

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

2 Laboratorinis darbas

Nr. 17

Atliko:

IFE-8 gr. studentas

Kemežys Martynas

Priėmė:

lekt. Andrius Kriščiūnas

KAUNAS, 2020

TURINYS

[1.UŽDUOTIS 3](#_Toc57369876)

[2.Pagrindinė dalis 4](#_Toc57369877)

[2.1 Netiesinių lygčių sistemų sprendimas(I lygčių sistema) 4](#_Toc57369878)

[2.1.1 Skirtinguose grafikuose pavaizduoti paviršiai: 4](#_Toc57369879)

[2.1.2 Netiesinių lygčių sistemos sprendimas grafiniu būdu: 5](#_Toc57369880)

[2.1.3 Metodo tikrinimas su laisvai pasirinktais artiniais: 5](#_Toc57369881)

[2.1.4 Sprendinių tikrinimas su išoriniais ištekliais: 5](#_Toc57369882)

[2.1.5 Kodo fragmentas: 7](#_Toc57369883)

[2.2 Netiesinių lygčių sistemų sprendimas(II lygčių Sistema) 7](#_Toc57369884)

[2.2.1 Netiesinių lygčių sistemos sprendimas su laisvai pasirinktu pradiniu artiniu: 7](#_Toc57369885)

[2.2.2 Sprendinių tikrinimas su išoriniais ištekliais: 8](#_Toc57369886)

[2.2.3 Kodo fragmentas: 9](#_Toc57369887)

[2.3 Optimizavimo uždavinys 11](#_Toc57369888)

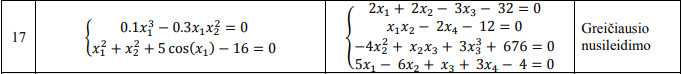
[2.3.1 Optimizavimo uždavinio sprendimas: 11](#_Toc57369889)

[2.3.2 Kodo fragmentas: 12](#_Toc57369890)

[3.IŠVADOS 15](#_Toc57369891)

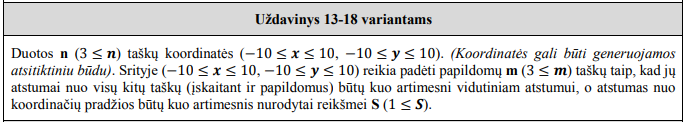
# 1.UŽDUOTIS

1. Netiesinių lygčių sistemų sprendimas.



*Pav. 1*

1. Optimizavimo uždavinys

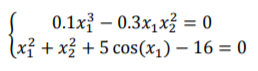


*Pav. 1*

# 2.Pagrindinė dalis

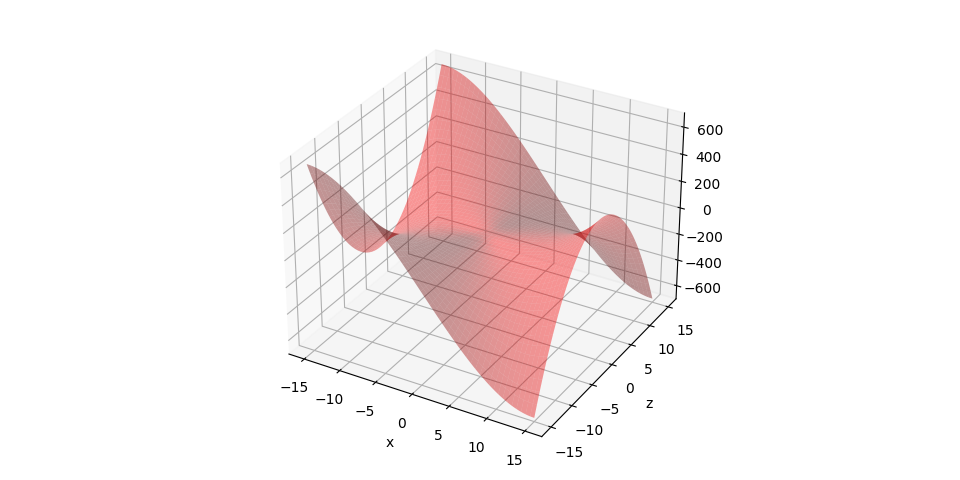
## 2.1 Netiesinių lygčių sistemų sprendimas(I lygčių sistema)

