki ardışık değer

$$(w_n, b_n) = (w_{n-1}, b_{n-1}) - \theta_{n-1} \nabla J(w_{n-1}, b_{n-1}), \qquad n = 1,2,3, \dots$$

genel iterasyon formülü ile elde edilir.

Minimum Probleminin Gradyan İnişi (Gradient Descent) Optimizasyon Yöntemi ile Çözümü

BS probleminin YSA yöntemi ile çözümünün bulunması bir minimum probleminin çözümünün bulunmasına dayanmaktadır. Aşağıdaki minimum problemi tekrar ele alınarak minimumunun gradyan iniş metodu ile bulunuşu açıklanır:

$$minE(\vec{p}) \to E(\vec{p}^*)$$

$$E(\vec{p}) = \frac{1}{2}\sum_{i=0}^{N}\sum_{j=1}^{M}\{e_{ij}\}^2, \qquad e_{ij} = \frac{\partial \varphi}{\partial t_j} + \frac{1}{2}A^2 {x_i}^2 \frac{\partial^2 \varphi}{\partial {x_i}^2} + B x_i \frac{\partial \varphi}{\partial x_i} - C\varphi(x_i, t_j)$$

burada $\varphi(x,t) = \frac{t}{T}P(x) + x(T-t)Net(x,t;\vec{p})$ YSA yöntemi için tanımlanan deneme fonksiyonudur. $\overrightarrow{p_k} = (\alpha_k, w_k, \Omega_k, b_k), k = 1,2,3, \dots, m$ optimal değerleri aranan parametreler vektörüdür. Bu minimum probleminin Gradyan İniş Algoritması ile çözümü aşağıdaki adımlardan oluşur:

Adım 1: $\overrightarrow{p_0} = (\alpha_0, w_0, \Omega_0, b_0)$ başlangıç değerleri seçilir.

Adım 2: Başlangıç gradyan değeri $\nabla E(\overrightarrow{p_0}) = \left(\frac{\partial E(\overrightarrow{p_0})}{\partial \alpha}, \frac{\partial E(\overrightarrow{p_0})}{\partial w}, \frac{\partial E(\overrightarrow{p_0})}{\partial \Omega}, \frac{\partial E(\overrightarrow{p_0})}{\partial \mathrm{b}}\right)$ hesaplanır.

Adım 3: İniş adım parametresi olarak θ_0 değeri seçilir.

Adım 4: Bir sonraki ardışık yaklaşım $\overrightarrow{p_1} = (\alpha_1, w_1, \Omega_1, b_1)$ değeri hesaplanır:

$$\begin{cases} \alpha_1 = \alpha_0 - \theta_0 \dfrac{\partial E(\overrightarrow{p_0})}{\partial \alpha}, & w_1 = w_0 - \theta_0 \dfrac{\partial E(\overrightarrow{p_0})}{\partial w}, \\ \Omega_1 = \Omega_0 - \theta_0 \dfrac{\partial E(\overrightarrow{p_0})}{\partial \Omega}, & b_1 = b_0 - \theta_0 \dfrac{\partial E(\overrightarrow{p_0})}{\partial b}. \end{cases}$$

Adım 5: Durdurma koşulunun yani

$$E(\overrightarrow{p_1}) = E(\alpha_1, w_1, \Omega_1, b_1) < \varepsilon$$

eşitsizliğinin sağlanıp sağlanmadığı kontrol edilir. Burada ε, önceden belirlenmiş durdurma parametresidir.

Adım 6: Koşul sağlanıyorsa algoritma sonlandırılır. Koşul sağlanmıyorsa Adım 2'ye gidilerek $\overrightarrow{p_0} := \overrightarrow{p_1}$ tanımlaması yapılarak süreç tekrarlanır.

Diğer taraftan, Gradyan İnişi metodunda $\theta_n,\ n = 0,1,2,\ldots$ parametresinin seçimi yöntemin yakınsaklığı açısından hayati önem arz etmektedir. Parametrenin seçilen çok küçük değerleri yöntemin uzun süre çalışmasına, dolayısı ile bir hesaplama zorluğuna neden olmaktadır. Aksi durumda ise yani parametrenin seçilen büyük değerleri için ise algoritma ıraksamaktadır. Üstelik parametrenin sabit tutulması büyük olasılıkla yerel optimal değerin atlanması durumunu oluşturacaktır. Bu olumsuzlukların üstesinden gelmek için θ_n parametresi azalan bir fonksiyon şeklinde tanımlanmıştır. Bu çalışmada θ_n iniş parametre değerleri için

$$\theta_n := \frac{10^{-6}}{\log(2+n)},\ \ n = 0,1,2,\ldots$$

tanımı kullanılmıştır.

Algoritmadan da görüleceği gibi yöntemin uygulanabilmesi için $\varphi(x,t)$ deneme fonksiyonu ve $Net(x,t;\vec{p})$ fonksiyonunun kısmi türevlerinin hesaplanması gerekmektedir. Deneme fonksiyonunun türev değerleri;

$$\frac{\partial \varphi}{\partial t} = P(x) - xNet(x,t;\vec{p}) + x(T-t)\frac{\partial N}{\partial t}$$

$$\frac{\partial \varphi}{\partial x} = \begin{cases} (T-t)N + x(T-t)\dfrac{\partial N}{\partial x}, & x \le E, \\ t + (T-t)N + x(T-t)\dfrac{\partial N}{\partial x}, & x > E \end{cases}$$

$$\frac{\partial^2 \varphi}{\partial x^2} = 2(T-t)\frac{\partial N}{\partial x} + x(T-t)\frac{\partial^2 N}{\partial x^2}$$

Burada,

$$\frac{\partial N}{\partial t} = \sum_{k=1}^{m} \alpha_k w_k f(z_k)(1 - f(z_k))$$

$$\frac{\partial N}{\partial x} = \sum_{k=1}^{m} \alpha_k \Omega_k f(z_k)(1 - f(z_k))$$

$$\frac{\partial^2 N}{\partial x^2} = \sum_{k=1}^{m} \alpha_k \Omega_k^2 f(z_k)(1 - f(z_k))(1 - 2f(z_k))$$

Deneme fonksiyonunun $(\overrightarrow{\alpha_k}, \overrightarrow{w_k}, \overrightarrow{\Omega_k}, \overrightarrow{b_k})$ parametrelerine göre türev değerleri;

$$\frac{\partial E}{\partial \alpha_k} = \sum_{i=0}^{N} \sum_{j=1}^{M} e_{ij} \frac{\partial e_{ij}}{\partial \alpha_k}$$

$$\frac{\partial e_{ij}}{\partial \alpha_k} = \frac{\partial^2 \varphi}{\partial \alpha_k \partial t_j} + \frac{1}{2} A x_i^2 \frac{\partial^3 \varphi}{\partial \alpha_k \partial x_i^2} + B x_i \frac{\partial^2 \varphi}{\partial \alpha_k \partial x_i} - C \frac{\partial \varphi}{\partial \alpha_k}$$

$$\frac{\partial \varphi}{\partial \alpha_k} = x(T-t)\frac{\partial N}{\partial \alpha_k}$$

$$\frac{\partial^2 \varphi}{\partial \alpha_k \partial t} = -x\frac{\partial N}{\partial \alpha_k} + x(T-t)\frac{\partial^2 N}{\partial \alpha_k \partial t}$$

$$\frac{\partial^2 \varphi}{\partial \alpha_k \partial x} = (T-t)\frac{\partial N}{\partial \alpha_k} + x(T-t)\frac{\partial^2 N}{\partial \alpha_k \partial x}$$

$$\frac{\partial^3 \varphi}{\partial \alpha_k \partial x^2} = 2(T-t)\frac{\partial^2 N}{\partial \alpha_k \partial x} + x(T-t)\frac{\partial^3 N}{\partial \alpha_k \partial x^2}$$

Burada $\frac{\partial N}{\partial \alpha_k} = f(z_k)$, $\frac{\partial^2 N}{\partial \alpha_k \partial t} = w_k f(z_k)(1 - f(z_k))$, $\frac{\partial^2 N}{\partial \alpha_k \partial x} = \Omega_k f(z_k)(1 - f(z_k))$, $\frac{\partial^3 N}{\partial \alpha_k \partial x^2} = \Omega_k^2 f(z_k)(1 - f(z_k))(1 - 2f(z_k)),\ k = 1,2,\dots,m$.

Benzer şekilde diğer parametreler olan w_k, Ω_k, b_k parametreleri için de türev değerleri aşağıdaki şekilde hesaplanır.

$$\frac{\partial E}{\partial w_k} = \sum_{i=0}^{N} \sum_{j=1}^{M} e_{ij} \frac{\partial e_{ij}}{\partial w_k}$$

$$\frac{\partial e_{ij}}{\partial w_k} = \frac{\partial^2 \varphi}{\partial w_k \partial t_j} + \frac{1}{2} A x_i^2 \frac{\partial^3 \varphi}{\partial w_k \partial x_i^2} + B x_i \frac{\partial^2 \varphi}{\partial w_k \partial x_i} - C \frac{\partial \varphi}{w_k}$$

$$\frac{\partial \varphi}{\partial w_k} = x(T-t) \frac{\partial N}{\partial w_k}$$

$$\frac{\partial^2 \varphi}{\partial w_k \partial t} = -x \frac{\partial N}{\partial w_k} + x(T-t) \frac{\partial^2 N}{\partial w_k \partial t}$$

$$\frac{\partial^2 \varphi}{\partial w_k \partial x} = (T-t) \frac{\partial N}{\partial w_k} + x(T-t) \frac{\partial^2 N}{\partial w_k \partial x}$$

$$\frac{\partial^3 \varphi}{\partial w_k \partial x^2} = 2(T-t) \frac{\partial^2 N}{\partial w_k \partial x} + x(T-t) \frac{\partial^3 N}{\partial w_k \partial x^2}$$

Burada $\frac{\partial N}{\partial w_k} = \alpha_k t f(z_k)(1 - f(z_k)$, $\frac{\partial^2 N}{\partial w_k \partial t} = \alpha_k f(z_k)(1 - f(z_k))(1 + w_k t - 2 w_k t f(z_k))$, $\frac{\partial^2 N}{\partial w_k \partial x} = \alpha_k \Omega_k t f(z_k)(1 - f(z_k)(1 - 2f(z_k)$, $\frac{\partial^3 N}{\partial w_k \partial x^2} = \alpha_k \Omega_k^2 t f(z_k)(1 - f(z_k))(1 - 2f(z_k)\{(1 - 2f(z_k))^2 - 2(1 - f(z_k))\}$, $k = 1,2, \dots, m$.

$$\frac{\partial E}{\partial \Omega_k} = \sum_{i=0}^{N} \sum_{j=1}^{M} e_{ij} \frac{\partial e_{ij}}{\partial \Omega_k}$$

$$\frac{\partial e_{ij}}{\partial \Omega_k} = \frac{\partial^2 \varphi}{\partial \Omega_k \partial t_j} + \frac{1}{2} A x_i^2 \frac{\partial^3 \varphi}{\partial \Omega_k \partial x_i^2} + B x_i \frac{\partial^2 \varphi}{\partial \Omega_k \partial x_i} - C \frac{\partial \varphi}{\Omega_k}$$

$$\frac{\partial \varphi}{\partial \Omega_k} = x(T-t) \frac{\partial N}{\partial \Omega_k}$$

$$\frac{\partial^2 \varphi}{\partial \Omega_k \partial t} = -x \frac{\partial N}{\partial \Omega_k} + x(T-t) \frac{\partial^2 N}{\partial \Omega_k \partial t}$$

$$\frac{\partial^2 \varphi}{\partial \Omega_k \partial x} = (T-t) \frac{\partial N}{\partial \Omega_k} + x(T-t) \frac{\partial^2 N}{\partial \Omega_k \partial x}$$

$$\frac{\partial^3 \varphi}{\partial w_k \partial x^2} = 2(T-t) \frac{\partial^2 N}{\partial \Omega_k \partial x} + x(T-t) \frac{\partial^3 N}{\partial \Omega_k \partial x^2}$$

Burada $\frac{\partial N}{\partial \Omega_k} = \alpha_k x f(z_k)(1 - f(z_k)$, $\frac{\partial^2 N}{\partial \Omega_k \partial t} = \alpha_k w_k x f(z_k)(1 - f(z_k))(1 - 2f(z_k))$, $\frac{\partial^2 N}{\partial w_k \partial x} = \alpha_k f(z_k)(1 - f(z_k)(1 + \Omega_k x - 2\Omega_k x f(z_k))$, $\frac{\partial^3 N}{\partial \Omega_k \partial x^2} = \alpha_k \Omega_k f(z_k)(1 - f(z_k))\{(2 - 4f(z_k) + \Omega_k x - 2\Omega_k f(z_k)\ \ (1 - f(z_k))\}$, $k = 1,2, \dots, m$.

$$\frac{\partial E}{\partial b_k} = \sum_{i=0}^{N} \sum_{j=1}^{M} e_{ij} \frac{\partial e_{ij}}{\partial b_k}$$

$$\frac{\partial e_{ij}}{\partial b_k} = \frac{\partial^2 \varphi}{\partial b_k \partial t_j} + \frac{1}{2} A x_i^2 \frac{\partial^3 \varphi}{\partial b_k \partial x_i^2} + B x_i \frac{\partial^2 \varphi}{\partial b_k \partial x_i} - C \frac{\partial \varphi}{b_k}$$

$$\frac{\partial \varphi}{\partial b_k} = x(T - t) \frac{\partial N}{\partial b_k}$$

$$\frac{\partial^2 \varphi}{\partial b_k \partial t} = -x \frac{\partial N}{\partial b_k} + x(T - t) \frac{\partial^2 N}{\partial b_k \partial t}$$

$$\frac{\partial^2 \varphi}{\partial b_k \partial x} = (T - t) \frac{\partial N}{\partial b_k} + x(T - t) \frac{\partial^2 N}{\partial b_k \partial x}$$

$$\frac{\partial^3 \varphi}{\partial b_k \partial x^2} = 2(T - t) \frac{\partial^2 N}{\partial b_k \partial x} + x(T - t) \frac{\partial^3 N}{\partial b_k \partial x^2}$$

Burada $\frac{\partial N}{\partial b_k} = \alpha_k x f(z_k)(1 - f(z_k)$, $\frac{\partial^2 N}{\partial b_k \partial t} = \alpha_k w_k f(z_k)(1 - f(z_k))(1 - 2f(z_k))$, $\frac{\partial^2 N}{\partial b_k \partial x} = \alpha_k \Omega_k f(z_k)(1 - f(z_k)(1 - 2f(z_k))$, $\frac{\partial^3 N}{\partial b_k \partial x^2} = \alpha_k {\Omega_k}^2 f(z_k)(1 - f(z_k)(1 - 2f(z_k) + 2f^2(z_k))$, $k = 1,2, \dots, m$.

Bulunan bu türev değerleri kullanılarak değer fonksiyonunun gradyanı elde edilir.

BÖLÜM 4
BLACK-SCHOLES PROBLEMİNİN NÜMERİK ÇÖZÜMÜ ÜZERİNE DENEYSEL ÇALIŞMALAR

Bu bölümde Black-Scholes probleminin nümerik çözümü için deneysel çalışmalar ve sonuçları tartışılacaktır. Bunun için Black-Scholes probleminin genel formunu tekrar ele alalım.

$$\begin{cases} \dfrac{\partial V}{\partial t} + \dfrac{1}{2}Ax^2\dfrac{\partial^2 V}{\partial x^2} + Bx\dfrac{\partial V}{\partial x} - CV = 0, & 0 < x < \infty, 0 \le t \le T \\ V(0,t) = 0, & 0 \le t \le T \\ V(x,t) \sim x, & x \to \infty, 0 \le t \le T \\ V(x,T) = P(x), & 0 < x < \infty \end{cases}$$

Black-Scholes probleminin yaklaşık gerçek çözümü aşağıdaki fonksiyonla ifade edilir.

$$V(x,t) = x\Phi(D_1) - E\exp(-r(T-t))\Phi(D_2)$$

Burada $D_{1,2} = \frac{\log(S/E)+(r\mp\sigma^2/2)(T-t)}{\sigma\sqrt{T-t}}$ ve $\Phi(\mathrm{z}) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{z}\exp(-y^2/2)dy$.

Örnek 4.1: Bu örnekte şu değerler kullanılmıştır. $E = 100, r = 0.12, T = 1,$ $\sigma = 0.10,\quad 70 < x < 130$ bu değerler için yaklaşık gerçek ve YSA çözüm fonksiyonları Şekil 4.1' de, elde edilen sayısal değerler Tablo 4.1'de gösterilmiştir. Bu tablodaki L^1 ve L^2 hata değerleri aşağıdaki şekilde tanımlanır.

