VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ STUDIJŲ PROGRAMA

Lab 1 (vienmatis optimizavimas)

Atliko: 4 kurso 1 grupės studentas
Ignas Šileika

2024 m. rugsėjo 22 d.

Turinys

1. ĮVADAS 3
2. DALIJIMO PUSIAU METODAS 3
3. AUKSINIO PJŪVIO METODAS 7
4. NIUTONO METODAS 9
5. IŠVADOS 10

1. Įvadas

Užduoties tikslas suprogramuoti vienmačio optimizavimo intervalo dalijimo pusiau, auksinio pjūvio ir Niutono metodo algoritmus, bei palyginti ju efektivumą su funkcija: $f(x) = \frac{(x^2-3)^2}{2} - 1$

Darbas buvo atliktas naudojant python programavimo kalbą su matplotlib biblioteka grafikam nupiešti.

2. Dalijimo pusiau metodas

```
4
5  def bisection ( f, rangeMin, rangeMax, tol = 1e-4 ):
6
7      x = np.linspace ( -1, 5, 1000 )
8      y = f ( x )
9
10     plt.title ( 'bisection' )
11     plt.plot ( x, y )
12
13     acc = abs ( rangeMax - rangeMin )
14     iterations = 0
15
16     mid = ( rangeMin + rangeMax ) / 2
17     fM = f ( mid )
18
19     while acc > tol:
20
21         left = ( rangeMin + mid ) / 2
22         right = ( mid + rangeMax ) / 2
23
24         fL = f ( left )
25         fR = f ( right )
26
27         if ( fL < fM ):
28             rangeMax = mid
29             mid = left
30             fM = fL
31         elif ( fR < fM ):
32             rangeMin = mid
33             mid = right
34             fM = fR
35         else:
36             rangeMin = left
37             rangeMax = right
38
39         acc = abs ( rangeMax - rangeMin )
40
41         plt.plot ( mid, fM, 'o' )
42         print ( 'mid point for iteration ' + str ( iterations ) + ': x: ' + str ( mid ) + ' y: ' + str ( fM ) )
43         print ( 'Left: ' + str ( rangeMin ) + ' right: ' + str ( rangeMax ) )
44         print ( 'acc: ' + str (acc) + '/r/n' )
45         iterations += 1
46
47     x_min = ( rangeMin + rangeMax ) / 2
48     return x_min, f ( x_min ), iterations, ( iterations * 2 ) + 1
49
```

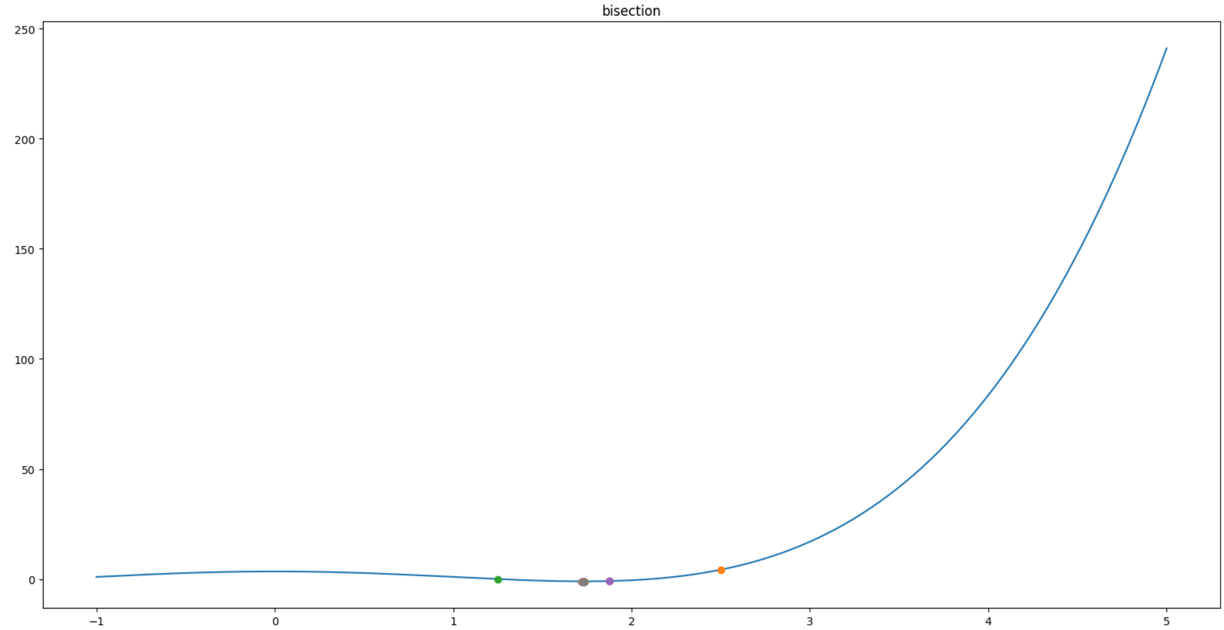
```
51
52 def f ( x ):
53     return ( ( ( ( x ** 2 ) - 3 ) ** 2 ) / 2 ) - 1
54
55 rez = bisection ( f, 0, 10 )
56
57 print ( rez )
58
59 plt.show ()
~
```

