VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS INSTITUTAS
INFORMATIKOS KATEDRA

Optimizavimo metodai

I laboratorinis darbas - vienmatis optimizavimas

Atliko: 3 kurso 2 grupės studentė
Gabrielė Rinkevičiūtė

Vilnius
2024

Turinys

Įvadas 2
1. Dalijimo pusiau metodas 3
2. Auksinio pjūvio metodas 6
3. Niutono metodas 9
Išvados 11

Įvadas

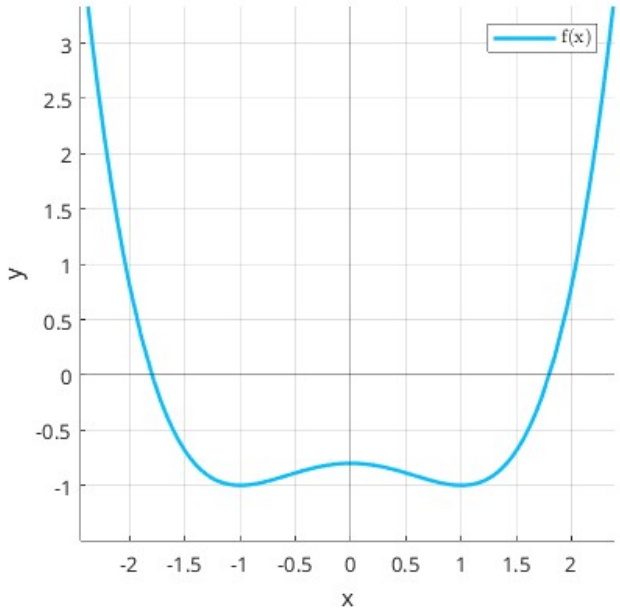
Šiame laboratoriniame darbe yra analizuojami trys funkcijos minimizavimo metodai: dalijimo pusiau metodas, auksinio pjūvio metodas, Niutono metodas. Šie metodai yra tiriami, juos pritaikant funkcijai

$$f(x) = \frac{(x^2 - a)^2}{b} - 1 \tag{1}$$

Šiuo atveju $a = 1, b = 5$, todėl

$$f(x) = \frac{(x^2 - 1)^2}{5} - 1 \tag{2}$$

Žemiau galima pamatyti šios funkcijos grafinį atvaizdavimą [1].



1 pav. Funkcijos $f(x)$ grafikas

Metodai yra lyginami pagal tai, kiek jiems prireikė iteracijų bei funkcijų skaičiavimų atlikti, kad būtų gautas norimo tikslumo rezultatas.

Metodai buvo įgyvendinti Matlab programavimo kalba.

1. Dalijimo pusiau metodas

Žemiau pateiktas algoritmo pseudokodas iliustruoja dalijimo pusiau metodo veikimo principą. Šio algoritmo įgyvendinimą MATLAB kalba galima rasti čia.

Algorithm 1 Dalijimo pusiau metodas funkcijai $f(x)$

Require: $f(x), l \in D_f, r \in D_f, l < r$
Ensure: $x_{min}, \min_{x \in [l;r]} f$
  $l \leftarrow 0$
  $r \leftarrow 10$
  $L \leftarrow r - l$
  $x_m \leftarrow \frac{l+r}{2}$
  while $L > 10^{-4}$ **do**
    $x_1 \leftarrow l + \frac{L}{4}$
    $x_2 \leftarrow r - \frac{L}{4}$
    $y_1 \leftarrow f(x_1)$
    $y_2 \leftarrow f(x_2)$
    $y_m \leftarrow f(x_m)$
    if $y_1 < y_m$ **then**
      $r \leftarrow x_m$
      $x_m \leftarrow x_1$
    else if $y_2 < y_m$ **then**
      $l \leftarrow x_m$
      $x_m \leftarrow x_2$
    else
      $l \leftarrow x_1$
      $r \leftarrow x_2$
    end if
    $L \leftarrow r - l$
  end while
  return $x_m, f(x_m)$