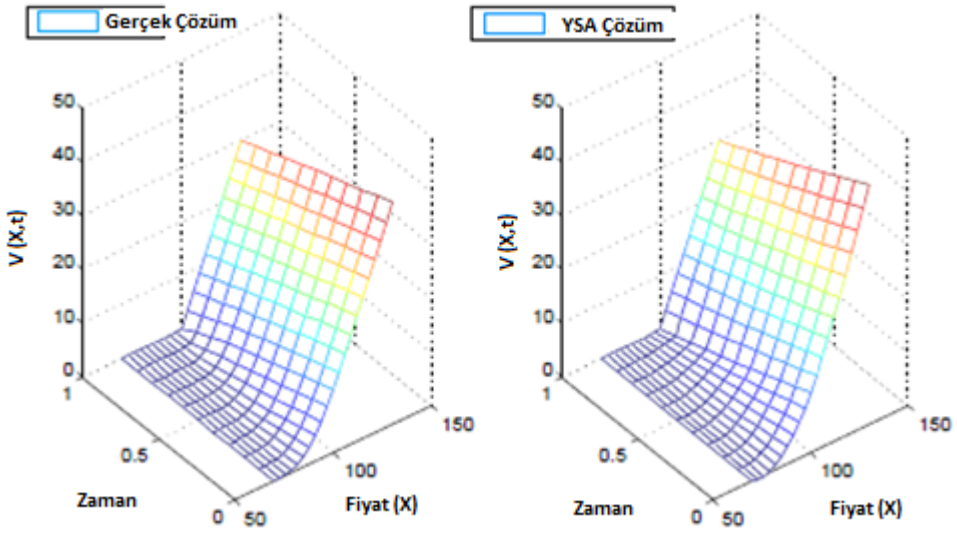
$\|v-\varphi\|_{L^1} := \frac{1}{N_X \times N_t}\sum_{i=1}^{N_X}\sum_{j=1}^{N_t}\left|V(x_i,t_j) - \varphi(x_i,t_j)\right|$: Ortalama Mutlak Hata (MAE)

$\|v-\varphi\|_{L^2} := \left(\frac{1}{N_X \times N_t}\sum_{i=1}^{N_X}\sum_{j=1}^{N_t}\left|V(x_i,t_j) - \varphi(x_i,t_j)\right|\right)^{1/2}$: Ortalama Karesel Hata (MSE)

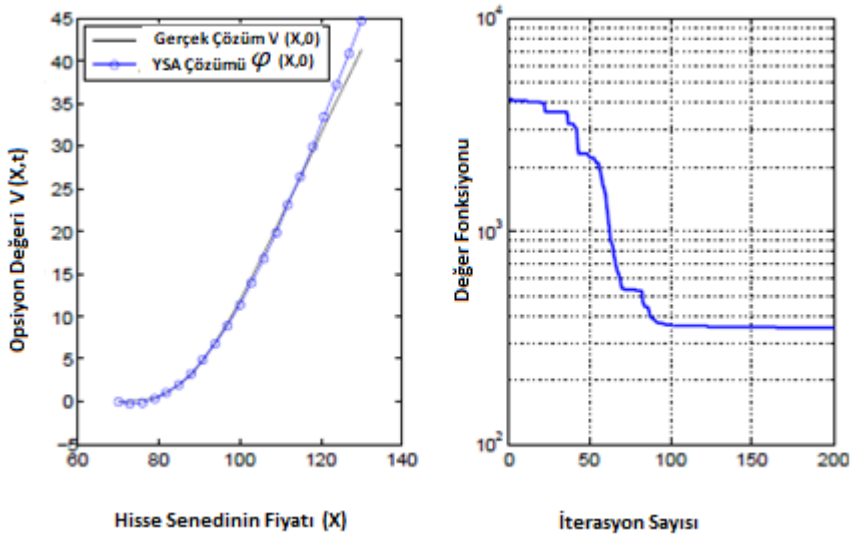
$\frac{\|v-\varphi\|_{L^2}}{\|v\|_{L^2}}$: Ortalama Karesel Bağıl Hata (MSRE)

Tablo 4.1. Farklı N, N_X, N_t şebeke parametrelerine karşılık gelen hata değerleri

N	N_X	N_t	iterasyon Sayısı	L^1-(MAE)	L^2-(MSE)	L^2-(MSRE)
5	11	11	100	0.4113	0.5914	0.0566
5	21	21	100	0.4483	0.6662	0.0461
5	31	31	100	0.4973	0.6855	0.0800
5	41	41	100	0.5944	0.9671	0.1359
5	81	81	100	0.4453	0.5850	0.0378
5	21	31	100	0.4711	0.6397	0.0719
5	21	31	150	0.4842	0.6257	0.4527
5	21	41	200	0.5062	0.7419	0.0941
5	21	41	100	0.3948	0.5332	0.0477
5	21	41	150	0.6952	1.2399	0.1846
5	21	41	200	0.5000	0.7314	0.0921
5	41	21	100	0.4575	0.6030	0.0558

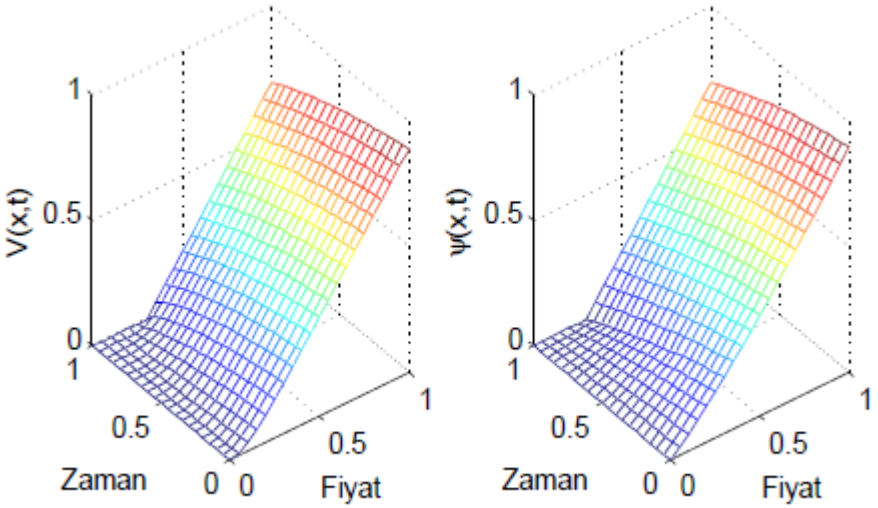


Şekil 4.1. Örnek 4.1 için Yaklaşık Gerçek çözüm ve YSA çözümü

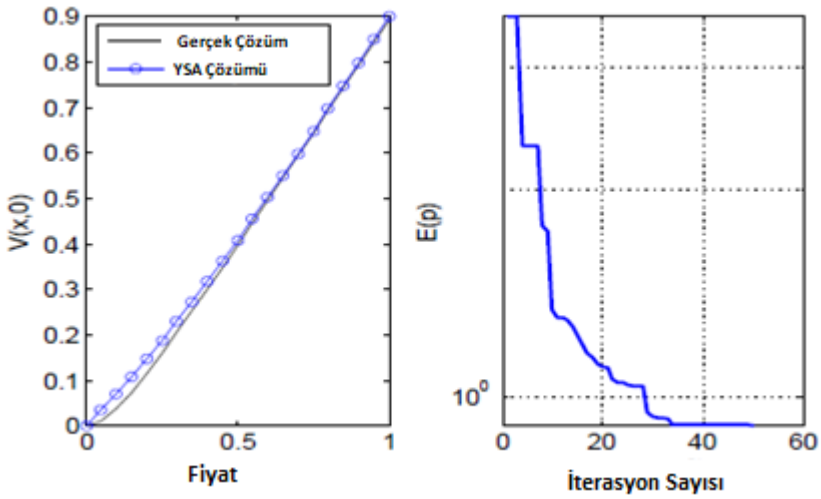


Şekil 4.2. Örnek 4.1 için Yaklaşık Gerçek ve YSA çözümünün $t = 0$ anındaki izdüşümü ve değer fonksiyonu

Örnek 4.2: Ele alınan bu ikinci örnekte değerler şu şekilde ele alınmıştır. $E = 0.3$, $r = 1,\ T = 1, \sigma = 1, 0 \leq x \leq 1$. Bu değerler için Yaklaşık gerçek ve YSA çözüm fonksiyonları Şekil 4.3 te gösterilmiştir.



Şekil 4.3. Örnek 4.2 için Yaklaşık Gerçek çözüm ve YSA çözümü

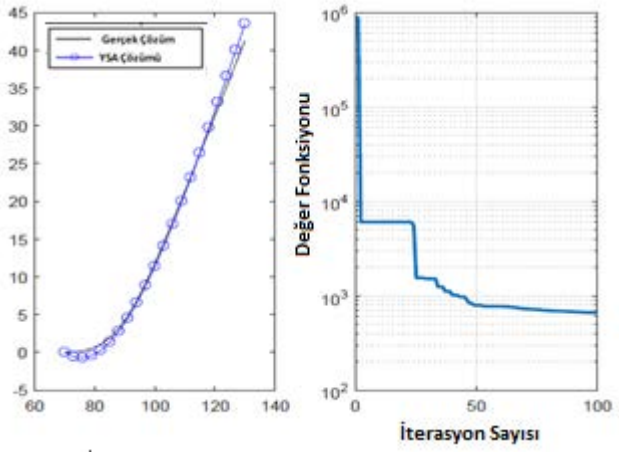


Şekil 4.4. Örnek 4.2 için Yaklaşık Gerçek ve YSA çözümünün $t = 0$ anındaki izdüşümü ve değer fonksiyonu

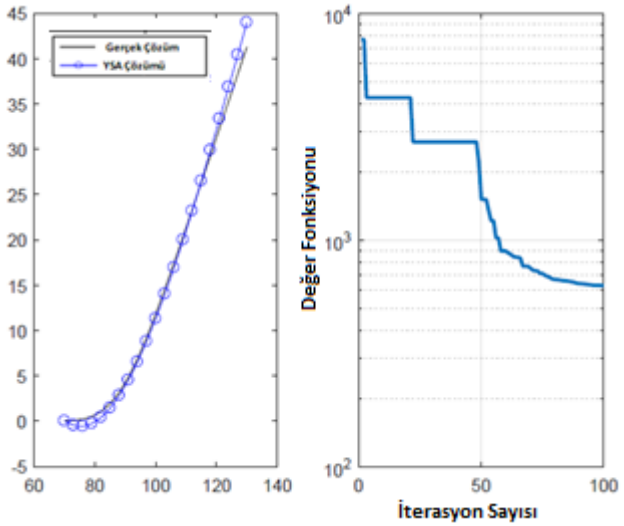
PSO algoritmasında yakınsaklık parçacıkların başlangıç konumlarına bağlıdır. Normal PSO algoritmasında bu değerler "random" fonksiyonu kullanılarak rastgele belirlenmektedir. PSO algoritmasında parçacıkların başlangıç konumlarını belirlemek için Sobol ve Halton dizilerinden faydalanılabilir.

Bunun için Matlab kütüphanesinde bulunan «sobolset()» ve «haltonset()» hazır fonksiyonları kullanılarak parçacıkların başlangıç konumları belirlenmektedir.

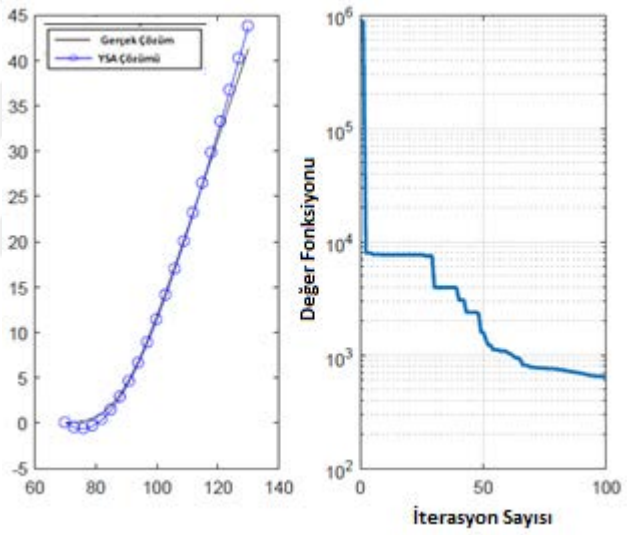
Elde edilen sonuçlar aşağıdaki tabloda ifade edilmiştir. Tablodan da görüleceği üzere Sobol ve Halton dizileri ile üretilen başlangıç konumları PSO'nun yakınsaklığına etkisi klasik yaklaşıma göre çok fark oluşturmamaktadır.



Şekil 4.5. Örnek 4.1 için Yaklaşık Gerçek ve PSO (Düzgün dağılım) çözümünün $t = 0$ anındaki izdüşümü ve değer fonksiyonu



Şekil 4.6. Örnek 4.1 için Yaklaşık Gerçek ve PSO (Sobol yaklaşımı) çözümünün $t = 0$ anındaki izdüşümü ve değer fonksiyonu



Şekil 4.7. Örnek 4.1 için Yaklaşık Gerçek ve PSO (Halton yaklaşımı) çözümünün $t = 0$ anındaki izdüşümü ve değer fonksiyonu

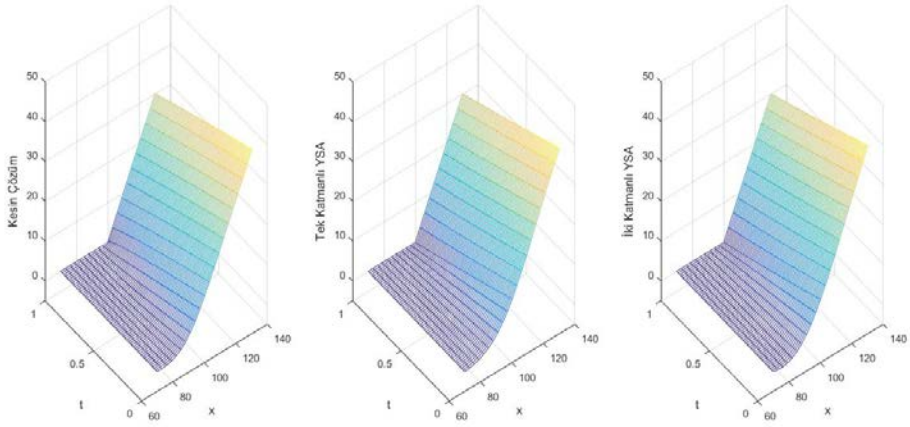
Tablo 4.2. Örnek 4.1 için PSO, Sobol ve Halton için hata değerlerinin karşılaştırılması

Hata Türleri	PSO	PSO_SOBOL	PSO_HALTON
L1 - (MAE)	0.42663	0.41486	0.41348
L2 - (MSE)	0.59264	0.56223	0.54897
L2 - (MSRE)	7.9983e-05	7.5879e-05	7.409e-05
İşlem Süresi	6.153102 saniye	6.325599 saniye	6.267847 saniye

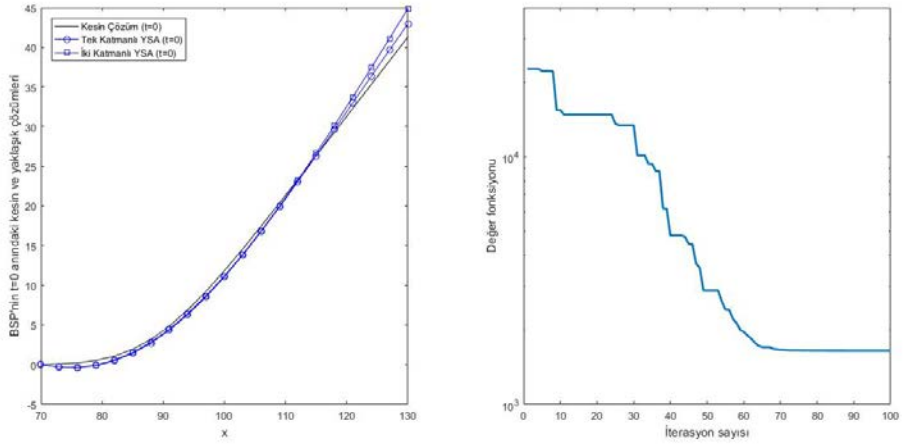
Örnek 4.3: (İki Katmanlı YSA ile Black-Sholes Probleminin (BSP) Çözümü) İki katmanlı yapay sinir ağı modeli benzer şekilde BSP'nin çözümü için kolaylıkla uygulanabilir.

$$\frac{\partial V}{\partial t}+\frac{1}{2}A^2x^2\frac{\partial^2 V}{\partial x^2}+Bx\frac{\partial V}{\partial x}-CV=0,$$
$$70\le x\le 130\,,0\le t<T$$
$$V(x,T)=\max(x-E,0)\,,\qquad 70<x<130$$

$A=0.1,\ B=C=0.12,\ T=1,\ E=100$ değerleri için çözümler Şekil 4.8'de verilmiştir. Black-Scholes Problemi için Yaklaşık gerçek çözüm ile Tek ve İki katmanlı YSA modelleri karşılaştırılacak olursa; Tablo 4.3 ile verilen değerler elde edilir.



