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

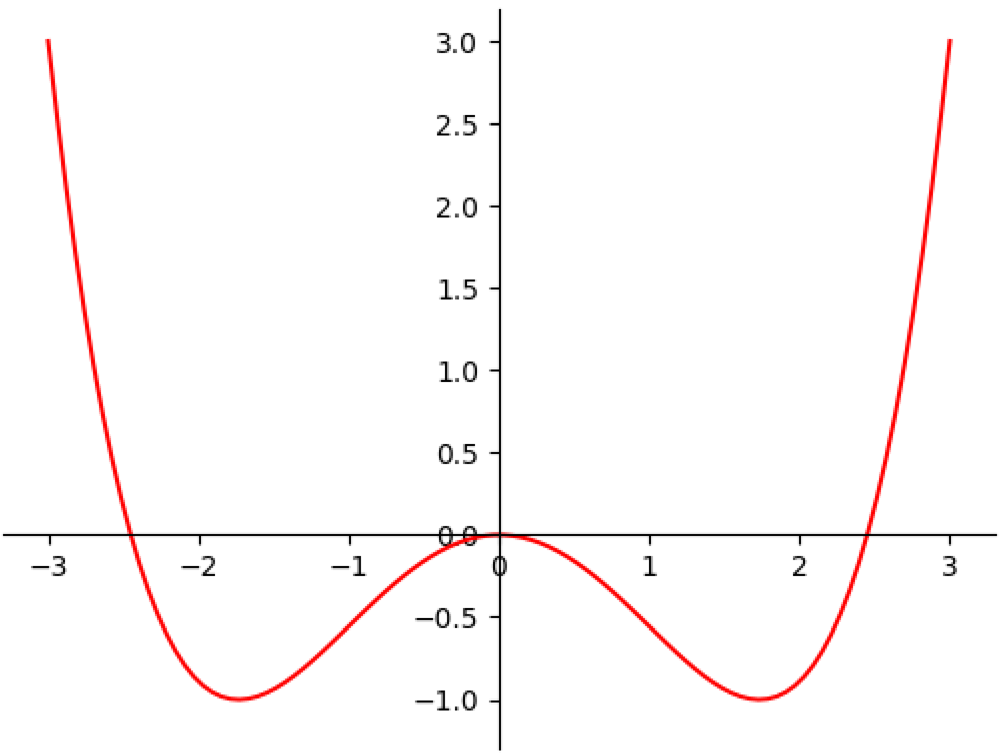
Laboratorinio darbo formulavimas: Suprogramuoti vienmačio optimizavimo intervalo dalijimo pusiau, auksinio pjūvio ir Niutono metodo algoritmus. Minimizuoti $f(x) = (x^2-3)^2/9-1$ funkciją intervalo dalijimo pusiau ir auksinio pjūvio metodais intervale [0,10] iki tikslumo 10^{-4} bei Niutono metodu nuo $x_0 = 5$ 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

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

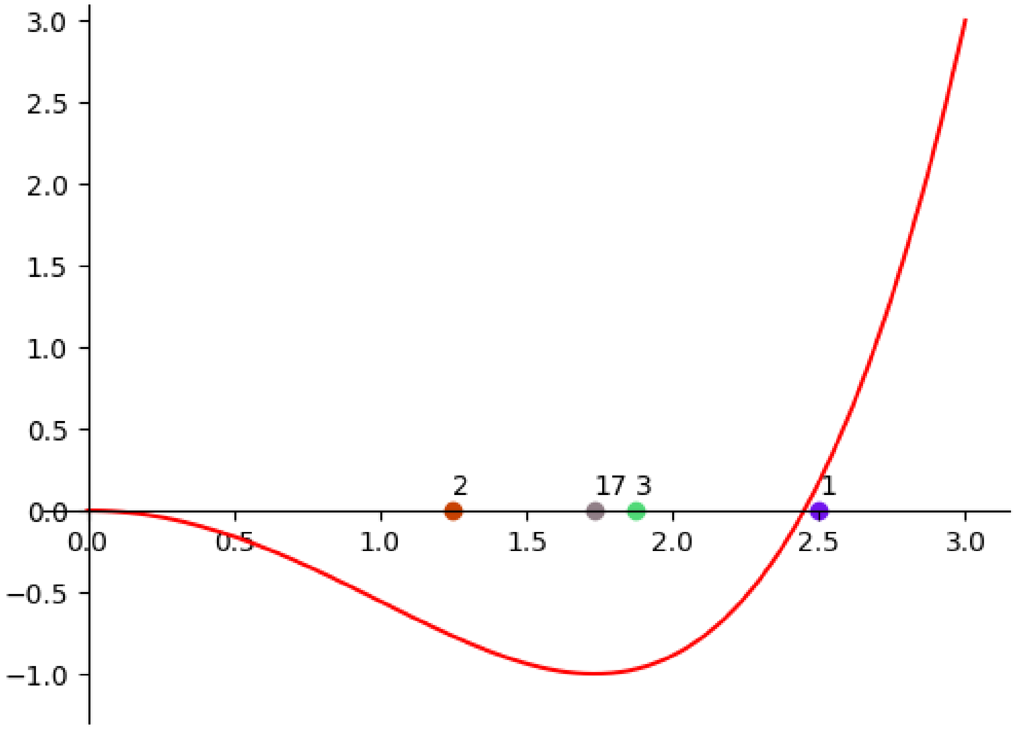
Pasirinkta programavimo kalba: Python.