Lygčių sistema:



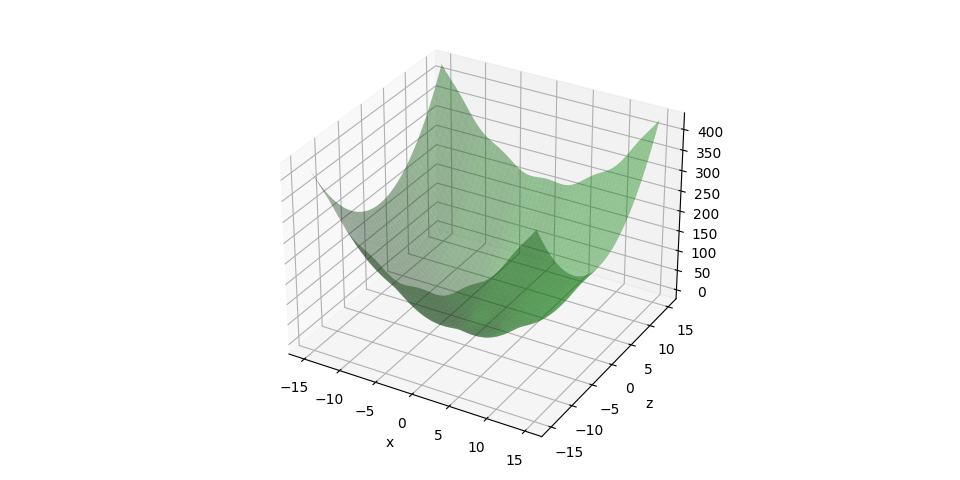
### 2.1.1 Skirtinguose grafikuose pavaizduoti paviršiai:

Z1()



*Pav. 4*

Z2()



*Pav. 5*

### 2.1.2 Netiesinių lygčių sistemos sprendimas grafiniu būdu:

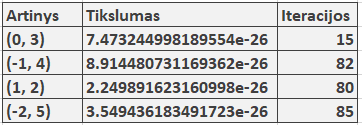
Apskaičiavus gradiento vektorių, jam priešinga kryptimi einama tol, kol funkcija tolydžio mažėja;

Funkcijai pradėjus vėl didėti, naujai apskaičuojame gradiento vektorių ir toliau minimizuojame priešinga jam kryptimi, kol gauname tinkama tikslumą.

Gautas rezultatas: x = [0, 3.31662479]

### 2.1.3 Metodo tikrinimas su laisvai pasirinktais artiniais:

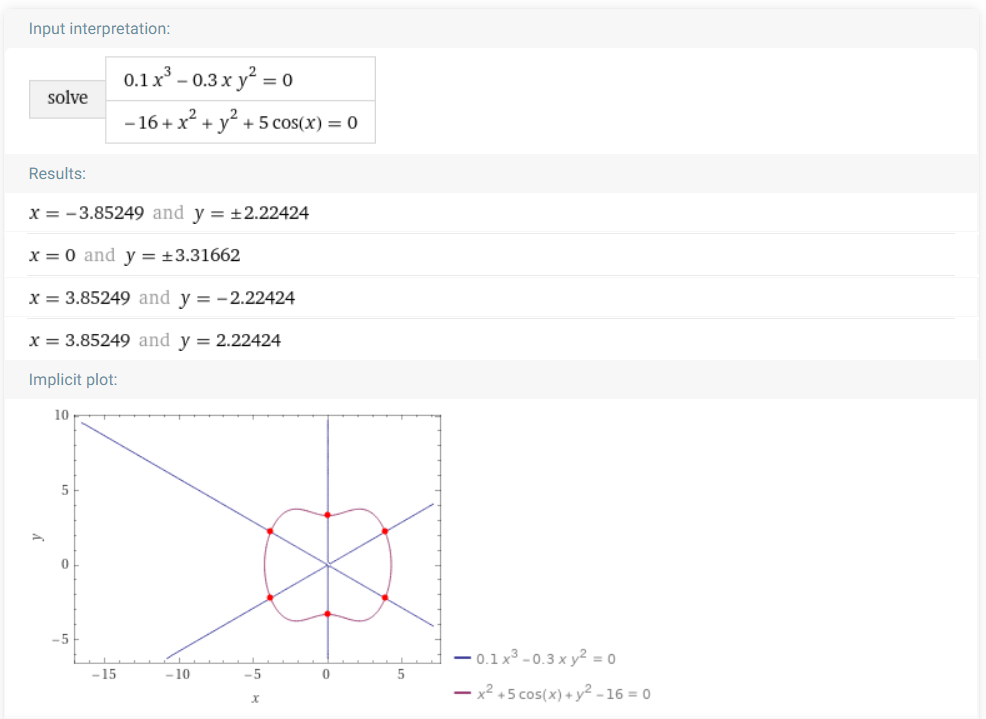
Gauti sprendiniai = **0** ir **3.31662479**, todėl artinius rinkausi aplink juos, iš lentelės galime pastebėti, kad paėmus tolimesnį artinį, iteracijų skaičius didėja.