5-48 eilutėse aprašytas pats dalijimo pusiau metodas kuris kaip parametrus gauna: funkcija, kairiįį intervalo kraštą, dešiniįį intervalo kraštą, bei tolerancijos lygį su standartine reikšme $10 * 1^{-4}$. 7-11 elutės naudojamos grafikui spauzdinti, 13 eilutė apskaičiuoja pradinį tikslumą, 16-17 eilutės apskaičiuoja pradinį vidurio tašką, bei funkcijos reikšmę tame taške. 19–45 eilutėse vygdomos kol tikslumas nepasiekia tolerancijos. 21–25 eilutės apskaičiuoja kairįįį ir dešinįįį tašką, bei šunkcijos rekšmes, 27-37 eilutėse palyginamos kairiosios ir dešiniosios funkcijos rekšmes su vidurio funkcijos reikšme ir nukerpami nereikalingi intervalo gabaliukai. 39 eiltėje apskaičiuojama nauja tikslumą, 41–44 eilutės pažymi vidurio tašką grafike ir atspauzina reikšmes į ekraną. Galiausiai funkcija gražina rastą x reikšmę, funkcijos reikšmę taške x, iteracijų skaičių, bei kiek kartų buvo kviesta minimizuojama funkcija. 52–53 eilutėse aprašyta funkcija kurę minimizuojame.

```
mid point for iteration 0: x: 2.5 y: 4.28125
Left: 0 right: 5.0
acc: 5.0

mid point for iteration 1: x: 1.25 y: 0.033203125
Left: 0 right: 2.5
acc: 2.5

mid point for iteration 2: x: 1.875 y: -0.8670654296875
Left: 1.25 right: 2.5
acc: 1.25

mid point for iteration 3: x: 1.875 y: -0.8670654296875
Left: 1.5625 right: 2.1875
acc: 0.625

mid point for iteration 4: x: 1.71875 y: -0.9989466667175293
Left: 1.5625 right: 1.875
acc: 0.3125

mid point for iteration 5: x: 1.71875 y: -0.9989466667175293
Left: 1.640625 right: 1.796875
acc: 0.15625

mid point for iteration 6: x: 1.71875 y: -0.9989466667175293
Left: 1.6796875 right: 1.7578125
acc: 0.078125

mid point for iteration 7: x: 1.73828125 y: -0.9997662509558722
Left: 1.71875 right: 1.7578125
acc: 0.0390625

mid point for iteration 8: x: 1.728515625 y: -0.9999251678746077
Left: 1.71875 right: 1.73828125
acc: 0.01953125

mid point for iteration 9: x: 1.7333984375 y: -0.9999890948815846
Left: 1.728515625 right: 1.73828125
acc: 0.009765625

mid point for iteration 10: x: 1.73095703125 y: -0.9999928264523703
Left: 1.728515625 right: 1.7333984375
acc: 0.0048828125
```

```
mid point for iteration 10: x: 1.73095703125 y: -0.9999928264523703
Left: 1.728515625 right: 1.7333984375
acc: 0.0048828125

mid point for iteration 11: x: 1.732177734375 y: -0.9999999033304316
Left: 1.73095703125 right: 1.7333984375
acc: 0.00244140625

mid point for iteration 12: x: 1.732177734375 y: -0.9999999033304316
Left: 1.7315673828125 right: 1.7327880859375
acc: 0.001220703125

mid point for iteration 13: x: 1.732177734375 y: -0.9999999033304316
Left: 1.73187255859375 right: 1.73248291015625
acc: 0.0006103515625

mid point for iteration 14: x: 1.732025146484375 y: -0.9999999960491109
Left: 1.73187255859375 right: 1.732177734375
acc: 0.00030517578125

mid point for iteration 15: x: 1.732025146484375 y: -0.9999999960491109
Left: 1.7319488525390625 right: 1.7321014404296875
acc: 0.000152587890625

mid point for iteration 16: x: 1.7320632934570312 y: -0.9999999990646088
Left: 1.732025146484375 right: 1.7321014404296875
acc: 7.62939453125e-05

(1.7320632934570312, -0.9999999990646088, 17, 35)
```