Şekil 4.8. Örnek 4.3 için Yaklaşık Gerçek çözüm ile Tek ve İki Katmanlı YSA modellerinden elde edilen yaklaşık çözümlerin karşılaştırılması



Şekil 4.9. Örnek 4.3 için $t=0$ anındaki Yaklaşık Gerçek çözüm ile Tek ve İki Katmanlı YSA çözümlerin karşılaştırılması

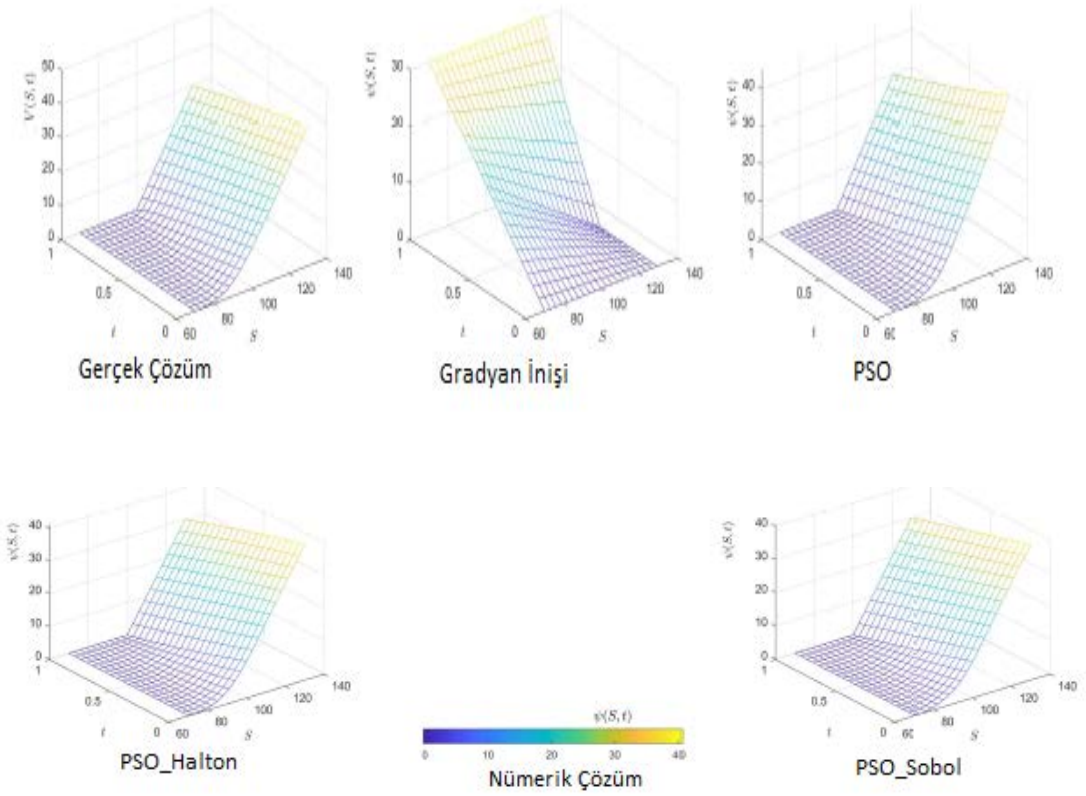
Tablo 4.3. Örnek 4.3 için iterasyon sayısına bağlı olarak Tek ve İki Katmanlı YSA da hata değerlerinin karşılaştırılması

N	N_X	N_t	İterasyon Sayısı	L^1-(MAE) (Tek Katmanlı YSA)	L^1-(MAE) (İki Katmanlı YSA)
5	21	61	50	7.09×10^{-5}	1.40×10^{-5}
5	21	61	100	2.39×10^{-5}	2.17×10^{-5}
5	21	61	150	2.82×10^{-5}	3.97×10^{-5}
5	21	21	50	2.88×10^{-4}	3.81×10^{-4}
5	21	21	100	3.81×10^{-4}	9.31×10^{-4}
5	21	21	150	6.87×10^{-5}	9.46×10^{-5}

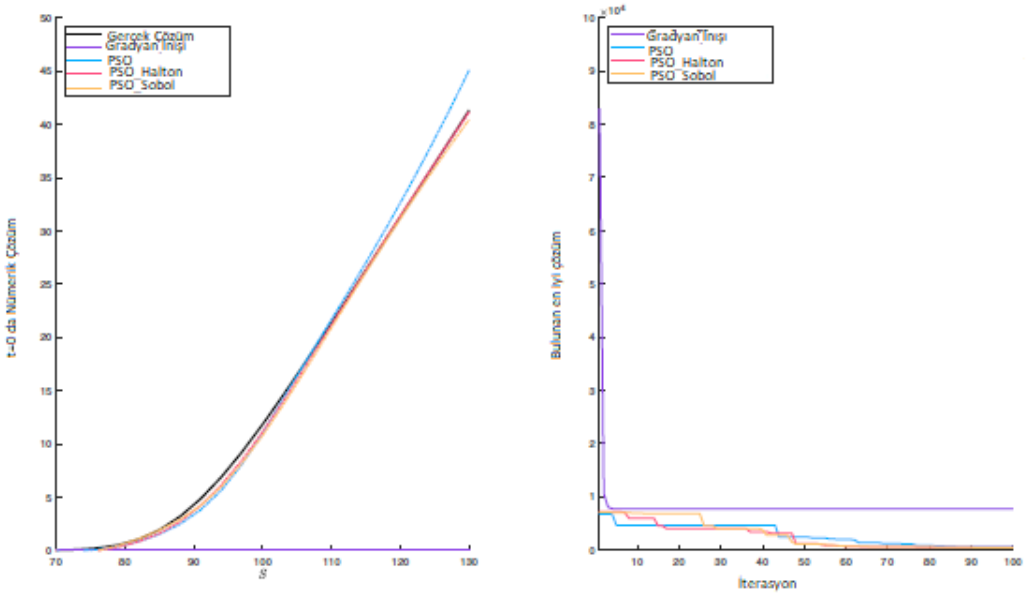
Örnek 4.4: Gradyan İnişi ile Black-Sholes Probleminin (BSP) Çözümü

$$\frac{\partial V}{\partial t} + \frac{1}{2}A^2x^2\frac{\partial^2 V}{\partial x^2} + Bx\frac{\partial V}{\partial x} - CV = 0,$$
$$70 \le x \le 130\,, 0 \le t < T$$
$$V(x,T) = \max(x - E, 0)\,, \qquad 70 < x < 130$$

$A = 0.1,\ B = C = 0.12,\ T = 1,\ E = 100$ değerleri için çözüm, Black-Scholes Problemi için Gerçek Çözüm, Gradyan İnişi, PSO, PSO_Halton, PSO_Sobol modelleri ile çözümleri aşağıdaki grafiklerde yer almaktadır;



Şekil 4.10. Örnek 4.4 için Yaklaşık Gerçek çözüm, Gradyan İnişi, PSO, PSO_Halton, PSO_Sobol modelleri ile elde edilen çözümlerin karşılaştırılması



Şekil 4.11. Örnek 4.4 için Yaklaşık Gerçek çözüm, Gradyan inişi, PSO, PSO_Halton, PSO_Sobol modelleri ile elde edilen çözümlerin $t = 0$ anındaki izdüşümlerinin karşılaştırılması

Tablo 4.4. PSO ve türevleri ile Gradyan iniş yöntemlerinin hata değerlerinin karşılaştırılması

Hata Türleri		PSO	PSO_Sobol	PSO_Halton	Gradyan İnişi
MAE	En iyi	0.3385	0.2556	0.3016	10.075
	En Kötü	0.5281	1.0410	0.5217	15.302
	Ortalama	0.4374	0.4543	0.4002	14.870
	Standart Sapma	0.0599	0.1440	0.0703	1.0002
MSE	En iyi	0.4742	0.3623	0.4170	14.897
	En Kötü	0.8182	2.0140	0.8048	18.563
	Ortalama	0.6212	0.6783	0.5644	18.176
	Standart Sapma	0.1141	0.3047	0.1203	0.6852
MSRE	En iyi	6.400×10^{-5}	4.890×10^{-5}	5.628×10^{-5}	2.0105×10^{-3}
	En Kötü	1.104×10^{-4}	2.718×10^{-4}	1.086×10^{-4}	2.5052×10^{-3}
	Ortalama	8.383×10^{-5}	9.154×10^{-5}	7.617×10^{-5}	2.453×10^{-3}
	Standart Sapma	1.540×10^{-5}	4.112×10^{-5}	1.623×10^{-5}	9.248×10^{-5}

Tablo 4.5. Farklı şebeke parametrelerine bağlı olarak Gradyan İnişi yönteminin hata değerlerinin karşılaştırılması

N_S	N_t	MAE	MSE	$MSRE$
11	11	1.8774	2.5847	1.229×10^{-3}
11	21	2.0402	2.7868	6.950×10^{-4}
11	41	2.2281	3.0650	3.917×10^{-4}
11	81	2.3492	3.2261	2.087×10^{-4}
21	11	2.1034	2.8425	7.316×10^{-4}
21	21	2.2514	3.0736	4.148×10^{-4}
21	41	2.2645	3.0204	2.088×10^{-4}
21	81	2.3955	3.2525	1.138×10^{-4}
41	11	2.1695	2.9456	3.950×10^{-4}
41	21	2.2361	2.9836	2.098×10^{-4}
41	41	2.4170	3.2762	1.180×10^{-4}
41	81	2.4193	3.2695	5.964×10^{-5}
81	11	2.4128	3.3246	2.276×10^{-4}
81	21	2.4232	3.2984	1.184×10^{-4}
81	41	2.4283	3.2853	6.045×10^{-5}
81	81	2.4307	3.2786	$\mathbf{3.054 \times 10^{-5}}$

Tablo 4.6. Farklı şebeke parametrelerine bağlı olarak PSO yöntemi için hata değerlerinin karşılaştırılması

N_S	N_t	MAE	MSE	$MSRE$
11	11	0.3066	0.4325	2.058×10^{-4}
11	21	0.2786	0.3918	9.772×10^{-5}
11	41	0.2996	0.4232	5.408×10^{-5}
11	81	0.2357	0.3512	2.273×10^{-5}
21	11	0.3166	0.4725	1.216×10^{-4}
21	21	0.3109	0.4096	5.528×10^{-5}
21	41	0.3078	0.4260	2.946×10^{-5}
21	81	0.3151	0.4372	1.531×10^{-5}
41	11	0.3248	0.4785	6.417×10^{-5}
41	21	0.3305	0.4785	3.365×10^{-5}
41	41	0.3304	0.4635	1.670×10^{-5}
41	81	0.3062	0.4279	7.807×10^{-6}
81	11	0.2927	0.4216	2.887×10^{-5}
81	21	0.3193	0.4693	1.685×10^{-5}
81	41	0.3921	0.5118	9.417×10^{-6}
81	81	0.3696	0.4888	$\mathbf{4.554 \times 10^{-6}}$

Tablo 4.7. Farklı şebeke parametrelerine bağlı olarak PSO_Halton yöntemi için hata değerlerinin karşılaştırılması

N_S	N_t	MAE	MSE	$MSRE$
11	11	0.2530	0.3917	1.864×10^{-4}
11	21	0.2460	0.3887	9.696×10^{-5}
11	41	0.2716	0.3987	5.096×10^{-5}
11	81	0.3043	0.4274	2.765×10^{-5}
21	11	0.2359	0.3823	9.840×10^{-5}
21	21	0.3056	0.4443	5.996×10^{-5}
21	41	0.2988	0.4107	2.840×10^{-5}
21	81	0.2568	0.4204	1.472×10^{-5}
41	11	0.3035	0.4130	5.540×10^{-5}
41	21	0.2916	0.3949	2.777×10^{-5}
41	41	0.3268	0.4320	1.557×10^{-5}
41	81	0.3261	0.4528	8.260×10^{-6}
81	11	0.2997	0.4881	3.343×10^{-5}
81	21	0.3475	0.4970	1.785×10^{-5}
81	41	0.3544	0.5181	9.534×10^{-6}
81	81	0.2983	0.4494	**4.187×10^{-6}**

Tablo 4.8. Farklı şebeke parametrelerine bağlı olarak PSO_Sobol yöntemi için hata değerlerinin karşılaştırılması

N_S	N_t	MAE	MSE	$MSRE$
11	11	0.2117	0.3381	1.609×10^{-4}
11	21	0.2509	0.3842	9.582×10^{-5}
11	41	0.2665	0.3995	5.106×10^{-5}
11	81	0.2575	0.3965	2.565×10^{-5}
21	11	0.3015	0.4123	1.061×10^{-4}
21	21	0.2690	0.4133	5.578×10^{-5}
21	41	0.3260	0.4354	3.011×10^{-5}
21	81	0.2679	0.4235	1.483×10^{-5}
41	11	0.3614	0.5092	6.830×10^{-5}
41	21	0.3289	0.4608	3.240×10^{-5}
41	41	0.3236	0.4881	1.759×10^{-5}
41	81	0.2943	0.4434	8.089×10^{-6}
81	11	0.3313	0.4587	3.142×10^{-5}
81	21	0.3087	0.4575	1.643×10^{-5}
81	41	0.3174	0.4829	8.886×10^{-6}
81	81	0.3385	0.4819	**4.490×10^{-6}**

Tablo 4.9. Farklı şebeke parametrelerine bağlı olarak Mutlak Hata değerlerinin karşılaştırılması

i	j	x_i	t_j	**Gradyan İniş**	**PSO**	**PSO_Sobol**	**PSO_Halton**
1	1	70	0.0	2.366×10^{-2}	2.366×10^{-2}	2.366×10^{-2}	2.366×10^{-2}
1	6	70	0.5	1.693×10^{-5}	1.693×10^{-5}	1.693×10^{-5}	1.693×10^{-5}
1	11	70	1.0	0.000×10^{-0}	0.000×10^{-0}	0.000×10^{-0}	0.000×10^{-0}
21	1	80	0.0	4.261×10^{-0}	3.151×10^{-1}	2.251×10^{-2}	6.623×10^{-1}
21	6	80	0.5	2.438×10^{-0}	1.171×10^{-1}	1.373×10^{-1}	4.283×10^{-1}
21	11	80	1.0	0.000×10^{-0}	0.000×10^{-0}	0.000×10^{-0}	0.000×10^{-0}
41	1	90	0.0	5.587×10^{-0}	1.238×10^{-0}	7.994×10^{-1}	5.844×10^{-1}
41	6	90	0.5	3.895×10^{-0}	3.614×10^{-2}	3.208×10^{-2}	1.718×10^{-2}
41	11	90	1.0	0.000×10^{-0}	0.000×10^{-0}	0.000×10^{-0}	0.000×10^{-0}
61	1	100	0.0	2.926×10^{-0}	5.273×10^{-1}	1.156×10^{-0}	8.983×10^{-1}
61	6	100	0.5	8.012×10^{-1}	1.604×10^{-0}	1.657×10^{-0}	1.441×10^{-0}
61	11	100	1.0	0.000×10^{-0}	0.000×10^{-0}	0.000×10^{-0}	0.000×10^{-0}
81	1	110	0.0	1.680×10^{-0}	4.053×10^{-1}	4.183×10^{-1}	5.880×10^{-1}
81	6	110	0.5	1.018×10^{-0}	1.447×10^{-1}	6.128×10^{-1}	3.617×10^{-1}
81	11	110	1.0	0.000×10^{-0}	0.000×10^{-0}	0.000×10^{-0}	0.000×10^{-0}
101	1	120	0.0	6.708×10^{-0}	7.858×10^{-1}	7.170×10^{-2}	4.436×10^{-1}
101	6	120	0.5	3.522×10^{-0}	1.311×10^{-1}	3.258×10^{-1}	2.285×10^{-1}
101	11	120	1.0	0.000×10^{-0}	0.000×10^{-0}	0.000×10^{-0}	0.000×10^{-0}
121	1	130	0.0	$1.178 \times 10^{+1}$	1.884×10^{-0}	7.492×10^{-1}	1.439×10^{-0}
121	6	130	0.5	6.061×10^{-0}	8.531×10^{-1}	6.082×10^{-1}	7.891×10^{-1}
121	11	130	1.0	0.000×10^{-0}	0.000×10^{-0}	0.000×10^{-0}	0.000×10^{-0}

BÖLÜM 5
TARTIŞMA VE SONUÇLAR

Bu tezde Black-Scholes probleminin matematiksel modeli ele alınmıştır. Yapay sinir ağları yönteminin matematiksel altyapısı açıklanarak ele alınan modelin yapay sinir ağları ile sayısal çözümünün elde edilişi ayrıntılı olarak açıklanmıştır. Sayısal çözüm ile yaklaşık analitik çözüm karşılaştırılarak yöntemin güvenilirliği test edilmiş ve yöntemin sayısal analizi yapılmıştır.

Sayısal çözümün bir parçası olan Parçacık Sürü Optimizasyon yönteminin başlangıç değerleri için standart yaklaşımdan farklı olarak Sobol ve Halton dizileri kullanılmış, elde edilen sayısal sonuçlar problemin var olan yaklaşık analitik çözümü ile karşılaştırılmıştır. Minimum probleminin çözümü için Parçacık Sürü Optimizasyon yöntemine alternatif olarak Gradyan İnişi (Gradient Descent) yöntemi önerilmiş ve minimum problemi, her iki yöntemle de çözülerek karşılaştırma yapılmıştır. Bu karşılaştırma sonucunda Parçacık Sürü Optimizasyon yönteminin Gradyan İniş yöntemine göre daha optimal çözümler verdiği görülmüştür.

Black-Scholes probleminin geri parabolik bir diferansiyel denklem olması dolayısıyla, yöntemin parabolik tipteki diğer diferansiyel denklemlere uygulanabilmesi mümkündür. Ayrıca problem tek katmanlı yapay sinir ağları ile çözülmüştür. İleriki çalışmalarda katman sayısının arttırılmasına bağlı olarak farklı problemlerin sayısal çözümleri karşılaştırılabilir.

KAYNAKÇA

Aarts, L. P. ve Veer, P. (2001). *Solving Nonlinear Differential Equations by a Neural Network Method*. Springer-Verlag Berlin, Heidelberg.

Bangyal W.H., Rauf, H.T., Batool, H., Bangyal S.A., Ahmed, J. ve Pervaiz, S. (2019). *An Improved Particle Swarm Optimization Algorithm with Chi-Square Mutation Strategy*. International Journal of Advanced Computer Science and Applications, 10(3), 481-491.

Black, F. ve Scholes, M.S. (1973). *The Pricing of Options and Corporate Liabilities.* Journal of Political Economy, 81(3), 637-654.

Cybenko, G. (1989). *Approximation by superpositions of a sigmoidal function*. Mathematics of Control, Signals and Systems, 2(4), 303-314.

