Particle Swarm Optimization

Povilas Asijavičius

Sausis 2015 m.

1 Teorinė dalis

Paprastai tariant, galima teigti, kad Particle swarm optimization (PSO) algoritmą apibūdina turima populiacija (swarm, būrys) ir galimi sprendiniai (particles, būrio dalelės). Šios dalelės juda po visą nagrinėjamą sritį. Jų judėjimo kryptį nusako kelios nesudėtingos formulės. Dalelių judėjimą lemia jų pačių bei viso būrio rastų optimalių taškų pozicijos. Radus geresnius optimalius taškus visas būrys atsižvelgia į šiuos naujus rezultatus. Procesas tęsiasi tol, kol randamas optimalus taškas arba kol įvykdomas maksimalus iteracijų skaičius.

Kalbant formaliau, galima apibrėžti išlaidų (cost) funkciją $f : \mathbb{R}^n \to \mathbb{R}$, kurią sieksime minimizuoti. Ši funkcija kiekvienam vektoriui prisiskirią realią reikšmę. Jos išvestinė nėra žinoma. Priešingu atveju, minimumą būtų galima rasti analiziškai. Taigi tikslas yra rasti tokį sprendinį x, kad $f(x) \leq f(y)$, kiekvienam y iš nagrinėjamos erdvės. Norint rasti funkcijos maksimumą, reikėtų minimizuoti funkciją $-f$.

Tarkime, kad S yra dalelių būryje skaičius. Kiekviena iš šių dalelių (i-toji) turi dabartinę savo poziciją erdvėje $x_i \in \mathbb{R}^n$ bei greitį $v_i \in \mathbb{R}^n$. Taip pat tariame, kad geriausia žinoma i-tosios dalelės pozicija yra $p_i.$, o g_i geriausia žinoma visos būrio pozicija. Tada PSO algoritmą galima užrašyti taip:

- Kiekvienai dalelei i, kur $i = 1, ..., S$ turime:
  - Nustatome dalelės pradinę poziciją pagal tolygųjį skirstinį $x_i \sim U([b_{il}, b_{iu}])$, kur

b_{il}, b_{iu} yra atitinkamai apatinė ir viršutinė nagrinėjamos srities ribos.

- Geriausia dalelės pozicija nustatome dabartinę jos poziciją $p_i \leftarrow x_i$.
- Atsitiktinai nustatome dabartinį dalelės greitį $v_i \sim U([-|b_{iu} - b_{il}|, |b_{iu} - b_{il}|])$

- Kol neįvykdoma nustatyta sąlyga (neįvyksta maksimalus iteracijų skaičius arba nėra rastas optimalus taškas) yra vykdoma:
  - Kiekvienai dalelei i, kur $i = 1, ..., S$ turime:
    - Generuojame atsitiktinius skaičius $r_p, r_g \sim U([0, 1])$.
    - Kiekvienam d, kur $d = 1, ..., n$:
      - atnaujiname dalelės greičius: $v_{id} = \omega v_{id} + \phi_p r_p(p_{id} - x_{id}) + \phi_g r_g(g_d - x_{id})$
    - Atnaujiname dalelės poziciją: $x_i \leftarrow x_i + v_i$.
    - Jei $f(x_i) < f(p_i)$:
      - Atnaujiname dalelės geriausią žinomą poziciją $p_i \leftarrow x_i$
      - Jei $f(p_i) < f(g)$, tai atnaujinama geriausia viso būrio pozicija $g \leftarrow p_i$
- Dabar g yra geriausias rastas sprendimas.

Parametrai ω, ϕ_p ir ϕ_p yra parenkami priklausomai nuo situacijos.

2 Praktinė dalis

Toliau bus bandoma pritaikyti ši algoritmą praktiniuose tyrimuose. Parinksime keletą funkcijų ir jas minimizuosime pasirinkus reikiamus parametrus. Darbui naudosime R programą ir jos paketą *pso*. Pagrindinė funkcija, kurią naudosime yra *pso*. Jos naudojimas apsirašo taip:

```
psoptim(par, fn, lower = -1, upper = 1, control = list())
```

Trumpai apibūdinkime parametrus:

- *par* - vektorius nusakantis kokios dimensijos yra erdvė. galima įrašyti tiesiog vektorių su tuščiais elementais. Visgi, jei nurodytos reikšmės paklius į vėliau nustatytas erdvės ribas, tai būtent ši reikšmė bus pradinė reikšmė.
- fn - funkcija, kuria minimizuojame, funkcijos rezultatas turi būti skaliaras.
- *lower* - apatinė kintamojo riba.
- *upper* - apatinė kintamojo riba.
- *control* - kiti parametrai (tarp jų: maksimalus iteracijų skaičius, konvergavimo tikslumas ir pan.).