Kodas į konsole kiekvienai iteracijai atspauzdina vidurio tašką, kairiji ir dešinijį intervalo kraštą, bei tikslumą. Baigus optimizuoti, atspauzina funkcijos gražintą rezultatą.



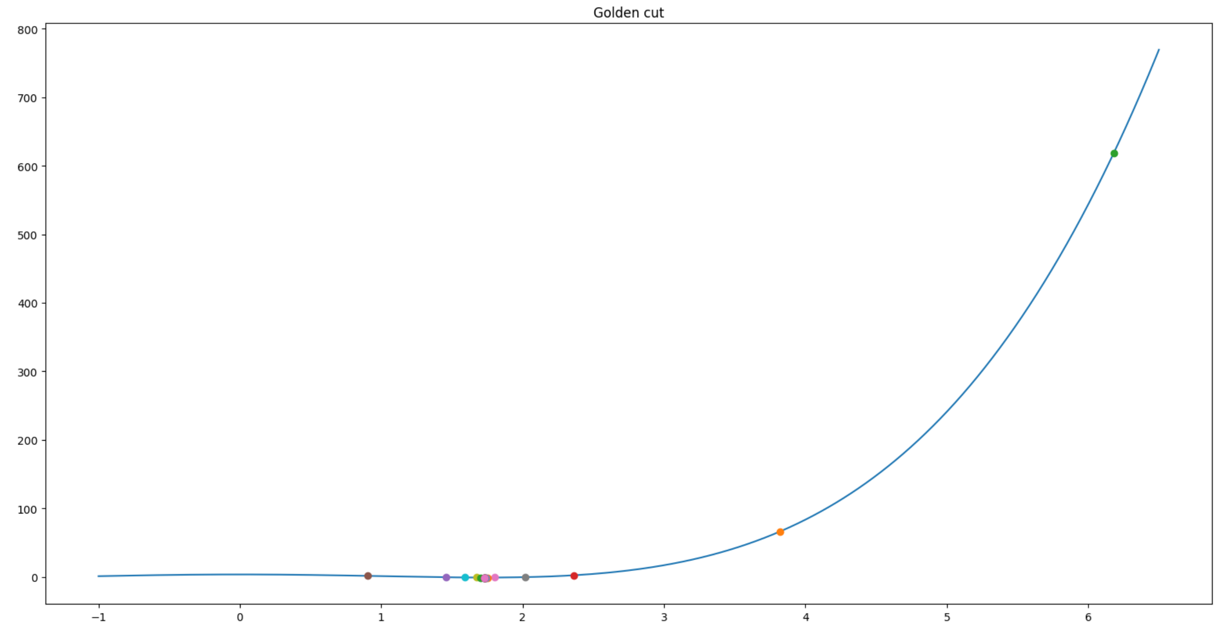
3. auksinio pjūvio metodas

```
 4
 5 def goldenCut ( f, l, r, tol = 1e-4 ):
 6
 7     t = 0.618
 8
 9     x = np.linspace ( -1, 6.5, 1000 )
10     y = f ( x )
11
12     plt.title ( 'Golden cut' )
13     plt.plot ( x, y )
14
15     iteration = 0
16
17     L = r - l
18
19     x1 = r - ( t * L )
20     x2 = l + ( t * L )
21
22     fx1 = f ( x1 )
23     fx2 = f ( x2 )
24
25     plt.plot ( x1, fx1, 'o' )
26     plt.plot ( x2, fx2, 'o' )
27
28     print ( 'iteration ' + str ( iteration ) + ': x1: ' + str ( x1 ) + ' x2: ' + str ( x2 ) )
29
30     while L > tol:
31
32         if ( fx2 < fx1 ):
33             l = x1
34             L = r - l
35
36             x1 = x2
37             fx1 = fx2
38
39             x2 = l + ( t * L )
40             fx2 = f ( x2 )
41
42             plt.plot ( x2, fx2, 'o' )
43         else:
44             r = x2
45             L = r - l
46
47             x2 = x1
48             fx2 = fx1
49
50             x1 = r - ( t * L )
51             fx1 = f ( x1 )
52
53             plt.plot ( x1, fx1, 'o' )
54
55         iteration += 1
56
57         print ( 'iteration ' + str ( iteration ) + ': x1: ' + str ( x1 ) + ' x2: ' + str ( x2 ) + ' L: ' + str(L) )
58
59     xMin = ( r + l ) / 2
60     return xMin, f ( xMin ), iteration, iteration + 2
61
62 def f ( x ):
63     return ( ( ( ( x ** 2 ) - 3 ) ** 2 ) / 2 ) - 1
64
65 rez = goldenCut ( f, 0, 10 )
66
67 print ( rez )
68
69 plt.show ()
```

