Erciyes Üniversitesi
Fen Bilimleri Enstitüsü Dergisi
Cilt 41, Sayı 1, 2025

Erciyes University
Journal of Institue Of Science and Technology
Volume 41, Issue 1, 2025

Doğrusal Olmayan Sistem Durum Tahmini İçin Yapay Arı Kolonisi Algoritması Tabanlı Yeni Bir Parçacık Filtresi

Fatma Selcen TURGUN*[1], Hasan ZORLU [2]

*[1] Yozgat Bozok Üniversitesi Mühendislik Mimarlık Fakültesi Bilgisayar Mühendisliği, YOZGAT
[2] Erciyes Üniversitesi Mühendislik Fakültesi Elektrik Elektronik Mühendisliği, KAYSERİ

(Alınış / Received: 29.01.2025, Kabul / Accepted: 14.04.2025, Online Yayınlanma / Published Online: 30.04.2025)

Anahtar Kelimeler
Durum tahmini,
Parçacık filtresi,
Yapay arı kolonisi,
Parçacık sürü optimizasyon algoritması

Öz: Sıralı Monte Carlo yöntemi olarak da bilinen Parçacık Filtresinin (PF) temel fikri, durumun minimum varyans tahminini elde etmek için sistemin olasılık yoğunluk fonksiyonunu bazı ayrık rastgele örnekleme noktaları ile yaklaştırmaktır. Olası sistem durumlarını temsil etmek için bir dizi parçacıklar, ağırlıkları ve gözlemleri dâhil ederek tahmini sürekli olarak günceller. Bununla birlikte, çoğu parçacığın ağırlığı birkaç yinelemeden sonra ihmal edilebilir hale geldiğinden parçacık fakirleşmesi ve parçacık çeşitliliğinin azalması sorunları ortaya çıkar. Bu makale, doğrusal olmayan ve Gauss olmayan sistemler için durum tahmininde Parçacık Filtresi (PF) algoritmasının performansını artırmak için Yapay Arı Kolonisi (YAK) algoritmasının standart PF'ye dâhil edilmesini tanıtmaktadır. PF'nin sorunlarını çözmek için YAK algoritması, PF'nin yeniden örnekleme aşamasından sonra algoritmaya dâhil edilir ve parçacıkların istenen olasılık dağılımına daha etkin bir şekilde yaklaşmalarını sağlar. Önerilen yaklaşım, YAK algoritmasının güçlü yönlerinden yararlanarak en yüksek ağırlıklara sahip parçacıkları geliştirerek, sistemin gerçek durumunu daha iyi temsil etmeyi ve önemli parçacıkların sayısını artırmayı amaçlamaktadır. Önerilen yöntem, geleneksel PF ve Parçacık Sürü Optimizasyon (PSO) algoritmasıyla optimize edilmiş PF ile karşılaştırılmıştır. YAK tabanlı algoritmanın etkinliğini göstermek için bir bilgisayar simülasyonu gerçekleştirilmiştir. Simülasyon sonuçları, önerilen YAK tabanlı yeni algoritmanın tahmin doğruluğunu standart Parçacık Filtresine kıyasla önemli ölçüde iyileştirdiğini göstermektedir.

A New Particle Filter Based On Artificial Bee Colony Algorithm For Nonlinear System State Estimation

Keywords
State estimation,
Particle filter,
Artificial bee colony,
Particle swarm optimization algorithm

Abstract: The basic idea of the Particle Filter (PF), also known as the sequential Monte Carlo method, is to approximate the probability density function of the system with some discrete random sampling points to obtain a minimum variance estimate of the state. A set of particles to represent possible system states continuously updates the estimate by incorporating weights and observations. However, problems of sample impoverishment and reduced particle diversity arise as the weight of most particles becomes negligible after a few iterations. This paper introduces the incorporation of the Artificial Bee Colony (ABC) algorithm into the standard PF to improve the performance of the Particle Filter (PF) algorithm in state estimation for nonlinear and non-Gaussian systems. To solve the problems of the PF, the ABC algorithm is incorporated into the PF algorithm after the resampling stage, allowing the particles to more efficiently converge to the desired probability distribution. The proposed approach aims to better represent the true state of the system and increase the number of important particles by developing particles with the highest weights by utilising the strengths of the ABC algorithm. The proposed method is compared with conventional PF and PF optimised by Particle Swarm Optimisation (PSO) algorithm. A computer simulation is carried out to demonstrate the effectiveness of the ABC-based algorithm. The simulation results show that the proposed new ABC-based algorithm significantly improves the prediction accuracy compared to the standard Particle Filter.

*İlgili Yazar, email: f.selcen.bozkurt@yobu.edu.tr

1. Giriş

Doğrusal olmayan sistem, girdi ve çıktı arasındaki ilişkinin basit bir doğrusal denklemle ifade edilemediği sistemlere verilen genel bir isimdir. Doğrusal bir denklemle tam olarak ifade edilemez, daha karmaşık ilişkiler söz konusudur. Süperpozisyon ilkesi geçerli değildir. Birden fazla girdinin etkisi, girdilerin toplamının etkisiyle aynı olmayabilir. Homojenlik ilkesi de geçerli değildir. Girdiyi bir sabit ile çarparsak, çıktı aynı sabit ile çarpılmayabilir. Doğrusal olmayan sistemler, gerçek dünyadaki birçok sistemin daha doğru bir modellemesini sağlar. Bu nedenle mühendislik, fizik, biyoloji, ekonomi gibi birçok alanda önemli bir çalışma alanıdır. Mühendislikte elektronik devre tasarımı, kontrol sistemleri, robotik, havacılık gibi alanlarda sıklıkla karşılaşılır. Doğrusal olmayan sistem durum tahmini, gerçek dünyadaki birçok sistemin karmaşık davranışlarını daha iyi anlamak ve kontrol etmek için kullanılan önemli bir tekniktir. Bu teknik, bir sistemin iç durumunu, direkt olarak ölçülemeyen değişkenlerini, mevcut ölçümler ve sistem modeli kullanılarak tahmin etmeyi amaçlar. Birçok mühendislik uygulamasında, dinamik bir sistemin durumlarının tahmin edilmesi gerekir. Bir durum tahmin problemi, dinamik bir sistemin matematiksel modeli verildiğinde, gürültülü bir ölçüm kullanarak zamanla değişen durumların tahmin edilmesini istemektedir [1]. Tahmin problemlerinde odak genellikle gözlemlerden gerçek sinyalin çıkarılması olarak adlandırılan filtrelemedir. Filtreler genellikle çalışırken belirli bir amaç fonksiyonunu en aza indirir. Bu tür filtrelere Optimal filtreler denir [2]. Optimal filtreler, özyinelemeli ve toplu filtreler olarak kategorize edilir [3][4]. Toplu filtre, örneğin en küçük kareler filtresi, bilinmeyen durumları tahmin etmek için tüm ölçüm geçmişini kullanır. Özyinelemeli filtreler ise ölçümleri sırayla alma ve işleme yeteneğine sahiptir. Özyinelemeli filtreler ayrıca doğrusal ve doğrusal olmayan filtreler olarak kategorize edilebilir [3][4]. Yinelemeli filtreler temelde iki aşamadan oluşur, bunlar; tahmin ve güncellemedir [4]. Tahmin, geçerli adımın ilk tahminini üretmek için önceki zaman adımının tahmini durumlarını kullanır. Bu aşama aynı zamanda önsel durum tahmini olarak da bilinir çünkü mevcut zaman adımında elde edilen gözlemleri kullanmaz. Güncelleme aşamasında, önsel durum tahmini, durum tahminini iyileştirmek için mevcut gözlemle birleştirilir. Bu iyileştirilmiş tahmin aynı zamanda sonsal durum tahmini olarak da adlandırılır. Dinamik durumlar, alınan ölçüme dayalı olarak elde edilen sonsal olasılık yoğunluk fonksiyonu (PDF) kullanılarak tahmin edilebilir. Literatürde, Kalman Filtresi (KF) [5], Genişletilmiş Kalman Filtresi (EKF) [6], Kokusuz Kalman Filtresi (UKF) [7], Parçacık Filtresi (PF) [8] olmak üzere birçok özyinelemeli filtre bulunur.

Mevcut filtreler arasında, Parçacık Filtreleri (PF'ler), sistem modelini işleme sürecinde neredeyse herhangi bir varsayıma ihtiyaç duymayan gelişigüzel olasılık yoğunluklarına yaklaşabilir. Bu nedenle, PF'ler, salgın tahmini, mobil robot lokalizasyonu ve haritalama, nesne izleme, jeofizik sistemler, yer bilimleri ve uzaktan algılama, ulaşım sistemleri ve orman yangını yayılma simülasyonu gibi çeşitli alanlarda yaygın olarak kullanılmaktadır [9]. Olasılık yoğunluklarının noktasal parçacık temsiline dayalı olarak çalışır [10]. PF'ler de ki her parçacık, bir sistemin olası bir durumunu temsil eder. Bu parçacıklar sistemin durumuna ilişkin gözlem ölçümlerini kabul ederek, sistemin gerçek durumuna sürekli olarak yaklaşmak için tahmine dayalı bilgilerle yeni parçacıklar üretir. PF'ler, çeşitli karmaşık problemlere, özellikle doğrusal olmayan ve Gauss olmayan problemlere uygulanır [11]. Örnekleme, ağırlık hesaplama ve yeniden örnekleme olmak üzere PF'lerin yinelemeli sürecinde üç ana adım vardır.