*Lentele. 1*

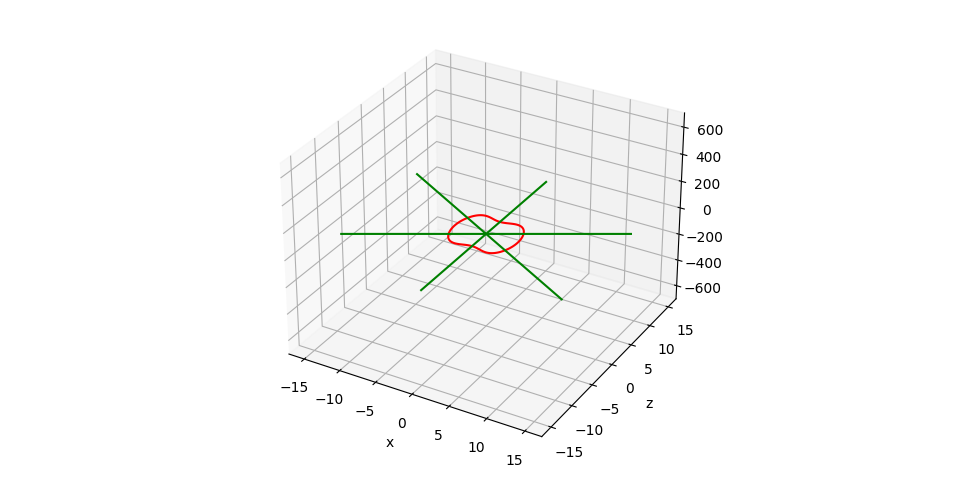
### 2.1.4 Sprendinių tikrinimas su išoriniais ištekliais:

Tikrinta su [**wolframalpha**](https://www.wolframalpha.com/input/?i=systems+of+equations+calculator&assumption=%7B%22F%22%2C+%22SolveSystemOf2EquationsCalculator%22%2C+%22equation1%22%7D+-%3E%220.1x%5E3-0.3x*y%5E2%3D0%22&assumption=%22FSelect%22+-%3E+%7B%7B%22SolveSystemOf2EquationsCalculator%22%7D%7D&assumption=%7B%22F%22%2C+%22SolveSystemOf2EquationsCalculator%22%2C+%22equation2%22%7D+-%3E%22x%5E2%2By%5E2%2B5*cos%28x%29-16%3D0%22)



*Pav. 6*

Gautas rezulatas:



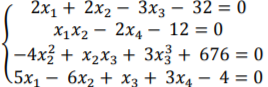
*Pav. 7*

### 2.1.5 Kodo fragmentas:

def gradientas(x): # gradiento funkcija  
 dx0 = (tikslumas([x[0] + 1e-13, x[1]]) - tikslumas(x)) / 1e-13  
 dx1 = (tikslumas([x[0], x[1] + 1e-13]) - tikslumas(x)) / 1e-13  
 return np.array([dx0, dx1])  
  
def tikslumas(x): # grazina tiksluma  
 return (lygciuSistema(x) \*\* 2).sum()  
  
def greiciausio\_nusileidimo(funkcija): # metodas  
 alpha = 0.01  
 x = np.array([0, 3.5])  
 g = gradientas(x)  
 for i in range(180):  
 buv\_tikslumas = tikslumas(x) # išsisaugom buvusį tikslumą  
 x = x - alpha \* g # nauja kryptis  
  
 if tikslumas(x) > buv\_tikslumas: # tikrinam ar naujas tikslenis  
 x = x + alpha \* g # atgalinis žingsnis  
 g = gradientas(x) # perskaičiuoja gradienta  
 x = x - alpha \* g # nauja kryptis  
  
 print(f'iteracijos: {i} tikslumas: {tikslumas(x)}')  
  
 if tikslumas(x) < 1e-25: break # tinkamas tikslumas - stabdom  
  
 print(f'Funkcijos reiksme: {funkcija(x)}')  
 print(f'x = {x}')  
  
def lygciuSistema(x): # grazina reiksmiu stulpeli  
 return np.array([  
 [0.1 \* x[0] \*\* 3 - 0.3 \* x[0] \* (x[1] \*\* 2)],  
 [x[0] \*\* 2 + x[1] \*\* 2 + 5 \* np.cos(x[0]) - 16]  
 ])  
  
greiciausio\_nusileidimo(lygciuSistema)

## 2.2 Netiesinių lygčių sistemų sprendimas(II lygčių Sistema)

Lygčių sistema:



### 2.2.1 Netiesinių lygčių sistemos sprendimas su laisvai pasirinktu pradiniu artiniu:

Gauti sprendiniai =

**[ 5, 2, -6, -1 ]**



*Lentele. 2*

### 2.2.2 Sprendinių tikrinimas su išoriniais ištekliais:

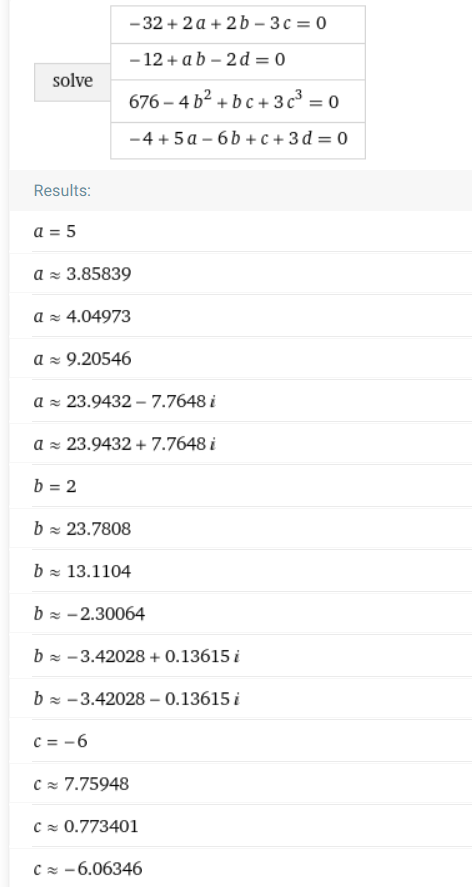
Tikrinta su [**wolframalpha**](https://www.wolframalpha.com/input/?i=systems+of+equations+calculator&assumption=%7B%22F%22%2C+%22SolveSystemOf4EquationsCalculator%22%2C+%22equation1%22%7D+-%3E%222a%2B2b-3c-32%3D0%22&assumption=%22FSelect%22+-%3E+%7B%7B%22SolveSystemOf4EquationsCalculator%22%7D%7D&assumption=%7B%22F%22%2C+%22SolveSystemOf4EquationsCalculator%22%2C+%22equation2%22%7D+-%3E%22ab-2d-12%3D0%22&assumption=%7B%22F%22%2C+%22SolveSystemOf4EquationsCalculator%22%2C+%22equation3%22%7D+-%3E%22-4b%5E2%2Bbc%2B3c%5E3%2B676%3D0%22&assumption=%7B%22F%22%2C+%22SolveSystemOf4EquationsCalculator%22%2C+%22equation4%22%7D+-%3E%225a-6b%2Bc%2B3d-4%3D0%22)