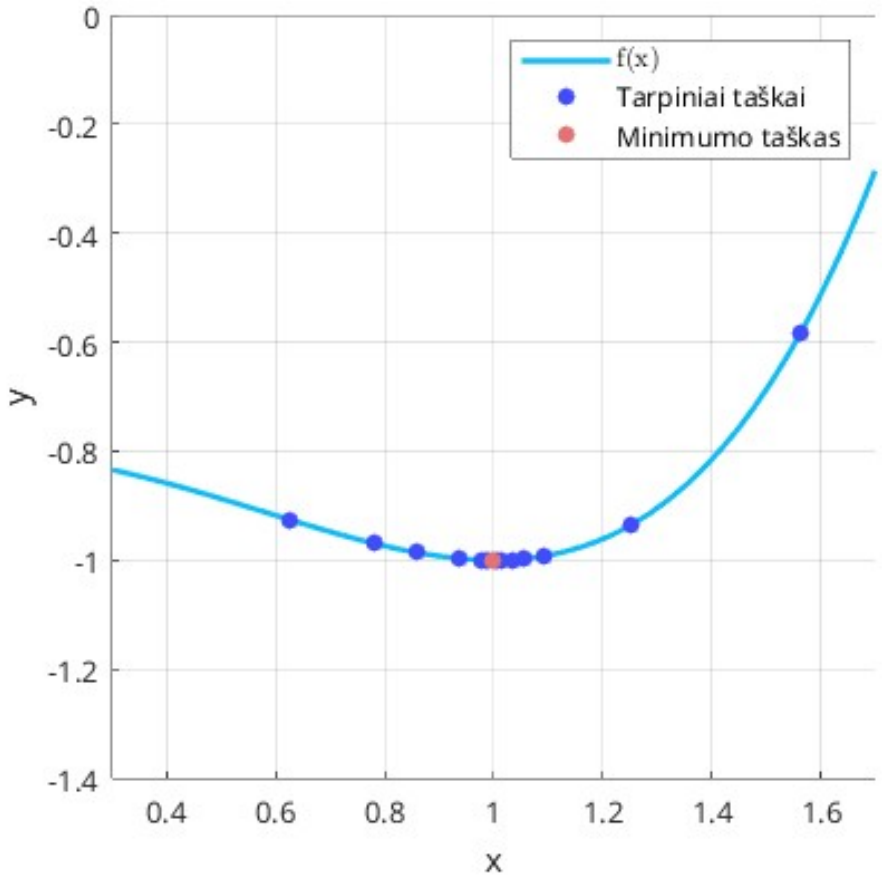
Galutinis rezultatas yra paskutinis apskaičiuotas x_m. Iš žemiau pateiktos lentelės [1] matome, kad šiuo atveju tai būtų $x_m = 0.999985$.

Toje pačioje lentelėje [1], galima pamatyti, kad algoritmas sustoja po 17 pilnų iteracijų, tačiau sustojimo sąlyga yra įdėta taip, kad dar yra įvykdoma pusė iteracijos x_m apskaičiuoti. Dėl šios priežasties bus laikoma, kad šis metodas reikalauja 18 iteracijų minimumo taškui rasti.

	l	x_1	x_m	x_2	r	L
1 iteracija	0.000000	2.500000	5.000000	7.500000	10.000000	10.000000
2 iteracija	0.000000	1.250000	2.500000	3.750000	5.000000	5.000000
3 iteracija	0.000000	0.625000	1.250000	1.875000	2.500000	2.500000
4 iteracija	0.625000	0.937500	1.250000	1.562500	1.875000	1.250000
5 iteracija	0.625000	0.781250	0.937500	1.093750	1.250000	0.625000
6 iteracija	0.781250	0.859375	0.937500	1.015625	1.093750	0.312500
7 iteracija	0.937500	0.976563	1.015625	1.054688	1.093750	0.156250
8 iteracija	0.976563	0.996094	1.015625	1.035156	1.054688	0.078125
9 iteracija	0.976563	0.986328	0.996094	1.005859	1.015625	0.039063
10 iteracija	0.986328	0.991211	0.996094	1.000977	1.005859	0.019531
11 iteracija	0.996094	0.998535	1.000977	1.003418	1.005859	0.009766
12 iteracija	0.998535	0.999756	1.000977	1.002197	1.003418	0.004883
13 iteracija	0.998535	0.999146	0.999756	1.000366	1.000977	0.002441
14 iteracija	0.999146	0.999451	0.999756	1.000061	1.000366	0.001221
15 iteracija	0.999756	0.999908	1.000061	1.000214	1.000366	0.000610
16 iteracija	0.999908	0.999985	1.000061	1.000137	1.000214	0.000305
17 iteracija	0.999908	0.999947	0.999985	1.000023	1.000061	0.000153
18 iteracija	–	–	0.999985	–	–	–

1 lentelė. Dalijimo pusiau metodu gautos iteracijos

Dalis šių parinktų taškų yra atvaizduota žemiau pateiktame paveikslėlyje [2]. Matome, kad parenkami taškai vis artėja prie minimumo taško $x = 1$.