Örnekleme aşamasında, sistem durumunu temsil eden önceki parçacıklar kullanılarak yeni bir parçacık seti oluşturulur. Daha sonra tüm parçacıkların ağırlıkları hesaplanır ve normalizasyon yapılır. Önsel durum tahmininde mevcut zaman adımında elde edilen ölçümler kullanılmaz [12]. Bu PF'nin parçacık fakirleşmesi sorunu olarak bilinir. Bu soruna çözüm olarak sunulan yeniden örnekleme aşamasında, yavru parçacıklar normalizasyon yapılmış ağırlıklara göre elde edilir ve bu yeniden örneklenen parçacıklar bir sonraki iterasyonun girdisi olarak kullanılacaktır. Ancak standart yeniden örnekleme prosedürü, önemli parçacıkları kopyalar ve uygunluklarına göre önemsiz olanları atar ama bu da parçacık yoksullaşmasına sebep olur [13].

Son zamanlarda, biyolojik evrim ilkelerine dayanan yeni bir teknik olarak, evrimsel hesaplama, çeşitli doğrusal olmayan ve çok amaçlı optimizasyon problemlerine giderek daha yaygın bir şekilde uygulanmaktadır. Evrimsel hesaplama ve parçacık filtreleri belirli benzer özelliklere sahip olduğundan, geçmiş göreceli araştırmalarda parçacık filtrelerinin parçacık yoksullaşma problemine çözüm geliştirmek için bazı evrimsel algoritmalar tanıtılmıştır. [14], [15] ve [16]'da yeniden örneklemeden sonra örneklerin çeşitliliğini artırmak için Genetik Algoritma (GA) ve PF birleştirilmiştir. Örnek büyüklüğünü azaltmak ve verimliliği artırmak için parçacık filtresine yerel bir arama yöntemi eklenmiştir [17]. [18] ve [19]'de yeniden örnekleme sürecini iyileştirmek için Karınca Kolonisi Optimizasyonu (ACO) kullanılmıştır. Parçacık Sürü Optimizasyonu (PSO), parçacık yoksullaştırma ve örnek boyutu bağımlılık problemlerini çözmek için PF'ye dâhil edilmiştir [20]. En yeni gözlemleri örnekleme sürecine dâhil ederek, parçacıklar en potansiyel bölgelere doğru hareket ettirilir ve

parçacıkların dağılımı optimize edilebilir sonucuna ulaşılmıştır. [21]'de PSO algoritması ve GA algoritması kombine edilip PF'ye dâhil edilmiştir. Bu algoritma, parçacık sürüsü optimizasyonunun hızlı yakınsama hızını genetik algoritmanın güçlü küresel arama yeteneğiyle birleştirmiştir.

Bu makalede, Yapay Arı Kolonisi (YAK) algoritması olarak bilinen meta-sezgisel yönteme dayalı olarak doğrusal olmayan sistemlerin durum tahmini için yeni bir Parçacık Filtresi önerilmiştir. Önerilen yaklaşımın ana fikrinde parçacıklar yeniden örnekleme adımından sonra tüm parçacık durumlarından küresel optimum çıktıyı daha da iyileştirmek için Yapay Arı Kolonisi algoritmasına dâhil edilir, böylelikle sonsal dağılıma daha da yaklaşılabilir. Yeniden örnekleme adımının getirdiği parçacık çeşitliliği azalması sorununa çözüm sunmak için tasarlanmış yeni bir değiştirilmiş PF önerilmiştir. Genel PF ve PSO algoritmasıyla optimize edilmiş PF ile karşılaştırıldığında, daha doğru tahmin sonuçları sağlamıştır. Tanımlanan doğrusal olmayan bir problem üzerinde önerilen YAK tabanlı PF ve PSO algoritmasıyla optimize edilmiş PF algoritması uygulanmıştır. Tahminin simülasyon sonuçları tatmin edicidir.

Makalenin geri kalanı şu şekilde düzenlenmiştir: Bölüm 2'de genel parçacık filtresi ile PSO ve YAK algoritması kısaca gözden geçirilmektedir. PSO ve YAK algoritması tabanlı PF tanıtılmaktadır. 3. bölümde bir durum tahmin problemi formüle edilmiştir ve yeni tahmin algoritması ile gerçekleştirilen deneysel sonuçlar verilmiştir. Sonuçlar 4. bölümde sunulmaktadır.

2. Materyal ve Metot

2.1. Standart parçacık filtresi

Parçacık Filtresi, Bayes filtre teorisinden türetilmiştir [22]. Bayes filtresinin ana fikri, bilinen sistem bilgilerini kullanarak sistem durumunun sonsal dağılımını oluşturmaktır. Sistemin durumu doğrudan ölçülemezken, Bayes teorisine göre sistemin durumunu tahmin etmek için sistemin gözlemini kullanmak gerekir [23]. Genel olarak, önceki dağılımı tahmin etmek için süreç modelini kullanır ve ardından sonsal dağılımı elde etmek için yeni gözlemleri dâhil ederek bu dağılımı günceller.

PF, yinelemeli Bayes filtrelemeyi gerçekleştirmek için sıralı Monte Carlo simülasyonunu kullanır. Parçacık filtresinin temel düşüncesi, sonsal dağılımı N ağırlıklı örnek kümesiyle temsil etmek ve tüm parçacıkların ağırlıklı toplamını kullanarak durum tahmini elde etmektir. Örnek kümesi boyutu sonsuz olma eğilimindeyse, Monte Carlo karakteri ve sonsal olasılık dağılımı eşdeğerdir [23].

PF, durum olasılık dağılımını temsil etmek için ağırlıklı parçacıklar kullanır. Doğrusal olmayan filtreleme problemini çözmek için tasarlanmış parametrik olmayan bir yaklaşım tekniğidir ve durum uzay modeliyle tanımlanabilen doğrusal olmayan herhangi bir sisteme uygulanabilir.
Bir sistemin durum uzay modeli şu şekilde tanımlanabilir:

$$x_k = f(x_{k-1}, u_{k-1}) \tag{1}$$

$$z_k = h(x_k, v_k) \tag{2}$$

Burada $f(.)$ ve $h(.)$ sırasıyla durum transfer ve gözlem ölçüm fonksiyonudur. x_k, k zamanındaki sistem durum değişkenidir, z_k gözlemlenen ölçüm değeridir, u_{k-1} süreç ve v_k gözlem ölçüm gürültüsüdür.

Standart Parçacık Filtresi algoritmasında ilk olarak önceki dağılımdan $x_k^i (i = 1,2, \dots, N)$ parçacık kümesi oluşturulur ve burada her parçacığın ağırlığı 1/N'dir. Yaklaşık sonsal dağılım ise şöyledir:

$$p(x_{0:k}|z_{1:k}) \approx \sum_{i=1}^{N} u_k^i \, \delta(x_{0:k} - x_{0:k}^i) \tag{3}$$

Burada $x_{0:k}^i = [x_0^i, x_1^i, \dots, x_k^i]$ $p(x_{0:k}|z_{1:k})$ tarafından üretilen rasgele bir örnektir. $z_{1:k} = [z_1, z_2, \dots z_k]$ ölçüm verisidir. u_k^i, x_k^i örneği için önem ağırlığıdır. $\delta()$ Dirac fonksiyonunu temsil eder. Ancak pratikte $p(x_{0:k}|z_{1:k})$ genellikle yanıltıcı olduğundan x_k^i için $q(x_{0:k}|z_{1:k})$'dan önem örneklemesi yapılır. u_k^i tahmini daha sonra şu şekilde hesaplanabilir:

$$u_k^i = \frac{p(x_{0:k}^i|z_{1:k})}{q(x_{0:k}^i|z_{1:k})} \tag{4}$$

Daha sonra ağırlık güncelleme denklemini şu şekilde belirlenebilir:

$$u_k^i = u_{k-1}^i \frac{p(z_k|x_k^i)\, p(x_k^i|x_{k-1}^i)}{q(x_k^i|x_{0:k-1}^i, z_{1:k})} \tag{5}$$

Burada $p(z_k|x_k^i)$ olasılık fonksiyonu ve $p(x_k^i|x_{k-1}^i)$ durum transfer dağılımıdır. Sistem Markov sürecini takip ediyorsa, ağırlık güncellemesi şu şekilde basitleştirilebilir:

$$u_k^i = u_{k-1}^i \frac{p(z_k|x_k^i)\, p(x_k^i|x_{k-1}^i)}{q(x_k^i|x_{k-1}^i, z_k)} \tag{6}$$