Eberhart, R.C. ve Kennedy, J. (1995). *A new optimizer using particle swarm theory*, in Symposium on Micro Machine and Human Science. Japan: Nagoya, Piscataway, NJ.

Halton, J.H. (1964). Algorithm 247: *Radical-inverse quasi-random point sequence*. Communications of the ACM, 1964, 7(12), 701-702.

Jang, H. ve Lee, J. (2019). *Generative Bayesian neural network model for risk-neutral pricing of American index options*, Quantitative Finance, 19:4, 587-603.

Kennedy, J. ve Eberhart, R. (1995). *Particle swarm optimization*," Proceedings of ICNN'95 - International Conference on Neural Networks, Perth,WA, Australia, 1995, pp. 1942-1948 vol.4. doi: 10.1109/ICNN.1995.488968

Kolmogorov, A.N. (1957). *On the representation of continuous functions of several variables by superpositions of Quantitative Finance draft continuous functions of one variable and addition*, Dokl. Akad. Nauk SSSR 114, 1957, 953-956. English Translation: Am. Math. Soc. Transl. 28(2), 1963, 55-59.

Kumar, M. ve Yadav, N. (2015). *Numerical Solution of Bratu's Problem Using Multilayer Perceptron Neural Network Method*, Springer India, National Academy Science Letters, 425-428.

Lee, H. ve Kang, I. (1990). *Neural Algorithm for Solving Differential Equation*s, Journal of Computational Physics 91, 110-131.

Liu, X., Cao, Y., Ma, C. ve Shen L. (2018). *Wavelet-based option pricing: An empirical study*, European Journal of Operational Research 272(3).

Liu, S., Oosterlee, C.W. ve Bohte, S.M. (2019). *Pricing Options and Computing Implied Volatilities using Neural Networks*, Risks.

Malek, A. ve Shekari Beidokhti, R. (2006). *Numerical solution for high order differential equations using a hybrid neural network-optimization method*. Appl. Math. Comput., 2006, 183, 260-271.

Macias-Diaz, J.E. ve Vargas-Rodriguez, H. (2018). *Some exact solutions of a hyperbolic model of energy transmission in non-homogeneous media*, Journal Of Computational And Applied Mathematics.

McFall, K. S. ve Mahan, J.R.(2009). *Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions*. IEEE Transactions on Neural Network 1221-33.

Muller, B., Reinhardt, J. ve Strickland, M.T. (2002). *Neural Networks*: An Introduction, second ed. Springer-Verlag, Berlin.

Otadi, M. ve Mosleh, M. (2011). *Numerical solution of quadratic Riccati differential equation by neural network*, Mathematical Sciences, 5-3, 249-257.

Ömür, U. (2008). *An introduction to computational Finance*, Imperial College Press, Series in Quantitative Finance World Scientific Publishing Co.

Raja, M.A.Z. ve Ahmad S. (2014). *Numerical treatment for solving one-dimensional Bratu problem using neural networks*, Neural Computing and Applications, Volume 24, Issue 3–4, pp 549–561.

Raja, M.A.Z., Ahmad S. ve Raza S.S. (2014). *Solution of the 2-dimensional Bratu problem using neural network, swarm intelligence and sequential quadratic programming*, Neural Computing and Applications, Volume 25, Issue 7–8, pp 1723–1739.

Slavova, A., ve Kyurkchiev, N. (2018). *On Cnn Model Of Black–Scholes Equatıon with Leland Correction*, Comptes rendus de l'Acad´emie bulgare des Sciences, Tome 71, No 2.

Sobol, I.Y.M. (1967). *On the distribution of points in a cube and the approximate evaluation of integrals*. Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki, 1967, 7(4), 784-802.

Shekari Beidokhti, R.ve Malek, A. (2007). *Solve initial-boundary value problems for systems of partial differential equations using neural networks and optimization techniques,* Journal of the Franklin Institute 346(2009),898-913

Vala, J. , Jarosova, P. ve Brno (2018). *Optimization Approaches to Some Problems of Building Design*, Applications Of Mathematics.

Velazquez, E.R., Sanchez, O.D., Quiroz, G., Pulido, G.O. (2018). *Parametric Identification of Sorensen model for glucose-insulin-carbohydrates dynamics using evolutive algorithms* Kybernetıka | Volume 54 (2018) , Number 1, Pages 110 – 134.

Yun, H., Lim, S.ve Lee, K. (2018). *The value of options for time charterparty extension: an artificial neural networks (ANN) approach*, Maritime Policy &Management, 45:2, 197-210.

EK - Matlab Kodları

Değer (Cost) Fonksiyonunun hesaplanması;

```
function fcost = Cost(xval,tval,p)
% This function computes values of the Cost Function defined as
% E(p)=(1/2)*Sum_x*Sum_t (eij)^2
% eij=U_t+(1/2)*x^2*A^2*U_xx+B*x*U_x-C*U,
% U(x,t) is the trial function defined as
% U(x,t)=(t/Tf)*P(x)+x*(Tf-t)*N(x,t;p)
global A B C E
Nx = length(xval);
Nt = length(tval);
Tf=tval(Nt);
xA=xval(1);
fE=max(xval-E,0);
fdPx=dP_x(xval);
fNet=Net(xval,tval,p);
fdNet_t=dNet_t(xval,tval,p);
fdNet_x=dNet_x(xval,tval,p);
fd2Net_xx=d2Net_xx(xval,tval,p);
fcost =0;
for j=1:Nt
   tj=tval(j);
   for i=1:Nx
      xi=xval(i);
      fdtrial_t=fE(i)/Tf-(xi-xA)*fNet(i,j)+(xi-xA)*(Tf-tj)*fdNet_t(i,j);
      ftrial=tj*fE(i)/Tf+(xi-xA)*(Tf-tj)*fNet(i,j);
      fdtrial_x=tj*fdPx(i)/Tf+(Tf-tj)*fNet(i,j)+(xi-xA)*(Tf-tj)*fdNet_x(i,j);
      fd2trial_xx=2*(Tf-tj)*fdNet_x(i,j)+(xi-xA)*(Tf-tj)*fd2Net_xx(i,j);
      fcost =  fcost+0.5*(fdtrial_t+0.5*xi^2*A^2*fd2trial_xx+B*xi*fdtrial_x-
C*ftrial)^2;
   end
end
```

Net Fonksiyonun hesaplanması;

```
function fNet = Net(xx,tt,p)
% This function computes values of the Net function N(x,t;p)
% fN is output of neural network corrsponding to the inputs x and t
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
alpha = p(1:M);
omegaA = p(M+1:2*M);
omegaB = p(2*M+1:3*M);
bias = p(3*M+1:4*M);
fNet=zeros(Nx,Nt);
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      for k=1:M
         zk= omegaA(k)*t+omegaB(k)*x+bias(k);
         fNet(i,j)=fNet(i,j)+alpha(k)*fsigma(zk);
      end
   end
end
end
```

Net Fonksiyonun x'e göre türevinin hesaplanması;

```
function fdNet_x = dNet_x(xx,tt,p)
% This function computes values of the spatial derivative of the Net function
N(x,t;p)
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
alpha = p(1:M);
omegaA = p(M+1:2*M);
omegaB = p(2*M+1:3*M);
bias = p(3*M+1:4*M);
fdNet_x=zeros(Nx,Nt); %outputs of neural network
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      for k=1:M
         zk= omegaA(k)*t+omegaB(k)*x+bias(k);
         fz=fsigma(zk);
         fdNet_x(i,j) = fdNet_x(i,j)+alpha(k)*omegaB(k)*fz*(1-fz);
      end
   end
end
end
```

Net Fonksiyonun x'e göre ikinci türevinin hesaplanması;

```
function fd2Net_xx = d2Net_xx(xx,tt,p)
% This function computes values of the 2nd-order
% spatial derivative of the Net function N(x,t;p)
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
alpha = p(1:M);
omegaA = p(M+1:2*M);
omegaB = p(2*M+1:3*M);
bias = p(3*M+1:4*M);
fd2Net_xx=zeros(Nx,Nt); %outputs of neural network
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      for k=1:M
         zk= omegaA(k)*t+omegaB(k)*x+bias(k);
         fz=fsigma(zk);
         fd2Net_xx(i,j) = fd2Net_xx(i,j)+alpha(k)*omegaB(k)^2*fz*(1-3*fz+2*fz^2);
      end
   end
end
end
```

Net Fonksiyonun t'ye göre türevinin hesaplanması;

```
function fdNet_t = dNet_t(xx,tt,p)
% This function computes values of the time derivative of the Net function N(x,t;p)
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
alpha = p(1:M);
omegaA = p(M+1:2*M);
omegaB = p(2*M+1:3*M);
bias = p(3*M+1:4*M);
fdNet_t=zeros(Nx,Nt); %outputs of neural network
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      for k=1:M
         zk= omegaA(k)*t+omegaB(k)*x+bias(k);
         fz=fsigma(zk);
         fdNet_t(i,j) = fdNet_t(i,j)+alpha(k)*omegaA(k)*fz*(1-fz);
      end
   end
end
end
```

Deneme (trial) Fonksiyonunun Değerlerinin hesaplanması;

```
function ftrial = trial_u(xx,tt,p)
global E
m = length(xx);
n = length(tt);
ftrial = zeros(n,m);
xA=xx(1);Tf=tt(end);
fNet=Net(xx,tt,p);
fPx=max(xx-E,0);
for i=1:m
for j=1:n
   ftrial(j,i)=tt(j)*fPx(i)/Tf+(xx(i)-xA)*(Tf-tt(j))*fNet(i,j);
end
end
```

Sigmoid Fonksiyonunun hesaplanması;

```
function fz = fsigma(x)
   fz= 1./(1+exp(-x));
end
```

Gerçek Çözümün hesaplanması;

```
function u=exact_u(xx,tt)
global A B  E
Nx=length(xx);Nt=length(tt);
Tf=tt(end);
u=zeros(Nt,Nx);
for i=1:Nx
   x=xx(i);
   for j=1:Nt-1
      t=tt(j);
      p1=log(x/E)+(B+A^2/2)*(Tf-t);
      pp=A*sqrt(Tf-t);
      p2=log(x/E)+(B-A^2/2)*(Tf-t);
      z1=p1/pp;z2=p2/pp;
      u(j,i)=x*0.5*erfc(-z1/sqrt(2))-0.5*E*exp(-B*(Tf-t))*erfc(-z2/sqrt(2));
   end
end
u(Nt,:)=max(xx-E,0);
```

P(x), Fonksiyonun değerlerinin hesaplanması;

```
function fpx = fPx(x)
% This function computes values of the function P(x) defined as P(x)=U(x,Tf);
% Here, Tf is the final time point.
EE=100;%sqrt(1/19);
fpx= max(x-EE,0);
end
```

P(x)'in x'e göre türevinin hesaplanması;

```
function fdpx = dP_x(xx)
% This function computes values of the derivative of the function P(x)
% defined as P(x)=U(x,Tf), here, Tf is the final time point.
global E
Nx=length(xx);
fdpx=zeros(1,Nx);
for i=1:Nx
   x=xx(i);
if x < E
   fdpx(i)= 0;
else
   fdpx(i)=1;
end
end
```

PSO Ana Programının BSP için hesaplanması;

```
% This program solves the Black-Scholes problem defined as
% U_t+(1/2)*x^2*A^2*U_xx+B*x*U_x-C*U=0, 0<x<1, 0<t<Tf,
% U(0,t)=0, left boundary condition
% U(x,Tf)=P(x),final time condition, P(x)=max{x-E,0},
% by using Artificial Neural Networks (ANN) and
% by Particle Swarm Optimization (PSO)
clc;
clear;
close;
global A B C E
%% Problem definition
CostFunction =@(x,t,p)  Cost(x,t,p);          % Cost function
A=0.1;B=0.12;C=B;                          % Coefficients of the equation
xA=70; xB=130;                            % Left and right boundary points
Tf=1;                                  % Final time point
E=100;                                  % Strike price
N=5;                                   % Number of neurons
%Nt=21;Nx=21;                                % Number of nodes
%xval=linspace(xA,xB,Nx);                        % Spatial vector x
%tval=linspace(0,Tf,Nt);                     % Time vector t
```

```
%Uexc = exact_u(xval,tval);                % Exact solution

%% Parameters of Pso
MaxIt = 200;                               % Maximum number of iterations
nPop = 100;                                % Population size ( swarm size )
ww=1;                                      % Inertia coefficient
wdamp=0.99;                                % Damping parameter of intertia coefficient
c1 = 1.5;                                  % Personal weight
c2 = 1.5;                                  % Social weight
nVar = 4*N;                                % Number of unknown variables in ANN-
algorithm
VarSize=[1 nVar];                          % matrix size of unknown variables
VarMin = -1;                               % Lower bound of unknown variables
VarMax = 1;                                % Upper bound of unknown variables
maxTrials = 25;                            % Maximum number of trials

%% Initialization
empty_particle.position = [];
empty_particle.velocity = [];
empty_particle.cost = [];
empty_particle.best.position = [];
empty_particle.best.cost = [];

empty_sol.BestCosts = [];
empty_sol.Cost = [];
empty_sol.p = [];
empty_sol.Uapp = [];
empty_sol.errorL1 = [];
empty_sol.errorL2 = [];
empty_sol.RelErrL2 = [];

for Nx = [11,21,41,81]
   for Nt = [11,21,41,81]
      fprintf('N_x = %2d \t N_t = %2d\n',Nx,Nt);

      xval=linspace(xA,xB,Nx);             % Spatial vector x
      tval=linspace(0,Tf,Nt);              % Time vector t
      Uexc = exact_u(xval,tval);           % Exact solution

      sol = repmat(empty_sol, maxTrials, 1);
      for trialId = 1:maxTrials

         %fprintf('\n\n-----------------------------------------------------\n');
         fprintf('\t\t Trial Id : %d\n', trialId);
         T=cputime;
         ww=1;
         % Generate initial population
         particle = repmat(empty_particle, nPop, 1);

         % Initialize global best
```

```
GlobalBest.cost = inf;

for i=1:nPop
   % Generate random solution
   particle(i).position = unifrnd(VarMin, VarMax, VarSize);

   % Initialize velocity
   particle(i).velocity = zeros(VarSize);

   % Evaluation
   particle(i).cost = CostFunction (xval,tval,particle(i).position) ;

   % Update the personel best
   particle(i).best.position = particle(i).position ;
   particle(i).best.cost = particle(i).cost ;

   % Update the global best
   if particle(i).best.cost < GlobalBest.cost
      GlobalBest = particle(i).best;
   end
end

% Array to hold the best cost value on each iteration
BestCosts = zeros(MaxIt,1);

%% Main loop of Pso
for it=1:MaxIt

   for i=1:nPop
      particle(i).velocity =  ww*particle(i).velocity...
         +c1*rand(VarSize).*(particle(i).best.position-particle(i).position)...
         +c2*rand(VarSize).*(GlobalBest.position-particle(i).position);

      particle(i).position = particle(i).position +  particle(i).velocity;

      particle(i).cost = CostFunction ( xval,tval,particle(i).position) ;

      if particle(i).cost < particle(i).best.cost

         particle(i).best.position = particle(i).position;
         particle(i).best.cost = particle(i).cost;

         % Update the global best
         if particle(i).best.cost < GlobalBest.cost
            GlobalBest = particle(i).best;
         end
      end
   end

   % Store best cost value
```

```
        BestCosts(it) = GlobalBest.cost;

        % Display iteration information
        %fprintf('\t\tIteration %4d : Best Cost = %4.3e\n', it, BestCosts(it) );

        % Damping intertia coefficient
        ww = ww*wdamp;
    end


    %% Construct approximate solution
    p_opt = GlobalBest.position;
    Uapp = trial_u(xval,tval,p_opt);

    errorL1=(sum(sum(abs(Uexc-Uapp))))/(Nx*Nt);
    errorL2=norm(Uexc-Uapp,'fro')/(sqrt(Nx*Nt));
    RelErrL2=errorL2/norm(Uexc,'fro')/(sqrt(Nx*Nt));

    sol(trialId).BestCosts = BestCosts;
    sol(trialId).Cost = GlobalBest.cost;
    sol(trialId).p = p_opt;
    sol(trialId).Uapp = Uapp;
    sol(trialId).errorL1 = errorL1;
    sol(trialId).errorL2 = errorL2;
    sol(trialId).RelErrL2 = RelErrL2;
    sol(trialId).elapsedTime = cputime - T;

end
Costs = [sol(1:maxTrials).Cost];
[~,minId] = min(Costs);

BestCosts = sol(minId).BestCosts ;
p_opt = sol(minId).p;
Uapp =  sol(minId).Uapp;
errorL1 = sol(minId).errorL1;
errorL2 = sol(minId).errorL2;
RelErrL2 = sol(minId).RelErrL2;
%errorL3 = sol(minId).errorL3;

%% Results
fileName = ['Results\PSO_' num2str(Nx) '_'  num2str(Nt) '.txt'];
fid = fopen(fileName,'w+');

minErrorL1 = min([sol(1:maxTrials).errorL1]);
worstErrorL1 = max([sol(1:maxTrials).errorL1]);
meanErrorL1 = mean([sol(1:maxTrials).errorL1]);
stdErrorL1 = std([sol(1:maxTrials).errorL1]);

minErrorL2 = min([sol(1:maxTrials).errorL2]);
worstErrorL2 = max([sol(1:maxTrials).errorL2]);
```

```
        meanErrorL2 = mean([sol(1:maxTrials).errorL2]);
        stdErrorL2 = std([sol(1:maxTrials).errorL2]);

        minRelErrL2 = min([sol(1:maxTrials).RelErrL2]);
        worstRelErrL2 = max([sol(1:maxTrials).RelErrL2]);
        meanRelErrL2 = mean([sol(1:maxTrials).RelErrL2]);
        stdRelErrL2 = std([sol(1:maxTrials).RelErrL2]);

        meanTime = mean([sol(1:maxTrials).elapsedTime]);
        stdTime = std([sol(1:maxTrials).elapsedTime]);

        fprintf('\n\n');
        pm = 177;               % ASCII code for plus minus symbol
        disp('********************* Errors *********************')
        fprintf('Min of AbsErrorL1 = %0.3e\n', minErrorL1);
        fprintf('Worst of AbsErrorL1 = %0.3e\n', worstErrorL1);
        fprintf('Mean of AbsErrorL1 = %0.3e %c %0.3e\n\n', meanErrorL1, pm,
stdErrorL1);

        fprintf('Min of AbsErrorL2 = %0.3e\n', minErrorL2);
        fprintf('Worst of AbsErrorL2 = %0.3e\n', worstErrorL2);
        fprintf('Mean of AbsErrorL2 = %0.3e %c %0.3e\n\n', meanErrorL2, pm,
stdErrorL2);

        fprintf('Min of RelErrorInf = %0.3e\n', minRelErrL2);
        fprintf('Worst of RelErrorInf = %0.3e\n', worstRelErrL2);
        fprintf('Mean of RelErrorInf = %0.3e %c %0.3e\n\n', meanRelErrL2, pm,
stdRelErrL2);

        fprintf('Mean of Elapsed Time = %0.3e %c %0.3e seconds\n\n', meanTime, pm,
stdTime);

        fprintf(fid,'Min of AbsErrorL1 = %0.3e\n', minErrorL1);
        fprintf(fid,'Worst of AbsErrorL1 = %0.3e\n', worstErrorL1);
        fprintf(fid,'Mean of AbsErrorL1 = %0.3e %c %0.3e\n\n', meanErrorL1, pm,
stdErrorL1);

        fprintf(fid,'Min of AbsErrorL2 = %0.3e\n', minErrorL2);
        fprintf(fid,'Worst of AbsErrorL2 = %0.3e\n', worstErrorL2);
        fprintf(fid,'Mean of AbsErrorL2 = %0.3e %c %0.3e\n\n', meanErrorL2, pm,
stdErrorL2);

        fprintf(fid,'Min of RelErrorInf = %0.3e\n', minRelErrL2);
        fprintf(fid,'Worst of RelErrorInf = %0.3e\n', worstRelErrL2);
        fprintf(fid,'Mean of RelErrorInf = %0.3e %c %0.3e\n\n', meanRelErrL2, pm,
stdRelErrL2);

        fprintf(fid,'Mean of Elapsed Time = %0.3e %c %0.3e seconds\n\n', meanTime,
pm, stdTime);
```

```
        fclose(fid);

        results.ErrorL1.min = minErrorL1;
        results.ErrorL1.worst = worstErrorL1;
        results.ErrorL1.mean = meanErrorL1;
        results.ErrorL1.std = stdErrorL1;

        results.ErrorL2.min = minErrorL2;
        results.ErrorL2.worst = worstErrorL2;
        results.ErrorL2.mean = meanErrorL2;
        results.ErrorL2.std = stdErrorL2;

        results.RelErrorL2.min = minRelErrL2;
        results.RelErrorL2.worst = worstRelErrL2;
        results.RelErrorL2.mean = meanRelErrL2;
        results.RelErrorL2.std = stdRelErrL2;

%       %% Plots
%       fig1 = figure;
%       % axis([xA xB 0 Tf 0 max(max(Uexc))])
%       subplot(1,2,1);mesh(xval,tval,Uexc);
%       xlabel('$x$','Interpreter','latex')
%       ylabel('$t$','Interpreter','latex')
%       zlabel('Exact Solution','Interpreter','latex')
%       % axis([xA xB 0 Tf 0 max(max(Uapp))])
%       subplot(1,2,2);mesh(xval,tval,Uapp)
%       xlabel('$x$','Interpreter','latex')
%       ylabel('$t$','Interpreter','latex')
%       zlabel('Numerical Solution','Interpreter','latex')
%       print(fig1,'Fig1.eps','-depsc','-r300');
%       print(fig1,'Fig1.jpg','-djpeg','-r300');
%
%       fig2=figure;
%       subplot(1,2,1);plot(xval,Uexc(1,:),'k-',xval,Uapp(1,:),'r-.','LineWidth',1.5)
%       xlabel('$x$','FontSize',12,'Interpreter','latex')
%       ylabel('Numerical solution at $t=0$','FontSize',12,'Interpreter','latex')
%       legend('Exact Solution','Numerical solution','Location','northwest')
%       subplot(1,2,2);semilogy(BestCosts(1:200), 'linewidth',2)
%       xlabel('Iterations','FontSize',12,'Interpreter','latex')
%       ylabel('Best Cost','FontSize',12,'Interpreter','latex')
%       grid on
%       print(fig2,'Fig2.eps','-depsc','-r300');
%       print(fig2,'Fig2.jpg','-djpeg','-r300');

        save(['Results\PSO_' num2str(Nx) '_'  num2str(Nt)
'.mat'],'sol','BestCosts','p_opt','Uexc','Uapp','results');
    end
end
```

