

Vilniaus universitetas

Matematikos ir informatikos fakultetas

Programų sistemų studijų programa

Optimizavimo metodų pirmo laboratorinio darbo ataskaita

Ataskaitą tikrino: Prof. Dr. Pranas Katauskis

Ataskaitą parengė: Dominykas Daunoravičius

Vilnius

2022

**Įvadas**

**Laboratorinio darbo formulavimas:** Suprogramuoti vienmačio optimizavimo intervalo dalijimo pusiau, auksinio pjūvio ir Niutono metodo algoritmus. Minimizuoti *f*(*x*) = (*x*2−*3*)2/*9*−1 funkciją intervalo dalijimo pusiau ir auksinio pjūvio metodais intervale [0,10] iki tikslumo 10−4 bei Niutono metodu nuo *x*0 = 5 kol žingsnio ilgis didesnis už 10−4. Palyginti rezultatus: gauti sprendiniai, rastas funkcijos minimumo įvertis, atliktų žingsnių ir funkcijų skaičiavimų skaičius. Vizualizuoti tikslo funkciją ir bandymo taškus.

**Laboratorinio darbo tikslas:** Naudojantis intervalo dalijimo pusiau, auksinio pjūvio ir Niutono metodais surasti funkcijos minimumą, funkcijos reikšmę minimumo taške, surasti iteracijų skaičių bei skaičiuotų funkcijų kiekį, palyginti gautus rezultatus ir įvertinti, kuris metodas yra efektyviausias sprendžiant tokio tipo uždavinį.

**Darbo eiga**

Pirmąjame laboratoriniame darbe iš viso buvo suprogramuotos trys funkcijos, kurios skirtingais metodais minimizavo tikslo funkciją f(x) = (x2 - 3)2/9-1. Naudojantis intervalo dalijimo pusiau, auksinio pjūvio ir Niutono metodais visais metodais buvo surastas funkcijos minimumas, funkcijos reikšmė minimumo taške, iteracijų skaičius bei skaičiuotų funkcijų kiekis beieškant rezultato.

Pasirinkta programavimo kalba: Python Chart, line chart

Description automatically generated

1 pav. Tikslo funkcijos grafikas

1. **Intervalo dalijimo pusiau metodas**Intervalo dalijimo pusiau metodo metu pradiniame intervale pasirenkami trys tolygiai pasiskirstę bandymo taškai ir kiekvienos iteracijos metu yra atmetama pusė intervalo. Po kiekvienos iteracijos yra tikrinama ar pasiektas atitinkamas tikslumas, jei pasiekėme – baigiame skaičiavimus, jei ne – kartojame procesą, tol kol pasieksime.

Chart, line chart

Description automatically generated

2 pav. Intervalo dalijimo pusiau metodo grafiko ir intervalų taškų vaizdavimas

Grafike galima matyti kaip kiekvienos iteracijos metu intervalas yra dalijamas pusiau ir intervalai siaurėja. Šiame grafike pavaizduoti taškai tik po 1, 2, 3, 4 ir 17 iteracijų, jog grafike aiškiai matytųsi sprendinių artėjimas link minimumo.

Visi taškai, po kiekvienos iteracijos[[1]](#footnote-1):

[[0, 5.0], [0, 2.5], [1.25, 2.5], [1.5625, 2.1875], [1.5625, 1.875], [1.640625, 1.796875], [1.6796875, 1.7578125], [1.71875, 1.7578125], [1.71875, 1.73828125], [1.728515625, 1.73828125], [1.728515625, 1.7333984375], [1.73095703125, 1.7333984375], [1.7315673828125, 1.7327880859375], [1.73187255859375, 1.73248291015625], [1.73187255859375, 1.732177734375], [1.7319488525390625, 1.7321014404296875], [1.732025146484375, 1.7321014404296875]].

1. **Auksinio pjūvio metodas**

Auksinio pjūvio metodo intervale yra du bandymo taškai, kurie yra vienodai nutolę nuo vidurio, kiekvienos iteracijos metu skaičiuojama viena tikslo funkcijos reikšmė ir atmetama intervalo dalis, kurioje minimumas neegzistuoja. Procesas kartojamas, kol pasiekiamas nurodytas tikslumas.