Pratik hesaplamalarda, yoğunluk fonksiyonu $q(x_k^i|x_{k-1}^i, z_k)$ sıklıkla önceki dağılım olan $p(x_k^i|x_{k-1}^i)$ ile değiştirilir. Her yineleme için önem ağırlık güncellemeleri aşağıdaki gibidir:

$$u_k^i = u_{k-1}^i p(z_k|x_k^i) \tag{7}$$

Ağırlıklar daha sonra şu şekilde normalizasyon yapılır:

$$u_k^i = \frac{u_k^i}{\sum_{i=1}^N u_k^i} \tag{8}$$

Dejenerasyondan kaçınmak için standart yaklaşım, düşük ağırlıklı parçacıkların yüksek ağırlıklı parçacıklarla tekrarlanmasını ortadan kaldırmaktır. Yeniden örnekleme adımı, yeniden örneklemeden sonra toplam parçacık sayısı değişmeden kalan ve u_k^i 1/N'ye eşit olan yeni bir parçacık kümesi elde etmek için u_k^i'ye göre örneği kopyalama işlemidir. Yeniden örneklemeden sonra, yeni bir parçacık seti $\{\hat{x}_k^i\}_{i=1}^N$ oluşturulur ve yeniden örnekleme eşiği, şu şekilde hesaplanan etkin örnek N_{eff} olarak tanımlanır:

$$N_{eff} \approx \frac{1}{\sum_{i=1}^N (u_k^i)^2} \tag{9}$$

Yeni hedef durum tahmini elde edilen yeni parçacıklara ve ağırlıklara dayalı olarak şu şekilde güncellenebilir:

$$\tilde{x}_k = \sum_{i=1}^N x_k^i\, u_k^i \tag{10}$$

Sonrasında algoritma önem örneklemesi adımına döner ve filtreleme bitene kadar filtrelemeye devam eder. Yukarıdaki adımlarda görüldüğü üzere, PF'nin yeniden örnekleme işleminin parçacıkların çeşitliliğini azalttığı ve hedef durum tahmininde büyük bir hataya yol açacağı gözlemlenir.

Algoritma 1'de, standart parçacık filtresinin sözde kodu gösterilmektedir.

Algoritma 1. Standart parçacık filtresi sözde kodu

$\left[\{x_k^j, u_k^j\}_{i=1}^N\right] = PF\left[\{x_{k-1}^j, u_{k-1}^j\}_{i=1}^N, z_k\right]$
For=1:N
    parçacık seti $x_k^i \sim p(x_k, x_{k-1}^i)$
    her bir parçacığın ağırlığı $u_k^i = p(z_k|x_k^i)$
End For
toplam ağırlıkları hesapla: $t = topla\left(\{u_k^i\}_{i=1}^N\right)$
For= 1:N
    Normalize: $u_k^i = t^{-1} u_k^i$
End For
Yeniden Örnekleme
$\left[\{x_k^j, u_k^j, \ldots\}_{i=1}^N\right] = Yeniden\ Örnekleme\left[\{x_k^j, u_k^i\}_{i=1}^N\right]$
Durum Tahmini

2.2. Parçacık filtresi sınırlamaları

Parçacık Filtresi, yaklaşık sonsal dağılımı hesaplamak için sıralı önem örneklemesi (SIS) yöntemini kullanır. Genel Parçacık filtresinin önem fonksiyonu optimalin altında olduğundan, bu yöntemde iki ciddi sorun vardır. İlk sorun, olasılık çok dar olduğunda veya olasılık önceki dağılımın sonunda olduğunda olasılık ve önceki dağılımın örtüşme bölgesi çok küçüktür ve bu da örnek fakirleşmesi sorununu ortaya çıkarır. Dolayısıyla, güncelleme adımından sonra, yalnızca birkaç parçacığın önemli önem ağırlıkları olacaktır. Bu nedenle, örnek seti yalnızca birkaç farklı parçacık içerir ve bazen birkaç yinelemeden sonra tek bir örneğe düşer. Sonuç olarak, önemli örnekler kaybolabilir. Genel Parçacık filtresinin diğer bir sorunu ise örnek boyutu bağımlılığıdır. Örnek seti boyutu küçükse, gerçek durum etrafında dağılmış parçacıklar olmayabilir. Bu nedenle, birkaç yinelemeden sonra parçacıkların gerçek duruma yakınsaması çok zordur. Genel Parçacık filtresi için, bu sorunu çözmenin bir yolu örnek set boyutunu artırmaktır. Ancak, başarılı bir tahmin sağlamak için tüm durum uzayını kapsayan yeterince büyük örnek seti sağlarsak, parçacık filtresinin hesaplama verimliliği düşecektir ve hatta sistemin gerçek zamanlı gereksinimlerini karşılayamama durumu meydana gelecektir [24].

2.3. Parçacık sürü optimizasyon algoritması

Parçacık Sürüsü Optimizasyonu (PSO), 1995 yılında Kennedy ve Eberhart tarafından geliştirilen, stokastik bir küresel optimizasyon tekniğidir [25]. Bu algoritma, kuş veya balık sürülerinin yiyecek arama davranışlarını model alarak, popülasyon tabanlı bir yaklaşım sergiler. PSO, evrimsel hesaplama ve sürü zekası özelliklerini bir araya getirir ve özellikle doğrusal olmayan, türevlenemeyen, çok modlu optimizasyon problemlerini çözmede sağlam ve hızlı bir yöntem olarak bilinir [26]. Bu özellikleri sayesinde PSO, optimizasyon problemlerinde geniş bir uygulama alanı bulmuş ve çeşitli mühendislik ve bilim dallarında başarıyla kullanılmıştır.

Algoritmanın temelinde, bireyler yani parçacıklar arasındaki iş birliği ve rekabet yoluyla karmaşık bir çözüm alanında arama yapmak ve optimum sonuca ulaşmak yer alır. PSO algoritması, başlangıçta uygulanabilir çözüm alanında bir popülasyon oluşturur. Parçacıkların ağırlığı ve hacmi bulunmaz. N boyutlu bir uzayda popülasyondaki her bir birey çözüm alanında bir hız vektörü ile hareket eder. Parçacıkların hareketi hem kendi bireysel deneyimlerinden hem de tüm parçacıkların genel deneyiminden etkilenerek güncellenir. Her nesilde, parçacıklar iki kritik değeri takip eder: biri, bireysel parçacığın şimdiye kadar bulduğu en iyi çözüm, diğeri ise tüm popülasyonun bulduğu en iyi çözüm. Bu süreç, parçacıkların uygunluk değerleri doğrultusunda yönlendirilir ve popülasyondaki diğer parçacıklarla yapılan iş birliği ve rekabet sayesinde optimizasyon arayışı sürdürülür.
PSO algoritmasının özü, parçacıkların bir sonraki konumlarını hem bireysel hem de küresel en iyi çözümlerin bilgilerini kullanarak belirlemeleridir. Bu yapı, algoritmayı çok çeşitli sürekli fonksiyonların optimizasyonunda etkili kılar [27].

m parçacıktan oluşan rastgele bir parçacık sürüsü için, i. parçacığın n boyutlu uzaydaki konumları ve hızları sırasıyla $X_i = (x_{i1}, x_{i2}, \dots, x_{in})$ ve $V_i = (v_{i1}, v_{i2}, \dots, v_{in})$ 'dir. İterasyonların her adımında, parçacıklar konumlarını ve hızlarını iki en iyi değere göre günceller. Bunlardan biri $Pbest_i = (p_{i1}, p_{i2}, \dots, p_{in})$, i. parçacığın mevcut adıma kadar olan bireysel önceki en iyi değeridir. Bir diğeri ise $Gbest_i = (g_1, g_2, \dots, g_n)$, popülasyondaki tüm parçacıkların en iyi değeridir. En iyi iki değer bulunduktan sonra, her parçacık konumunu ve hızını aşağıdaki denklemlere göre günceller.

$$v_{id}^{k+1} = w * v_{id}^{k} + c_1 * r_1 * \left(x_{pbest} - x_{id}^{k}\right) + c_2 * r_2 * \left(x_{gbest} - x_{id}^{k}\right) \quad (11)$$

$$x_{id}^{k+1} = x_{id}^{k} + v_{id}^{k+1} \quad (12)$$

Burada, v_{id}^{k} k. iterasyondaki i. parçacığın v_i hızının d. bileşeni, i =1,2, ... ,N. w eylemsizlik katsayısı veya diğer bir ifadeyle atalet ağırlığı, $c_1$ve c_2 ivme katsayıları pozitif sabitlerdir. r_1 ve r_2 U(0,1) dağılımına uyan tekdüze olasılık dağılımına göre üretilen rasgele sayılardır. Genel olarak PSO büyük atalet ağırlığı kullanarak güçlü küresel arama yeteneğine sahiptir ve bunun tersi de geçerlidir.

2.3.1. Parçacık sürü optimizasyon algoritmasına dayalı parçacık filtresi

Yukarıdaki PSO ve Parçacık Filtresinin tanıtımından ikisinin birçok ortak yönü olduğunu görebiliriz. Öncelikle, PSO parçacıkların konumlarını ve hızlarını arama alanında yinelemeli olarak güncelleyerek en iyi değeri elde ederken, PF parçacıkların konumlarını ve ağırlıklarını yinelemeli olarak güncelleyerek arka olasılık dağılıma en iyi yaklaşımı elde eder. İkinci olarak, PSO'da en iyi uygunluk değerine sahip bir parçacık arama alanındaki en uygun noktayı belirtir. Benzer şekilde, PF'de en yüksek ağırlığa sahip bir parçacık en olası sistem durumu olarak

kabul edilir. Üçüncüsü, hem PSO hem de PF'de amaç parçacıkları en iyi konumlara doğru hareket ettirmektir. PSO parçacıkları, her parçacığın konumunu ve hızını her parçacığın önceki en iyi değerine ve tüm parçacıkların küresel en iyi değerine göre güncelleyerek en uygun noktaya doğru hareket ettirirken, PF her parçacığın önce durum modelini kullanarak konumunu günceller ve ardından ölçüm modeli aracılığıyla her parçacık kendi ağırlık değerini günceller, parçacığı gerçek arka olasılık dağılıma doğru hareket ettirir. Bu nedenle, PSO algoritması yukarıdaki benzerliklere dayanarak standart parçacık filtresinin performansını iyileştirebilir.

