Jednodimenzione metode

Njutn-Rapson

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

dok $|x_{k+1} - x_k| < \xi$

Jedno pocetno pogadjanje. Rezultat je stacionarna tacka, mi moramo proveriti da li je minimum ili maksimum (plot ili slobodna optimizacija).
Osetljiviji na pocetno pogadjanje nego metod secice.
Brzi dolazak do resenja nego metod secice.
Zasto je kriterijum zaustavljanja apsolutna greska a ne relativna? Deljenje 0.

Sečica

$$x_{k+1} = x_k - \frac{f'(x_k)*(x_k - x_{k-1})}{f'(x_k) - f'(x_{k-1})}$$

dok $|x_{k+1} - x_k| < \xi$

Potrebne dve pocetne tacke. Rezultat je stacionarna tacka, mi moramo proveriti da li je minimum ili maksimum (plot ili slobodna optimizacija).

Fibonačijev metod

Pretraga zatvorenog intervala unimodalne funkcije. Prednost je činjenica da možemo odrediti optimum bez informacije o diferencijabilnosti funkcije.
Fiksan broj iteracija odredjen fibonacijevim brojem (for i in range (2, n+1)).

Trazimo prvi fibonacijev broj koji zadovoljava relaciju:

$$F_n > \frac{b-a}{\xi}$$

$$x_1 = a + (b - a)\frac{F_{n-2}}{F_n}$$

$$x_2 = a + b - x_1$$

```
if(func(x1)<=func(x2)):
                b=x2
                x2=x1
                x1=a+b-x2
            else:
                a=x1
                x1=x2
                x2=a+b-x1
```

Pseudokod:

1. Odrediti interval [a,b] koji sadrzi x* i specifirati rezoluciju (tacnost aproksimacije) ξ
2. Odrediti najmanji prirodan broj n koji zadovoljava uslov $F_n > \frac{b-a}{\xi}$
3. Izracunati prvu iteraciju

   $x_1 = a + (b - a)\frac{F_{n-2}}{F_n}$

   $x_2 = a + b - x_1$
4. Izracunati k-ti interval i ponavljati sve dok k=n:

   if $f(x_1) \le f(x_2)$ then b=x_2, $x_2 = x_1$, $x_1 = a + b - x_1$

   if $f(x_1) \ge f(x_2)$ then a=x_1, $x_1 = x_2$, $x_2 = a + b - x_2$

Zlatni presek

Pretraga zatvorenog intervala <u>unimodalne</u> funkcije. Ne moramo imati podatak o diferencijabilnosti.

Uopstenje Fibonacijevog metoda jer se kolicnik $\frac{F_{n-2}}{F_n}$ posle ~15 iteracija ponasa konstantno.

Prednost je to što ne treba unapred da odredimo broj iteracija sto ga cini brzim od Fibonacija.

$c = \frac{3-\sqrt{5}}{2}$

$x_1 = a + c \cdot (b - a)$

$x_2 = a + b - x_1$

Kriterijum zaustavljanja: (b-a)<ξ

```
while (abs(b-a))>tol :
    iter+=1
    if(func(x1)<=func(x2)):
        b=x2
    else:
        a=x1
    x1=a+c*(b-a)
    x2=a+b-x1
```

Pseudokod:

1. Odrediti interval [a,b] koji sadrzi x* i specifirati rezoluciju (tacnost aproksimacije) ξ
2. $c = \frac{3-\sqrt{5}}{2}$
3. Izracunati prvu iteraciju

   $x_1 = a + c \cdot (b - a)$

   $x_2 = a + b - x_1$
4. Izracunati k-ti interval i ponavljati sve dok (b-a)<ξ :