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, kol pasieksime.



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

Grafike galima matyti kaip po kiekvienos iteracijos intervalo vidurio taškas artėja minimumo link. Šiame grafike pavaizduoti taškai tik po 1, 2, 3 ir 17 iteracijų, jog grafike aiškiai matytųsi artėjimas link minimumo.

1 lentelė. Intervalo dalijimo pusiau metodo rezultatai po pirmų trijų ir paskutinių trijų iteracijų.

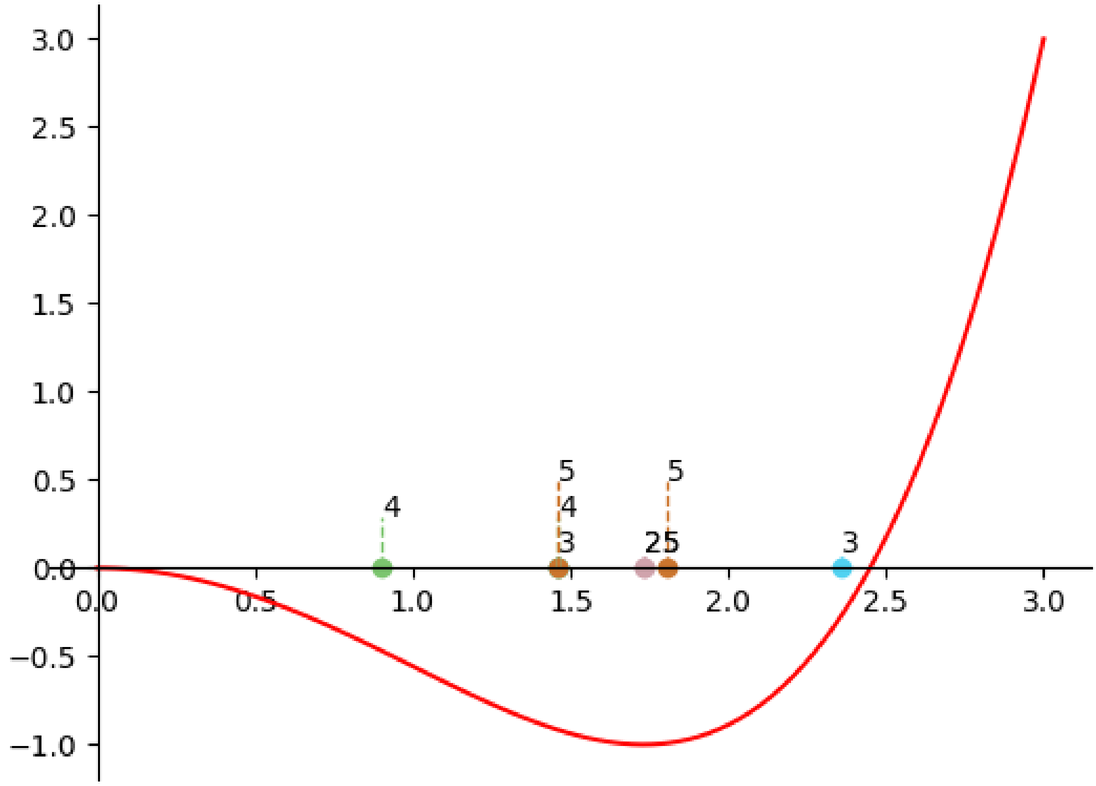
Iteracija	X_M[1]	$f(x_M)$
1	2.5	0.17361111111111116
2	1.25	-0.7703993055555556
3	1.875	-0.970458984375
15	1.732025146484375	-0.9999999991220246
16	1.732025146484375	-0.9999999991220246
17	1.7320632934570312	-0.9999999997921353

2. 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

[1] Indeksas M žymi intervalo vidurio tašką.

intervalo dalis, kurioje minimumas neegzistuoja. Procesas kartojamas, kol pasiekiamas nurodytas tikslumas.