Bu bölümde önerilen PSO-PF algoritması tanıtılmıştır. Parçacık filtrelemenin eksikliklerini gidermek için, literatürde [28], yeniden örnekleme sürecini iyileştirmek, parçacık bozulmasını azaltmak ve durum tahmininin doğruluğunu ve algoritmanın yakınsamasını iyileştirmek için parçacık filtrelemeye parçacık sürüsü optimizasyonu eklenmiştir [29]. Parçacık sürüsü optimizasyon yöntemi, parçacık filtresinin yeniden örnekleme aşamasından sonra tüm parçacık durumlarından küresel optimum çıktıyı daha da rafine etmek için kullanılır. Algoritma 2, öncelikle tüm parçacık durumlarını başlatan ve ardından tahmin, ölçüm ve yeniden örneklemeyi gerçekleştirmek için parçacık filtresini kullanan PSO-PF algoritmasının sözde kodunu gösterir.

PSO-PF algoritmasının adımları aşağıdaki gibidir:

(1) Adım 1'de ölçüm değerini elde edilir. İlk olarak, bir uygunluk fonksiyonu tanımlanarak her parçacığın uygunluğu uygunluk fonksiyonu denklem (13) ile hesaplanır. Geleneksel parçacık filtresi, optimal olmayan bir önem fonksiyonu kullanır, bu nedenle parçacık örnekleme süreci optimal olmayan haldedir, parçacık filtresi örnekleme sürecini optimize etmek için, en son ölçülen değerler örnekleme sürecine duruma dahil edilir.

$$z_k(fitness) = exp\left[-\frac{1}{2R_k}\left(z_k - \widehat{z^i}(k|k\text{-}1)\right)^2\right] \tag{13}$$

Burada z_k en son ölçülen değer, R_k, gözlemlenen ölçüm gürültü varyansıdır; $\widehat{z^i}_{(k|k-1)}$ ise tahmin değeridir.

(2) Parçacık seti oluşturulur. Başlangıç parçacıkların belirtmek için önem yoğunluk fonksiyonundan N adet $\{x^i_{0:k}, w^i_k\}^N_{i-1}$ parçacık örneklenir ve parçacık sürüsü başlatılır. Her bir örneğin başlangıç ağırlıklarını $\{w^i_k = 1/N\}$ olarak tanımlanır, önem yoğunluk fonksiyonunun transferinin önsel olasılığı denklem (14) kullanılarak bulunur.

$$x^i_k \sim q\left(x^i_k | x^i_{k\text{-}1}, z_k\right) = p\left(x^i_k | x^i_{k\text{-}1}\right) \tag{14}$$

(3) Her parçacığın uygunluk değeri hesaplanır ve en uygun konumu belirlenir. En son ölçülen değerlere göre, mevcut parçacık ağırlıklarını denklem (15) ile hesaplanır.

$$w^i_k = w^i_{k\text{-}1}\, p\left(z_k | x^i_k\right) \tag{15}$$

(4) Parçacıklar yeniden örnekleme sürecine girer. Parçacık sürüsü optimizasyon yöntemi, en iyi çözümü elde etmeye yönelik bir algoritma olduğundan, parçacık filtresinin yeniden örneklemesinden sonra tüm parçacık durumlarından küresel en iyi çıktıyı daha da hassaslaştırmak için kullanılır.

(5) Tüm parçacık durumları parçacık sürüsü optimizasyon yöntemine kopyalanır. Her parçacığın hızını ve konumunu güncellemek için PSO algoritması ve sırasıyla denklemler (11) ve (12)'u kullanılır. Böylece parçacıklar gerçek duruma yakın hale getirilir. Parçacıklar gerçek duruma yakın konumlanmışsa, parçacık sürüsündeki her parçacığın uygunluğu çok yüksektir. Tersine, parçacık sürüsündeki her parçacığın bireysel optimum değerleri ve parçacık sürüsünün küresel optimum değeri çok düşükse, parçacıklar gerçek duruma yakın konumlanmamış demektir, bu sırada parçacık kümesi, PSO algoritmasını kullanarak her parçacığın hızını ve konumunu optimum değere göre günceller ve parçacıkların gerçek duruma hareket etmesini sağlar. PSO algoritmasının özü, tüm parçacıkları yüksek olasılık olasılığına sürmektir. En son parçacık ağırlıkları denklem (16) ile normalize edilir.

$$\widetilde{w}^i_k = \frac{w^i_k}{\sum_{i=1}^N w^i_k} \tag{16}$$

(6) Elde edilen yeni parçacıklara ve ağırlıklara dayalı olarak durum tahminleri hesaplanır.

$$\tilde{x}_k = \sum_{i=1}^{N} x_k^i \, u_k^i \tag{17}$$

Maksimum yineleme sayısına ulaşılıp ulaşılmadığı kontrol edilir. Koşullar sağlanırsa algoritma durur, sağlanmazsa yinelemeli olarak Adım 2'ye döner.

Algoritma 2'de PSO algoritması ile optimize edilmiş parçacık filtresinin sözde kodu gösterilmiştir.

Algoritma 2. PSO algoritması ile optimize edilmiş parçacık filtresinin sözde kodu

(1)$İlklendirme$
$For\ i = 1, \dots, N, ilk\ parçacıkları\ oluşturma, x_0^{n,(i)} \sim p(x_0^n)$
(2)$For\ k > 0$
(2.1)PF ölçüm $güncellemesi$
$\left[\{x_k^i\}_{i=1}^N\right] = PF\left[\{x_k^i\}_{i=1}^N, z_k\right]$
(2.2)$Parçacıkların\ ağırlıklarının\ hesaplanması$
(2.3)$Yeniden$ Örnekleme
(2.4)$Tüm\ parçacık\ durumlarının\ PSO\ yöntemine\ kopyalanması$
$Parçacık\ En\ İyisini\ Hesapla\ P', P_{pbest}[t]);$
$Ağırlıklı\ Yeleği\ Hesapla\ P_{pbest}[t], P_{gbest}[t]);$
$Hızları\ Hesapla\ (V[t]);$
$Parçacık\ güncellemesi\ P'$
$t = t + 1;$
(2.5)$Ağırlıkların\ normalize\ edilmesi$
$\tilde{w}_k^i = \frac{w_k^i}{\sum_{i=1}^N w_k^i}$
(2.6)$Durum\ Tahmini$
(2.6) $k = k + 1\ ve\ (2.1)'den\ itibaren\ yineleme\ gerçekleştirilir.$

2.4. Yapay arı kolonisi algoritması

Yapay Arı Kolonisi (YAK) algoritması, bal arılarının yiyecek arama davranışlarından ilham alan popülasyon tabanlı bir metasezgisel optimizasyon tekniğidir [30]. Derviş Karaboğa tarafından 2005 yılında önerilen bu algoritma, karmaşık sorunlara en uygun çözümleri bulmak için bir arı kolonisinin arama sürecini, işbirlikçi davranışını taklit eder [31].

YAK algoritmasında, yiyecek arayan arılar üç gruba ayrılır, bunlar görevli işçi, gözcü ve keşifçi arılardır. Kolonin ilk yarısı işçi yapay arılardan oluşur ve ikinci yarısı gözcü arıları içerir. Her yiyecek kaynağı için yalnızca bir işçi arı vardır. İşçi arılar belirli yiyecek kaynaklarını kullanır ve diğer arılarla bilgi paylaşır. Başka bir deyişle, kovanın yakınındaki yiyecek kaynaklarının sayısı işçi arıların sayısına karşılık gelir. Bir yiyecek kaynağının konumu, optimizasyon problemine olası bir çözümdür. Gözcü arılar, işçi arılar tarafından paylaşılan bilgilere dayanarak yiyecek kaynaklarını seçerler ve en umut verici alanlara odaklanırlar. Keşifçi arılar ise potansiyel çözümler için yeni alanları keşfederek arama sürecinde çeşitliliği garanti ederler [30].

İlgili problemin kalitesi, bir yiyecek kaynağında bulunan nektar miktarıyla ilgilidir. İşçi veya gözcü arıların sayısı, koloni büyüklüğüne eşittir. YAK algoritmasının üç temel kontrol parametresini vardır ve bunlar sırayla; koloni büyüklüğü, sınır değeri ve maksimum döngü sayısının (MCN) değeridir [30].