5-60 eilutėse aprašytas auksinio pjūvio metodas kuris kaip parametrus gauna: funkcija, kairijį intervalo kraštą, dešinijį intervalo kraštą, bei tolerancijos lygį su standartine reikšme $10 * 1^{-4}$. 9-13 elutės naudojamos grafikui spauzdinti, 17-23 eilutėse apskaičiuojamas pradinis intervalo ilgis, x1 ir x2 taškai, bei funkciju reikšmės tuose taškuose. 25-28 eilutės nupaišo taškus grafikei, bei atspauzdina reiksmes į ekraną. 30-60 eilutėse kartojasi kol intervalo ilgis nesumažėja iki tolerancijos ribos. 32-57 eilutėse paliginama kuriame taške x1 ar x2 funkcijos reikšmė mažesnė, ir pagal tai nukerpa reikiama intervalo dalį. Galiausiai funkcija gražina rastą x reikšmę, funkcijos reikšmę taške x, iteracijų skaičių, bei kiek kartų buvo kviesta minimizuojama funkcija. 62-63 52-53 eilutėse aprašyta funkcija kurę minimizuojame.

```
iteration 0: x1: 3.8200000000000003 x2: 6.18
iteration 1: x1: 2.36076 x2: 3.8200000000000003 L: 6.18
iteration 2: x1: 1.4592400000000003 x2: 2.36076 L: 3.8200000000000003
iteration 3: x1: 0.90181032 x2: 1.4592400000000003 L: 2.36076
iteration 4: x1: 1.4592400000000003 x2: 1.80344122224 L: 1.45894968
iteration 5: x1: 1.80344122224 x2: 2.01637936 L: 0.9015199999999997
iteration 6: x1: 1.6720672355200001 x2: 1.80344122224 L: 0.5571393599999999
iteration 7: x1: 1.5907248668956802 x2: 1.6720672355200001 L: 0.3442012222399997
iteration 8: x1: 1.6720672355200001 x2: 1.72218357449847 L: 0.21271635534431987
iteration 9: x1: 1.72218357449847 x2: 1.75325635931296 L: 0.1313739867199999
iteration 10: x1: 1.7030814808089108 x2: 1.72218357449847 L: 0.08118912379295984
iteration 11: x1: 1.72218357449847 x2: 1.7340895557244131 L: 0.05017487850404922 5
iteration 12: x1: 1.7340895557244131 x2: 1.7413865555138248 L: 0.03107278481449005
iteration 13: x1: 1.7295191132463354 x2: 1.7340895557244131 L: 0.01920298101535489 2
iteration 14: x1: 1.7340895557244131 x2: 1.7368531925676438 L: 0.01186744226748937
iteration 15: x1: 1.7323207315470752 x2: 1.7340895557244131 L: 0.00733407932130836 2
iteration 16: x1: 1.731265022272961 x2: 1.7323207315470752 L: 0.004570442478077696 4
iteration 17: x1: 1.7323207315470752 x2: 1.7330105839459584 L: 0.002824533451452060 4
iteration 18: x1: 1.731931826832046 x2: 1.7323207315470752 L: 0.001745561672997331 1
iteration 19: x1: 1.7316683032156728 x2: 1.731931826832046 L: 0.001055709274114136
iteration 20: x1: 1.731931826832046 x2: 1.7320715039244794 L: 0.000652428331402443 7
iteration 21: x1: 1.7320715039244794 x2: 1.732172169945934 L: 0.000388904715029170 23
iteration 22: x1: 1.7320236379015512 x2: 1.7320715039244794 L: 0.000240343113887986 35
iteration 23: x1: 1.7320715039244794 x2: 1.7321154307049798 L: 0.000148532044382809 76
iteration 24: x1: 1.732058702752461 x2: 1.7320715039244794 L: 9.179280342852536e-05
(1.7320695343032655, -0.9999999978958337, 24, 26)
```

