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

1. Netiesinių lygčių sistemų sprendimas.

17	$\begin{cases} 0.1x_1^3 - 0.3x_1x_2^2 = 0 \\ x_1^2 + x_2^2 + 5\cos(x_1) - 16 = 0 \end{cases}$	$\begin{cases} 2x_1 + 2x_2 - 3x_3 - 32 = 0 \\ x_1x_2 - 2x_4 - 12 = 0 \\ -4x_2^2 + x_2x_3 + 3x_3^3 + 676 = 0 \\ 5x_1 - 6x_2 + x_3 + 3x_4 - 4 = 0 \end{cases}$	Greičiausio nusileidimo

Pav. 1

2. Optimizavimo uždavinys

Uždavinys 13-18 variantams
Duotos **n** ($3 \leq \boldsymbol{n}$) taškų koordinatės ($-10 \leq \boldsymbol{x} \leq 10$, $-10 \leq \boldsymbol{y} \leq 10$). *(Koordinatės gali būti generuojamos atsitiktiniu būdu)*. Srityje ($-10 \leq \boldsymbol{x} \leq 10$, $-10 \leq \boldsymbol{y} \leq 10$) reikia padėti papildomų **m** ($3 \leq \boldsymbol{m}$) taškų taip, kad jų atstumai nuo visų kitų taškų (įskaitant ir papildomus) būtų kuo artimesni vidutiniam atstumui, o atstumas nuo koordinačių pradžios būtų kuo artimesnis nurodytai reikšmei **S** ($1 \leq \boldsymbol{S}$).

Pav. 1

2.Pagrindinė dalis

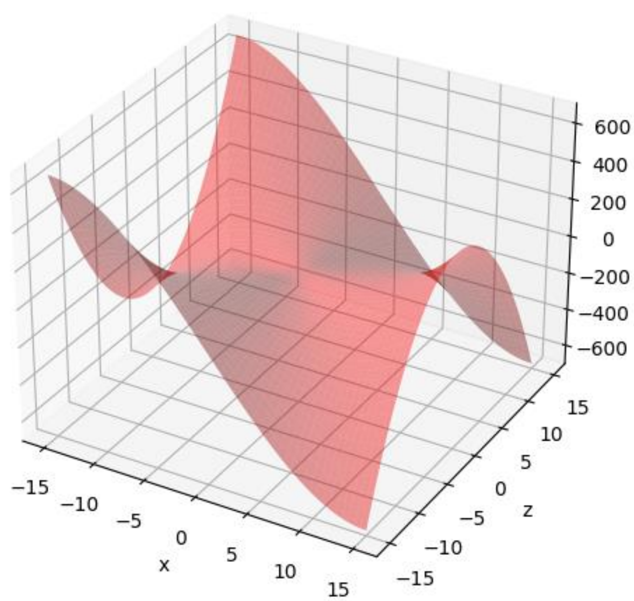
2.1 Netiesinių lygčių sistemų sprendimas(I lygčių sistema)

Lygčių sistema:

$$\begin{cases} 0.1x_1^3 - 0.3x_1x_2^2 = 0 \\ x_1^2 + x_2^2 + 5\cos(x_1) - 16 = 0 \end{cases}$$

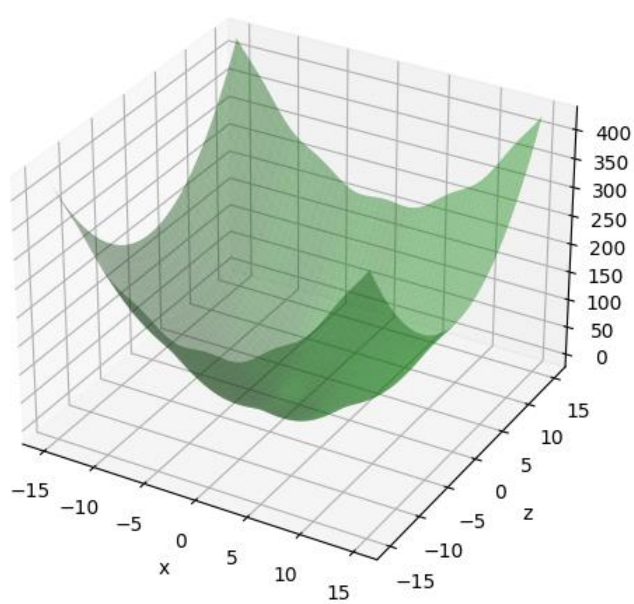
2.1.1 Skirtinguose grafikuose pavaizduoti paviršiai:

Z1($0.1x_1^3 - 0.3x_1x_2^2 = 0$)



Pav. 4

Z2($x_1^2 + x_2^2 + 5\cos(x_1) - 16 = 0$)



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:

