VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS

Laboratorinis darbas 1

Vienmatis optimizavimas

Nikita Gainulin

VILNIUS 2024

Turinys

1 Tikslo funkcija

Visi šioje ataskaitoje minėti metodai bus pritaikyti šiai tikslo funkcijai:

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

Mano *Python* kode aprašyta taip:

```
def f(x):
    return ((x**2-5)**2)/7-1
```

2 Vienmačiai optimizavimo metodai ir jų algoritmų kodai

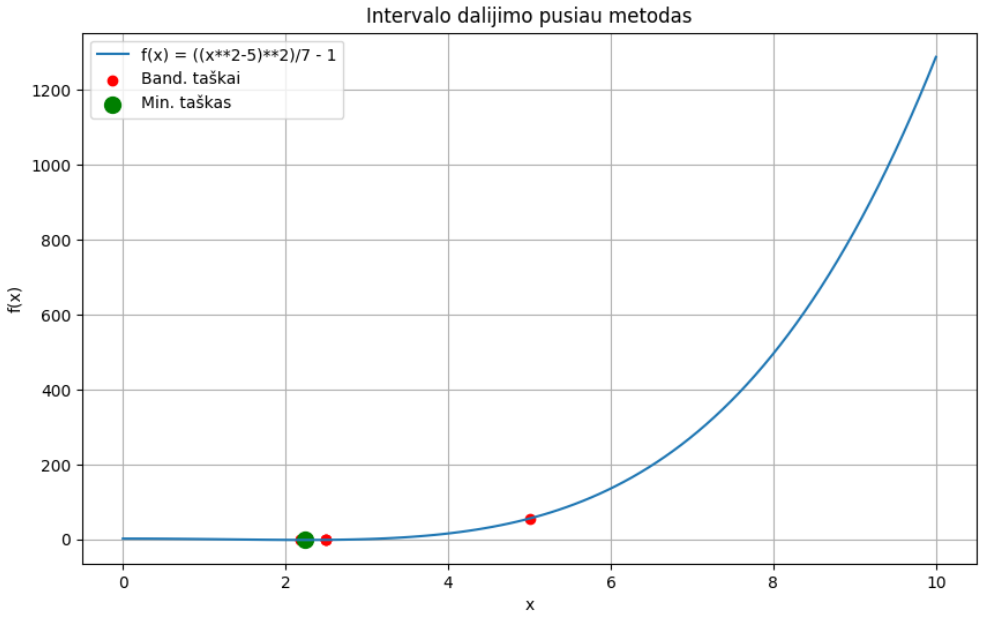
2.1 Intervalo dalijimo pusiau metodas

Pirmiausia reikia nustatyti intervalą, kuriame ieškosime minimumo taško. Minėto intervalo ribas įsiminsime kintamaisiais l (kairioji riba) ir r (dešinioji riba). Pradiniame intervale pasirenkam tris bandymo taškus x_1, x_2 ir x_m, taip, kad x_m būtų intervalo vidurio taškas, o x_1 ir x_2 tarp l ir vidurio taško bei r ir vidurio taško atitinkamai. Tada apskaičiuojame $f(x_m)$, $f(x_1)$ ir $f(x_2)$ reikšmes. Po to palyginame gautas reikšmes, t. y. kuriame taške funkcija įgyja didesnę reikšmę. Tas taškas ir nustato intervalą, kurį reikia atmesti. Kadangi ieškome minimumo taško, atmetamas atitinkamas intervalas, atitinkmai keičiami x_1, x_2 ir x_m taškai bei perskaičiuojami $f(x_m)$, $f(x_1)$ ir $f(x_2)$ reikšmės. Algoritmas tęsiamas tol, kol gaunamas pakankamai tikslus minimumo taškas, t.y. kol intervalo ilgis $r - l$ nėra mažiau už tam tikrą ε reikšmę.
Štai kaip tai atrodo mano *Python* kode:

```
def f(x):
    return ((x**2-5)**2)/7-1

def halfCut(f, l, r, eps=1e-4):
    iterNum = 0
    funcNum = 0

    xm = (l+r)/2
    fxm = f(xm)
    funcNum+=1

    xVal = [xm]
    yVal = [fxm]
    while(r-l) > eps:
        iterNum += 1
        L = r-l

        x1 = l+L/4
        x2 = r-L/4

        fx1 = f(x1)
        fx2 = f(x2)

        funcNum += 2

        if fx1 < fxm:
            r = xm
            xm = x1
            fxm = fx1
```

```
30              xVal.append(x1)
31              yVal.append(fx1)
32
33          elif fx2 < fxm:
34              l = xm
35              xm = x2
36              fxm = fx2
37              xVal.append(x2)
38              yVal.append(fx2)
39
40          else:
41              l = x1
42              r = x2
43              xVal.append(xm)
44              yVal.append(fxm)
45
46      xm = (l+r)/2
47      xVal.append(xm)
48      yVal.append(f(xm))
49      return xm, f(xm), iterNum, funcNum, xVal, yVal
```

Vizualizuojant mūsų tikslo funciją (1) šiuo metodu gauname štai tokį grafą:



1 pav.: Intervalo dalijimo pusiau metodo vizualizacija (1) tikslo funkcijai.