if $f(x_1) \leq f(x_2)$ then b=x_2

if $f(x_1) \geq f(x_2)$ then a=x_1

and $x_1 = a + c \cdot (b - a)$

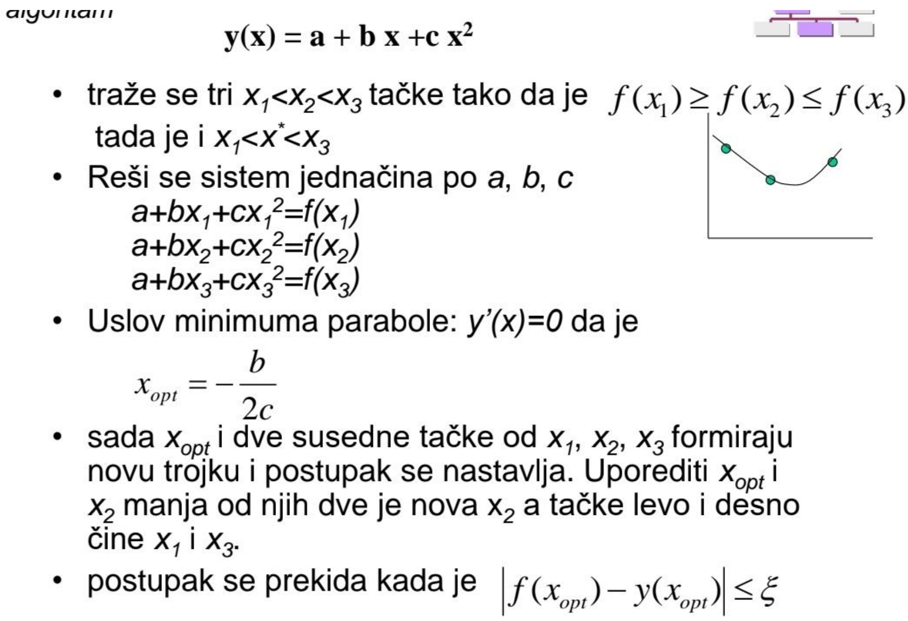
$x_2 = a + b - x_1$

Parabola

<u>Unimodalna</u> funkcija f se aproksimira polinomom *y(x)* na intervalu I koji sadrzi optimum; odredi se minimum *y(x)=x**; u okolini x* formira se novi interval i vrsi se aproksimacija.

$y(x) = a + bx + cx^2$

$|f(x^*) - y(x^*)| < \xi$



Višedimenzione metode

Najbrzi pad

$$x_{k+1} = x_k - \gamma \cdot \nabla f(x_k)$$

Kriterijum zaustavljanja $|\nabla f(x_k)| < \xi$

Treba nma jedno pocetno pogadjanje. Algoritam mnogo zavisi od izbora γ, izvrsava se poprilicno dugo, a najveca mana je da staje u prvoj kriticnoj tacki na koju naidje (lokalni minimum, prevojna tacka,..).

Uopstenje ovog algoritma- Gradijentni algortam sa normalnizovanim (fiksnim) korakom:

$$x_{k+1} = x_k - \gamma \cdot \frac{\nabla f(x_k)}{|\nabla f(x_k)|}$$

Brzi, ali neprecizan, kruzi oko resenja za γ.

Gradijentni sa momentom

$v_k = \omega v_{k-1} + \gamma \cdot \nabla f(x_k)$

$x_{k+1} = x_k - v_k$

Kriterijum zaustavljanja $|\nabla f(x_k)| < \xi$

$\omega \in (0, 1)$ ("koliko prethodnog gradijenta zadrzati")
Intercija- teznja zadrazti prethodno stanje kretanja
Za γ nam treba iskustvo u nekoj klasi problema.
=> Komponente brzine se sabiraju tokom iteracija (isti smer), dok ce se komponente promenljivog smera medjusobno ponistavati.
* eksplozija algoritma - nestabilnost jedne cestice dovodi do raspada roja ~ nestabilnost sistema
Glavna prednost - sposobnost prelaska preko prevojne tacke! (takodje i smanjenje oscilacija)

Nestorov

Racunanje gradijentra u pretpostavljenoj tacki x'. Ona omogucava brzu reakciju na promenu gradijenta funkcije. Direktniji put do optimuma - smanjenje oscilacija.

$x' = x_{k-1} - \omega v_{k-1}$

$v_k = \omega v_{k-1} + \gamma \cdot \nabla f(x')$

$x_{k+1} = x_k - v_k$

ADAGRAD

Mane prethodnih algoritama: ista brzina obuke u svim pravcima; male promene nekih parametara mnogo uticu na kriterijum optimalnosti, dok velike promene nekih drugih ne uopste.
Uvodimo adaptivni gradijent - brzina adaptacije razliicta za svaku osu (svaku promenljivu)
=> γ obrnuto proporcionalni gradijentu

$g_{i,k} = \nabla f(x_k)_i$ gradijent kriterijuma optimlanosti po i-toj promenljivoj u k-toj iteraciji