PSO, PSO_SOBOL, PSO_HALTON Matlab Kodları

Klasik PSO dan farklı olarak sadece ana program üzerinde Sobol ve Halton başlangıç dizileri ile başlamak için değişiklik yapılır.

Ana Programının BSP için hesaplanması;

```
% This program solves the Black-Scholes problem defined as
% U_t+(1/2)*x^2*A^2*U_xx+B*x*U_x-C*U=0, 0<x<1, 0<t<Tf,
% U(0,t)=0, left boundary condition
%U(x,Tf)=P(x),final time condition, P(x)=max{x-E,0},
% by using Artificial Neural Networks (ANN) and
% by Particle Swarm Optimization (PSO)
clc;
clear;
close;
global A B C E
%% Problem definition
CostFunction =@(x,t,p)  Cost(x,t,p);          % Cost function
A=0.1;B=0.12;C=B;                             % Coefficients of the equation
xA=70; xB=130;                                % Left and right boundary points
Tf=1;                                         % Final time point
E=100;                                        % Strike price
N=5;Nt=21;Nx=21;                              % Number of nodes
xval=linspace(xA,xB,Nx);                      % Spatial vector x
tval=linspace(0,Tf,Nt);                       % Time vector t
Uexc = exact_u(xval,tval);                    % Exact solution

%% Parameters of Pso
MaxIt = 1000;                                 % Maximum number of iterations
nPop = 100;                                   % Population size ( swarm size )
ww=1;                                         % Inertia coefficient
wdamp=0.99;                                   % Damping parameter of intertia coefficient
c1=1.5;                                       % Personal weight
c2=1.5;                                       % Social weight
nVar = 4*N;                                   % Number of unknown variables in ANN-
algorithm
VarSize=[1 nVar];                             % matrix size of unknown variables
VarMin = -1;                                  % Lower bound of unknown variables
VarMax = 1;                                   % Upper bound of unknown variables
maxTrials = 25;                               % Maximum number of trials
%method = 'sobol';                            % Halton or Sobol

%% Initialization
empty_particle.position = [];
empty_particle.velocity = [];
empty_particle.cost = [];
empty_particle.best.position = [];
empty_particle.best.cost = [];
```

```
empty_sol.BestCosts = [];
empty_sol.Cost = [];
empty_sol.p = [];
empty_sol.Uapp = [];
empty_sol.errorL1 = [];
empty_sol.errorL2 = [];
empty_sol.RelErrL2 = [];

for method ={'sobol','halton'}
   method = char(method);
   for Nx = [11,21,41,81]
      for Nt = [11,21,41,81]
         fprintf('N_x = %2d \t N_t = %2d\n',Nx,Nt);

         xval=linspace(xA,xB,Nx);                    % Spatial vector x
         tval=linspace(0,Tf,Nt);                   % Time vector t
         Uexc = exact_u(xval,tval);                  % Exact solution

         sol = repmat(empty_sol, maxTrials, 1);
         for trialId = 1:maxTrials

            %fprintf('\n\n-------------------------------------------------\n');
            fprintf('\t\t Trial Id : %d\n', trialId);
            T=cputime;
            ww=1;
            % Generate initial population
            particle = repmat(empty_particle, nPop, 1);

            % Initialize global best
            GlobalBest.cost = inf;

            %% Construct the initial simplex
            leapInd = randi(maxTrials);
            skipInd = randi(maxTrials);
            Q = qrandstream(method,nVar,'Leap',leapInd,'Skip',skipInd);
            Seq = qrand(Q,nPop);       % Halton or Sobol sequence

            for i=1:nPop
               % Generate population
               %     particle(i).position = unifrnd(VarMin, VarMax, VarSize);
               particle(i).position = VarMin+(VarMax-VarMin)*Seq(i,:);

               % Initialize velocity
               particle(i).velocity = zeros(VarSize);

               % Evaluation
               particle(i).cost = CostFunction(xval,tval,particle(i).position) ;

               % Update the personel best
               particle(i).best.position = particle(i).position ;
```

```
    particle(i).best.cost = particle(i).cost ;

    % Update the global best
    if particle(i).best.cost < GlobalBest.cost
        GlobalBest = particle(i).best;
    end
end

% Array to hold the best cost value on each iteration
BestCosts = zeros(MaxIt,1);

%% Main loop of Pso
for it=1:MaxIt

    for i=1:nPop
        particle(i).velocity =  ww*particle(i).velocity...
            +c1*rand(VarSize).*(particle(i).best.position-particle(i).position)...
            +c2*rand(VarSize).*(GlobalBest.position-particle(i).position);

        particle(i).position = particle(i).position +  particle(i).velocity;

        particle(i).cost = CostFunction ( xval,tval,particle(i).position) ;

        if particle(i).cost < particle(i).best.cost

            particle(i).best.position = particle(i).position;
            particle(i).best.cost = particle(i).cost;

            % Update the global best
            if particle(i).best.cost < GlobalBest.cost
                GlobalBest = particle(i).best;
            end
        end
    end

    % Store best cost value
    BestCosts(it) = GlobalBest.cost;

    % Display iteration information
    %fprintf('\t\tIteration %4d : Best Cost = %4.3e\n', it, BestCosts(it) );

    % Damping intertia coefficient
    ww = ww*wdamp;
end


%% Construct approximate solution
p_opt = GlobalBest.position;
Uapp = trial_u(xval,tval,p_opt);
```

```
    errorL1=(sum(sum(abs(Uexc-Uapp))))/(Nx*Nt);
    errorL2=norm(Uexc-Uapp,'fro')/(sqrt(Nx*Nt));
    RelErrL2=errorL2/norm(Uexc,'fro')/(sqrt(Nx*Nt));

    sol(trialId).BestCosts = BestCosts;
    sol(trialId).Cost = GlobalBest.cost;
    sol(trialId).p = p_opt;
    sol(trialId).Uapp = Uapp;
    sol(trialId).errorL1 = errorL1;
    sol(trialId).errorL2 = errorL2;
    sol(trialId).RelErrL2 = RelErrL2;
    sol(trialId).elapsedTime = cputime - T;

end
Costs = [sol(1:maxTrials).Cost];
[~,minId] = min(Costs);

BestCosts = sol(minId).BestCosts ;
p_opt = sol(minId).p;
Uapp =  sol(minId).Uapp;
errorL1 = sol(minId).errorL1;
errorL2 = sol(minId).errorL2;
RelErrL2 = sol(minId).RelErrL2;
%errorL3 = sol(minId).errorL3;

%% Results
fileName = ['Results\PSO_' method '_' num2str(Nx) '_'  num2str(Nt) '.txt'];
fid = fopen(fileName,'w+');

minErrorL1 = min([sol(1:maxTrials).errorL1]);
worstErrorL1 = max([sol(1:maxTrials).errorL1]);
meanErrorL1 = mean([sol(1:maxTrials).errorL1]);
stdErrorL1 = std([sol(1:maxTrials).errorL1]);

minErrorL2 = min([sol(1:maxTrials).errorL2]);
worstErrorL2 = max([sol(1:maxTrials).errorL2]);
meanErrorL2 = mean([sol(1:maxTrials).errorL2]);
stdErrorL2 = std([sol(1:maxTrials).errorL2]);

minRelErrL2 = min([sol(1:maxTrials).RelErrL2]);
worstRelErrL2 = max([sol(1:maxTrials).RelErrL2]);
meanRelErrL2 = mean([sol(1:maxTrials).RelErrL2]);
stdRelErrL2 = std([sol(1:maxTrials).RelErrL2]);

meanTime = mean([sol(1:maxTrials).elapsedTime]);
stdTime = std([sol(1:maxTrials).elapsedTime]);

fprintf('\n\n');
pm = 177;               % ASCII code for plus minus symbol
disp('********************* Errors *********************')
```

```
        fprintf('Min of AbsErrorL1 = %0.3e\n', minErrorL1);
        fprintf('Worst of AbsErrorL1 = %0.3e\n', worstErrorL1);
        fprintf('Mean of AbsErrorL1 = %0.3e %c %0.3e\n\n', meanErrorL1, pm,
stdErrorL1);

        fprintf('Min of AbsErrorL2 = %0.3e\n', minErrorL2);
        fprintf('Worst of AbsErrorL2 = %0.3e\n', worstErrorL2);
        fprintf('Mean of AbsErrorL2 = %0.3e %c %0.3e\n\n', meanErrorL2, pm,
stdErrorL2);

        fprintf('Min of RelErrorInf = %0.3e\n', minRelErrL2);
        fprintf('Worst of RelErrorInf = %0.3e\n', worstRelErrL2);
        fprintf('Mean of RelErrorInf = %0.3e %c %0.3e\n\n', meanRelErrL2, pm,
stdRelErrL2);

        fprintf('Mean of Elapsed Time = %0.3e %c %0.3e seconds\n\n', meanTime,
pm, stdTime);

        fprintf(fid,'Min of AbsErrorL1 = %0.3e\n', minErrorL1);
        fprintf(fid,'Worst of AbsErrorL1 = %0.3e\n', worstErrorL1);
        fprintf(fid,'Mean of AbsErrorL1 = %0.3e %c %0.3e\n\n', meanErrorL1, pm,
stdErrorL1);

        fprintf(fid,'Min of AbsErrorL2 = %0.3e\n', minErrorL2);
        fprintf(fid,'Worst of AbsErrorL2 = %0.3e\n', worstErrorL2);
        fprintf(fid,'Mean of AbsErrorL2 = %0.3e %c %0.3e\n\n', meanErrorL2, pm,
stdErrorL2);

        fprintf(fid,'Min of RelErrorInf = %0.3e\n', minRelErrL2);
        fprintf(fid,'Worst of RelErrorInf = %0.3e\n', worstRelErrL2);
        fprintf(fid,'Mean of RelErrorInf = %0.3e %c %0.3e\n\n', meanRelErrL2, pm,
stdRelErrL2);

        fprintf(fid,'Mean of Elapsed Time = %0.3e %c %0.3e seconds\n\n',
meanTime, pm, stdTime);

        fclose(fid);

        results.ErrorL1.min = minErrorL1;
        results.ErrorL1.worst = worstErrorL1;
        results.ErrorL1.mean = meanErrorL1;
        results.ErrorL1.std = stdErrorL1;

        results.ErrorL2.min = minErrorL2;
        results.ErrorL2.worst = worstErrorL2;
        results.ErrorL2.mean = meanErrorL2;
        results.ErrorL2.std = stdErrorL2;

        results.RelErrorL2.min = minRelErrL2;
        results.RelErrorL2.worst = worstRelErrL2;
```

```
        results.RelErrorL2.mean = meanRelErrL2;
        results.RelErrorL2.std = stdRelErrL2;

        %% Plots
        %fig1 = figure;
        % axis([xA xB 0 Tf 0 max(max(Uexc))])
        %subplot(1,2,1);mesh(xval,tval,Uexc);
        %xlabel('$x$','Interpreter','latex')
        %ylabel('$t$','Interpreter','latex')
        %zlabel('Exact Solution','Interpreter','latex')
        %% axis([xA xB 0 Tf 0 max(max(Uapp))])
        %subplot(1,2,2);mesh(xval,tval,Uapp)
        %xlabel('$x$','Interpreter','latex')
        %ylabel('$t$','Interpreter','latex')
        %zlabel('Numerical Solution','Interpreter','latex')
        %print(fig1,['Fig1_' method '.eps'],'-depsc','-r300');
        %print(fig1,['Fig1_' method '.jpg'],'-djpeg','-r300');
        %
        %fig2=figure;
        %subplot(1,2,1);plot(xval,Uexc(1,:),'k-',xval,Uapp(1,:),'r-.','LineWidth',1.5)
        %xlabel('$x$','FontSize',12,'Interpreter','latex')
        %ylabel('Numerical solution at $t=0$','FontSize',12,'Interpreter','latex')
        %legend('Exact Solution','Numerical solution','Location','northwest')
        %subplot(1,2,2);semilogy(BestCosts(1:200), 'linewidth',2)
        %xlabel('Iterations','FontSize',12,'Interpreter','latex')
        %ylabel('Best Cost','FontSize',12,'Interpreter','latex')
        %grid on
        %print(fig2,['Fig2_' method '.eps'],'-depsc','-r300');
        %print(fig2,['Fig2_' method '.jpg'],'-djpeg','-r300');

        save(['Results\PSO_' method '_' num2str(Nx) '_'  num2str(Nt)
'.mat'],'sol','BestCosts','p_opt','Uexc','Uapp','results');
      end
   end
end
```