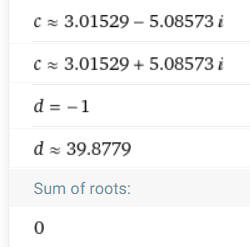
Gauti sprendiniai =

**[ 5, 2, -6, -1 ]**

Gauti sprendiniai iš išorinių išteklių=

**A=5, B=2, C=-6, D=-1**



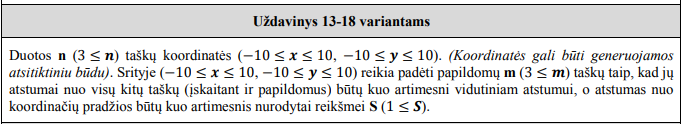


*Pav. 8*

### 2.2.3 Kodo fragmentas:

def lygciuSistema(x): # grazina lygciu sistema  
 return np.array([  
 2 \* x[0] + 2 \* x[1] - 3 \* x[2] - 32,  
 x[0] \* x[1] - 2 \* x[3] - 12,  
 -4 \* x[1] \*\* 2 + x[1] \* x[2] + 3 \* x[2] \*\* 3 + 676,  
 5 \* x[0] - 6 \* x[1] + x[2] + 3 \* x[3] - 4  
 ])  
  
def tikslumas(x): # grazina tiksluma  
 return (lygciuSistema(x) \*\* 2).sum()  
  
def gradientas(x): # gradiento funkcija  
 dx0 = (tikslumas([  
 x[0] + 1e-14, x[1], x[2], x[3]  
 ]) - tikslumas(x)) / 1e-14  
 dx1 = (tikslumas(  
 [x[0], x[1] + 1e-14, x[2], x[3]  
 ]) - tikslumas(x)) / 1e-14  
 dx2 = (tikslumas(  
 [x[0], x[1], x[2] + 1e-14, x[3]  
 ]) - tikslumas(x)) / 1e-14  
 dx3 = (tikslumas(  
 [x[0], x[1], x[2], x[3] + 1e-14]  
 ) - tikslumas(x)) / 1e-14  
 g = np.array([dx0, dx1, dx2, dx3])  
 return g  
  
# turi x'sai gautis = 5 2 -6 -1  
def greiciausio\_nusileidimo(): # metodas  
 alpha = 1.1  
 x = np.array([  
 4.9, 1.9, -5.9, -0.9  
 ])  
  
 x\_1 = 0  
 g = gradientas(x)  
 for i in range(1000000):  
 buv\_tikslumas = tikslumas(x)  
 if buv\_tikslumas < 1e-17: # jei tikslumas pasiektas - stabdom  
 break  
  
 x = x - alpha \* g # nauja kryptis  
  
 if buv\_tikslumas < tikslumas(x):  
 x = x + alpha \* g # atgalinis žingsnis  
 alpha \*= 0.4 # sumažinam alpha  
 g = gradientas(x) # perskaičiuoja gradienta  
 else:  
 x\_1 += 1  
 if x\_1 > 10: # kas 10 žingsnių didinam alpha  
 x\_1 = 0  
 alpha \*= 10000  
  
  
 print(f'Tikslumas: {tikslumas(x)} iteracijos: {i}')  
 print()  
 print(f'x = {x}')  
  
greiciausio\_nusileidimo()