Chart, line chart

Description automatically generated

3 pav. Auksinio pjūvio metodo grafiko ir intervalų taškų vaizdavimas

Grafike vaizduojama kaip po kiekvienos iteracijos taškai artėja sprendinio link. Taškai vaizduojami tik 1, 2, 3, 4 ir 24 iteracijų metu, kad matytųsi aiškus intervalo trumpėjimas ir artėjimas link minimumo.

Grafike vaizduojamas intervalo siaurėjimas kiekvienos iteracijos metu.

Visi taškai, po kiekvienos iteracijos[[2]](#footnote-2):

[[0, 6.1803], [0, 3.8197], [0, 2.360689191], [0.9017124502862701, 2.360689191], [1.459010809, 2.360689191], [1.459010809, 2.01627509942746], [1.459010809, 1.8034038453495767], [1.590558617094448, 1.8034038453495767], [1.671869050014577, 1.8034038453495767], [1.671869050014577, 1.7531614995754667], [1.70292032697335, 1.7531614995754667], [1.722103353512965, 1.7531614995754667], [1.722103353512965, 1.741298219523973], [1.7294352164831899, 1.741298219523973], [1.7294352164831899, 1.736766908252485], [1.7294352164831899, 1.7339708788766361], [1.7311677034476145, 1.7339708788766361], [1.7311677034476145, 1.7329001499580128], [1.7318294460411914, 1.7329001499580128], [1.7318294460411914, 1.7324911731829045], [1.7318294460411914, 1.7322357027883075], [1.7319846239308874, 1.7322357027883075], [1.7319846239308874, 1.7321397981971387], [1.7319846239308874, 1.7320822059575116]].

1. **Niutono metodas**

Niutono metode kairiosiose lygčių sistemos pusėse esančios funkcijos ištiesinamos taške Xi ir, išsprendus tiesinę lygčių sistemą, turėtasis taškas pakeičiamas tiesinės lygčių sistemos sprendiniu Xi+1. Šiai idėjai realizuoti reikia pasinaudoti tikslo funkcijos Teiloro eilutės nariais iki antrosios eilės. Vėliau prilyginę nuliui, gausime iteracinę metodo formulę:

Xi+1 = Xi – f’(Xi)/f’’(Xi).

Diagram

Description automatically generated

4 pav. Niutono metodo grafiko ir jo sprendinių vaizdavimas

Grafike vaizduojama kaip po kiekvienos iteracijos taškai artėja sprendinio link. Taškai vaizduojami tik 1, 2, 3, 4 ir 7 iteracijų metu, kad matytųsi aiškus intervalo trumpėjimas ir artėjimas link minimumo.

Visi taškai, po kiekvienos iteracijos:

[3.472234310326069, 2.524189319050012, 1.9960729950020806, 1.7766174793315364, 1.733669235400448, 1.732048098582492, 1.7320458075855936]

**Rezultatų palyginimai**

1 lentelė. Tikslo funkcijos rezultatų palyginimai

|  |  |  |  |
| --- | --- | --- | --- |
| Rezultatai skaičiuojant funkciją  f(x) = (x2 - 3)2/9-1 | Intervalo dalijimo pusiau metodas | Auksinio pjūvio metodas | Niutono metodas |
| Iteracijų skaičius | 17 | 24 | 7 |
| Minimumo Xmin reikšmė | 1.7320632934570312 | 1.7320438958453674 | 1.732046 |
| Funkcijos f(xmin) reikšmė | -0.9999999997921353 | -0.9999999999363044 | -1.000000 |
| Skaičiuotų funkcijų kiekis | 29 | 26 | 14 |

Iš lentelės duomenų akivaizdžiai matosi, jog pagal iteracijų ir skaičiuotų funkcijų kiekį geriausiai pasirodė Niutono metodas, kuriam prireikė tik 7 iteracijų ir 14 apskaičiuotų funkcijų.

