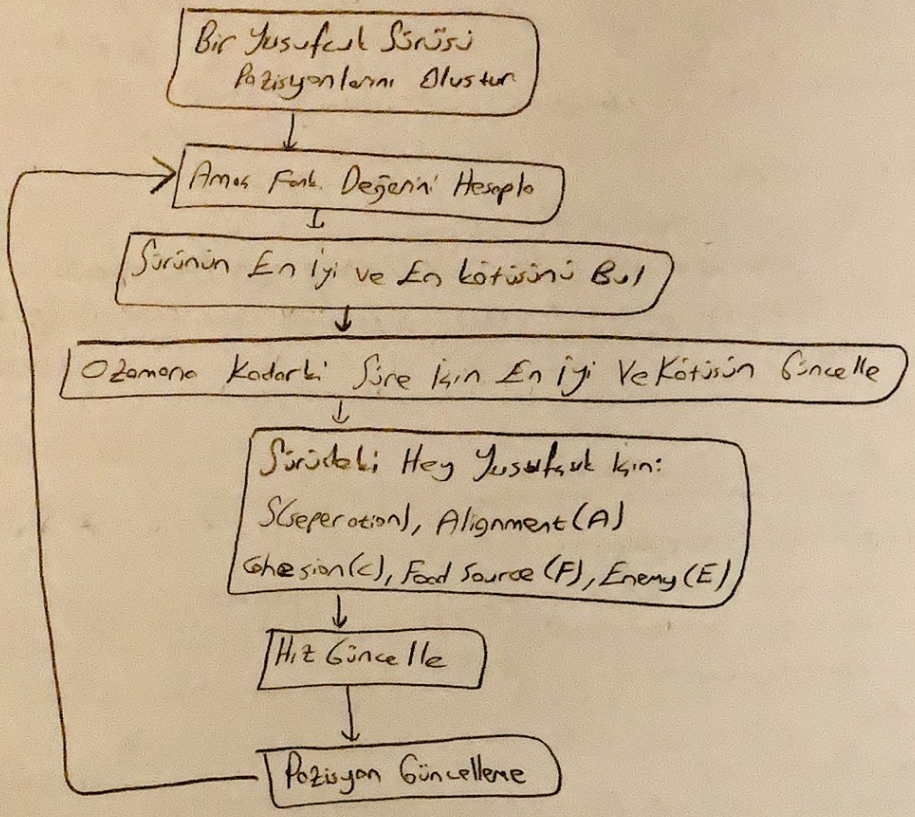
Yusufcuk (Dragonfly) Algoritması - Matlab

- PSO'ya çok benziyor. Daha çok global arama ağırlıklı iş için yapılıyor.



DF-1

→ Current Folder → New → Function → dragonfly.m

$\min f(x) = \sum x_i^2 = \|x1\|$ [-5, 5] → Aralık — Problemimiz

d=5 0.25 4.5 ±5 -4 -3

$= (0,25)^2 + (4,5)^2 + (5)^2 + (-4)^2 + (-3)^2$

↳ Amacımız sıfıra yaklaştırmak.

(- - - - - -) → 1. çözüm
(- - - - - -) → 2. çözüm
(- - - - - -) → n. çözüm
(n x d)

→ Öncelikimiz bunu oluşturmak.
→ Yusufçuk sürüsünü oluşturmak.
(n = satır sayısı, sürü sayısı
d = sütün sayısı, boyut)

- suru = unifrnd (as, us, [surusayisi boyut]); //başlangıç popülasyonu oluşturduk

- amac = zeros (surusayisi, 1);
- for i = 1 : surusayisi
  amac(i) = sum (suru(i,:).^2);
  end

//Amaç Fonk hesaplıyoruz.

amac = [ ] surusayisi x 1

→ ① Seperation = yusufçukların birbirine çarpmaması, ayrı — ayrı durması için.

$S_i = -\sum_{j=1}^{N} X - X_{fen}$ → kendi pozisyonu, komşunun n pozisyonu

② Alignment = birlikte uçacaklar yani hızları birbirine yakın olacak.

$A_i = \frac{\sum_{j=1}^{N} V_j}{N}$ → Tüm komşuların hız ortalaması

③ Chaisen = pozisyonları birbirine yakın olmalı.

$C_i = \frac{\sum_{j=1}^{N} X_j}{N} - X$ → komşuların pozisyon ortalaması

④ Food Force = Yemek kaynağı. En iyi çözümü simgeler.

$F_i = X^+ - X$

⑤ Enemy = Düşman. En kötü çözümü simgeler.

$E_i = X^- + X$

- enkucuk = min(amac); // Amaç fonk. en küçük değerini bulduk.
  idx = find(amac==enkucuk); // en küçük değeri veren index bulduk.
  ? idx = idx(1);
  yemek = suru(idx,:); // En iyi sonuç.