GRADYAN İNİŞİ Matlab Kodları

Değer (Cost) Fonksiyonunun hesaplanması;

```
function fcost = Cost(xval,tval,p)
% This function computes values of the Cost Function defined as
% E(p)=(1/2)*Sum_x*Sum_t (eij)^2
% eij=U_t+(1/2)*x^2*A^2*U_xx+B*x*U_x-C*U,
% U(x,t) is the trial function defined as
% U(x,t)=(t/Tf)*P(x)+x*(Tf-t)*N(x,t;p)
global A B C E
Nx = length(xval);
Nt = length(tval);
Tf=tval(Nt);
xA=xval(1);
fE=max(xval-E,0);
fdPx=dP_x(xval);
fNet=Net(xval,tval,p);
fdNet_t=dNet_t(xval,tval,p);
fdNet_x=dNet_x(xval,tval,p);
fd2Net_xx=d2Net_xx(xval,tval,p);
fcost =0;
for j=1:Nt
   tj=tval(j);
   for i=1:Nx
      xi=xval(i);
      fdtrial_t=fE(i)/Tf-(xi-xA)*fNet(i,j)+(xi-xA)*(Tf-tj)*fdNet_t(i,j);
      ftrial=tj*fE(i)/Tf+(xi-xA)*(Tf-tj)*fNet(i,j);
      fdtrial_x=tj*fdPx(i)/Tf+(Tf-tj)*fNet(i,j)+(xi-xA)*(Tf-tj)*fdNet_x(i,j);
      fd2trial_xx=2*(Tf-tj)*fdNet_x(i,j)+(xi-xA)*(Tf-tj)*fd2Net_xx(i,j);
      fcost =  fcost+0.5*(fdtrial_t+0.5*xi^2*A^2*fd2trial_xx+B*xi*fdtrial_x-
C*ftrial)^2;
   end
end
```

Net fonksiyonun türevinin hesaplanması (d2Net_dalpha);

```
function d2fNet = d2Net_dalpha(xx,tt,p)
Nx=length(xx);Nt=length(tt);
d2fNet=zeros(Nx,Nt);
```

Net fonksiyonun uzay değişkenine göre türevinin hesaplanması (d2Net_dalpha_dS);

```
function fdNet_t = d2Net_dalpha_dS(xx,tt,p)
% This function computes values of the time derivative of the Net function N(x,t;p)
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
%alpha = p(1:M);
omegaA = p(M+1:2*M);
omegaB = p(2*M+1:3*M);
bias = p(3*M+1:4*M);
fdNet_t=zeros(M,Nx,Nt); %outputs of neural network
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      for k=1:M
         zk = omegaA(k)*t+omegaB(k)*x+bias(k);
         dfz = dsigma(zk);
         fdNet_t(k,i,j) = omegaB(k)*dfz;
      end
   end
end
end
```

Net fonksiyonun zaman değişkenine göre türevinin hesaplanması (d2Net_dalpha_dt);

```
function fdNet_t = d2Net_dalpha_dt(xx,tt,p)
% This function computes values of the time derivative of the Net function N(x,t;p)
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
alpha = p(1:M);
omegaA = p(M+1:2*M);
omegaB = p(2*M+1:3*M);
bias = p(3*M+1:4*M);
fdNet_t=zeros(M,Nx,Nt); %outputs of neural network
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      for k=1:M
         zk = omegaA(k)*t+omegaB(k)*x+bias(k);
         dfz = dsigma(zk);
         fdNet_t(k,i,j) = omegaA(k)*dfz;
      end
   end
end
end
```

Net fonksiyonun türevinin hesaplanması (d2Net_dbeta);

```
function dfNet = d2Net_dbeta(xx,tt,p)
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
alpha = p(1:M);
omegaA = p(M+1:2*M);
omegaB = p(2*M+1:3*M);
bias = p(3*M+1:4*M);
dfNet=zeros(Nx,Nt);
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      for k=1:M
         zk = omegaA(k)*t+omegaB(k)*x+bias(k);
         dfNet(i,j) = alpha*d2sigma(zk);
      end
   end
end
end
```

Net fonksiyonun uzay değişkenine göre türevinin hesaplanması (d2Net_dbeta_dS);

```
function dfNet = d2Net_dbeta_dS(xx,tt,p)
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
alpha = p(1:M);
omegaA = p(M+1:2*M);
omegaB = p(2*M+1:3*M);
bias = p(3*M+1:4*M);
dfNet=zeros(M,Nx,Nt);
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      for k=1:M
         zk = omegaA(k)*t + omegaB(k)*x + bias(k);
         dfNet(k,i,j) = alpha(k)*omegaB(k)*d2sigma(zk);
      end
   end
end
end
```

Net fonksiyonun zaman değişkenine göre türevinin hesaplanması (d2Net_dbeta_dt);

```
function fdNet_t = d2Net_dbeta_dt(xx,tt,p)
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
alpha = p(1:M);
omegaA = p(M+1:2*M);
omegaB = p(2*M+1:3*M);
bias = p(3*M+1:4*M);
fdNet_t=zeros(M,Nx,Nt); %outputs of neural network
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      for k=1:M
         zk = omegaA(k)*t+omegaB(k)*x+bias(k);
         d2fz = d2sigma(zk);
         fdNet_t(k,i,j) = alpha(k)*omegaA(k)*d2fz;
      end
   end
end
end
```

Net fonksiyonun türevinin hesaplanması (d2Net_domega);

```
function dfNet = d2Net_domega(xx,tt,p)
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
alpha = p(1:M);
omegaA = p(M+1:2*M);
omegaB = p(2*M+1:3*M);
bias = p(3*M+1:4*M);
dfNet=zeros(Nx,Nt);
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      for k=1:M
         zk = omegaA(k)*t+omegaB(k)*x+bias(k);
         dfNet(i,j) = alpha(k)*t(j)^2*d2sigma(zk);
      end
   end
end
end
```

Net fonksiyonun uzay değişkenine göre türevinin hesaplanması (d2Net_domega_dS);

```
function dfNet = d2Net_domega_dS(xx,tt,p)
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
alpha = p(1:M);
omegaA = p(M+1:2*M);
omegaB = p(2*M+1:3*M);
bias = p(3*M+1:4*M);
dfNet=zeros(M,Nx,Nt);
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      for k=1:M
         zk = omegaA(k)*t + omegaB(k)*x + bias(k);
         dfNet(k,i,j) = alpha(k)*omegaB(k)*tt(j)*d2sigma(zk);
      end
   end
end
end
```

Net fonksiyonun zaman değişkenine göre türevinin hesaplanması (d2Net_domega_dt);

```
function dfNet = d2Net_domega_dt(xx,tt,p)
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
alpha = p(1:M);
omegaA = p(M+1:2*M);
omegaB = p(2*M+1:3*M);
bias = p(3*M+1:4*M);
dfNet=zeros(M,Nx,Nt);
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      for k=1:M
         zk = omegaA(k)*t + omegaB(k)*x + bias(k);
         dfNet(k,i,j) = alpha(k)*tt(j)*dsigma(zk)*(1+omegaA(k)*tt(j)-
2*omegaA(k)*tt(j)*sigma(zk));
      end
   end
end
end
```

Net fonksiyonun türevinin hesaplanması (d2Net_domegaB);

```
function dfNet = d2Net_domegaB(xx,tt,p)
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
alpha = p(1:M);
omegaA = p(M+1:2*M);
omegaB = p(2*M+1:3*M);
bias = p(3*M+1:4*M);
dfNet=zeros(Nx,Nt);
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      for k=1:M
         zk = omegaA(k)*t+omegaB(k)*x+bias(k);
         dfNet(i,j) = alpha(k)*x(i)^2*d2sigma(zk);
      end
   end
end
end
```

Net fonksiyonun uzay değişkenine göre türevinin hesaplanması (d2Net_domegaB_dS);

```
function dfNet = d2Net_domegaB_dS(xx,tt,p)
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
alpha = p(1:M);
omegaA = p(M+1:2*M);
omegaB = p(2*M+1:3*M);
bias = p(3*M+1:4*M);
dfNet=zeros(M,Nx,Nt);
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      for k=1:M
         zk = omegaA(k)*t + omegaB(k)*x + bias(k);
         dfNet(k,i,j) = alpha(k)*d2sigma(zk)*(1+omegaB(k)*x-
2*omegaB(k)*x*sigma(zk));
      end
   end
end
end
```

Net fonksiyonun zaman değişkenine göre türevinin hesaplanması (d2Net_domegaB_dt);

```
function dfNet = d2Net_domegaB_dt(xx,tt,p)
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
alpha = p(1:M);
omegaA = p(M+1:2*M);
omegaB = p(2*M+1:3*M);
bias = p(3*M+1:4*M);
dfNet=zeros(M,Nx,Nt);
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      for k=1:M
         zk = omegaA(k)*t + omegaB(k)*x + bias(k);
         dfNet(k,i,j) = alpha(k)*omegaA(k)*xx(i)*d2sigma(zk);
      end
   end
end
end
```

Net fonksiyonun zaman değişkenine göre türevinin hesaplanması (d2Net_S);

```
function fdNet_t = d2Net_S(xx,tt,p)
% This function computes values of the space derivative of the Net function N(x,t;p)
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
alpha = p(1:M);
omegaA = p(M+1:2*M);
omegaB = p(2*M+1:3*M);
bias = p(3*M+1:4*M);
fdNet_t=zeros(Nx,Nt); %outputs of neural network
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      for k=1:M
         zk = omegaA(k)*t+omegaB(k)*x+bias(k);
         d2fz = d2sigma(zk);
         fdNet_t(i,j) = fdNet_t(i,j)+alpha(k)*omegaB(k)^2*d2fz;
      end
   end
end
end
```

Net fonksiyonun zaman değişkenine göre türevinin hesaplanması (d2Net_xx);

```
function fd2Net_xx = d2Net_xx(xx,tt,p)
% This function computes values of the 2nd-order
% spatial derivative of the Net function N(x,t;p)
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
alpha = p(1:M);
omegaA = p(M+1:2*M);
omegaB = p(2*M+1:3*M);
bias = p(3*M+1:4*M);
fd2Net_xx=zeros(Nx,Nt); %outputs of neural network
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      for k=1:M
         zk= omegaA(k)*t+omegaB(k)*x+bias(k);
         fz=fsigma(zk);
         fd2Net_xx(i,j) = fd2Net_xx(i,j)+alpha(k)*omegaB(k)^2*fz*(1-3*fz+2*fz^2);
      end
   end
end
end
```

Aktivasyon fonksiyonunun (Sigma) 2. Mertebe türev değerlerinin hesaplanması d2sigma;

```
function y = d2sigma(z)
  f = sigma(z);
  y = f.*(1-f).*(1-2*f);
end
```

Deneme (trial) fonksiyonun zaman değişkenine göre türevinin hesaplanması (d2trialu_dalpha_dS);

```
function ftrial = d2trialu_dalpha_dS(xx,tt,p)
%global E
M = floor(length(p)/4);
m = length(xx);
n = length(tt);
ftrial = zeros(M,m,n);
xA=xx(1);Tf=tt(end);
%fNet=Net(xx,tt,p);
%dfNet_dt = dNet_dt(xx,tt,p);
dfNet_dalpha = dNet_dalpha(xx,tt,p);
d2fNet_dalpha_dS = d2Net_dalpha_dS(xx,tt,p);
```

```
%fPx=max(xx-E,0);
for k = 1:M
for i=1:m
for j=1:n
   ftrial(k,i,j)= (Tf-tt(j))*dfNet_dalpha(k,i,j) + (xx(i)-xA)*(Tf-
tt(j))*d2fNet_dalpha_dS(k,i,j);
end
end
end
```

Deneme (trial) fonksiyonun zaman değişkenine göre türevinin hesaplanması (d2trialu_dalpha_dt);

```
function ftrial = d2trialu_dalpha_dt(xx,tt,p)
%global E
M = floor(length(p)/4);
m = length(xx);
n = length(tt);
ftrial = zeros(M,m,n);
xA=xx(1);Tf=tt(end);
%fNet=Net(xx,tt,p);
%dfNet_dt = dNet_dt(xx,tt,p);
dfNet_dalpha = dNet_dalpha(xx,tt,p);
d2fNet_dalpha_dt = d2Net_dalpha_dt(xx,tt,p);
%fPx=max(xx-E,0);
for k=1:M
for i=1:m
for j=1:n
   ftrial(k,i,j)= -(xx(i)-xA)*dfNet_dalpha(k,i,j)+(xx(i)-xA)*(Tf-
tt(j))*d2fNet_dalpha_dt(k,i,j);
end
end
end
```

Deneme (trial) fonksiyonun uzay değişkenine göre türevinin hesaplanması (d2trialu_dbeta_dS);

```
function ftrial = d2trialu_dbeta_dS(xx,tt,p)
%global E
M = length(p)/4; %number of neurons
m = length(xx);
n = length(tt);
ftrial = zeros(M,m,n);
xA=xx(1);Tf=tt(end);
dfNet_dbeta=dNet_dbeta(xx,tt,p);
d2fNet_dbeta_dS=d2Net_dbeta_dS(xx,tt,p);
%fPx=max(xx-E,0);
for k=1:M
```

```
for i=1:m
for j=1:n
   ftrial(k,i,j) = (Tf-tt(j))*dfNet_dbeta(k,i,j) + (xx(i)-xA)*(Tf-
tt(j))*d2fNet_dbeta_dS(k,i,j);
end
end
end
```

Deneme (trial) fonksiyonun zaman değişkenine göre türevinin hesaplanması (d2trialu_dbeta_dt);

```
function ftrial = d2trialu_dbeta_dt(xx,tt,p)
%global E
M = length(p)/4; %number of neurons
m = length(xx);
n = length(tt);
ftrial = zeros(M,m,n);
xA=xx(1);Tf=tt(end);
%fNet=Net(xx,tt,p);
%dfNet_dt = dNet_dt(xx,tt,p);
dfNet_dbeta = dNet_dbeta(xx,tt,p);
d2fNet_dbeta_dt = d2Net_dbeta_dt(xx,tt,p);
%fPx=max(xx-E,0);
for k=1:M
for i=1:m
for j=1:n
   ftrial(k,i,j)= -(xx(i)-xA)*dfNet_dbeta(k,i,j)+(xx(i)-xA)*(Tf-
tt(j))*d2fNet_dbeta_dt(k,i,j);
end
end
end
```

Deneme (trial) fonksiyonun uzay değişkenine göre türevinin hesaplanması (d2trialu_domega_dS);

```
function ftrial = d2trialu_domega_dS(xx,tt,p)
%global E
M = length(p)/4; %number of neurons
m = length(xx);
n = length(tt);
ftrial = zeros(M,m,n);
xA=xx(1);Tf=tt(end);
dfNet_domega=dNet_domega(xx,tt,p);
d2fNet_domega_dS=d2Net_domega_dS(xx,tt,p);
%fPx=max(xx-E,0);
for k=1:M
for i=1:m
for j=1:n
   ftrial(k,i,j) = (Tf-tt(j))*dfNet_domega(k,i,j) + (xx(i)-xA)*(Tf-
tt(j))*d2fNet_domega_dS(k,i,j);
```

```
end
end
end
```

Deneme (trial) fonksiyonun zaman değişkenine göre türevinin hesaplanması (d2trialu_domega_dt);

```
function ftrial = d2trialu_domega_dt(xx,tt,p)
%global E
M =length(p)/4;
m = length(xx);
n = length(tt);
ftrial = zeros(M,m,n);
xA=xx(1);Tf=tt(end);
%fNet=Net(xx,tt,p);
%dfNet_dt = dNet_dt(xx,tt,p);
dfNet_domega = dNet_domega(xx,tt,p);
d2fNet_domega_dt = d2Net_domega_dt(xx,tt,p);
%fPx=max(xx-E,0);
for k=1:M
for i=1:m
for j=1:n
   ftrial(k,i,j)= -(xx(i)-xA)*dfNet_domega(k,i,j)+(xx(i)-xA)*(Tf-
tt(j))*d2fNet_domega_dt(k,i,j);
end
end
end
```

Deneme (trial) fonksiyonun uzay değişkenine göre türevinin hesaplanması (d2trialu_domegaB_dS);

```
function ftrial = d2trialu_domegaB_dS(xx,tt,p)
%global E
M = length(p)/4; %number of neurons
m = length(xx);
n = length(tt);
ftrial = zeros(M,m,n);
xA=xx(1);Tf=tt(end);
dfNet_domegaB=dNet_domegaB(xx,tt,p);
d2fNet_domegaB_dS=d2Net_domegaB_dS(xx,tt,p);
%fPx=max(xx-E,0);
for k=1:M
for i=1:m
for j=1:n
   ftrial(k,i,j) = (Tf-tt(j))*dfNet_domegaB(k,i,j) + (xx(i)-xA)*(Tf-
tt(j))*d2fNet_domegaB_dS(k,i,j);
end
end
end
```

Deneme (trial) fonksiyonun zaman değişkenine göre türevinin hesaplanması (d2trialu_domegaB_dt);

```
function ftrial = d2trialu_domegaB_dt(xx,tt,p)
%global E
M = length(p)/4; %number of neurons
m = length(xx);
n = length(tt);
ftrial = zeros(M,m,n);
xA=xx(1);Tf=tt(end);
%fNet=Net(xx,tt,p);
%dfNet_dt = dNet_dt(xx,tt,p);
dfNet_domegaB = dNet_domegaB(xx,tt,p);
d2fNet_domegaB_dt = d2Net_domegaB_dt(xx,tt,p);
%fPx=max(xx-E,0);
for k=1:M
for i=1:m
for j=1:n
   ftrial(k,i,j)= -(xx(i)-xA)*dfNet_domegaB(k,i,j)+(xx(i)-xA)*(Tf-
tt(j))*d2fNet_domegaB_dt(k,i,j);
end
end
end
```