## 2.3 Optimizavimo uždavinys

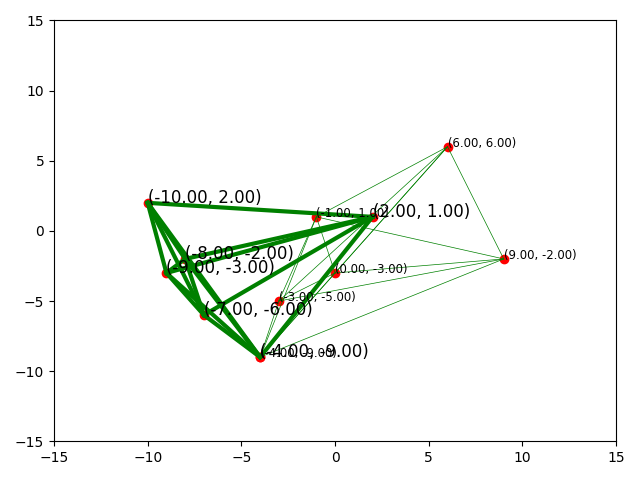


*Pav. 9*

### 2.3.1 Optimizavimo uždavinio sprendimas:

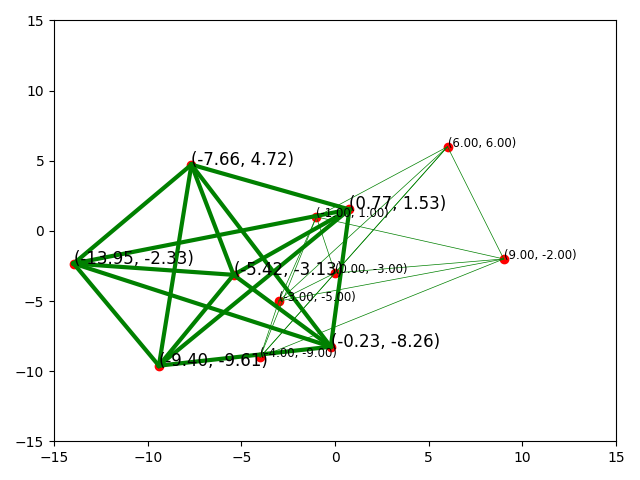
Pridedame n taškų, tada pridedam m taškų, tada m taškų kordinates koreguojam taip kad atstumai tarp taškų būtų artimesni vidutiniam atstumui. Apskaičiuojame vidutinį atstumą(naudojam formule atstumui tarp taškų skaičiuoti(sqrt((x1-x2)^2+(y1-y2)^2)), tada kol negaunam tinkamo tikslumo, skaičiuojam gradienta, nustatom naują kryptį arba darome atgalinį žingsnį.

Taškai ir atstumai prieš pertvarkyma:



*Pav. 10*

Taškai po pertvarkymo:

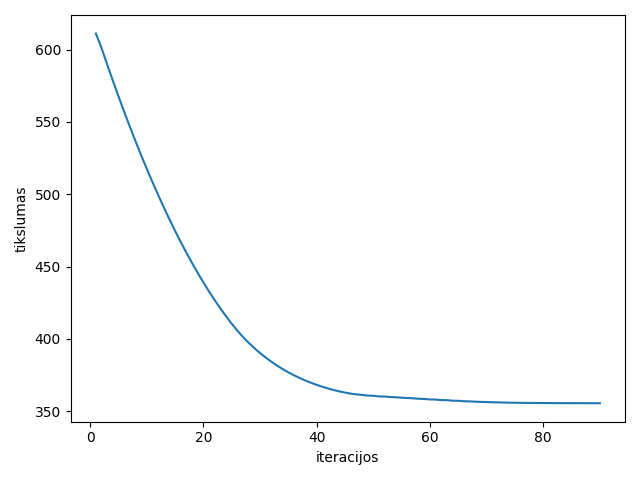


*Pav. 11*



*Lentele. 3*

Tikslo funkcijos priklausomybės nuo iteracijų skačiaus grafikas



*Pav. 12*

### 2.3.2 Kodo fragmentas:

def taskai(n):  
 taskai = []  
 kiekis = n  
 for \_ in range(kiekis):  
 x = np.random.randint(-10, 10)  
 y = np.random.randint(-10, 10)  
 taskai.append((x, y))  
 return np.array(taskai)  
  
def papildomi\_taskai(n):  
 ptaskai = []  
 kiekis = n  
 for i in range(kiekis):  
 rx = np.random.randint(-10, 10)  
 ry = np.random.randint(-10, 10)  
 ptaskai.append((rx, ry))  
 return np.array(ptaskai)  
  