2 pav. Dalijimo pusiau metodu pasirinkti tarpiniai taškai ir minimumo taškas

Neskaičiuojant iškvietimų, reikalingų grafikui pateikti, dalijimo pusiau metodo algoritmui bendrai prireikė 35 tikslo funkcijos iškvietimo.

2. Auksinio pjūvio metodas

Žemiau pateiktas algoritmo pseudokodas iliustruoja auksinio pjūvio metodo veikimo principą. Šio algoritmo įgyvendinimą MATLAB kalba galima rasti čia.

Algorithm 2 Auksinio pjūvio metodas funkcijai $f(x)$

Require: $f(x), l \in D_f, r \in D_f, l < r$
Ensure: $x_{min}, \min_{x\in[l;r]} f$
  $L \leftarrow r - l$
  $x_1 \leftarrow r - \tau \cdot L$ ▷ Čia ir toliau τ - Fibonačio skaičius ($\tau = 0{,}61803\ldots$).
  $x_2 \leftarrow l + \tau \cdot L$
  while $L > 10^{-4}$ **do**
    $x_1 \leftarrow l + \frac{L}{4}$
    $x_2 \leftarrow r - \frac{L}{4}$
    $y_1 \leftarrow f(x_1)$
    $y_2 \leftarrow f(x_2)$
    if $y_2 < y_1$ **then**
      $l \leftarrow x_1$
      $L \leftarrow r - l$
      $x_1 \leftarrow x_2$
      $x_2 \leftarrow l + \tau \cdot L$
    else
      $r \leftarrow x_2$
      $L \leftarrow r - l$
      $x_2 \leftarrow x_1$
      $x_1 \leftarrow r - \tau \cdot L$
    end if
  end while
  $x_m \leftarrow \frac{x_1+x_2}{2}$
  return $x_m, f(x_m)$

Šiame metode galutinis rezultatas yra skaičiuojamas pagal formulę

$$\frac{x_1 + x_2}{2}, \tag{3}$$

kur x_1 ir x_2 yra paskutinės gautos x_1 ir x_2 reikšmės. Iš žemiau pateiktos lentelės [2] matome, kad šiuo atveju tai būtų

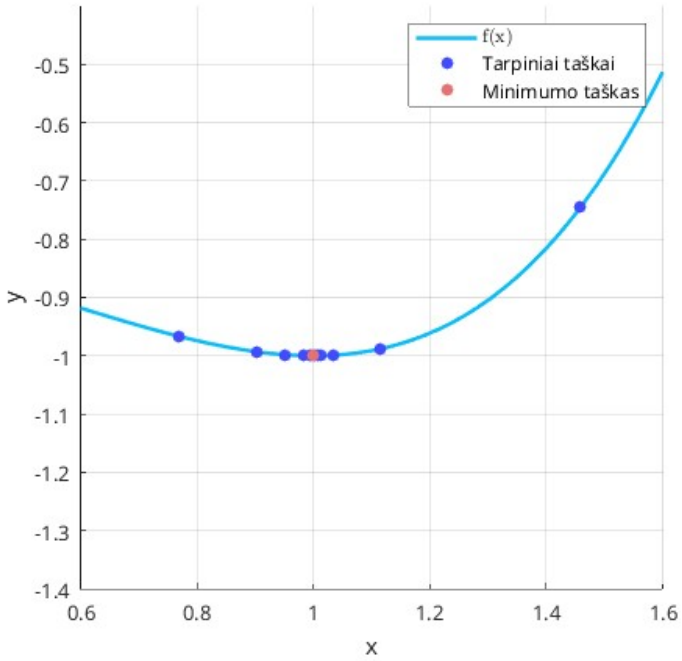
$$\frac{x_1 + x_2}{2} \approx \frac{1.0000001 + 1.000023}{2} \approx 1.000012 \tag{4}$$

Taip pat toje pačioje lentelėje [2] matome, kiekvienos iteracijos rezultatus bei tai, kad jai prireikė 25 iteracijų darbui pabaigti.