Lyginant Intervalo dalijimo pusiau ir auksinio pjūvio metodus, efektyvesnis metodas šiai funkcijai skaičiuoti yra intervalo dalijimo pusiau, jei teigtume, kad iteracijų skaičius nusako efektyvumą. Tačiau, nors auksinio pjūvio metodui reikėjo daugiau iteracijų atrasti sprendinį, jis efektyvesnis iš skaičiuotų funkcijų kiekio – jam per didesnį kiekį iteracijų, prireikė suskaičiuoti mažiau funkcijų. Tai būtų naudingas rodiklis, jei iteracijų kiekis būtų kur kas didesnis nei dabartinis ir galėtume pamatyti kaip intervalo dalijimo pusiau metodui reikia apskaičiuoti daugiau funkcijų kiekvienos iteracijos metu, kad galėtų surasti teisingą atsakymą.

Visų metodų apskaičiavimai funkcijos minimumo taške yra panašūs, tačiau tiksliausiai jį sugebėjo apskaičiuoti intervalo dalijimo pusiau metodas, o Niutono metodas pateikė mažiausiai tikslią reikšmę.

**Išvados**

Apibendrinant galima būtų teigti, jog visi metodai tinkamai atliko savo darbą – surado funkcijos minimumą. Vis dėlto, iš 1 lentelės duomenų galima spręsti, jog tikslo funkcijos f(x) = (x2 - 3)2/9-1 minimumui apskaičiuoti efektyviausias būtų Niutono metodas, kuriam prireikė tik 7 iteracijų ir 14 apskaičiuotų funkcijų. Intervalo dalijimo pusiau ir auksinio pjūvio metodai pasirodė apylygiai, tačiau intervalo dalijimo pusiau metodui prireikė mažiau iteracijų pasiekti tikslą, o auksinio pjūvio metodui prireikė mažiau funkcijų skaičiavimų.

**Priedas**

import random  
  
import numpy as np  
from matplotlib import pyplot as plt  
  
bisection\_method\_points = []  
golden\_section\_method\_points = []  
newtons\_method\_points = []  
  
def bisection\_method(func, l, r, eps):  
 def f(x):  
 return eval(func)  
  
 counter = 0  
  
 # 1.  
 xm = (l + r) / 2  
 L = r - l  
 fxm = f(xm)  
 counter += 1  
  
 for i in range(1, 100):  
 # 2.  
 x1 = l + L / 4  
 x2 = r - L / 4  
 fx1 = f(x1)  
 counter += 1  
 if fx1 >= fxm:  
 fx2 = f(x2)  
 counter += 1  
  
 # 3. Atmetamas (xm , r]  
 if fx1 < fxm:  
 r = xm  
 xm = x1  
 fxm = fx1  
 L = r - l  
 # 4. Atmetamas [l, xm)  
 elif fx2 < fxm:  
 l = xm  
 xm = x2  
 fxm = fx2  
 L = r - l  
 # 5. Atmetamas [l, x1) ir (x2, r]  
 else:  
 l = x1  
 r = x2  
 L = r - l  
  
 bisection\_method\_points.append([l, r])  
  
 # 6.  
 if L < eps:  
 break  
  
 print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
 print("Bisection method")  
 print("Number of iterations: %d" % i)  
 print("f(Xmin) =", fxm)  
 print("Xmin =", xm)  
 print("Number of functions calculated: %d" % counter)  
 print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
  
def golden\_section\_method(func, l, r, eps, tau):  
 counter = 0  
 def f(x):  
 return eval(func)  
  
 # 1.  
 L = r - l  
 x1 = r - tau \* L  
 x2 = l + tau \* L  
 fx1 = f(x1)  
 fx2 = f(x2)  
 counter += 2  
  
 for i in range(1, 100):  
 # 2. Atmetamas [l, x1)  
 if fx2 < fx1:  
 l = x1  
 L = r - l  
 x1 = x2  
 fx1 = fx2  
 x2 = l + tau \* L  
 fx2 = f(x2)  
 counter += 1  
 # 3. Atmetamas (x2, r]  
 else:  
 r = x2  
 L = r - l  
 x2 = x1  
 fx2 = fx1  
 x1 = r - tau \* L  
 fx1 = f(x1)  
 counter += 1  
  
 golden\_section\_method\_points.append([l, r])  
  
