Optimizavimo metodai - 1 laboratorinio darbo ataskaita

Autorius: Vilius Minkevičius, Programų Sistemos 4 kursas 2 grupė

Optimizuota funkcija: f(x) = $\frac{(x^2-3)^2}{9} - 1$, (a = 3, b = 9). Skaitant užduoties aprašymą pasirodė, kad galima funkciją interpretuoti šitaip: f(x) = $\frac{(x^2-3)^2}{8}$, bet ši interpretacija atmesta.

Optimizavimo uždavinys išspręstas naudojant Octave įrankį su Matlab kalba. Analitiškai išsprendus uždavinį gauta minimumo taško x koordinatė ($\sqrt{3}$) sutampa su apytiksliais atsakymais (1.7320...), gautais naudojant algoritmus.

Optimizavimas dalinant intervalą pusiau

Sukurta programinė funkcija, kuriai paduodama optimizuojama tikslo funkcija, intervalo galai bei norima paklaida. Funkcijos išeiga – kiekvienos iteracijos intervalo vidurio taškai. Paskutinis taškas – minimumo taško x koordinatės aproksimacija. Funkcijos pradžioje apskaičiuojamos trijų taškų y reikšmės, o kiekvieną iteraciją – po du kartus. Kai intervalas tampa mažesnis nei paklaida, gražinamas vidurys.

Optimizavimas dalinant aukso pjūviais

Sukurta funkcija, kuri turi tą pačią įeitį ir išeitį kaip ir ankstesnė. Funkcijos pradžioje tikslo funkcija apskaičiuojama 2 kartus, o kiekvieną iteraciją po vieną kartą. Grąžinami iteracijų intervalų vidurio taškai. Pastaba: atstumas tarp imamų taškų yra mažesnis nei atstumas tarp kurio nors taško ir intervalo galo.

Niutono metodo pritaikymas optimizavime

Sukurta funkcija, kuriai perduodama pradinė x koordinatė, tikslo funkcijos pirmoji ir antroji išvestinės bei norimas tikslumas. Žymus trūkumas – nenaudojant bibliotekų, kurios duotai funkcijai apskaičiuotų išvestines, labai nepatogu keisti optimizuojamą funkciją. Reikia po kiekvieno pakeitimo perrašyti išvestines. Priklausomai nuo naudojamo įrankio gali būti lengvai pasiekiamos ir paprasto panaudojimo išvestinių skaičiavimo funkcijos arba sunkiai pasiekiamos ir nepatogios.

Optimizuojant hiperbolinės formos funkciją pastebėti šie Niutono metodo trūkumai:

1. Parinkus pradinį tašką toli nuo minimumo galima iteracijų metu išeiti už intervalo ribų, peršokti trūkio tašką.
2. Tad reikia parinkti pradinės iteracijos tašką pakankamai arti minimumo taško.
3. Dėl šuolių galima surasti ne tą minimumo tašką, kurio ieškota, arba rasti maksimumą.
4. Reikia papildomo patikrinimo, kad iteracijų metu nevyktų dalyba iš labai mažų skaičių. Gautųsi didelė paklaida ir galimai iteracinis metodas diverguotų.

Informacijos surinkimas ir atvaizdavimas

Parašytos funkcijos modifikuotos taip, kad vykdymo metu būtų kaupiama iteracijų informacija: įvykdytų ciklų skaičius, kiek kartų apskaičiuota tikslo funkcija. Kiekviena funkcija grąžina iteracijos taškų X koordinates. Jos prieinamos pagrindinėje funkcijoje, kuri kviečia optimizavimo metodus, ir panaudojamos atvaizduojant metodą grafike.

Metodų palyginimas

Metodo pavadinimas	Iteracijų kiekis	Tikslo funkcijos apskaičiavimai	Pasiekta paklaida
Dalinant intervalą pusiau	15	30	0.0001
Dalinant intervalą aukso pjūviu	21	21	0.0001
Niutono metodas	7	14 (po 7 kartus apskaičiuotos abi išvestinės)	0.0001

Išvados

1. Intervalo dalinimo aukso pjūviu algoritmas paprastesnis ir efektyvesnis nei dalinant intervalą pusiau.
2. Niutono metodas efektyviausias, tačiau jį keisti sudėtinga. Jis gali būti mažiau efektyvus, jei išvestinių skaičiavimas yra sudėtingesnis nei pačios tikslo funkcijos.
3. Nagrinėjant metodus pastebėta, kad iš minėtų trijų metodų vienintelis Niutono metodas gali iššokti iš nagrinėjamo intervalo ribų ir dėl to veikti nenuspėjamai.

Priedas 1: Kodo fragmentai

```
while (intervalo_ilgis > paklaida)
  x1 = a + intervalo_ilgis / 4;
  x2 = b - intervalo_ilgis / 4;
  f1 = f(x1);
  f2 = f(x2);
  if f1 < f_vid
    # Nagrinejame pirmaja puse [a, x_vid]:
    b = x_vid;
    x_vid = x1;
    f_vid = f1;
  elseif f2 < f_vid
    # Nagrinejame antraja puse [x_vid, b]:
    a = x_vid;
    x_vid = x2;
    f_vid = f2;
  else
    # Nagrinejame aplinkui viduri [x1, x2]:
    a = x1;
    b = x2;
  end
  intervalo_ilgis = b - a;
  X = [X, x_vid];
  ciklai += 1;
end
```

Nuotrauka 1 - kodo fragmentas, vaizduojantis intervalo dalinimo pusiau algoritmą.

```
while (intervalo_ilgis > paklaida)
  intervalo_ilgis = b - x1; # tiksl
  if f1 < f2
    # Nagrinejame [a, x2]:
    b = x2;
    x2 = x1;
    f2 = f1;
    x1 = b - fi * intervalo_ilgis;
    f1 = f(x1);
  else
    # Nagrinejame [x1, b]:
    a = x1;
    x1 = x2;
    f1 = f2;
    x2 = a + fi * intervalo_ilgis;
    f2 = f(x2);
  end
  ciklai += 1;
  X = [X (a+b)/2];
end
```

Nuotrauka 2 - intervalo dalinimo aukso pjūviais algoritmas.