} (Yemeği Bulduk.)

- enbuyuk = max(amac); // Amaç fonk. en büyük değeri bulduk.
  idx = find(amac==enbuyuk); // En büyük değeri veren index.
  ? idx = idx(1);
  dusman = suru(idx,:); // En kötü sonuç

} (Düşmanı Bulduk)

⇒ S, A, C, Y, D

(S, A, C → Hız; hepsi → Yer Değiştirme)

$$V = S_{xs} + A_{xa} + C_{xc}$$

V → Hız, S → Seperation, A → Aligment, C → Chaisen

(Pozisyon) $X_1^{(1)} = [1 \ 2 \ 1 \ 0 \ 2]$ // X_1 çözümünün 1. zamandaki değeri

$$X_1^{(2)} = V + \vec{Y}.y + \vec{D}.d + w.\Delta x$$

Y → Yemek katsayı, D → Düşman katsayı, w → Eylemsizlik katsayısı, Δx → Eylemsizlik isteği

- S_s, - A_a, - C_c) Bunlar hesaplanacak Hız Gelecek.

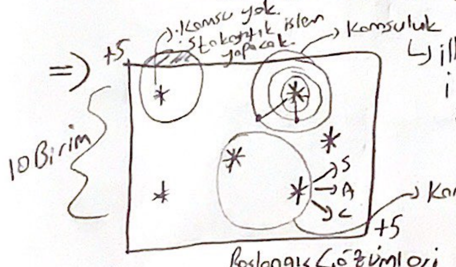
- Y_y, - D_d) Hesaplanacak pozisyon Değişikliği 1

Δx → pozisyon değişikliği 2

s, a, c, y, d, w → Bunlar katsayılar, biz gireceğiz.

- hiz = zeros(surusayisi, boyut); // hızın kalıbı oluşturuldu.
  deltaX = zeros(surusayisi, boyut); // Yer değiştirmenin kalıbı oluşturuldu.

⇒



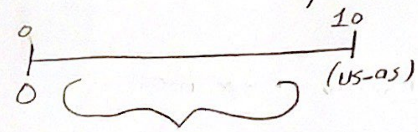
↳ ilk başta çok küçük sonra arttırıyoruz. ilerledikçe tüm popülasyonu kapsayabilir. iterasyonla komşuluk büyüyecek.

→ komşu var. ⇒ Her yusufçuk için komşuluk hesaplanıp işlemler yapılacak. Yoksa rastsal sayı ataması yapılacak.

10 Birim → 100 birimlik bir alandayız.

1. iterasyon → 1 birimlik çevre
⋮
100 iterasyon → 100 birimlik çevre

us - as = (5)-(-5) = 10) problem aralığı



max iterasyon → Yani max iterasyon neyse o kadar parçaya bölmeliyiz.

DF-3

→ squareform(pdist(suru))

Hesaplanan sonucu matrise çevirir

Süru elemanlarının uzaklıklarını hesaplar

Kare ve simetrik bir matris elde ederiz.

n×n

A = [1, 2, 3]
Setdiff(A,1) = 2 3

- iterasyon = 1; // Bu esit parçalara göre artırılacak
  eniyiiterasyon(iterasyon) = en kucuk;
  while iterasyon <= maksiterasyon

    uzunluk = squareform(pdist(suru)); // komşuluk hesabı yapılıyor.
    komşuluk = iterasyon * (us-as)/maksiterasyon; // çevreyi hesaplıyoruz

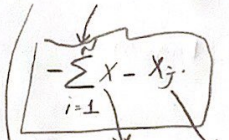
    for i = 1 : surusayisi

komşuların indexlerini bulur

    komsu = find(uzaklik(i,:) <= komsuluk); // yarıçapı içinde başka kuş varmı? varsa indexini alıyoruz.
    komsu = setdiff(komsu,i); // komşu matrisinin içinden kendinin indexini çıkarıyor.
    if isempty(komsu) // komşu boşmu? dolu mu? onun kontrolü!!

    else
    - komsu = suru(komsu,:); // Sürudeki komşuları aldık
      %indexlerini bildiğim komşuların kendilerini al
      - komsularinhizi = hiz(komsu,:);
    - seperation = - sum(repmat(suruli,:),size(komsular,1),1) - komsular);

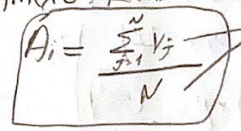
↳ Çıkarma işlemi yapacağımız için diğer matrisin boyutu kadar çoğaltmak lazım.



Sürünün satır sayısını komşulara eşitledik
Sütün sayıları zaten eşit.