 # 4.  
 if L < eps:  
 break  
  
 print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
 print("Golden section method")  
 print("Number of iterations: %d" % i)  
 print("f(Xmin) =", fx1 if fx1 < fx2 else fx2)  
 print("Xmin =", x1 if fx1 < fx2 else x2)  
 print("Number of functions calculated: %d" % counter)  
 print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
  
def newtons\_method(func, x0, eps):  
 counter = 0  
  
 for i in range(1, 100):  
 xn = x0 - first\_deriv(x0, func) / second\_deriv(x0, func)  
 counter += 2  
 newtons\_method\_points.append(xn)  
 if abs(xn - x0) < eps:  
 break  
 x0 = xn  
  
 print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
 print("Newton's method")  
 print("Number of iterations: %d" % i)  
 print("f(Xmin): %f" % f(xn, func))  
 print("Xmin: %f" % xn)  
 print("Number of functions calculated: %d" % counter)  
 print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
  
def f(x, func):  
 return eval(func)  
  
def first\_deriv(x, func):  
 h = 1e-5  
 return (f(x + h, func) - f(x, func)) / h  
  
def second\_deriv(x, func):  
 h = 1e-5  
 return (first\_deriv(x + h, func) - first\_deriv(x, func)) / h  
  
def drawGraph(func, l, r, char):  
 x = np.linspace(l, r, 100)  
 y = eval(func)  
  
 print("bisection: ", bisection\_method\_points)  
 print("golden: ", golden\_section\_method\_points)  
 print("newtons: ", newtons\_method\_points)  
  
 fig1 = plt.figure()  
 ax = fig1.add\_subplot(1, 1, 1)  
 ax.set(ylim=(-2, 5))  
 ax.spines['left'].set\_position('zero')  
 ax.spines['bottom'].set\_position('zero')  
 ax.spines['right'].set\_color('none')  
 ax.spines['top'].set\_color('none')  
 ax.xaxis.set\_ticks\_position('bottom')  
 ax.yaxis.set\_ticks\_position('left')  
  
 if char == 'b':  
 drawPoints(bisection\_method\_points)  
 elif char == 'g':  
 drawPoints(golden\_section\_method\_points)  
 elif char == 'n':  
 drawPoints(newtons\_method\_points)  
  
 plt.plot(x, y, 'r')  
 plt.show()  
  
def drawPoints(point\_array):  
 show = [1, 2, 3, 4, len(point\_array)]  
 for i in range(0, len(point\_array)):  
 r = random.random()  
 b = random.random()  
 g = random.random()  
 a = 1  
 color = (r, g, b, a)  
 if i+1 in show:  
 if isinstance(point\_array[i], list):  
 plt.scatter(point\_array[i][0], 0, color=color)  
 plt.annotate(i + 1, (point\_array[i][0], 0))  
 plt.scatter(point\_array[i][1], 0, color=color)  
 plt.annotate(i + 1, (point\_array[i][1], 0))  
 else:  
 plt.scatter(newtons\_method\_points[i], 0, color=color)  
 plt.annotate(i + 1, (newtons\_method\_points[i], 0))  
  
def main():  
 function = "((x \*\* 2 - 3) \*\* 2) / 9 - 1"  
 l = 0  
 r = 10  
 # function = "(100 - x) \*\* 2"  
 # l = 60  
 # r = 150  
 eps = 0.0001  
 bisection\_method(function, l, r, eps)  
 tau = 0.61803  
 golden\_section\_method(function, l, r, eps, tau)  
 x0 = 5  
 newtons\_method(function, x0, eps)  
 drawGraph(function, 0, 4, 'n')  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

1. Taškai vaizduojami formatu [XL, XR], kur L yra kairioji intervalo pusė, o R – dešinioji. [↑](#footnote-ref-1)
2. Taškai vaizduojami formatu [XL, XR], kur L yra kairioji intervalo pusė, o R – dešinioji. [↑](#footnote-ref-2)