Kodas į konsole kiekvienai iteracijai atspauzdina x1 ir x2 reikšmes, bei intervalo ilgį. Baigus optimizuoti, atspauzina funkcijos gražintą rezultatą.



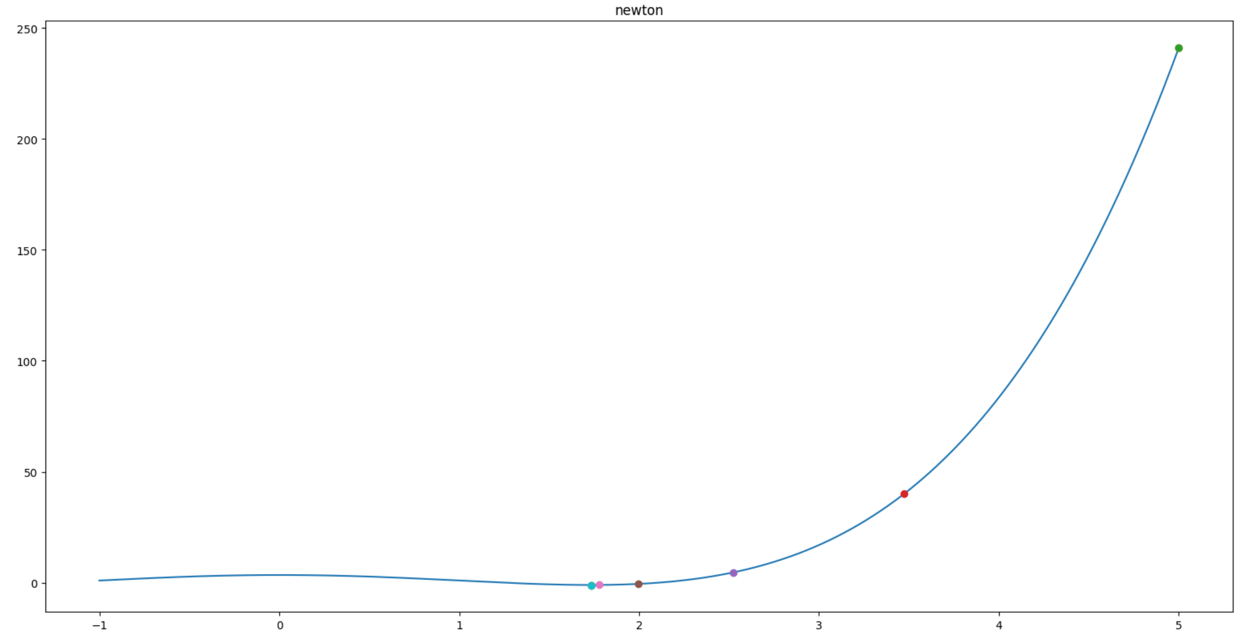
4. Niutono metodas

```
5 def newton ( f, fPrime, fDoublePrime, xi = 5, tol = 1e-4 ):
6
7     x = np.linspace ( -1, 5, 1000 )
8     y = f ( x )
9
10    plt.title ( 'newton' )
11    plt.plot ( x, y )
12
13    plt.plot ( xi, f ( xi ), 'o' )
14
15    iteration = 0
16
17    while 1:
18
19        fPrimeX = fPrime ( xi )
20        fDoublePrimeX = fDoublePrime ( xi )
21
22        if fDoublePrime == 0:
23            print ("Double prime = 0")
24            break
25
26        xi1 = xi - ( fPrimeX / fDoublePrimeX )
27
28        plt.plot ( xi, f ( xi ), 'o' )
29        print ( "iteration: " + str ( iteration ) + " xi: " + str ( xi ) + ' acc: ' + str ( abs ( xi1 - xi ) ) )
30        iteration += 1
31
32        if abs ( xi1 - xi ) < tol:
33            plt.plot ( xi1, f ( xi1 ), 'o' )
34            return xi1, f ( xi1 ), iteration, iteration * 2
35
36        xi = xi1
37
38
39 def f ( x ):
40     return ( ( ( ( x ** 2 ) - 3 ) ** 2 ) / 2 ) - 1
41
42 def fPrime ( x ):
43     return ( ( ( x ** 2 ) - 3 ) * 2 * x )
44
45 def fDoublePrime ( x ):
46     return ( 6 * ( x ** 2 ) ) - 6
47
48 rez = newton ( f, fPrime, fDoublePrime )
49
50 print ( rez )
51
52 plt.show ()
```