Deneme (trial) fonksiyonun uzay değişkenine göre türevinin hesaplanması (d2trialu_dS);

```
function ftrial = d2trialu_dS(xx,tt,p)
global E
m = length(xx);
n = length(tt);
ftrial = zeros(m,n);
xA=xx(1);Tf=tt(end);
%fNet=Net(xx,tt,p);
dfNet_dS = dNet_S(xx,tt,p);
d2fNet_dS = d2Net_S(xx,tt,p);
%fPx=max(xx-E,0);
for i=1:m
for j=1:n
    ftrial(i,j) = 2*(Tf-tt(j))*dfNet_dS(i,j) + (xx(i)-xA)*(Tf-tt(j))*d2fNet_dS(i,j);
end
end
```

Net fonksiyonun uzay değişkenine göre türevinin hesaplanması (d3Net_dalpha_dS2);

```
function fdNet_t = d3Net_dalpha_dS2(xx,tt,p)
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
alpha = p(1:M);
omegaA = p(M+1:2*M);
omegaB = p(2*M+1:3*M);
bias = p(3*M+1:4*M);
fdNet_t=zeros(M,Nx,Nt); %outputs of neural network
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      for k=1:M
         zk = omegaA(k)*t+omegaB(k)*x+bias(k);
         d2fz = d2sigma(zk);
         fdNet_t(M,i,j) = omegaB(k)^2*d2fz;
      end
   end
end
end
```

Net fonksiyonun uzay değişkenine göre türevinin hesaplanması (d3Net_dbeta_dS2);

```
function fdNet_t = d3Net_dbeta_dS2(xx,tt,p)
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
alpha = p(1:M);
omegaA = p(M+1:2*M);
omegaB = p(2*M+1:3*M);
bias = p(3*M+1:4*M);
fdNet_t=zeros(M,Nx,Nt); %outputs of neural network
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      for k=1:M
         zk = omegaA(k)*t+omegaB(k)*x+bias(k);
         d3fz = d3sigma(zk);
         fdNet_t(k,i,j) = alpha(k)*omegaB(k)^2*d3fz;
      end
   end
end
end
```

Net fonksiyonun uzay değişkenine göre türevinin hesaplanması (d3Net_domega_dS2);

```
function dfNet = d3Net_domega_dS2(xx,tt,p)
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
alpha = p(1:M);
omegaA = p(M+1:2*M);
omegaB = p(2*M+1:3*M);
bias = p(3*M+1:4*M);
dfNet=zeros(M,Nx,Nt);
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      for k=1:M
         zk = omegaA(k)*t + omegaB(k)*x + bias(k);
         fz = sigma(zk);
         d2fz = d2sigma(zk);
         dfNet(i,j) = alpha(k)*omegaB(k)^2*tt(j)*d2fz*((1-2*fz)^2 -2*(1-fz));
      end
   end
end
end
```

Net fonksiyonun uzay değişkenine göre türevinin hesaplanması (d3Net_domegaB_dS2);

```
function dfNet = d3Net_domegaB_dS2(xx,tt,p)
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
alpha = p(1:M);
omegaA = p(M+1:2*M);
omegaB = p(2*M+1:3*M);
bias = p(3*M+1:4*M);
dfNet=zeros(M,Nx,Nt);
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      for k=1:M
         zk = omegaA(k)*t + omegaB(k)*x + bias(k);
         fz = sigma(zk);
         dfz = dsigma(zk);
         dfNet(k,i,j) = alpha(k)*omegaB(k)*dfz*(2 - 4*fz +omegaB(k)*x -
2*omegaB(k)*dfz);
      end
   end
end
end
```

Aktivasyon fonksiyonunun (Sigmoid) 3. Mertebe türev değerlerinin hesaplanması (d3sigma);

```
function y = d3sigma(z)
  f = sigma(z);
  y = f.*(1-f).*((1-2*f).^2 - 2*(1-f));
end
```

Deneme (trial) fonksiyonunun uzay değişkenine göre türev değerlerinin hesaplanması (d3trialu_dalpha_dS2);

```
function fdNet = d3trialu_dalpha_dS2(xx,tt,p)
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
xA=xx(1);Tf=tt(end);
fdNet = zeros(M,Nx,Nt); %outputs of neural network
d2fNet_dalpha_dS = d2Net_dalpha_dS(xx,tt,p);
d3fNet_dalpha_dS2 = d3Net_dalpha_dS2(xx,tt,p);
for k=1:M
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      fdNet(k,i,j) = 2*(Tf-t)*d2fNet_dalpha_dS(k,i,j) + (x - xA)*(Tf-
t)*d3fNet_dalpha_dS2(k,i,j);
   end
end
end
end
```

Deneme (trial) fonksiyonunun uzay değişkenine göre türev değerlerinin hesaplanması (d3trialu_dbeta_dS2);

```
function ftrial = d3trialu_dbeta_dS2(xx,tt,p)
%global E
M = length(p)/4; %number of neurons
m = length(xx);
n = length(tt);
ftrial = zeros(M,m,n);
xA=xx(1);Tf=tt(end);
d2fNet_dbeta_dS=d2Net_dbeta_dS(xx,tt,p);
d3fNet_dbeta_dS2=d3Net_dbeta_dS2(xx,tt,p);
%fPx=max(xx-E,0);
for k=1:M
for i=1:m
for j=1:n
   ftrial(k,i,j) = 2*(Tf-tt(j))*d2fNet_dbeta_dS(k,i,j) + (xx(i)-xA)*(Tf-
tt(j))*d3fNet_dbeta_dS2(k,i,j);
end
end
end
```

Deneme (trial) fonksiyonunun uzay değişkenine göre türev değerlerinin hesaplanması (d3trialu_domega_dS2);

```
function ftrial = d3trialu_domega_dS2(xx,tt,p)
%global E
M = length(p)/4; %number of neurons
m = length(xx);
n = length(tt);
ftrial = zeros(M,m,n);
xA=xx(1);Tf=tt(end);
d2fNet_domega_dS=d2Net_domega_dS(xx,tt,p);
d3fNet_domega_dS2=d3Net_domega_dS2(xx,tt,p);
%fPx=max(xx-E,0);
for k=1:M
for i=1:m
for j=1:n
   ftrial(k,i,j) = 2*(Tf-tt(j))*d2fNet_domega_dS(k,i,j) + (xx(i)-xA)*(Tf-
tt(j))*d3fNet_domega_dS2(k,i,j);
end
end
end
```

Deneme (trial) fonksiyonunun uzay değişkenine göre türev değerlerinin hesaplanması (d3trialu_domegaB_dS2);

```
function ftrial = d3trialu_domegaB_dS2(xx,tt,p)
%global E
M = length(p)/4; %number of neurons
m = length(xx);
n = length(tt);
ftrial = zeros(M,m,n);
xA=xx(1);Tf=tt(end);
d2fNet_domegaB_dS=d2Net_domegaB_dS(xx,tt,p);
d3fNet_domegaB_dS2=d3Net_domegaB_dS2(xx,tt,p);
%fPx=max(xx-E,0);
for k=1:M
for i=1:m
for j=1:n
   ftrial(k,i,j) = 2*(Tf-tt(j))*d2fNet_domegaB_dS(k,i,j) + (xx(i)-xA)*(Tf-
tt(j))*d3fNet_domegaB_dS2(k,i,j);
end
end
end
```

Net fonksiyonunun türev değerlerinin hesaplanması (dNet_dalpha);

```
function dfNet = dNet_dalpha(xx,tt,p)
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
%alpha = p(1:M);
omegaA = p(M+1:2*M);
omegaB = p(2*M+1:3*M);
bias = p(3*M+1:4*M);
dfNet = zeros(M,Nx,Nt);
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      for k=1:M
         zk = omegaA(k)*t+omegaB(k)*x+bias(k);
         dfNet(k,i,j) = sigma(zk);
      end
   end
end
end
```

Net fonksiyonunun türev değerlerinin hesaplanması (dNet_dbeta);

```
function dfNet = dNet_dbeta(xx,tt,p)
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
alpha = p(1:M);
omegaA = p(M+1:2*M);
omegaB = p(2*M+1:3*M);
bias = p(3*M+1:4*M);
dfNet=zeros(M,Nx,Nt);
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      for k=1:M
         zk = omegaA(k)*t+omegaB(k)*x+bias(k);
         dfNet(k,i,j) = alpha(k)*dsigma(zk);
      end
   end
end
end
```

Net fonksiyonunun türev değerlerinin hesaplanması (dNet_domega);

```
function dfNet = dNet_domega(xx,tt,p)
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
alpha = p(1:M);
```

```
omegaA = p(M+1:2*M);
omegaB = p(2*M+1:3*M);
bias = p(3*M+1:4*M);
dfNet=zeros(M,Nx,Nt);
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      for k=1:M
         zk = omegaA(k)*t+omegaB(k)*x+bias(k);
         dfNet(k,i,j) = alpha(k)*t*dsigma(zk);
      end
   end
end
end
```

Net fonksiyonunun türev değerlerinin hesaplanması (dNet_domegaB);

```
function dfNet = dNet_domegaB(xx,tt,p)
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
alpha = p(1:M);
omegaA = p(M+1:2*M);
omegaB = p(2*M+1:3*M);
bias = p(3*M+1:4*M);
dfNet=zeros(M,Nx,Nt);
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      for k=1:M
         zk = omegaA(k)*t+omegaB(k)*x+bias(k);
         dfNet(k,i,j) = alpha(k)*x*dsigma(zk);
      end
   end
end
end
```

Net fonksiyonunun uzay değişkenine göre türev değerlerinin hesaplanması (dNet_S);

```
function fdNet_t = dNet_S(xx,tt,p)
% This function computes values of the time derivative of the Net function N(x,t;p)
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
alpha = p(1:M);
omegaA = p(M+1:2*M);
omegaB = p(2*M+1:3*M);
bias = p(3*M+1:4*M);
fdNet_t=zeros(Nx,Nt); %outputs of neural network
```

```
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      for k=1:M
         zk = omegaA(k)*t+omegaB(k)*x+bias(k);
         dfz = dsigma(zk);
         fdNet_t(i,j) = fdNet_t(i,j)+alpha(k)*omegaB(k)*dfz;
      end
   end
end
end
```

Net fonksiyonunun zaman değişkenine göre türev değerlerinin hesaplanması (dNet_t);

```
function fdNet_t = dNet_t(xx,tt,p)
% This function computes values of the time derivative of the Net function N(x,t;p)
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
alpha = p(1:M);
omegaA = p(M+1:2*M);
omegaB = p(2*M+1:3*M);
bias = p(3*M+1:4*M);
fdNet_t=zeros(Nx,Nt); %outputs of neural network
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      for k=1:M
         zk = omegaA(k)*t+omegaB(k)*x+bias(k);
         dfz = dsigma(zk);
         fdNet_t(i,j) = fdNet_t(i,j)+alpha(k)*omegaA(k)*dfz;
      end
   end
end
end
```

Net fonksiyonunun zaman değişkenine göre türev değerlerinin hesaplanması (dNet_x);

```
function fdNet_x = dNet_x(xx,tt,p)
% This function computes values of the spatial derivative of the Net function
N(x,t;p)
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
alpha = p(1:M);
omegaA = p(M+1:2*M);
omegaB = p(2*M+1:3*M);
bias = p(3*M+1:4*M);
```

```
fdNet_x=zeros(Nx,Nt); %outputs of neural network
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      for k=1:M
         zk = omegaA(k)*t+omegaB(k)*x+bias(k);
         dfz = dsigma(zk);
         fdNet_x(i,j) = fdNet_x(i,j)+alpha(k)*omegaB(k)*dfz;
      end
   end
end
end
```

P(x) fonksiyonunun türev değerlerinin hesaplanması (dP_x);

```
function fdpx = dP_x(xx)
% This function computes values of the derivative of the function P(x)
% defined as P(x)=U(x,Tf), here, Tf is the final time point.
global E
Nx=length(xx);
fdpx=zeros(1,Nx);
for i=1:Nx
   x=xx(i);
if x < E
   fdpx(i)= 0;
else
   fdpx(i)=1;
end
end
```

Aktivasyon fonksiyonunun (sigmoid) türev değerlerinin hesaplanması (dsigma);

```
function z = dsigma(x)
  y = sigma(x);
  z = y.*(1-y);
end
```

Deneme (trial) fonksiyonunun türev değerlerinin hesaplanması (dtrialu_dalpha);

```
function ftrial = dtrialu_dalpha(xx,tt,p)
%global E
M = floor(length(p)/4);
m = length(xx);
n = length(tt);
ftrial = zeros(M,m,n);
xA=xx(1);Tf=tt(end);
dfNet_dalpha=dNet_dalpha(xx,tt,p);
%fPx=max(xx-E,0);
```

```
for k =1:M
for i=1:m
for j=1:n
   ftrial(k,i,j)=(xx(i)-xA)*(Tf-tt(j))*dfNet_dalpha(k,i,j);
end
end
end
```

Deneme (trial) fonksiyonunun türev değerlerinin hesaplanması (dtrialu_dbeta);

```
function ftrial = dtrialu_dbeta(xx,tt,p)
%global E
M = length(p)/4; %number of neurons
m = length(xx);
n = length(tt);
ftrial = zeros(M,m,n);
xA=xx(1);Tf=tt(end);
dfNet_dbeta=dNet_dbeta(xx,tt,p);
%fPx=max(xx-E,0);
for k=1:M
for i=1:m
for j=1:n
   ftrial(k,i,j)=(xx(i)-xA)*(Tf-tt(j))*dfNet_dbeta(k,i,j);
end
end
end
```

Deneme (trial) fonksiyonunun türev değerlerinin hesaplanması (dtrialu_domega);

```
function ftrial = dtrialu_domega(xx,tt,p)
%global E
M =length(p)/4;
m = length(xx);
n = length(tt);
ftrial = zeros(M,m,n);
xA=xx(1);Tf=tt(end);
dfNet_domega=dNet_domega(xx,tt,p);
%fPx=max(xx-E,0);
for k=1:M
for i=1:m
for j=1:n
   ftrial(k,i,j)=(xx(i)-xA)*(Tf-tt(j))*dfNet_domega(k,i,j);
end
end
end
```

Deneme (trial) fonksiyonunun türev değerlerinin hesaplanması (dtrialu_domegaB);

```
function ftrial = dtrialu_domegaB(xx,tt,p)
%global E
M = length(p)/4; %number of neurons
m = length(xx);
n = length(tt);
ftrial = zeros(M,m,n);
xA=xx(1);Tf=tt(end);
dfNet_domegaB=dNet_domegaB(xx,tt,p);
%fPx=max(xx-E,0);
for k=1:M
for i=1:m
for j=1:n
   ftrial(k,i,j)=(xx(i)-xA)*(Tf-tt(j))*dfNet_domegaB(k,i,j);
end
end
end
```

Deneme (trial) fonksiyonunun uzay değişkenine göre türev değerlerinin hesaplanması (dtrialu_dS);

```
function ftrial = dtrialu_dS(xx,tt,p)
global E
m = length(xx);
n = length(tt);
ftrial = zeros(m,n);
xA=xx(1);Tf=tt(end);
fNet=Net(xx,tt,p);
dfNet_dS = dNet_S(xx,tt,p);
%fPx=max(xx-E,0);
for i=1:m
for j=1:n
   term = (Tf-tt(j))*fNet(i,j) + (xx(i)-xA)*(Tf-tt(j))*dfNet_dS(i,j);
   if xx(i)<=E
      ftrial(i,j) = term;
   else
      ftrial(i,j) = tt(j) + term;
   end
end
end
```

Deneme (trial) fonksiyonunun zaman değişkenine göre türev değerlerinin hesaplanması (dtrialu_dt);

```
function ftrial = dtrialu_dt(xx,tt,p)
global E
m = length(xx);
n = length(tt);
ftrial = zeros(m,n);
xA=xx(1);Tf=tt(end);
fNet=Net(xx,tt,p);
dfNet_dt = dNet_dt(xx,tt,p);
fPx=max(xx-E,0);
for i=1:m
for j=1:n
   ftrial(i,j)=fPx(i)-(xx(i)-xA)*fNet(i,j)+(xx(i)-xA)*(Tf-tt(j))*dfNet_dt(i,j);
end
end
```

Kesin çözüm fonksiyonunun değerlerinin hesaplanması (exact_u);

```
function u=exact_u(xx,tt)
global A B  E
Nx=length(xx);Nt=length(tt);
Tf=tt(end);
u=zeros(Nt,Nx);
for i=1:Nx
   x=xx(i);
   for j=1:Nt-1
      t=tt(j);
      p1=log(x/E)+(B+A^2/2)*(Tf-t);
      pp=A*sqrt(Tf-t);
      p2=log(x/E)+(B-A^2/2)*(Tf-t);
      z1=p1/pp;z2=p2/pp;
      u(j,i)=x*0.5*erfc(-z1/sqrt(2))-0.5*E*exp(-B*(Tf-t))*erfc(-z2/sqrt(2));
   end
end
u(Nt,:)=max(xx-E,0);
```