3 pav. Auksinio pjūvio metodo grafiko ir taškų vaizdavimas

Grafike vaizduojama kaip po kiekvienos iteracijos intervalas artėja minimumo link. Taškai vaizduojami tik 3, 4, 5 ir 25 iteracijų metu, kad matytųsi aiškus artėjimas link minimumo.

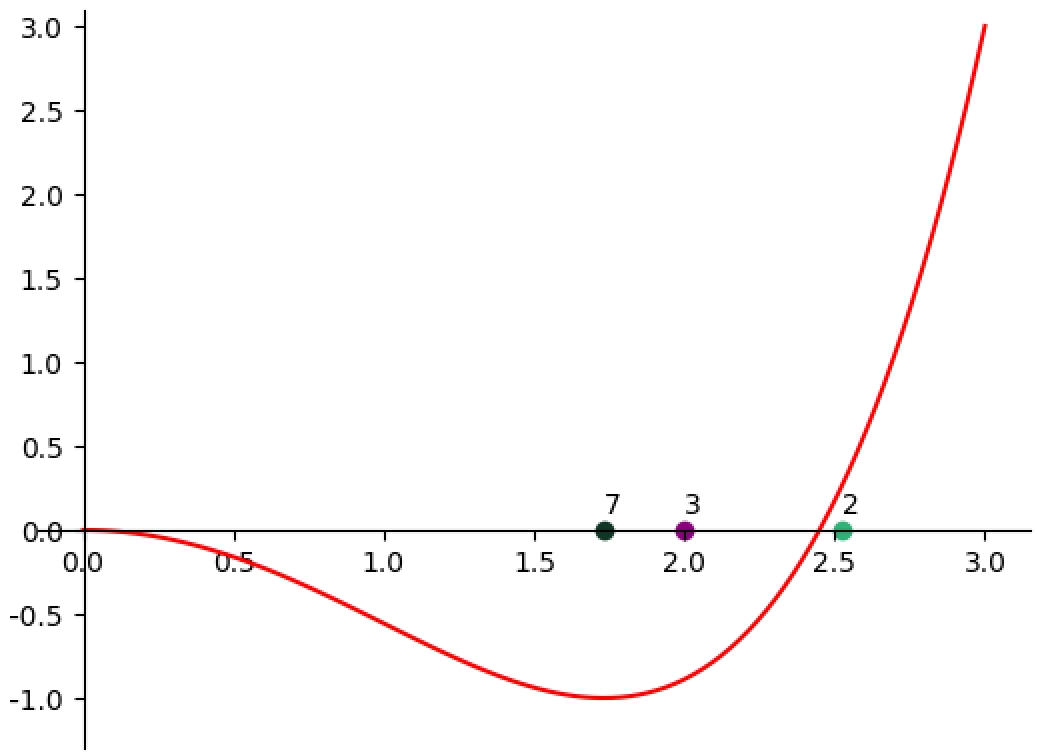
2 lentelė. Auksinio pjūvio metodo rezultatai po pirmų trijų ir paskutinių trijų iteracijų.

Iteracija	X_1	$f(x_1)$	X_2	$f(x_2)$
1	3.8197	13.92562283754261	6.1803	136.64066940921813
2	2.360689191	-0.2644916768171347	3.8197	13.92562283754261
3	1.459010809	-0.9156509070448806	2.360689191	-0.2644916768171347
23	1.7320822059575116	-0.9999999986854977	1.7321397981971387	-0.9999999894403483
24	1.7320438958453674	-0.9999999999363044	1.7320822059575116	-0.9999999986854977
25	1.732021897337597	-0.9999999988856166	1.7320438958453674	-0.9999999999363044

3. Niutono metodas

Lokalus minimumas pasiekiamas, kai funkcijos išvestinė lygi nuliui. Norint rasti, kur funkcijos išvestinė lygi nuliui naudojame iteracinę formulę $X_{i+1} = X_i - f'(X_i)/f''(X_i)$. Kiekvienos iteracijos metu X_i pakeičiamas naujai gautu X_{i+1} ir taip artėjama prie minimumo taško.

Niutono metodas konverguoja tik jei pradedamas netoli minimumo taško. Jis gali nekonverguoti net ir iškilos tikslo funkcijos atveju. Taip pat, šiam metodui ir kitiems, kurie pagrįsti išvestinėmis, reikia, kad tikslo funkcija būtų diferencijuojama.