$G_{i,k} = \sum_{i=0}^{k+1} g_{i,k}^2 \approx G_{i,k-1} + \nabla f(x_k)_i{}^2$

$x_{k+1} = x_k - \frac{\gamma}{\sqrt{G_{i,k} + \xi_1}} g_{i,k}$ (ξ_1 sluzi da bi se izbeglo deljenje 0)

Kriterijum zaustavljanja $|\nabla f(x_k)| < \xi$

Posto se stalno dodaje na sumu, brzina obuke postaje sve slabija (akumulacija gradijenta u G) sto vise napedujemo sa iteracijama tako da se ovakav algoritam kao takav i ne koristi toliko.

ADAM

Adaptivno momentno pretpostavljanje, najcesce koriscen gradijentni algoritam, veoma efikasan.
Ideja:

1. Imamo momentni element koji filtriramo pomocu prethodnih pravaca i gradijenta

2. kvadrat gradijenta
3. skaliranje velicina
4. idemo u <u>pravcu</u> momenta, a <u>brzinu</u> dobijamo pomocu v = posebno racunanje po koordinatama

(nije problem izabrati gamu)

$m_k = \omega_1 m_{k-1} + (1 - \omega_1) \cdot \nabla f(x_k)$

$v_k = \omega_2 v_{k-1} + (1 - \omega_2) \cdot \nabla f(x_k)^2$

[$m_h = \frac{m_k}{1-\omega_1}$; $v_h = \frac{v_k}{1-\omega_2}$] normalizovane vrednosti

$x_{k+1} = x_k - \frac{\gamma}{\sqrt{v_h + \xi_1}} \cdot m_h$

Kriterijum zaustavljanja $|\nabla f(x_k)| < \xi$

ω_1, ω_2 – $(0,1)$ služe za određivanje dela pozicije koji zadržavamo iz prethodne iteracije

γ – konstanta za skaliranje duzine koraka
m - momentni element filtriran pomocu gradijenta
v - brzina

Njutnov algoritam

Efikasniji od svih gradijentnih metoda, ali prakticno neprimenjiv za jako velike probleme. Pokusaj odredjivanja pravca, duzine skoka i korekcije ugla gradijenta.

Genetski

*problem maksimizacije
* funkcija ne mora biti diferencijalna, ne moramo znati nista o njoj, moze biti multimodalna
KO - kriterijum prilagodjenosti (mera kvaliteta jedinke)
jedinke - 1 potencijalno resenje problema
populacija - skup razmatranih jedninki (N - broj jedinki)

0. Reprezentacija jedinki **realno ili binarno** (predstavljanje realnih brojeva binarno se vrsi diskretizacijom- nivo diskretizacije, broj bita za predstavljanje svakog broja,...)
1. Selekcija - $\frac{N}{2}$ izbora parova roditelja tkd svaki par generise par potomaka (jedinka moze biti roditelj vise puta); bolje prilagodjene jedinke imaju vecu verovatnocu izbora
Ruletska selekcija - trazimo 2 maksimalna $f_k r_k$ k $\in$ 1, ...N ; moguce je i dodavanje ranga (redni broj u nizu sortiran po "dobroti", vrednosti funkcije prilagodjenosti) u taj proizvod i trazenja 2 makismalna
2. Ukrstanje - kod realno kodiranih:

$p_1 = r_1 R_1 + (1 - r_1) R_2$

$p_2 = r_2 R_1 + (1 - r_2) R_1$ (p - potomak ; R - roditelj)

- kod binarno kodiranih - p_1 dobija bit jednog roditelja, a p_2 od preostalog ukrstanje u jednoj tacki, ukrstanje u svakoj tacki (nasumicno)...

3. **Mutacija** - sa nekom malom verovatnocom promeniti (unos novine razlicite od osobina roditelja) jednog potomka (promena moze biti velika); smisao: unos novih osobina u populaciju
- kod realno kodiranih: mutacija= p+r (r- slucajan vektor/broj iz (0,1))
- kod binarno kodiranih: mutacija inverzijom - random invertovanje 1 ili vise bita

4. Izbor nove populacije (**prirodna selekcija**)
prva opcija: ubiti sve roditelje i pustiti decu da zive
druga opcija: rangiranje svih 2N jedinki po prilagodjenosti i pustimo N najboljih da prezive; ova opcija je mnogo bolja (biodiverzitet), mada moze da dovede do brze konvergencije ka nekom lokalnom ekstremu umesto ka globalnom

5. Neka resenja za probleme GA
- uvodjenje zivotnog veka promenljive (sa drugom opcijom izbora nove populacije)
- **Elitizam** - mali broj najbolje prilagodjenih jedinki se pusti da zivi u svim generacijama gde se podrazumava da ce svi roditelji (preostale jedinke) umirati.