	l	x_1	x_2	r	L
1 iteracija	0.000000	3.819660	6.180340	10.000000	10.000000
2 iteracija	0.000000	2.360680	3.819660	6.180340	6.180340
3 iteracija	0.000000	1.458980	2.360680	3.819660	3.819660
4 iteracija	0.000000	0.901699	1.458980	2.360680	2.360680
5 iteracija	0.000000	0.557281	0.901699	1.458980	1.458980
6 iteracija	0.557281	0.901699	1.114562	1.458980	0.901699
7 iteracija	0.557281	0.770143	0.901699	1.114562	0.557281
8 iteracija	0.770143	0.901699	0.983006	1.114562	0.344419
9 iteracija	0.901699	0.983006	1.033256	1.114562	0.212862
10 iteracija	0.901699	0.951949	0.983006	1.033256	0.131556
11 iteracija	0.951949	0.983006	1.002199	1.033256	0.081306
12 iteracija	0.983006	1.002199	1.014062	1.033256	0.050250
13 iteracija	0.983006	0.994868	1.002199	1.014062	0.031056
14 iteracija	0.994868	1.002199	1.006730	1.014062	0.019194
15 iteracija	0.994868	0.999399	1.002199	1.006730	0.011862
16 iteracija	0.994868	0.997668	0.999399	1.002199	0.007331
17 iteracija	0.997668	0.999399	1.000469	1.002199	0.004531
18 iteracija	0.999399	1.000469	1.001130	1.002199	0.002800
19 iteracija	0.999399	1.000060	1.000469	1.001130	0.001731
20 iteracija	0.999399	0.999808	1.000060	1.000469	0.001070
21 iteracija	0.999808	1.000060	1.000216	1.000469	0.000661
22 iteracija	0.999808	0.999964	1.000060	1.000216	0.000409
23 iteracija	0.999808	0.999904	0.999964	1.000060	0.000253
24 iteracija	0.999904	0.999964	1.000001	1.000060	0.000156
25 iteracija	0.999964	1.000001	1.000023	1.000060	0.000096

2 lentelė. Auksinio pjūvio metodu gautos iteracijos

Dalį pasirinktų taškų galima pamatyti žemiau pateiktame paveikslėlyje [3]. Matome, kad su kiekviena iteracija pasirinkti taškai yra vis arčiau minimumo taško.



3 pav. Auksinio pjūvio metodu pasirinkti tarpiniai taškai ir minimumo taškas

Neskaičiuojant iškvietimų, reikalingų grafikui pateikti, auksinio pjūvio metodo algoritmui bendrai prireikė 26 tikslo funkcijos iškvietimų.

3. Niutono metodas

Šiame skyriuje pateikiamas metodas skiriasi nuo aukščiau paminėtų metodų tuo, kad jis veikia ne intervalų atmetimo principu, bet iki norimo tikslumo iteratyviai pritaikant formulę

$$x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)} \tag{5}$$

kur

$$f'(x) = \frac{4}{5}x(x^2 - 1) \tag{6}$$

$$f''(x) = \frac{4}{5}(3x^2 - 1) \tag{7}$$

Šio metodo kontekste funkcijos iškvietimu laikysime išvestinių funkcijų iškvietimus.

Žemiau pateiktas algoritmo pseudokodas iliustruoja Niutono metodo veikimo principą. Šio algoritmo įgyvendinimą MATLAB kalba galima rasti čia.

Algorithm 3 Niutono metodas $f(x)$

Require: $x_0 \in R$
Ensure: $x_{min}, \min f$
  $p \leftarrow x_0$ ▷ Čia p ir n atitinka formulės x_i ir x_{i+1} reikšmes.
  $n \leftarrow p - \frac{f'(p)}{f''(p)}$
  while $\mid p - n \mid > 10^{-4}$ **do**
    $p \leftarrow n$
    $n \leftarrow p - \frac{f'(p)}{f''(p)}$
  end while
  return $n, f(n)$

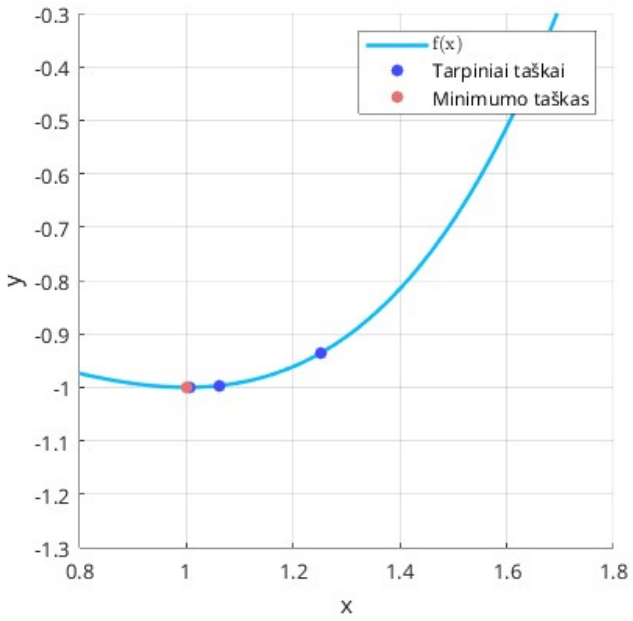
Šio metodo algoritmui prireikė mažiausiai - tik 9 iteracijų. Galutinis rezultatas - paskutinis suskaičiuotas x_{i+1}.