5-36 eilutėse aprašytas Niutono metodas kuris kaip parametrus gauna: funkcija, funkcijos pirmają išvestinę, funkcijos antrają išvestinę, pradinę x reikšmę su standartine 5, bei tolerancijos lygį su standartine reikšme $10 * 1^{-4}$. 7-13 elutės naudojamos grafikui spauzdinti. 17-36 kartojasi kol skirtumas tarp naujos x reikšmęs ir senos tampa mažesnis nei tolerancija. 19-20 eilutės apskaičiuoja funkcijos išvestinių reikšmes taške x_i, 22-24 eilutės patikrina ar antrosios išvestinės reikšmė nelygi 0. 26 eilutė apskaičiuoja nauja x reikšmę, 28-29 pažimi x reikšmę grafike, bei atspauzdina reikšmes į ekraną. Galiausiai 32-34 eilutės patikrina ar nepasiektas norimas tikslumas ir jei pasiektas gražiną x reikšmę, funkcijos reikšmę, bei iteracijas ir kiek kartu buvo skaičiuojamos išvestinės. 39-46 eilutės aprašo nagrinėjamą funkciją ir jos pirmą ir antrą išvestines.

```
iteration: 0 xi: 5 acc: 1.5277777777777777
iteration: 1 xi: 3.4722222222222223 acc: 0.9480417852425669
iteration: 2 xi: 2.5241804369796554 acc: 0.5281120736632223
iteration: 3 xi: 1.9960683633164331 acc: 0.21945015091224707
iteration: 4 xi: 1.776618212404186 acc: 0.04294462989568837
iteration: 5 xi: 1.7336735825084977 acc: 0.0016204993242796562
iteration: 6 xi: 1.732053083184218 acc: 2.2756108561949873e-06
(1.7320508075733618, -1.0, 7, 14)
```

Kodas į koncolę kiekvieną iteracija atspauzdina x reikšmę, bei tikslumą. Baigus optimizuoti, atspauzina funkcijos gražintą rezultatą.



5. Išvados

Visi trys metodai funkciją minimizavo į ta pati $x \approx 1.732$ ir $f(x) \approx -1.0$. Tiksliausisi su duota funkciją tai atliko Niutono metodas.

Paga iteracijų skaičių, geriausiai pasirodė Niutono metodas su 7 iteracijomis, o blogiau auksinio pjūvio metodas su 24. Dalijimo pusiau metodas minimizavo iki norimo tikslumo per 17 iteracijų.

Lyginant kiek kartų buvo apskaičiuota minimizuojama funkcija ar Niutono atvėju išvestinė, vėl geriausiai pasirodė Niutono metodas su 14 skaičevimų. Tačiau skirtingai negu vertinant įteracijas prasčiauses metodas buvo dalijimo pusiau su 35 funkcijos skaičevimais, kai auksinio pjųvio skaičiavo tik 26 kartus.

Apibendrinus su duotąja funkcija $f(x) = \frac{(x^2-3)^2}{2} - 1$ geriausis veikė Niutono metodas kuris su mažiausiai įteracijų ir funkcijos skaičiavimų minimizavo funkciją iki norimo tikslumo.