YAK algoritması, işçi, gözcü ve keşifçi arı aşamasını iteratif olarak tekrar eder. Bu süreç, maksimum iterasyon sayısına ulaşmak veya tatmin edici bir çözüm bulmak gibi bir sonlandırma kriteri karşılanana kadar devam eder. YAK algoritması, fonksiyon optimizasyonu, sınıflandırma sorunları, parametre tahmini ve özellik seçimi dahil olmak üzere çeşitli optimizasyon problemlerine başarıyla uygulanmıştır [32]. Sadeliği, çok yönlülüğü ve çok çeşitli optimizasyon problemlerini çözmedeki etkinliği, onu sürü zekası tabanlı optimizasyon algoritmalarında popüler bir seçim haline getirmiştir [33].

YAK algoritması ana adımları şu şekilde açıklanabilir [33][34]:

YAK algoritmasında süreç, ilk yiyecek kaynaklarının rastgele belirlenmesiyle başlar. Rastgele pozisyon üretim süreci, (18) denklemini kullanarak her parametrenin alt ve üst sınırları arasında bir değer üreterek gerçekleştirilir.

$$x_{ij} = x_j^{min} + rand(0,1) * (x_j^{max} - x_j^{min}) \quad (18)$$

Burada, $i = 1 \ldots SN$ ve $j = 1 \ldots PN$ 'dir. Burada SN, yiyecek kaynaklarının sayısı ve PN ise optimize edilecek parametre sayısıdır. x_j^{min}, j parametresinin alt sınırını ve x_j^{max}, j parametresinin üst sınırını tanımlar.

İşçi arılar, (19) denkleminde verilen eşitliği kullanarak yiyecek kaynağının civarında yeni bir komşu yiyecek kaynağı belirler ve kalitesini değerlendirir. Burada, j, [1, D] aralığında rastgele üretilen bir tam sayıdır. γ, [–1, 1] aralığında alınan rastgele bir değerdir. (19) denkleminde, v_i kaynağı, x_i'nin yalnızca bir parametresini değiştirerek bulunur.

$$v_{ij} = x_{ij} + \gamma * (x_{ij} - x_{kj}) \quad k \in \{1, \ldots, SN\} \; j = 1, \ldots, D \quad (19)$$

Eğer v_i, x_i'den iyiyse, işçi arı o adresi hafızasından siler. Aksi takdirde, eski adrese gitmeye devam eder. YAK algoritmasında gözcü arılar bu bilgiler ışığında nektar miktarını hesaplar ve bu ilgili olasılıklara ek olarak farklı yenilebilir kaynakları seçer [33]. Temel YAK algoritmasında Bir rulet çarkı seçimi (RWS) yöntemi kullanılarak kalite değerine dayalı bir seçim süreci gerçekleştirilir [35]. Rulet çarkı yöntemine göre, kaynakların seçim olasılığı denklem (20) ile hesaplanır. Burada, uygunluk i, i'inci kaynağın kalitesini gösterir ve SN, işçi arı sayısını temsil eder. Tüm işçi arıların arama süreçleri tamamlandıktan sonra gözcü arıların birbirleri ile yaptıkları titreşim hareketleri ile yiyecek kaynakları ile ilgili bilgiler aktarılır.

$$P_i = \frac{fitness_i}{\sum_{i=0}^{SN} fitness_j} \quad (20)$$

Gözcü arılar bir yiyecek kaynağı seçtiklerinde, her arı yiyecek kaynağının yakınında denklem (20) ile seçilen yeni bir çözüm bulur. Yeni çözüme ve eldeki çözümü iyileştirmek için seçilen yiyecek kaynağına açgözlü bir seçim uygulanır. Konum, sınırlı sayıda yinelemeyle daha fazla geliştirilemediğinde, yiyecek kaynağı atılır. Keşifçi arı denklem (18)'i yeni bir yiyecek kaynağı belirlemek için kullanır. Gereksinimler karşılanırsa, YAK algoritması sona erer. Aksi takdirde, ilk adıma geri dönülür.

Parçacık filtresini iyileştirmek için YAK algoritmaların kullanılması, YAK algoritmanın parçacıkların kullanım verimliliğini artırabilen, sonsal olasılık dağılımına yaklaşmak için gereken parçacıkları daha az yapan ve yeniden örneklemenin getirdiği sorunları önleyen benzersiz optimizasyon yeteneğine sahip olmasından kaynaklanmaktadır. Hesaplamayı belirli bir ölçüde azaltır, algoritmanın gerçek zamanını etkili bir şekilde iyileştirebilir. Ayrıca, YAK algoritması parçacık çeşitliliğini etkili bir şekilde artırabildiğinden, parçacık bozulmasını çözebilir böylece algoritmanın doğruluğunu artırabilir, filtre sapması olgusunu etkili bir şekilde önleyebilir ve durum tahmin doğruluğunu artırabilir.

2.4.1. Yapay arı kolonisi algoritmasına dayalı parçacık filtresi

Bu makalede Parçacık Filtresi algoritması parçacık bozulmasını iyileştirmek, parçacıkların çeşitliliğini artırmak ve algoritmanın yerel optimuma düşmesini önlemek için yeni bir yöntem olarak YAK algoritması ile optimize edilmiştir. Parçacık filtresinin sınırlamalarının üstesinden gelmek için, YAK algoritmasının örnekleme sürecini optimize etmesi düşünülmüştür. PF ve YAK algoritmalarını bir araya getirmek ve daha verimli bir Parçacık filtresi oluşturmak mümkündür [29].

İlk olarak, bir uygunluk fonksiyonu denklem (13) ile en son gözlem dikkate alınır.

YAK algoritması, tüm parçacıkları en iyi uygunluk değerine sahip olana doğru hareket ettirir. Tüm parçacıklar yüksek olabilirlik bölgesi etrafında dağılmışsa, her parçacığın uygunluk değeri yüksektir.

YAK-PF 'nin adımları aşağıdaki gibidir:[29]
(1) Parçacık seti oluşturulur. Denklem (14) ile başlangıç parçacıkları olarak önem yoğunluğu fonksiyonundan N adet $\{x_0^i, i = 1, \ldots, N\}$ parçacık örneklenir.

(2) Her parçacığın uygunluk değeri hesaplanır ve en uygun konumu belirlenir.

(3) Parçacıkların ağırlıkları denklem (15) ile hesaplanır.

(4) Yeniden örnekleme.

(5) Parçacık seti YAK algoritmasının popülasyonunu oluşturacak şekilde tanımlanır. YAK algoritması ana döngüsü parametreleri belirlenir. İşçi arılar için hızlanma katsayısı tanımlanır, yeni yiyecek pozisyonu belirlenir ve sınırlar uygulanır. Değerlendirme ve karşılaştırma gerçekleştirilir. Uygunluk değerleri ve seçim olasılıkları hesaplanır. Gözcü arılar için rulet çarkı seçim yöntemi tanımlanır. Hızlanma katsayısı tanımlanır ve yeni yiyecek pozisyonu belirlenir. Sınırlar uygulanır, değerlendirme ve karşılaştırma gerçekleştirilir. Bulunan en iyi çözüm güncellenir ve şimdiye kadar bulunan en iyi hafıza da tutulur [29].

(6) Parçacık ağırlıkları denklem (16) ile normalize edilir.

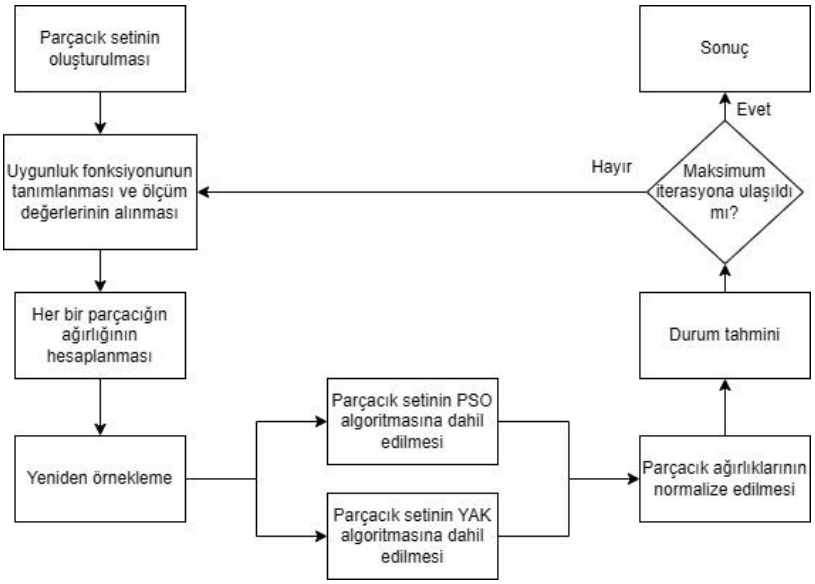
(7) Durum tahmini gerçekleştirilir. Maksimum yineleme sayısına ulaşılıp ulaşılmadığı kontrol edilir. Koşullar sağlanırsa algoritma durur, sağlanmazsa yinelemeli olarak Adım 2'ye döner.

Algoritma 3'de YAK algoritması ile optimize edilmiş parçacık filtresinin sözde kodu gösterilmiştir.