P(x) fonksiyonunun değerlerinin hesaplanması (fPx);

```
function fpx = fPx(x)
% This function computes values of the function P(x) defined as P(x)=U(x,Tf);
% Here, Tf is the final time point.
EE=100;%sqrt(1/19);
fpx= max(x-EE,0);
end
```

fsigma;

```
function fz = fsigma(x)
   fz= 1./(1+exp(-x));
end
```

Net fonksiyonunun değerlerinin hesaplanması (Net);

```
function fNet = Net(xx,tt,p)
% This function computes values of the Net function N(x,t;p)
% fN is output of neural network corrsponding to the inputs x and t
Nx=length(xx);Nt=length(tt);
M = length(p)/4; %number of neurons
alpha = p(1:M);
omegaA = p(M+1:2*M);
omegaB = p(2*M+1:3*M);
bias = p(3*M+1:4*M);
fNet=zeros(Nx,Nt);
for i=1:Nx
   x=xx(i);
   for j=1:Nt
      t=tt(j);
      for k=1:M
         zk = omegaA(k)*t+omegaB(k)*x+bias(k);
         fNet(i,j)=fNet(i,j)+alpha(k)*sigma(zk);
      end
   end
end
end
```

Aktivasyon fonksiyonunun (Sigmoid) değerlerinin hesaplanması (sigma);

```
function y = sigma(z)
   y = 1./(1+exp(-z));
end
```

Deneme (trial) fonksiyonunun değerlerinin hesaplanması (trial_u);

```
function ftrial = trial_u(xx,tt,p)
global E
m = length(xx);
n = length(tt);
ftrial = zeros(m,n);
xA=xx(1);Tf=tt(end);
fNet=Net(xx,tt,p);
fPx=max(xx-E,0);
for i=1:m
for j=1:n
   ftrial(i,j)=tt(j)*fPx(i)/Tf+(xx(i)-xA)*(Tf-tt(j))*fNet(i,j);
end
end
```

Gradyan inişi yöntemi için ana program (GradientDescent_MainPr);

```
% This program solves the Black-Scholes problem defined as
% U_t+(1/2)*x^2*A^2*U_xx+B*x*U_x-C*U=0, 0<x<1, 0<t<Tf,
% U(0,t)=0, left boundary condition
%U(x,Tf)=P(x),final time condition, P(x)=max{x-E,0},
% by using Artificial Neural Networks (ANN) trained by Gradient Descent
tic
clc;
clear;
close;
global A B C E
%% Problem definition
CostFunction = @(x,t,p)  Cost(x,t,p);          % Cost function
A=0.1;B=0.12;C=B;                              % Coefficients of the equation
xA=70; xB=130;                                 % Left and right boundary points
Tf=1;                                          % Final time point
E=100;                                         % Strike price
N=5;                                           % Number of neurons
% Nt=21;Nx=21;                                 % Number of nodes
% xval=linspace(xA,xB,Nx);                     % Spatial vector x
% tval=linspace(0,Tf,Nt);                      % Time vector t
% Uexc = exact_u(xval,tval);                   % Exact solution

%% Parameters of Gradient Descent
MaxIt = 1000;                                  % Maximum number of iterations
nVar = 4*N;                                    % Mumber of unknown variables in ANN
VarSize=[1 nVar];                              % Matrix size of unknown variables
VarMin = -1;                                   % Lower bound of unknown variables
VarMax = 1;                                    % Upper bound of unknown variables
lambda_damp = 0.99;                            % Damping parameter of learning coefficient
maxTrials = 25;                                % Maximum number of trials

%% Initialization
empty_sol.BestCosts = [];
empty_sol.Cost = [];
empty_sol.p = [];
empty_sol.Uapp = [];
empty_sol.errorL1 = [];
empty_sol.errorL2 = [];
empty_sol.RelErrL2 = [];

for Nx = [11,21,41,81]
   for Nt = [11,21,41,81]
      fprintf('N_x = %2d \t N_t = %2d\n',Nx,Nt);

      xval=linspace(xA,xB,Nx);                 % Spatial vector x
      tval=linspace(0,Tf,Nt);                  % Time vector t
      Uexc = exact_u(xval,tval);               % Exact solution
```

```
        sol = repmat(empty_sol, maxTrials, 1);
        for trialId = 1:maxTrials

            %fprintf('\n\n-----------------------------------------------------\n');
            fprintf('\t\t Trial Id : %d\n', trialId);
            T=cputime;

            % Initialize global best
            GlobalBest.cost = inf;
            p = VarMin + (VarMax - VarMin)*rand(1,nVar);                % Initial
parameters of Neural Net
            Tf=tval(Nt);
            xA=xval(1);
            fE=max(xval-E,0);
            fdPx=dP_x(xval);

            % Array to hold the best cost value on each iteration
            BestCosts = zeros(MaxIt,1);
            M = length(p)/4; %number of neurons

            %% Main loop of Gradient Descent
            for it=1:MaxIt
               lambda = 1e-6/log(1+it);              % learning coefficient

               % Calculate Cost
               F = CostFunction(xval, tval, p);

               fNet=Net(xval,tval,p);
               fdNet_t=dNet_t(xval,tval,p);
               fdNet_x=dNet_x(xval,tval,p);
               fd2Net_xx=d2Net_xx(xval,tval,p);
               ftrial = zeros(Nx,Nt);
               fdtrial_t = zeros(Nx,Nt);
               fdtrial_x = zeros(Nx,Nt);
               fd2trial_xx = zeros(Nx,Nt);
               for j=1:Nt
                  tj=tval(j);
                  for i=1:Nx
                     xi=xval(i);
                     ftrial(i,j) = tj*fE(i)/Tf+(xi-xA)*(Tf-tj)*fNet(i,j);
                     fdtrial_t(i,j) = fE(i)/Tf-(xi-xA)*fNet(i,j)+(xi-xA)*(Tf-tj)*fdNet_t(i,j);
                     fdtrial_x(i,j) = tj*fdPx(i)/Tf+(Tf-tj)*fNet(i,j)+(xi-xA)*(Tf-
tj)*fdNet_x(i,j);
                     fd2trial_xx(i,j) = 2*(Tf-tj)*fdNet_x(i,j)+(xi-xA)*(Tf-
tj)*fd2Net_xx(i,j);
                  end
               end

               fdtrialu_dalpha = dtrialu_dalpha(xval,tval,p);
               fd2trialu_dalpha_dS = d2trialu_dalpha_dS(xval,tval,p);
```

```
fd2trialu_dalpha_dt = d2trialu_dalpha_dt(xval,tval,p);
fd3trialu_dalpha_dS2 = d3trialu_dalpha_dS2(xval,tval,p);

fdtrialu_domega = dtrialu_domega(xval,tval,p);
fd2trialu_domega_dt = d2trialu_domega_dt(xval,tval,p);
fd2trialu_domega_dS = d2trialu_domega_dS(xval,tval,p);
fd3trialu_domega_dS2 = d3trialu_domega_dS2(xval,tval,p);

fdtrialu_domegaB = dtrialu_domegaB(xval,tval,p);
fd2trialu_domegaB_dt = d2trialu_domegaB_dt(xval,tval,p);
fd2trialu_domegaB_dS = d2trialu_domegaB_dS(xval,tval,p);
fd3trialu_domegaB_dS2 = d3trialu_domegaB_dS2(xval,tval,p);

fdtrialu_dbeta = dtrialu_dbeta(xval,tval,p);
fd2trialu_dbeta_dt = d2trialu_dbeta_dt(xval,tval,p);
fd2trialu_dbeta_dS = d2trialu_dbeta_dS(xval,tval,p);
fd3trialu_dbeta_dS2 = d3trialu_dbeta_dS2(xval,tval,p);

e = zeros(Nx,Nt);
de_dalpha = zeros(M,Nx,Nt);
de_domega = zeros(M,Nx,Nt);
de_domegaB = zeros(M,Nx,Nt);
de_dbeta = zeros(M,Nx,Nt);
for j=1:Nt
   tj=tval(j);
   for i=1:Nx
      xi=xval(i);
      e(i,j) =
(fdtrial_t(i,j)+0.5*xi^2*A^2*fd2trial_xx(i,j)+B*xi*fdtrial_x(i,j)-C*ftrial(i,j));
      for k=1:M
         de_dalpha(k,i,j) =
fd2trialu_dalpha_dt(k,i,j)+0.5*xi^2*A^2*fd3trialu_dalpha_dS2(k,i,j)+B*xi*fd2trialu
_dalpha_dS(k,i,j)-C* fdtrialu_dalpha(k,i,j);
         de_domega(k,i,j) =
fd2trialu_domega_dt(k,i,j)+0.5*xi^2*A^2*fd3trialu_domega_dS2(k,i,j)+B*xi*fd2tri
alu_domega_dS(k,i,j)-C* fdtrialu_domega(k,i,j);
         de_domegaB(k,i,j) =
fd2trialu_domegaB_dt(k,i,j)+0.5*xi^2*A^2*fd3trialu_domegaB_dS2(k,i,j)+B*xi*fd
2trialu_domegaB_dS(k,i,j)-C* fdtrialu_domegaB(k,i,j);
         de_dbeta(k,i,j) =
fd2trialu_dbeta_dt(k,i,j)+0.5*xi^2*A^2*fd3trialu_dbeta_dS2(k,i,j)+B*xi*fd2trialu_d
beta_dS(k,i,j)-C* fdtrialu_dbeta(k,i,j);
      end
   end
end

DeltaAlpha = zeros(1,M);
DeltaOmegaA = zeros(1,M);
DeltaOmegaB = zeros(1,M);
DeltaBeta = zeros(1,M);
```

```
        for k=1:M
            DeltaAlpha(k) = sum(sum(e.*squeeze(de_dalpha(k,:,:))));
            DeltaOmegaA(k) = sum(sum(e.*squeeze(de_domega(k,:,:))));
            DeltaOmegaB(k) = sum(sum(e.*squeeze(de_domegaB(k,:,:))));
            DeltaBeta(k) = sum(sum(e.*squeeze(de_dbeta(k,:,:))));
        end


        % Update parameters
        alpha = p(1:M);
        omegaA = p(M+1:2*M);
        omegaB = p(2*M+1:3*M);
        bias = p(3*M+1:4*M);
        alpha = alpha - lambda*DeltaAlpha;
        omegaA = omegaA - lambda*DeltaOmegaA;
        omegaB = omegaB - lambda*DeltaOmegaB;
        bias = bias - lambda*DeltaBeta;

        p = [alpha omegaA omegaB bias];
        J = CostFunction(xval, tval, p);
        % Store best cost value
        BestCosts(it) = J;

        % Display iteration information
        %fprintf('\t\tIteration %4d : Best Cost = %5.4e\n', it, BestCosts(it) );

        % Damping intertia coefficient
        %lambda = lambda*lambda_damp;
        %lambda = 1e-7/sqrt(it)
    end

    %% Construct approximate solution
    p_opt = p;
    Uapp = trial_u(xval,tval,p_opt)'; %%%% Transpose ???

    errorL1=(sum(sum(abs(Uexc-Uapp))))/(Nx*Nt);
    errorL2=norm(Uexc-Uapp,'fro')/(sqrt(Nx*Nt));
    RelErrL2=errorL2/norm(Uexc,'fro')/(sqrt(Nx*Nt));

    sol(trialId).BestCosts = BestCosts;
    sol(trialId).Cost = GlobalBest.cost;
    sol(trialId).p = p_opt;
    sol(trialId).Uapp = Uapp;
    sol(trialId).errorL1 = errorL1;
    sol(trialId).errorL2 = errorL2;
    sol(trialId).RelErrL2 = RelErrL2;
    sol(trialId).elapsedTime = cputime - T;

end
```

```
Costs = [sol(1:maxTrials).Cost];
[~,minId] = min(Costs);

BestCosts = sol(minId).BestCosts ;
p_opt = sol(minId).p;
Uapp =  sol(minId).Uapp;
errorL1 = sol(minId).errorL1;
errorL2 = sol(minId).errorL2;
RelErrL2 = sol(minId).RelErrL2;
%errorL3 = sol(minId).errorL3;

%% Results
fileName = ['Results\GradientDescent_' num2str(Nx) '_'  num2str(Nt) '.txt'];
fid = fopen(fileName,'w+');

minErrorL1 = min([sol(1:maxTrials).errorL1]);
worstErrorL1 = max([sol(1:maxTrials).errorL1]);
meanErrorL1 = mean([sol(1:maxTrials).errorL1]);
stdErrorL1 = std([sol(1:maxTrials).errorL1]);

minErrorL2 = min([sol(1:maxTrials).errorL2]);
worstErrorL2 = max([sol(1:maxTrials).errorL2]);
meanErrorL2 = mean([sol(1:maxTrials).errorL2]);
stdErrorL2 = std([sol(1:maxTrials).errorL2]);

minRelErrL2 = min([sol(1:maxTrials).RelErrL2]);
worstRelErrL2 = max([sol(1:maxTrials).RelErrL2]);
meanRelErrL2 = mean([sol(1:maxTrials).RelErrL2]);
stdRelErrL2 = std([sol(1:maxTrials).RelErrL2]);

meanTime = mean([sol(1:maxTrials).elapsedTime]);
stdTime = std([sol(1:maxTrials).elapsedTime]);

fprintf('\n\n');
pm = 177;              % ASCII code for plus minus symbol
disp('********************* Errors *********************')
fprintf('Min of AbsErrorL1 = %0.4e\n', minErrorL1);
fprintf('Worst of AbsErrorL1 = %0.4e\n', worstErrorL1);
fprintf('Mean of AbsErrorL1 = %0.4e %c %0.4e\n\n', meanErrorL1, pm,
stdErrorL1);

fprintf('Min of AbsErrorL2 = %0.4e\n', minErrorL2);
fprintf('Worst of AbsErrorL2 = %0.4e\n', worstErrorL2);
fprintf('Mean of AbsErrorL2 = %0.4e %c %0.4e\n\n', meanErrorL2, pm,
stdErrorL2);

fprintf('Min of RelErrorInf = %0.4e\n', minRelErrL2);
fprintf('Worst of RelErrorInf = %0.4e\n', worstRelErrL2);
fprintf('Mean of RelErrorInf = %0.4e %c %0.4e\n\n', meanRelErrL2, pm,
stdRelErrL2);
```

```
        fprintf('Mean of Elapsed Time = %0.4e %c %0.4e seconds\n\n', meanTime, pm,
stdTime);

        fprintf(fid,'Min of AbsErrorL1 = %0.4e\n', minErrorL1);
        fprintf(fid,'Worst of AbsErrorL1 = %0.4e\n', worstErrorL1);
        fprintf(fid,'Mean of AbsErrorL1 = %0.4e %c %0.4e\n\n', meanErrorL1, pm,
stdErrorL1);

        fprintf(fid,'Min of AbsErrorL2 = %0.4e\n', minErrorL2);
        fprintf(fid,'Worst of AbsErrorL2 = %0.4e\n', worstErrorL2);
        fprintf(fid,'Mean of AbsErrorL2 = %0.4e %c %0.4e\n\n', meanErrorL2, pm,
stdErrorL2);

        fprintf(fid,'Min of RelErrorInf = %0.4e\n', minRelErrL2);
        fprintf(fid,'Worst of RelErrorInf = %0.4e\n', worstRelErrL2);
        fprintf(fid,'Mean of RelErrorInf = %0.4e %c %0.4e\n\n', meanRelErrL2, pm,
stdRelErrL2);

        fprintf(fid,'Mean of Elapsed Time = %0.4e %c %0.4e seconds\n\n', meanTime,
pm, stdTime);

        fclose(fid);

        results.ErrorL1.min = minErrorL1;
        results.ErrorL1.worst = worstErrorL1;
        results.ErrorL1.mean = meanErrorL1;
        results.ErrorL1.std = stdErrorL1;

        results.ErrorL2.min = minErrorL2;
        results.ErrorL2.worst = worstErrorL2;
        results.ErrorL2.mean = meanErrorL2;
        results.ErrorL2.std = stdErrorL2;

        results.RelErrorL2.min = minRelErrL2;
        results.RelErrorL2.worst = worstRelErrL2;
        results.RelErrorL2.mean = meanRelErrL2;
        results.RelErrorL2.std = stdRelErrL2;

        % %% Plots
        % fig1 = figure;
        % % axis([xA xB 0 Tf 0 max(max(Uexc))])
        % subplot(1,2,1);mesh(xval,tval,Uexc);
        % xlabel('$x$','Interpreter','latex')
        % ylabel('$t$','Interpreter','latex')
        % zlabel('Exact Solution','Interpreter','latex')
        % % axis([xA xB 0 Tf 0 max(max(Uapp))])
        % subplot(1,2,2);mesh(xval,tval,Uapp)
        % xlabel('$x$','Interpreter','latex')
        % ylabel('$t$','Interpreter','latex')
```

```
        % zlabel('Numerical Solution','Interpreter','latex')
        % print(fig1,'Fig1.eps','-depsc','-r300');
        % print(fig1,'Fig1.jpg','-djpeg','-r300');
        %
        % fig2=figure;
        % subplot(1,2,1);plot(xval,Uexc(1,:),'k-',xval,Uapp(1,:),'r-.','LineWidth',1.5)
        % xlabel('$x$','FontSize',12,'Interpreter','latex')
        % ylabel('Numerical solution at $t=0$','FontSize',12,'Interpreter','latex')
        % legend('Exact Solution','Numerical solution','Location','northwest')
        % subplot(1,2,2);semilogy(BestCosts(1:200), 'linewidth',2)
        % xlabel('Iterations','FontSize',12,'Interpreter','latex')
        % ylabel('Best Cost','FontSize',12,'Interpreter','latex')
        % grid on
        % print(fig2,'Fig2.eps','-depsc','-r300');
        % print(fig2,'Fig2.jpg','-djpeg','-r300');

        save(['Results\GradientDescent_' num2str(Nx) '_'  num2str(Nt)
'.mat'],'sol','BestCosts','p_opt','Uexc','Uapp','results');
    end
end
```