Toliau išbandykime šią funkciją visiškai paprastam pavyzdžiui:

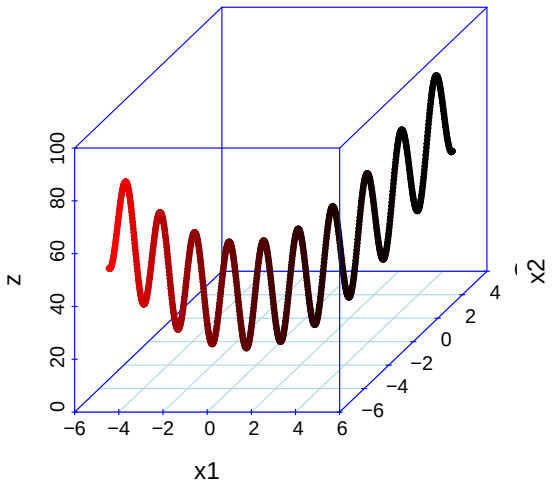
```
> set.seed(1)
> psoptim(c(NA),function(x) x^2-3,lower=-5,upper=5,control=list(abstol=1e-8))
$par
[1] 0.7285336

$value
[1] -2.469239

$counts
 function iteration  restarts
       12         1         0

$convergence
[1] 0

$message
[1] "Converged"
```

Šiuo atveju buvo bandoma rasti minimumą funkcijos $f(x) = x^2 - 3$. Erdvės ribos buvo nustatytos nuo -5 iki 5. Gavome, kad algoritmas konvergavo bet gavo nepatį tiksliausią rezultatą: minimumo taškas yra 0,73 (vietoj 0 iš tikrųjų), funkcijos minimumo taškas -2,47 (-3). Leidžiant simuliaciją daugiau kartų randami ir geresni rezultatai.

Toliau pereikime prie sudėtingesnių situacijų. Bandysime minimizuoti dviejų kintamųjų funkciją:

$$f(x) = 20 + \sum_{i=1}^{2}(x_i^2 - 10\cos(2\pi x_i)) \tag{1}$$

Kadangi funkcija yra dviejų kintamųjų, mes negalime taip greit nuspėti tikrojo optimumo taško, tačiau mes galime ją pavaizduoti 3D grafike:



Grafike gana aiškiai matyti kad minimumas yra maždaug ties reikšme $(0, 0)$. Galime pabandyti šį uždavinį išspręst PSO algoritmo pagalba.

```
> set.seed(1)
> psoptim(rep(NA,2),function(x) 20+sum(x^2-10*cos(2*pi*x)),
+         lower=-5,upper=5,control=list(abstol=1e-8))
$par
[1]  5.798684e-06 -1.490932e-08

$value
[1] 6.670927e-09

$counts
 function iteration  restarts
     1404       117          0

$convergence
[1] 0
```

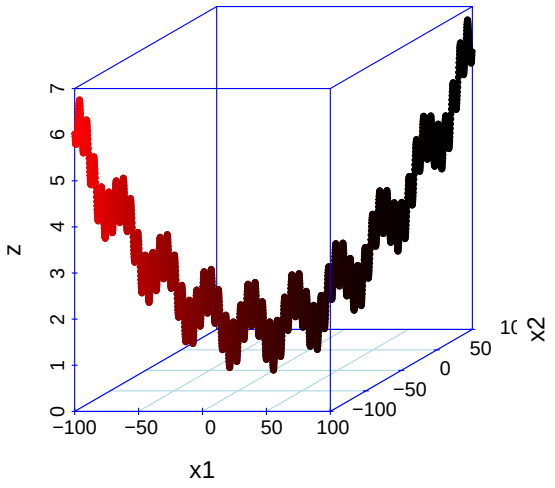
```
$message
[1] "Converged"
```

Matome, kad gavome tikrai gana tikslius rezultatus. *par* reikšmės yra labai arti nulio, kaip ir pačios funkcijos reikšmė *value*.

Pabandysime minimizuoti dar vieną dviejų kintamųjų funkciją:

$$f(x) = \frac{\sum_{i=1}^{2} x_i^2}{4000} - \sum_{i=1}^{2} \cos \frac{x_i}{\sqrt{i}} + 1 \tag{2}$$

Vizualizuokime šią funkciją:



Matome, gana panašų vaizdą kaip ir turėjome prieš tai. Pabandykime pritaikyti *pso*:

```
> psoptim(rep(NA,2),function(x) sum(x*x)/4000-prod(cos(x/sqrt(1:2)))+1,
+         lower=-100,upper=100,control=list(abstol=1e-2))
$par
[1] -0.04160654 -0.09827913

$value
[1] 0.003279911

$counts
 function iteration  restarts
```

```
     1020         85          0

$convergence
[1] 0

$message
[1] "Converged"
```

Matome, kad šiuo atveju rezultatai nėra tokie geri. Įdomu tai, kad paleidus algoritmą daugiau kartų, neretai rezultatai buvo dar blogesni. Tai rodo, kad nepaisant vizualinio panašumo į prieš tai buvusį atvejį, algoritmas šią funkciją minimizuoja kur kas sunkiau .