2.2 Auksinio pjūvio metodas

Šis metodas yra labai panašus į intervalo dalijimo pusiau metodą, tačiau per kiekvieną iteraciją naudojami tik 2, o ne 3 bandomieji taškai. Mums vis dar reikia pasirinkti intervalą ir įsiminti jo kairiąją l ir dešiniąją r ribas, tačiau šį kartą x_1 ir x_2 apskaičiuojami kitaip, naudojant *Fibonačio*

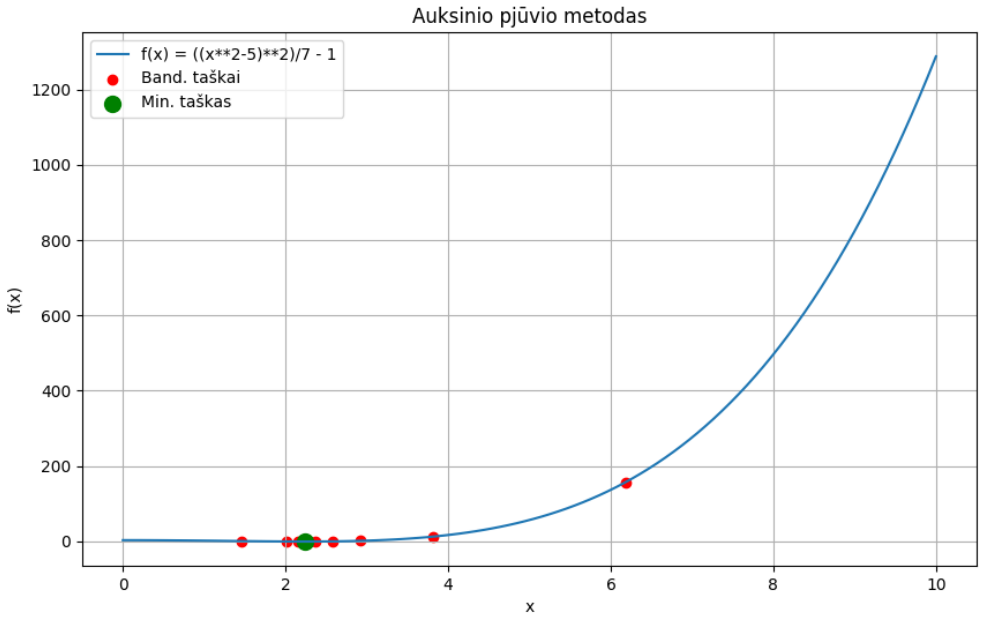
reikšmę, kuri apytiksliai lygi:

$$\tau = \frac{\sqrt{5}-1}{2} \approx 0.61803...$$

Apskaičiavę $x_1 = r - \tau(r-l)$ ir $x_2 = l + \tau(r-l)$ taškus, apskaičiuojame funkcijos reikšmes $f(x_1)$ ir $f(x_2)$ šiuose taškuose. Po to dar kartą palyginame funkcijos reikšmes ir pašaliname intervalus pagal didesnę vertę, perskaičiuojame taškus ir naujas funkcijos reikšmes ir kartojame tuos pačius veiksmus kiekvieną iteraciją, kol vėl gauname pakankamai tikslų minimumo tašką.
Algoritmo *Python* kodas:

```
def goldCut(f, l, r, eps=1e-4):
    tau = (math.sqrt(5)-1)/2
    print(tau)

    L = r-l
    x1 = l+(1-tau)*L
    x2 = l+tau*L
    fx1 = f(x1)
    fx2 = f(x2)

    iterNum = 0
    funcNum = 2

    xVal = [x1, x2]
    yVal = [fx1, fx2]
    while(r-l) > eps:
        iterNum += 1
        if fx1<fx2:
            r = x2
            x2 = x1
            fx2 = fx1
            L = r-l
            x1 = l+(1-tau)*L
            fx1 = f(x1)
            funcNum += 1
            xVal.append(x1)
            yVal.append(fx1)
        else:
            l = x1
            x1 = x2
            fx1 = fx2
            L = r-l
            x2 = l+tau*L
            fx2 = f(x2)
            funcNum += 1
            xVal.append(x2)
            yVal.append(fx2)

    xm = (l+r)/2
    xVal.append(xm)
    yVal.append(f(xm))
    return xm, f(xm), iterNum, funcNum, xVal, yVal
```

Vizualizuojant mūsų tikslo funciją (1) šiuo metodu gauname štai tokį grafą:



2 pav.: Auksinio pjūvio metodo vizualizacija (1) tikslo funkcijai.