Žemiau esančioje lentelėje [3] yra pateikti kiekvienos iteracijos rezultatai.

	x_i
1 iteracija	5.000000
2 iteracija	3.378378
3 iteracija	2.320009
4 iteracija	1.648781
5 iteracija	1.252803
6 iteracija	1.060412
7 iteracija	1.004799
8 iteracija	1.000034
9 iteracija	1.000000

3 lentelė. Niutono metodu gautos iteracijos

Grafinis paskutinių keturių taškų atvaizdavimas matomas žemiau pateiktame paveikslėlyje [4]. Šiame paveikslėlyje akivaizdžiai matosi, kad su kiekviena iteracija metodas artėja prie $x_i = 1$.



4 pav. Niutono metodo paskutinių 4 iteracijų taškai

Neskaičiuojant iškvietimų, reikalingų grafikui pateikti, Niutono metodo algoritmui bendrai prireikė 16 funkcijos skaičiavimų.

Išvados

Žemiau pateiktoje lentelėje [4] galima pamatyti, kaip skiriasi metodų efektyvumas, bei jų gauti rezultatai.

Metodas	Dalijimo pusiau	Auksinio pjūvio	Niutono
Iteracijų sk.	18	25	9
Funkcijų skaičiavimo sk.	35	26	16
Gautas minimumo taškas	0.999985	1.000012	1.000000
Gautas minimumas	-0.999999	-0.999999	-1.000000

4 lentelė. Metodų palyginimo lentelė

Lyginant metodus pagal iteracijų skaičių, su duotos funkcijos minimumo radimu geriausiai susitvarko Niutono metodas. Dalijimo pusiau metodas yra antras pagal efektyvumą, atlikdamas darbą apytiksliai 2 kartus ilgiau. Paskutinėje vietoje pagal iteracijų skaičių yra auksinio pjūvio metodas, kuriam prireikia 3 kartus daugiau iteracijų negu Niutono metodui.

Lyginant metodus pagal funkcijų skaičiavimo kiekį, situacija yra kiek kitokia. Pirmoje vietoje taip pat yra Niutono metodas, tačiau vėlesnėse vietose rezultatai skiriasi nuo aukščiau aptartų. Antroje vietoje pagal funkcijų skaičiavimo kiekį yra auksinio pjūvio metodas, o trečioje - dalijimo pusiau metodas.

Kadangi funkcijų efektyvumas yra vertinamas pagal tikslo funkcijos iškvietimų skaičių, galima būtų teigti, kad Niutono metodas vienareikšmiškai yra efektyvesnis už kitus. Tačiau prisiminus formulę, naudojamą Niutono metode, matome, kad kviečiama ne pati tikslo funkcija, o jos pirmos ir antros eilės išvestinės. Taigi vienareikšmiškai vertinti šių metodų negalima, bet tarus, kad išvestinių skaičiavimas užima tiek pat laiko ir resursų, kiek ir tikslo funkcijos skaičiavimas, matome, kad šiai funkcijai skaičiuoti efektyviausias yra Niutono metodas.

Teigti, kad kažkuris metodas apskritai yra geresnis už kitus, negalima, nes metodai šio laboratorinio ribose metodai buvo pritaikyti tik vienai funkcijai ir rezultatai gali skirtis pritaikius kitoms.

Taip pat vertėtų atkreipti dėmesį ir į funkcijų tikslumą. Dalijimo pusiau ir auksinio pjūvio metodams buvo pateikta sąlyga, kad metodas turi dirbti iki tol, kol intervalo dydis yra didesnis 10^{-4}. Niutono metodui sąlyga buvo veikti iki tol, kol žingsnio skirtumas nebus mažesnis už 10^{-4}. Matome, kad Niutono metodu gauti rezultatai yra artimiausi teoriškai apskaičiuotam rezultatui. Tolimiausi rezultatai, tuo tarpu, yra gaunami panaudojus dalijimo pusiau metodą.

Apibendrinus, galima teigti, kad atsižvelgus į tam tikras prielaidas, efektyviausias metodas pateiktai funkcijai minimizuoti yra Niutono metodas - tiek iteracijų, tiek funkcijų skaičiavimo kiekio atžvilgiais.