Ponavljati 1-4 dok populacija ne iskonvergira.

```
 Main loop - sklopljen genetski algoritam
- Generišemo početnu populaciju veličine *population_size*
- Vrtimo glavnu petlju maksimalno *max_iter* puta, pri čemu svaka iteracija
petlje predstavlja jednu generaciju
- Rangiramo jedinke po prilagođenosti, funkcijom **rank_chromosomes**
- Funkcijom **natural_selection** biramo roditelje (ovo je visak)
- Za binarno: Funkcijom **bin_encode_chromosomes** kodiramo celu roditeljsku
populaciju
- Funkcijom **roulette_selection** odvajamo parove roditeljskih hromozoma
koje ćemo ukrstiti
- Nekom metodom ukrštanja uparujemo roditeljske hromozome i dobijemo
populaciju dece
- Spajamo novonastale hromozome sa roditeljskim u novu populaciju
*chromosomes*
- Na ovoj populaciji vršimo mutaciju funkcijom sa zadatim *mutation_rate*
- Za binarno: Dekodiramo celu populaciju funkcijom
**bin_decode_chromosomes**
- Proveravamo da li populacija konvergira ili da li smo došli do optimalnog
rešenja
- Ispisujemo statistike za svaku generaciju: prosečna prilagođenost,
najbolji hromozom i sastav najboljeg hromozoma
```

PSO

Ideja: algoritam ponasanja socijalnih ponasanja zivotinja koje se pomeraju u velikim grupama (ptice). Optimizujemo poziciju cestice, koja se nalazi u populaciji cestica. Svaka cestica ima poziciju x i brzinu v, pri cemu je brzina razlika izmedju trenutne i prethodne pozicije. Svaka cestica pamti svoju <u>licnu najbolju poziciju</u>, a populacija pamti <u>najbolju globalnu poziciju</u> pri cemu ce cela populacija biti vodjena sa ova 2 faktora. Preracunavamo poziciju i brzinu u svakoj iteraciji sa:

$v_k = \omega v_{k-1} + c_p r_p (p_k - x_k) + c_g r_g (g_k - x_k)$
$x_{k+1} = x_k + v_k$
$\omega \in (0, 1)$ - inercijlani faktor (deterministicka velicina)
c_p - kognitivni faktor >0
c_g - socijalni faktor >0
r_p, r_g - slucajni faktori, random rasporedjeni u [0,1]
*sa vremenom ω i c_p opadaju, dok c_g raste (preciznija pretraga oko optimuma, manje oslanjanje na licno iskustvo, a vise na iskustvo cele grupe u kasnijim iteracijama)
*ovo je dinamicki sistem - podesavamo oscilatornost i vreme smirenja pri cemu je neophodno da cestice imaju oscilatoran karakter (potrebno je promasiti cilj kako bismo se vratili u njega)

Postupak (pseudokod):

1∘ Inicijalizacija: postaviti parametre algoritma - broj iteracija, broj cestica; postaviti vrednosti pozicija i brzina cestica (random)

2∘ Optimizacija:

a) Izracunati vrednost funkcije za svaku cesticu
b) Podesiti vrednosti licne i globalne najbolje pozicije
   if $f_k \leq f_{p_{best}}$ then $f_{p_{best}} = f_k$ and $p_{best} = x_k$
   if $f_k \leq f_{g_{best}}$ then $f_{g_{best}} = f_k$ and $g_{best} = x_k$
c) Podesiti nove vrednosti pozicije i brzine
   $v_k = \omega v_{k-1} + c_p r_p (p_k - x_k) + c_g r_g (g_k - x_k)$
   $x_{k+1} = x_k + v_k$

3∘ Proveriti kriterijum zaustavljanja:

a) Dostignut postavljen broj iteracija
b) Globalno najbolja pozicija se minimalno promenila (za neko zadato ξ)