Algoritma 3. YAK algoritması ile optimize edilmiş parçacık filtresinin sözde kodu

(1)İ$lklendirme$
$For\ i = 1, \dots, N, ilk\ parçacıkları\ oluşturma, x_0^{n,(i)} \sim p(x_0^n)$
$(2) For\ k > 0$
$(2.1) PF$ ölçüm $güncellemesi$
$\left[\{x_k^i\}_{i=1}^N\right] = PF\left[\{x_k^i\}_{i=1}^N, z_k\right]$
$(2.2) Parçacıkların\ ağırlıklarının\ hesaplanması$
$(2.3) Yeniden$ Örnekleme
$(2.4) Her\ bir\ parçacığın\ YAK\ algoritmasına\ dahil\ edilmesi$
$\left[\{x_k^{i*}\}_{i=1}^N\right] = YAK\left[\{x_k^i, w_k^i\}_{i=1}^N, z_k\right]$
$(2.5) Ağırlıkların\ normalize\ edilmesi$
$\widetilde{w}_k^i = \frac{w_k^i}{\sum_{i=1}^N w_k^i}$
$(2.5) Durum\ Tahmini$
$(2.6)\ k = k + 1\ ve\ (2.1)'den\ itibaren\ yineleme\ gerçekleştirilir$

YAK ve PSO algoritmalarıyla optimize edilmiş PF'nin akış diyagramı Şekil 1'de gösterilmiştir.



Şekil 1. PSO-YAK-PF akış diyagramı

3. Bulgular

Bu makalede bahsedilen algoritmaların etkili performansını test etmek için doğrusal olmayan bir sistem olarak tek değişkenli bir büyüme modeli seçilmiştir. Model tahmin yöntemlerinin performanslarını değerlendirmek için yaygın olarak kullanılır. Modelin simülasyon sonuçlarının elde edildiği bilgisayarın temel frekansı 2.6GHZ'dir; sabit disk kapasitesi 80G'dir. Simülasyon yazılımı MATLAB (R2015b)'dir. Modelin durum denklemi ve ölçüm denklemi şunlardır [36]:

Durum denklemi:

$$x(t) = 1 + sin(0.04\pi t) + 0.5x(t-1) + u(t-1) \tag{21}$$

Ölçüm denklemi:

$$z(t) = \begin{cases} 0.2x(t)^2 + v(t) & 1 \le t \le 30 \\ 0.5x(t) - 2 + v(t) & 30 < t \le T \end{cases} \tag{22}$$

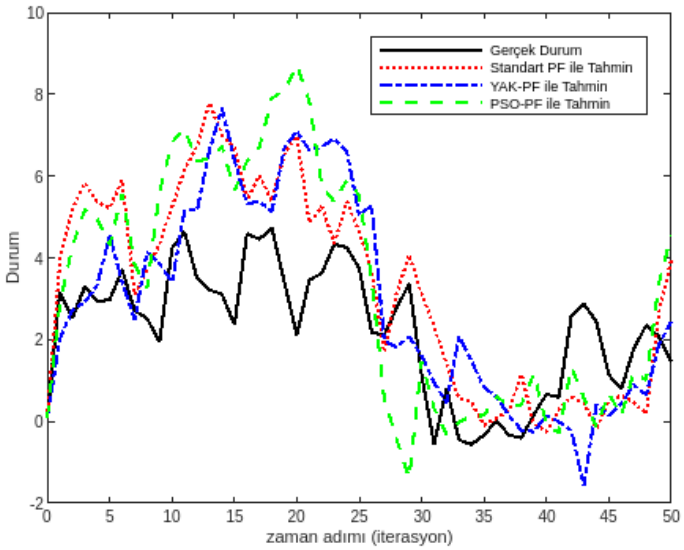
Formül de, $u(t-1)$ bir Gamma (3,2) rasgele değişkenidir ve $v(t) \sim N(0, 0.00001)$ ölçüm gürültüsüdür, bunların her ikisi de Gauss gürültüsüdür. T=50 iterasyon sayısıdır. Parçacık filtresi, [37] ve [38]'den referans alarak geliştirilen Parçacık Sürü Optimizasyon algoritmasıyla optimize edilmiş parçacık filtresi ve geliştirilen yeni yöntem kullanılarak modelin sistem gürültü varyansı R=1, parçacık sayısı N=10 ve ölçüm gürültü varyansının (Q) farklı değerleri için filtrelerin durum tahmini simüle edilmiştir. Bu çalışmada, gürültü varyansı olarak 1, 3 ve 6 değerleri seçilmiştir. Bu değerler, parçacık filtre algoritmalarının performansını düşük (1), orta (3) ve yüksek (6) gürültü seviyelerinde değerlendirmek amacıyla tercih edilmiştir. Literatürde, parçacık filtrelerinin performansının gürültü seviyelerine bağlı olarak değiştiği ve yüksek gürültü varyanslarında algoritmaların kararlılığının azaldığı belirtilmektedir. Örneğin, Zhang ve diğerleri, sensör gürültüsünün büyük olduğu durumlarda parçacık filtrelerinin performansının önemli ölçüde düştüğünü ve doğru durum tahminlerinin elde edilmesinin zorlaştığını göstermiştir [39]. Benzer şekilde, Liu, ölçüm gürültüsünün yüksek olduğu durumlarda geleneksel parçacık filtrelerinin performansının ciddi şekilde bozulduğunu belirtmektedir [40]. Bu nedenle, çalışmamızda gürültü varyansının 9 gibi daha yüksek bir değeri değerlendirmeye alınmamıştır; çünkü literatürde de belirtildiği gibi, bu seviyelerde parçacık filtrelerinin performansı önemli ölçüde düşmekte ve algoritmaların kararlılığı azalmaktadır. Farklı Q değerleri ile yapılan karşılaştırmalarda Kök Ortalama Karesel Hata (RMSE) ölçütleri, denklem (23), performans değerlendirmesi için kullanılmıştır. PSO algoritması için $c_1 = 1.5$ ve $c_2 = 2.0$ ve atalet ağırlığı 1 alınmıştır. Bu parametrelerden c_1 değeri parçacıkların kendi en iyi konumlarına yönelme eğilimini, c_2 ise topluluğun en iyi konumuna yönelme eğilimini ifade eder. Bu bağlamda, kullanılan değerler, parçacıkların hem kendi tecrübelerinden hem de topluluğun ortak bilgisinden faydalanarak daha etkili bir arama yapmalarını sağlamak amacıyla seçilmiştir. YAK algoritması parametreleri ise maksimum iterasyon sayısı (limit) 40 ve arı koloni boyutu (SN) 10 olarak belirlenmiştir. Karar değişkenlerinin sayısı 10 alınmıştır. Bu değer karar değişkenleri matris boyutunu ve de buna bağlı olarak ivme katsayısının değişkeni olarak etkilemektedir. Karar değişkenleri alt sınırı -20 ve karar değişkenleri üst sınırı +20 seçilmiştir. Bu çalışmada, belirlenen parametrelerin denendiklerinde performans/test süresi bakımından en verimli olduğu değerler dikkate alınmıştır.

Parçacık fakirleşmesi, ölçüm gürültüsünün büyüklüğü ve filtrede kullanılan parçacık sayısı ile ilişkilidir. Algoritmaların ölçüm gürültüsünün büyüklüğü faktörüne karşı sağlamlığını değerlendirmek için, bu çalışmada ölçüm gürültüsünün varyansı üç farklı değere ayarlanmıştır. Parçacık sayısı farklı ölçüm gürültü varyans değerlerinde sabit tutulmuştur. Bunun sebebi pratik uygulamalarda standart parçacık filtresindeki parçacık setinin boyutu çok büyük olamaz ve parçacık sayısı ne kadar büyükse o kadar fazla hesaplama karmaşası ortaya çıkar.

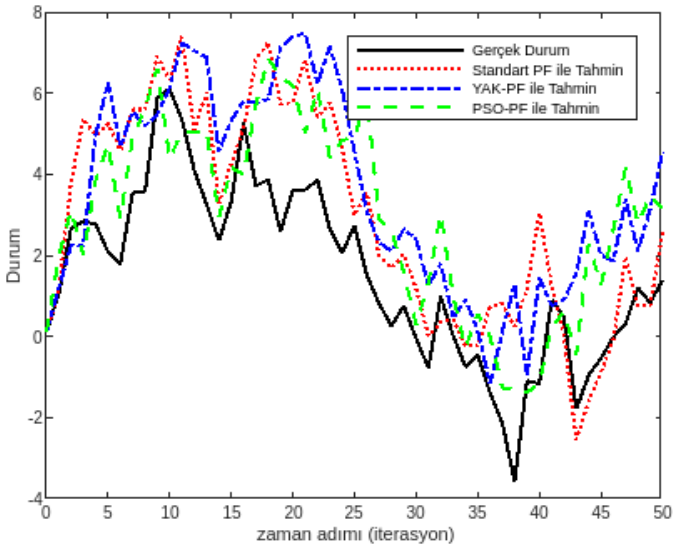
Kök Ortalama Karesel Hata (RMSE), çeşitli alanlardaki modellerin tahmine dayalı performansını değerlendirmek için yaygın olarak kullanılan Ortalama Kare Hata (MSE) metriğinin bir çeşididir [41]. RMSE, MSE'nin karekökü alınarak elde edilir, böylece değerlendirme kriterinin orijinal verilerle aynı birimde sunulması sağlanır. RMSE değeri, gözlemlenen ve tahmin edilen değerler arasındaki tipik sapmanın kolayca yorumlanabilir bir ölçüsünü sunar. Daha düşük bir RMSE, gözlenen ve tahmin edilen değerler arasında mükemmel bir uyumu temsil eden sıfır ile gelişmiş tahmin doğruluğunu ifade eder. Denklem (23)'de x_n beklenen değerleri, $\tilde{x}_n$ tahmin edilen değerleri ve N örnek sayısını göstermektedir.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{n}(x_n - \tilde{x}_n)^2} \tag{23}$$