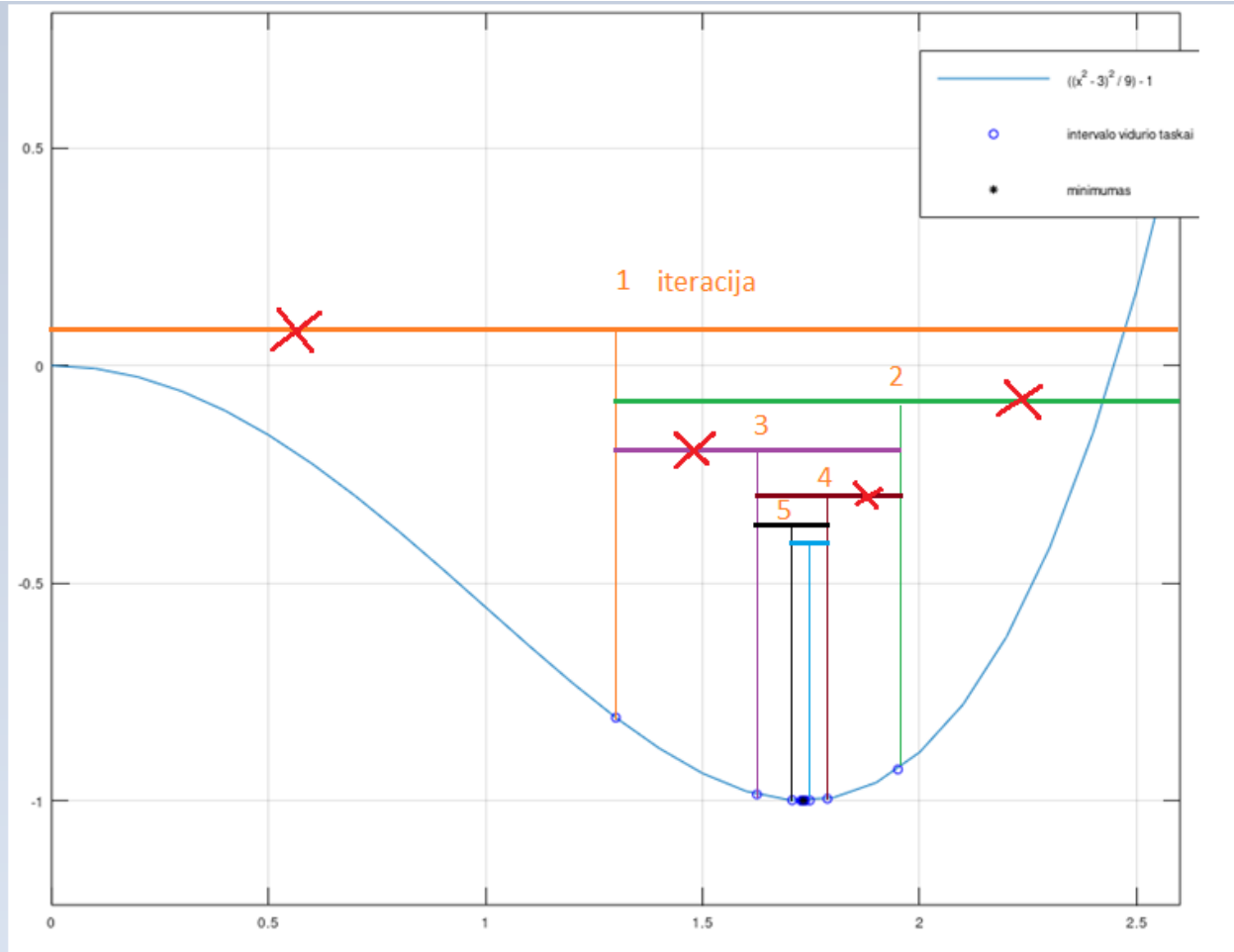
```
while (abs(xi-x_ankstesnis) > tikslumas)
  x_ankstesnis = xi;
  xi = xi - (f_isvestine_1(xi) / f_isvestine_2(xi));
  ciklai += 1;
end
```

Nuotrauka 3 – Niutono metodo iteracija.

```
f_isvestine_1 = @(x) (4*x^3 - 12*x) / 9;
f_isvestine_2 = @(x) (4/3) * (x^2 - 1);
x_min_niutonu = niutono_metodas(x0, f_isvestine_1, f_isvestine_2, tikslumas);
minimumo_taskas_niutonu = [x_min_niutonu, f(x_min_niutonu)]
```

Nuotrauka 4 - Niutono metodo panaudojimas.

Priedas 2: Iliustracija, kaip, dalinant intervalą pusiau, artėjama prie minimumo



Nuotrauka 5 - optimizavimas dalinant intervalą pusiau. Horizontalios linijos – analizuojami intervalai. Vertikalios linijos – intervalų viduriai. Nubrauktos horizontalios linijos – atmesti intervalai.