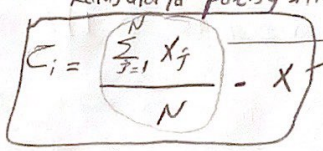
      % komşular ile birbirine çarpmamak için.

      % Aligment: komşuların hızı eşit olsun.



N tane komşunun hızını topla ve ortalamasını al.

    - aligment = mean(komsularinhizi);
      % cohesion = komşularla pozisyonları yakın olsun.



Komşuların pozisyonları ortalamasından, kendini çıkar.

    - cohesion = mean(komsular) - suru(i,:);
      %i. yusufcuk hızını güncelle
    - hiz(i,:) = s*seperation + a*alignment + c*cohesion;

→ s, a, c katsayılarının belirlenmesi gerekir. Bunları biz belirliyoruz.

↳ Genelde bunların toplamı 1 olacak şekilde verilir. (0.2 / 0.1 / 0.7) — (s / a / c)

→ max hız = (us − as)/2 ↳ Bu hızı geçmemeli

min. hız = −(us − as)/2 ↳ Bu hızı geçmemeli

- hiz(hiz > (us-as)/2) = (us-as)/2; // max. hız kontrolü yapıldı.
- hiz(hiz < (as-us)/2) = -(as-us)/2; // min hız. // //.

→ $F_i = X^t - X$ → En iyi sonuçla fark, kapatmalıyız!!

F_i: En iyi sonuç, X: yemek

⇒ $E_i = X^- + X$ → En kötü sonuçla fark, açmalıyız!!

E_i: En kötü sonuç, X^-: Düşman

- yemekyonu = yemek - suru(i,:);
  dusmanyonu = dusman + suru(i,:);
  deltax(i,:) = hiz(i,:) + yemekyonu * y + dusmanyonu * dt + deltax(i,:) * w;

y: sabit katsayılar, w: eylemsizlik katsayısı

Else kısmı bu kadar. (Komşu var)

— end

Komşu yoksa:

⇒ $X_{t+1} = X_t + Levy(d) \times X_t$ → şuanki konum

Δx

$$Levy(x) = (0.01) \times \left(\frac{r_1 + \sigma}{|r_2|^{1/\beta}} \right)$$ → (r_1, r_2 = (0,1) rastsal sayı)

$$\sigma = \left(\frac{\Gamma(1+\beta) \times \sin\left(\frac{\pi\beta}{2}\right)}{\Gamma\left(\frac{1+\beta}{2}\right) \times \beta \times 2^{\left(\frac{\beta-1}{2}\right)}} \right)^{1/\beta}$$ where $\Gamma(x) = (x-1)!$

gamma tam sayı ise $(x-1)!$ olarak düşün.

*. Problemin boyutu azaltıldıkça ve iterasyon sayısı arttırıldıkça sonuçlar daha minimize olacaktır.

*. Çıkan sonuçu direk kullanmak önerilmez. Daha sonra tavlama benzetimi gibi algoritmalarla birleştirme yapılabilir.

DF-5

```
if isempty(Komsu)
    %sigma(G) hesabı:
    pay = gamma(1+beta) * sin(pi*beta/2);
    payda = gamma((1+beta)/2) * beta * 2^((beta-1)/2);
    sigma = (pay/payd)^(1/beta);
    Levy = 0.01 * unifrnd(0,1) * sigma / (unifrnd(0,1)^(1/beta));

    %DeltaX hesabı:
    deltax(i,:) = levy * suru(i,:);        > Komşu Yoksa
else
                                           > Komşu Varsa
end
end
  Suru = suru + deltax;
  Suru(suru > us) = us; // Sürünün üst sınırdan büyük olan elemanlarını üst sınıra eşitle.
  Suru(suru < as) = as; //  //    alt  //  küçük //  //  alt //  //.

  %amaç fonk. değerini bul.
  for i = 1 : surusayisi
      amac(i) = sum(suru(i,:).^2);
  end
  if min(amac) < enkucuk
      enkucuk = min(amac);
      idx = find(amac == enkucuk);      // Tüm iterasyonlardaki en küçüğü bul.
      idx = idx(1); // Bir çok değer olabilir, bunlardan ilkini al.
      en' Yemek = suru(idx,:); //En iyi sonucu değiştirdik.
  end
  if max(amac) > enbuyuk
      enbuyuk = max(amac);
      idx = find(amac == enbuyuk);
      idx = idx(1);
      dusman = suru(idx,:);
  end
  iterasyon = iterasyon + 1;
  eniyi iterasyon(iterasyon) = enkucuk;
end // while'in endi.
plot(eniyi iterasyon, 'LineWidth', 2) // Bulunan sonuçları çizdirir.
end // fonksiyonun endi.
```

DF-6