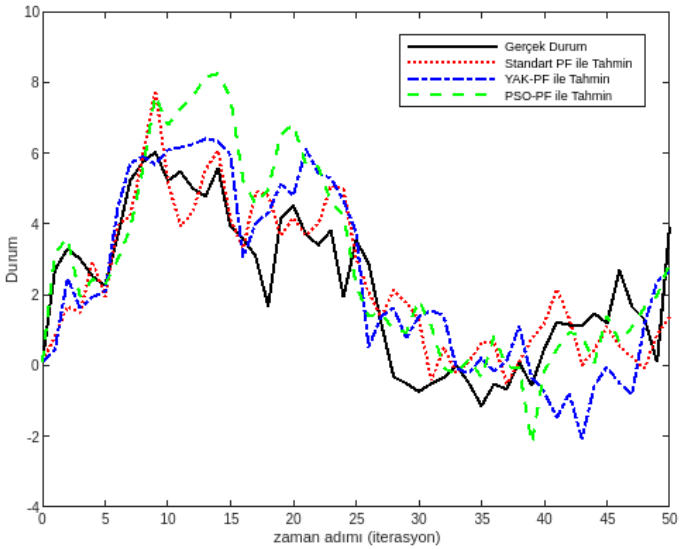
Aşağıdaki şekiller durum tahminlerinin tek bir çalışmasından üretilen farklı filtrelerin tahminlerini karşılaştırır. Şekil 2,3 ve 4'de Standart PF filtresinin, PSO algoritmasıyla optimize edilmiş PF filtresinin ve YAK tabanlı PF'nin, ölçüm gürültü varyansının (Q) sırasıyla 1, 3 ve 6 değerlerinde ve iterasyon sayısının 50 olduğu durumdaki deneysel simülasyonu görülmektedir.



Şekil 2. PF algoritmalarının Q=1 için deneysel simülasyonu



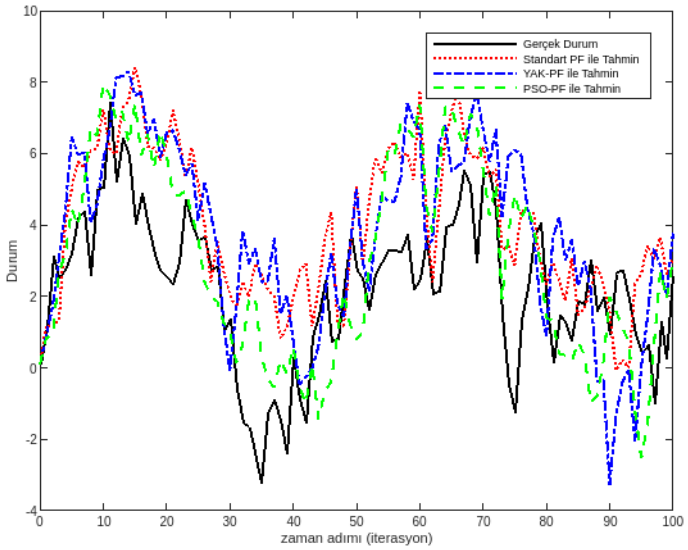
Şekil 3. PF algoritmalarının Q=3 için deneysel simülasyonu



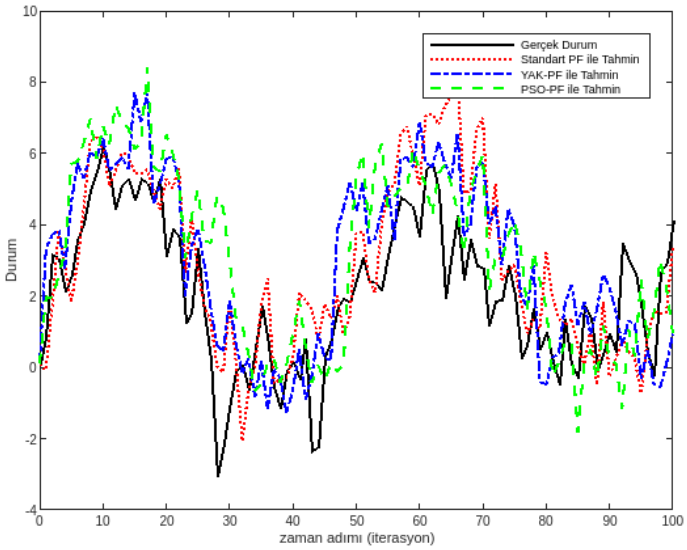
Şekil 4. PF algoritmalarının Q=6 için deneysel simülasyonu

Şekil 2, 3 ve 4'den, YAK algoritma tabanlı PF ile tahmin sonuçlarının standart parçacık filtresi ve PSO algoritması ile optimize edilmiş PF ile gerçekleştirilen tahmin sonuçlarına göre gerçek duruma çok daha yakın olduğunu görülmektedir. Tablo 1'de gerçekleştirilen simülasyonların RMSE değerleri verilmiştir. Buradan anlaşıldığı üzere YAK tabanlı PF'nin gerçek duruma daha yakındır. Bunun temel nedeni, YAK algoritmasının parçacık çeşitliliğini artırması ve daha iyi uygunluğa sahip parçacıkları tutulmasıdır. Birden fazla iterasyondan sonra, sistemin gerçek durumuna daha da yakınlaşır. Parçacık sayısının ve algoritmanın karmaşıklığının artmasıyla birlikte çalışma süresi de uzamaktadır.

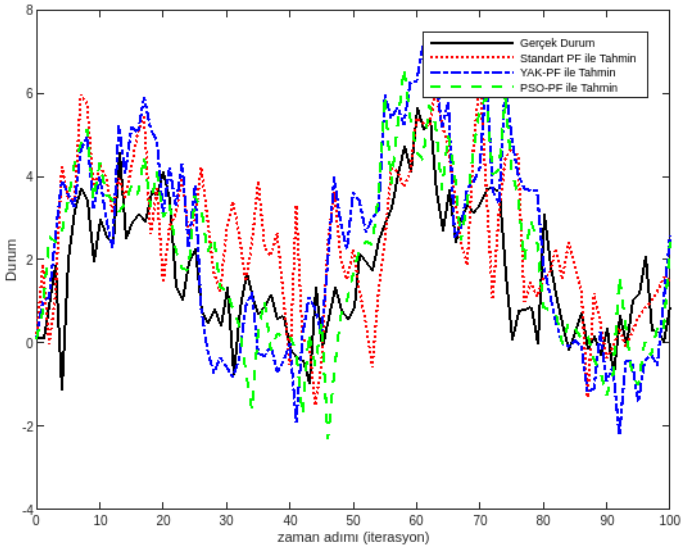
Şekil 4, 5 ve 6'da ise filtrelerin ölçüm gürültü varyansının (Q) sırasıyla 1, 3 ve 6 değerlerinde ve iterasyon sayısının 100 olduğu durumdaki deneysel simülasyonu görülmektedir.



Şekil 5. PF algoritmalarının Q=1 için deneysel simülasyonu 100



Şekil 5. PF ve YAK tabanlı PF'nin Q=3 için deneysel simülasyonu 100



Şekil 6. PF ve YAK tabanlı PF'nin Q=6 için deneysel simülasyonu 100

Şekil 4, 5 ve 6'ya bakıldığında, YAK algoritması destekli parçacık filtresinin (PF), standart parçacık filtresi ve PSO algoritmasıyla optimize edilmiş PF'ye göre gerçek duruma daha yakın tahminler sağladığı iterasyon sayısı artmasıyla da sonucun değişmediği gözlemlenmiştir. Tablo 1'de ki RMSE değerleri de YAK tabanlı PF'nin daha doğru sonuçlar ürettiğini kanıtlamaktadır.