taskai = taskai(6)  
x = taskai[:, 0]  
y = taskai[:, 1]  
ptaskai = papildomi\_taskai(6)  
rx = ptaskai[:, 0]  
ry = ptaskai[:, 1]  
  
def skaiciuoti\_vid\_ilgi(rx, ry, x ,y):  
 suma = 0  
 n = len(rx+x) # viso tasku suma  
 for i in range(n):  
 for j in range(i + 1, n):  
 suma += np.sqrt((rx[j]+x[j] - rx[i]+x[i]) \*\* 2 + (ry[j]+y[j] - ry[i]+y[i]) \*\* 2) # atstumo tarp tasku formule  
 kampu\_kiekis = n \* (n - 1) / 2 # kampu kiekis  
 vidutinis\_ilgis = suma / kampu\_kiekis # vidutinis atstumas  
 return vidutinis\_ilgis, suma  
  
  
def tikslumas(rx, ry, vidutinis, s, x, y):  
 suma = 0  
 n = len(rx)  
 \_, ilgis = skaiciuoti\_vid\_ilgi(rx, ry, x, y)  
 for i in range(n):  
 for j in range(i + 1, n):  
 atst = ((rx[j] - rx[i]) \*\* 2 + (ry[j] - ry[i]) \*\* 2) \*\* 0.5  
 suma += (atst - vidutinis) \*\* 2  
 return suma + abs(ilgis - s)  
  
  
def gradientas(rx, ry, vidutinis, s, x ,y):  
 g = []  
 t = 1e-12  
 f0 = tikslumas(rx, ry, vidutinis, s, x ,y)  
 for i in range(len(rx)):  
 xx = np.array(rx, copy=True)  
 yy = np.array(ry, copy=True)  
 xx[i] += t  
 yy[i] += t  
 dx = (tikslumas(xx, ry, vidutinis, s, x, y) - f0) / t  
 dy = (tikslumas(rx, yy, vidutinis, s, x, y) - f0) / t  
 g.append((dx, dy))  
 g = np.array(g).T  
 return g / np.linalg.norm(g)  
  
  
def nusileidimo\_gradientas(rx, ry, s, x, y):  
 iteracijos = 0  
 eTikslumas = 1e10  
 log = []  
 vidutinis\_ilgis, \_ = skaiciuoti\_vid\_ilgi(rx, ry, x, y)  
 print(f'{vidutinis\_ilgis}')  
  
 alpha = 0.2  
 while eTikslumas > 1e-6:  
 iteracijos += 1  
 buv\_tikslumas = tikslumas(rx, ry, vidutinis\_ilgis, s, x, y) # išsisaugom tikslumą  
  
 grad = gradientas(rx, ry, vidutinis\_ilgis, s, x, y) # apsiskaičiuoja gradienta  
  
 log.append((iteracijos, buv\_tikslumas))  
 rx = rx - alpha \* grad[0] # nauja kryptis  
 ry = ry - alpha \* grad[1] # nauja kryptis  
  
 esam\_tikslumas = tikslumas(rx, ry, vidutinis\_ilgis, s, x , y) # apskaičiuoja dabartinį tikslumą po krypties keitimo  
 eTikslumas = np.abs(esam\_tikslumas - buv\_tikslumas) / (np.abs(buv\_tikslumas) + np.abs(esam\_tikslumas))  
 if eTikslumas < 1e-6:  
 atvaizduoti\_taskus(rx, ry, x, y)  
 rodyti\_tiksluma(np.array(log))  
 break # stabdom pasiekus tinkama tiksluma  
 if esam\_tikslumas > buv\_tikslumas:  
 rx = rx + alpha \* grad[0] # atgalinis žingsnis  
 ry = ry + alpha \* grad[1] # atgalinis žingsnis  
 alpha /= 2  
 print(f'{iteracijos}')

# 3.IŠVADOS

Minimizuojant priešinga gradientui kryptimi, gradiento vektorių tenka apskaičiuoti kiekviename žingsnyje. Tai užima nemažai skaičiavimo laiko.