4 pav. Niutono metodo grafiko ir taškų vaizdavimas

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

3 lentelė. Niutono metodo rezultatai po kiekvienos iteracijos.

Iteracija	X	f(x)
1	3.472234310326069	8.113175790817573
2	2.524189319050012	0.2630251253576461
3	1.9960729950020806	-0.8923487710661436
4	1.7766174793315364	-0.9972831696634855
5	1.733669235400448	-0.9999965043244012
6	1.732048098582492	-0.9999999999902152
7	1.7320458075855936	-0.999999999966667

Rezultatų palyginimai

4 lentelė. Tikslo funkcijos rezultatų palyginimai

Rezultatai skaičiuojant funkciją $f(x) = (x^2 - 3)^2/9-1$	Intervalo dalijimo pusiau metodas	Auksinio pjūvio metodas	Niutono metodas
Iteracijų skaičius	17	25	7
Minimumo X_{min} reikšmė	1.7320632934570312	1.7320438958453674	1. 7320458075855936

Funkcijos f(x_{min}) reikšmė	-0.9999999997921353	-0.9999999999363044	-0. 999999999966667
Skaičiuotų funkcijų kiekis	29	26	14
Apytikslis atstumas iki realaus X_{min}	0.0000124858881539	0.0000069117235099	0.0000049999832837

Intervale [0,10] funkcijos tikrasis $X_{min} = \sqrt{3} \approx 1.7320508075688773$

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. Intervalo dalijimo pusiau metode sugebėjau sumažinti skaičiuotų funkcijų kiekį neskaičiuojant $f(x_2)$, jeigu $f(x_1) < f(x_m)$.

Visų metodų apskaičiavimai funkcijos minimumo taške yra panašūs, tačiau tiksliausiai jį sugebėjo apskaičiuoti Niutono metodas, o intervalo dalijimo pusiau 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 $f(x) = (x^2 - 3)^2/9-1$ 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, l + L/2, 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

    golden_section_method_points.append([x1, x2])

    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([x1, x2])

        # 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(xn, func))
    print("Xmin:",  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)
    getPointsWithF(bisection_method_points)
    print("golden: ", golden_section_method_points)
    getPointsWithF(golden_section_method_points)
    print("newtons: ", newtons_method_points)
    getPointsWithF(newtons_method_points)

    fig1 = plt.figure()
    ax = fig1.add_subplot(1, 1, 1)
    ax.set(ylim=(-1.2, 3.2))
    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 getPointsWithF(point_array):
    def f(x):
        return eval("((x ** 2 - 3) ** 2) / 9 - 1")

    i = 1
    if isinstance(point_array[0], list) and len(point_array[0]) == 3:
        for array in point_array:
            print("Iteracija:", i,". Xl:", array[0], ",Yl:", f(array[0]), ".
```

```
Xm:", array[1], ",Ym:", f(array[1]), ". Xr:", array[2], ",Yr:", f(array[2]))
            i+=1
    elif isinstance(point_array[0], list) and len(point_array[0]) == 2:
        for array in point_array:
            print("Iteracija:", i,". Xl:", array[0], ",Yl:", f(array[0]), ".
Xr:", array[1], ",Yr:", f(array[1]))
            i+=1
    else:
        for point in point_array:
            print("Iteracija:", i,". X:", point, ",Y:", f(point))
            i += 1



def drawPoints(point_array):
    show = [3, 4, 5, 6, 7, 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)
                a = getA(i)
                plt.annotate(i + 1, (point_array[i][0], a))
                plt.scatter(point_array[i][1], 0, color=color)
                plt.annotate(i + 1, (point_array[i][1], a))
                plt.vlines(x=point_array[i][0], ymin=0, ymax=a, colors=color,
ls='--', lw=1, label='vline_single - partial height')
                plt.vlines(x=point_array[i][1], ymin=0, ymax=a, colors=color,
ls='--', lw=1, label='vline_single - partial height')
            else:
                plt.scatter(point_array[i], 0, color=color)
                plt.annotate(i + 1, (point_array[i], 0.09))

def getA(i):
    if i == 3:
        return 0.29
    elif i == 4:
        return 0.49
    elif i == 5:
        return 0.69
    elif i == 6:
        return 0.89
    else:
        return 0.09
def main():
    #sqrt(3) atsakymas
    function = "((x ** 2 - 3) ** 2) / 9 - 1"
    l = 0
    r = 10
    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, 3, 'g')
```

```python
if __name__ == '__main__':
    main()
```