Tablo 1. Farklı algoritmaların kök ortalama kare hatalarının (RMSE) karşılaştırması

	Algoritma		
Parametre	PF	PSO-PF	YAK-PF
T=50, Q=1	2.560659	1.377214	1.263523
T=50, Q=3	2.676719	1.654520	1.557265
T=50, Q=6	2.276009	1.494696	1.271453
T=100, Q=1	2.276854	1.806892	1.482641
T=100, Q=3	2.480433	1.697316	1.469528
T=100, Q=6	2.033237	1.601216	1.786575

4. Tartışma ve Sonuç

Bu makalede, örnek fakirleşmesi ve parçacık çeşitliliğinin azalması sorununu çözmek için akıllı optimizasyon algoritması ile doğrusal olmayan filtreleme için yeni bir parçacık filtresi algoritması geliştirilmiştir. Bu algoritma, yeniden örneklemeden sonra parçacık durumlarından en uygun durumu arayan parçacık setini optimize etmek için YAK algoritması aracılığıyla çok amaçlı optimizasyondan yararlanır. Geliştirilmiş algoritma, yukarıda ifade edilen RMSE değerlerine bakıldığında parçacıkların çeşitliliğini ve sistem durumu tahmininin doğruluğunu

artırdığı görülmektedir. YAK algoritmasının parçacık filtresine dâhil edilmesi bu makalenin en önemli katkısıdır. Simülasyon sonuçları, standart parçacık filtresi algoritması, PSO tabanlı parçacık filtresi ve YAK tabanlı parçacık filtresi algoritmasının karşılaştırıldığını göstermektedir. Deneyler, yeni parçacık filtresinin diğer parçacık filtrelerine göre daha etkili olduğunu göstermektedir. Ayrıca parçacık filtre algoritmalarının tek boyutlu doğrusal olmayan sistemlerdeki performansını farklı gürültü varyansları altında inceleyerek literatürdeki önemli bir boşluğu doldurmaktadır. Özellikle 1, 3 ve 6 gibi gerçekçi varyans seviyelerinde yapılan testler, yöntemin ölçüm belirsizliğine karşı dayanıklılığını ortaya koymuştur. Çalışma, bu yönüyle hem tek boyutlu sistemlerde parçacık filtre uygulamalarına dair kapsamlı bir değerlendirme sunmakta hem de filtre performansının gürültüye duyarlılığına ilişkin literatüre yeni bulgular kazandırmaktadır. Gelecek çalışmalarda, önerilen algoritma doğrusal olmayan Gauss olmayan radar hedef takibi gibi belirli sorunlara uygulanabilir.

Kaynakça

[1] Nobahari, H., Sharifi, A., 2012, A novel heuristic filter based on ant colony optimization for non-linear systems state estimation, Computational Intelligence and Intelligent Systems: 6th International Symposium, ISICA 2012, Wuhan, China, October 27-28.

[2] Brayson, J., Ho, Y., 1969, Applied Optimal Control, Blaisdell Publishing Company, Waltham.

[3] Siouris, J., 1995, An Engineering Approach to Optimal Control and Estimation Theory, Air Force Institute of Technology, New York.

[4] Ristic, B., Arulampalam, S., Gordon, N., 2004, Beyond the Kalman Filter, Artech House, London.

[5] Kalman, R. E., 1960, A new approach to linear filtering and prediction problems, Transaction of the ASME Journal Basic Engineering 82(Series D), 35–45.

[6] Jazwinski, A. H., 1970, Stochastic Processes and Filtering Theory, Academic Press, New York.

[7] Julier, S. J., Uhlmann, J. K., 1997, A new extension of the Kalman filter to nonlinear systems, In: AeroSense 11th International Symposium Aerospace Defense Sensing, Simulation and Controls, pp. 182–193.

[8] Carpenter, J., Clifford, P., Fernhead, P., 1997, An improved particle filter for non-linear problems, IEE Proceedings, Radar Sonar and Navigation 146, 2–7.

[9] Cao, T., et al., 2020, Differential Evolution Optimized a Second-Order Divided Difference Particle Filter, J. Electr. Comput. Eng. 2020, 1-9.

[10] Arulampalam, M. S., Maskell, S., Gordon, N., Clapp, T., 2002, A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking, IEEE Transactions on Signal Processing 50(2), 174–188.

[11] Hou, Y., et al., 2022, A multi-objective discrete particle swarm optimization method for particle routing in distributed particle filters, Knowledge-Based Systems 240: 108068.

[12] Zhang, Q. B., Wang, P., Chen, Z. H., 2019, An improved particle filter for mobile robot localization based on particle swarm optimization, Expert Syst. Appl. 135, 181-193.

[13] Meng, Z., Mei, J., Yu, C., 2022, Firefly optimized particle filter algorithm based on adaptive differential evolution, J. Phys. Conf. Ser. Vol. 2187. No. 1. IOP Publishing.

[14] Higuchi, T., 1997, Monte Carlo filter using the genetic algorithm operators, J. Stat. Comput. Simul. 59(1), 1–23.

[15] Park, S., Hwang, J. P., Kim, E., Kang, H., 2009, A new evolutionary particle filter for the prevention of sample impoverishment, IEEE Transection on Evolutionary Computation 13(4), 801–809.

[16] Turgun, F. S., Zorlu, H., 2023, Parçacık Filtresinin Optimizasyonu için Genetik Algoritma Tabanlı Yeni Bir Yaklaşım, Bozok Journal of Engineering and Architecture, 2(1), 24-33.

[17] Troma, P., Szepesvari, C., 2001, LS-N-IPS: An improvement of particle filters by means of local search, In: Proc. Non-Linear Control Systems (NOLCOS 2001), St. Petersburg, Russia.

[18] Zhong, J., Fung, Y., Dai, M., 2010, A biologically inspired improvement strategy for particle filter: Ant Colony Optimization Assisted Particle Filter, Int. J. Control Autom. Syst. 8(3), 519–526.

[19] Hao, Z., Zhang, X., Yu, P., Li, H., 2010, Video object tracing based on particle filter with ant colony optimization, IEEE, 232–236.

[20] Tong, G., Fang, Z., Xu, X., 2006, A particle swarm optimized particle filter for nonlinear system state estimation, In: IEEE Congress on Evolutionary Computation, pp. 438–442.

[21] Yang, J., et al., 2021, Particle filter algorithm optimized by genetic algorithm combined with particle swarm optimization, Procedia Comput. Sci. 187, 206-211.

[22] Gordon, N. J., Salmond, D. J., Smith, A. F. M., 1993, Novel approach to nonlinear/non-Gaussian Bayesian state estimation, IEEE Proceedings-F, vol. 140, no. 2, pp. 107-113.

[23] Zhang, T., et al., 2023, Remaining useful life prediction for rolling bearings with a novel entropy-based health indicator and improved particle filter algorithm, IEEE Access 11, 3062-3079.

[24] Kung, H., He, Z., 2018, Improved particle filter based on fruit fly optimization and its application in target tracking, Journal of Hunan University (Natural Sciences), 45(10), 130-138.

[25] Kennedy, J., Eberhart, R., 1995, Particle swarm optimization, In Proceedings of the IEEE International Conference on Neural Networks (Perth, Australia), IEEE Service Center, Piscataway, NJ, IV: pp. 1941-1948.

[26] Eberhart, R. C., Shi, Y., 2001, Tracking and optimizing dynamic systems with particle swarms, In Proceedings of the Congress on Evolutionary Computation, Seoul, Korea, 27–30 May, pp. 94–100.

[27] Park, J. B., 2010, An improved particle swarm optimization for nonconvex economic dispatch problems, IEEE Transactions on Power Systems, 25(1), 156-166.

[28] Tong, G., Fang, Z., Xu, X., 2006, A particle swarm optimized particle filter for nonlinear system state estimation, 2006 IEEE International Conference on Evolutionary Computation, IEEE.

[29] Bozkurt, F.S. 2023. Doğrusal olmayan parçacık filtresinin esnek hesaplama yöntemleri ile optimizasyonu. Erciyes Üniversitesi, Fen Bilimleri Enstitüsü, Yüksek Lisans Tezi, 74 s, Kayseri.

[30] Karaboğa, D., Kaya, E., 2017, Training ANFIS by using the artificial bee colony algorithm, Turkish Journal of Electrical Engineering and Computer Sciences, 25(3), 1669-1679.

[31] Karaboga, D., 2005, An idea based on honey bee swarm for numerical optimization, Vol. 200. Technical report-tr06, Erciyes University, Engineering Faculty, Computer Engineering Department.

[32] Aslan, S., Arslan, S., 2022, A modified artificial bee colony algorithm for classification optimisation, International Journal of Bio-Inspired Computation, 20(1), 11-22.

[33] Karaboga, D., Basturk, B., 2008, On the performance of artificial bee colony (ABC) algorithm, Appl. Soft Comput., 8, 687–697.

[34] Karaboga, D., Basturk, B., 2007, A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm, J. Global Optim., vol. 39, no. 3, pp. 459-471.

[35] Özbay, E., 2023, An active deep learning method for diabetic retinopathy detection in segmented fundus images using artificial bee colony algorithm, Artif Intell Rev 56, 3291–3318.

[36] Zhang, C., Li, L., 2014, Hybrid differential evolution particle filter for nonlinear filtering, The Applied Computational Electromagnetics Society Journal (ACES), 1133-1139.

[37] Pozna, C., et al., 2022, Hybrid particle filter–particle swarm optimization algorithm and application to fuzzy controlled servo systems, IEEE Transactions on Fuzzy Systems, 30.10, 4286-4297.

[38] Hou, Y., et al., 2022, A multi-objective discrete particle swarm optimization method for particle routing in distributed particle filters, Knowledge-Based Systems 240: 108068.

[39] Jingquan Zhang, Dian Wang, Peng Li, Shiyu Liu, Han Yu, Yuxin Xu, Ming Teng, Application of an improved particle filter for random seismic noise suppression, Journal of Geophysics and Engineering, Volume 18, Issue 6, December 2021, Pages 943–953, https://doi.org/10.1093/jge/gxab064

[40] Liu, B. 2017, March. Robust particle filter by dynamic averaging of multiple noise models. In 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 4034-4038). IEEE.

[41] Hodson, T. O., 2022, Root-mean-square error (RMSE) or mean absolute error (MAE): when to use them or not, Geosci. Model Dev., 15(14), 5481-5487.