2.3 Niutono metodas

Skirtingai nuo dviejų ankstesnių metodų, kurie remiasi konkretaus intervalo pjovimu, Niutono metodas remiasi vienu pradiniu tašku x_i ir Teiloro eilutėmis iki antros eilės. Pasirinkus pradinį tašką x_i, kitas taškas apskaičiuojamas pagal šią formulę:

$$x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)} \tag{2}$$

Tokiu būdu iteracija po iteracijos taškas artėja prie minimumo.
Vienas įdomus faktas, kurį pastebėjau, yra tai, kad daugelyje internetinių šaltinių kaip Niutono metodo pagrindas naudojama ši formulė:

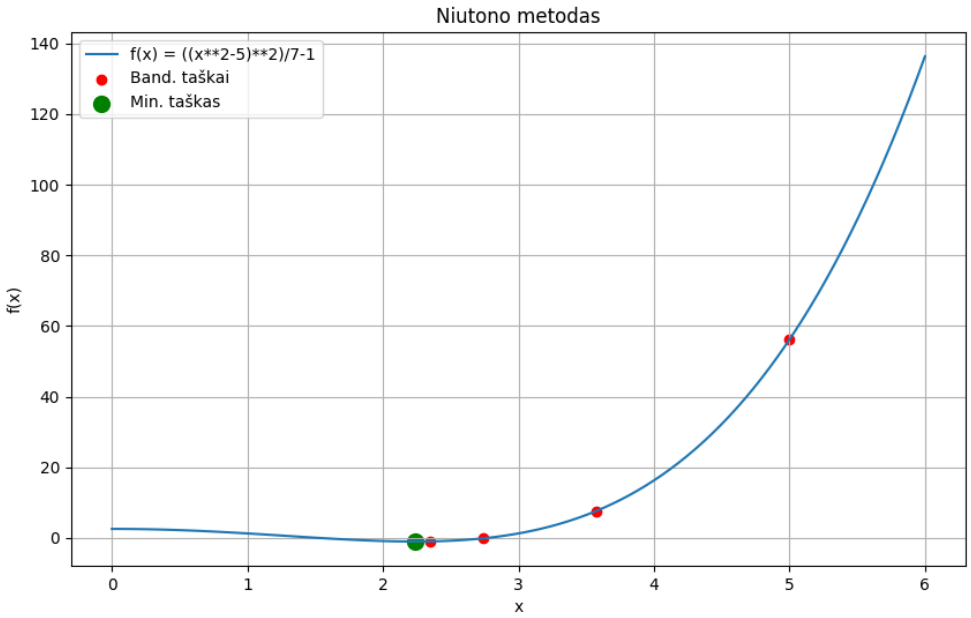
$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Nors ši formulė paprastai naudojama dažniau ir bendriau, formulė (2) yra tikslesnė optimizavimo kontekste, todėl ją ir naudojame vietoj šios bendrosios formulės.
Algoritmo *Python* kodas:

```
1 def newton(fSym, x0, eps=1e-4, maxIter=100):
2     x = sp.symbols('x')
3 
4     fIsv = sp.diff(fSym, x)
5     fIsv2 = sp.diff(fIsv, x)
6 
7     fSk = sp.lambdify(x, fSym)
```

```
    fIsvSk = sp.lambdify(x, fIsv)
    fIsv2Sk = sp.lambdify(x, fIsv2)

    xi = x0
    iterNum = 0
    funcNum = 0

    xVal = [xi]
    yVal = [fSk(xi)]

    while iterNum < maxIter:
        fIsv_xi = fIsvSk(xi)
        fIsv2_xi = fIsv2Sk(xi)
        funcNum += 2

        if abs(fIsv2_xi) < eps:
            break

        xiNext = xi-fIsv_xi/fIsv2_xi
        iterNum += 1

        xVal.append(xiNext)
        yVal.append(fSk(xiNext))

        if abs(xiNext-xi) < eps:
            break

        xi = xiNext

    return xi, fSk(xi), iterNum, funcNum, xVal, yVal

x = sp.symbols('x')
fSym = ((x**2-5)**2)/7-1
```

Vizualizuojant mūsų tikslo funciją (1) šiuo metodu gauname štai tokį grafą:



3 pav.: Niutono metodo vizualizacija (1) tikslo funkcijai.

3 Rezultatai ir jų palyginimai

3.1 Minimumo taškas ir funkcijos reikšmė tame taške

Kaip ir galima tikėtis, taikant visus metodus gaunami daugiau ar mažiau panašūs rezultatai, kai kalbama apie minimumo tašką:

	Intervalo dalijimo pusiau metodas	Auksinio pjūvio metodas	Niutono metodas
Minimumo taškas	2.236061	2.236057	2.236105
F-jos reikšmė	-0.999999	-0.999999	-0.999999

Matomas šiek tiek didesnis Niutono metodo nuokrypis nuo kitų metodų. Tačiau tai nebūtinai yra blogai, tiesiog rodo, kad Niutono metodo tikslumas priklauso nuo tokių parametrų kaip pradinis taškas ir funkcijos pobūdis.