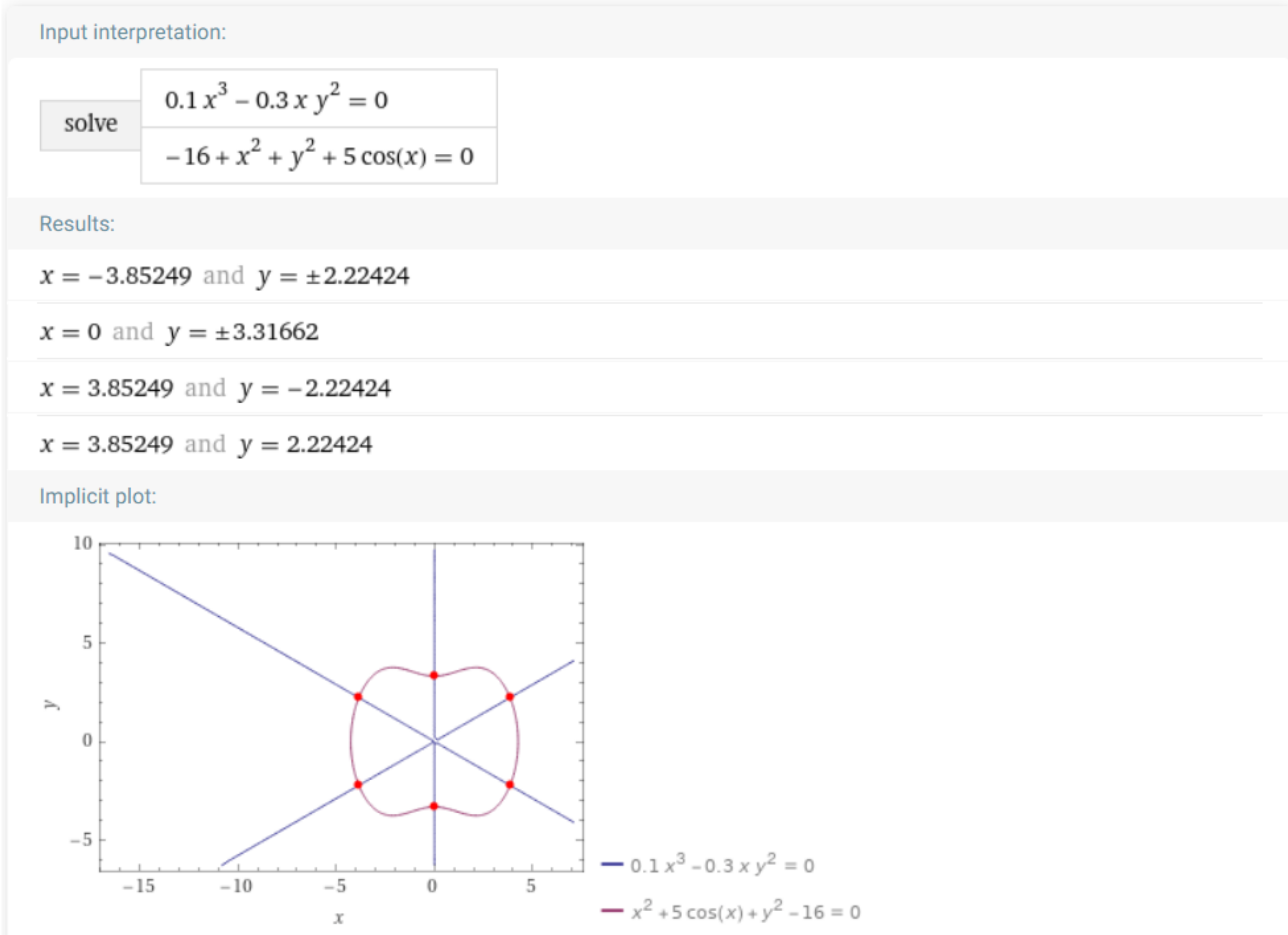
Gauti sprendiniai = **0** ir **3.31662479**, todėl artinius rinkausi aplink juos, iš lentelės galime pastebėti, kad paėmus tolimesnį artinį, iteracijų skaičius didėja.

Artinys	Tikslumas	Iteracijos
(0, 3)	7.473244998189554e-26	15
(-1, 4)	8.914480731169362e-26	82
(1, 2)	2.249891623160998e-26	80
(-2, 5)	3.549436183491723e-26	85

Lentele. 1

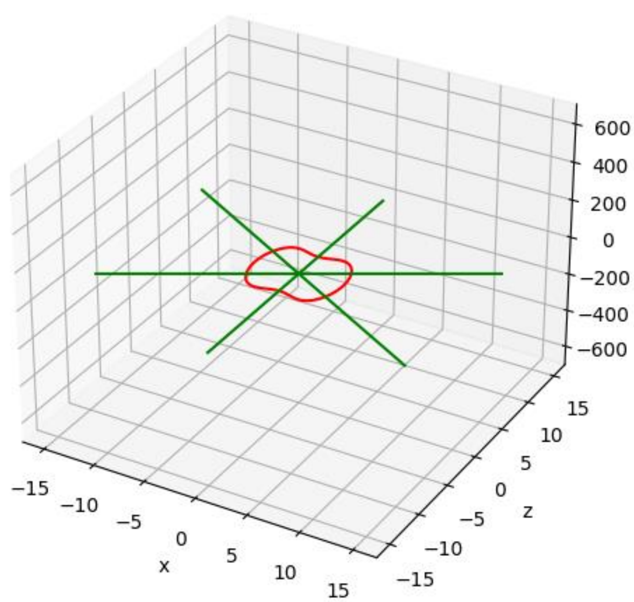
2.1.4 Sprendinių tikrinimas su išoriniais ištekliais:

Tikrinta su **wolframalpha**



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:

$$\begin{cases} 2x_1 + 2x_2 - 3x_3 - 32 = 0 \\ x_1x_2 - 2x_4 - 12 = 0 \\ -4x_2^2 + x_2x_3 + 3x_3^3 + 676 = 0 \\ 5x_1 - 6x_2 + x_3 + 3x_4 - 4 = 0 \end{cases}$$

2.2.1 Netiesinių lygčių sistemos sprendimas su laisvai pasirinktu pradiniu artiniu:

Gauti sprendiniai =

[5, 2, -6, -1]

Pasirinktas artinys	Tikslumas	Iteracijos
(4.9, 1.9, -5.9, -0.9)	9.981719093621009e-18	736131

Lentele. 2

2.2.2 Sprendinių tikrinimas su išoriniais ištekliais:

Tikrinta su **wolframalpha**

Gauti sprendiniai =
[5, 2, -6, -1]

Gauti sprendiniai iš išorinių išteklių=
A=5, B=2, C=-6, D=-1

solve

$$-32 + 2a + 2b - 3c = 0$$
$$-12 + ab - 2d = 0$$
$$676 - 4b^2 + bc + 3c^3 = 0$$
$$-4 + 5a - 6b + c + 3d = 0$$

Results:

$a = 5$

$a \approx 3.85839$

$a \approx 4.04973$

$a \approx 9.20546$

$a \approx 23.9432 - 7.7648\,i$

$a \approx 23.9432 + 7.7648\,i$

$b = 2$

$b \approx 23.7808$

$b \approx 13.1104$

$b \approx -2.30064$

$b \approx -3.42028 + 0.13615\,i$

$b \approx -3.42028 - 0.13615\,i$

$c = -6$

$c \approx 7.75948$

$c \approx 0.773401$

$c \approx -6.06346$

$c \approx 3.01529 - 5.08573\, i$

$c \approx 3.01529 + 5.08573\, i$

$d = -1$

$d \approx 39.8779$

Sum of roots:

0

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):
```

```python
            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

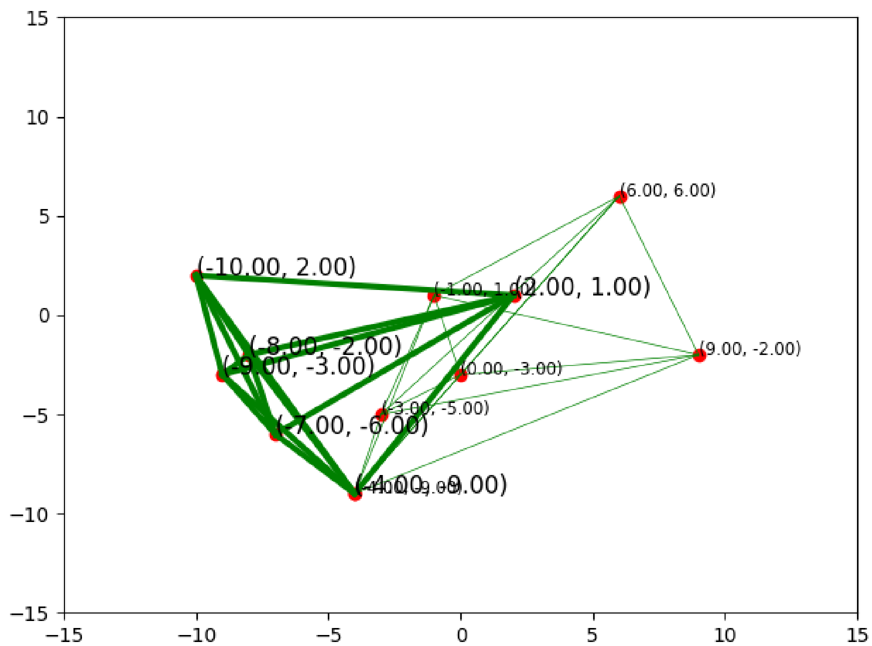
Uždavinys 13-18 variantams
Duotos **n** ($3 \le \boldsymbol{n}$) taškų koordinatės ($-10 \le \boldsymbol{x} \le 10$, $-10 \le \boldsymbol{y} \le 10$). *(Koordinatės gali būti generuojamos atsitiktiniu būdu)*. Srityje ($-10 \le \boldsymbol{x} \le 10$, $-10 \le \boldsymbol{y} \le 10$) reikia padėti papildomų **m** ($3 \le \boldsymbol{m}$) taškų taip, kad jų atstumai nuo visų kitų taškų (įskaitant ir papildomus) būtų kuo artimesni vidutiniam atstumui, o atstumas nuo koordinačių pradžios būtų kuo artimesnis nurodytai reikšmei **S** ($1 \le \boldsymbol{S}$).

Pav. 9

2.3.1 Optimizavimo uždavinio sprendimas:

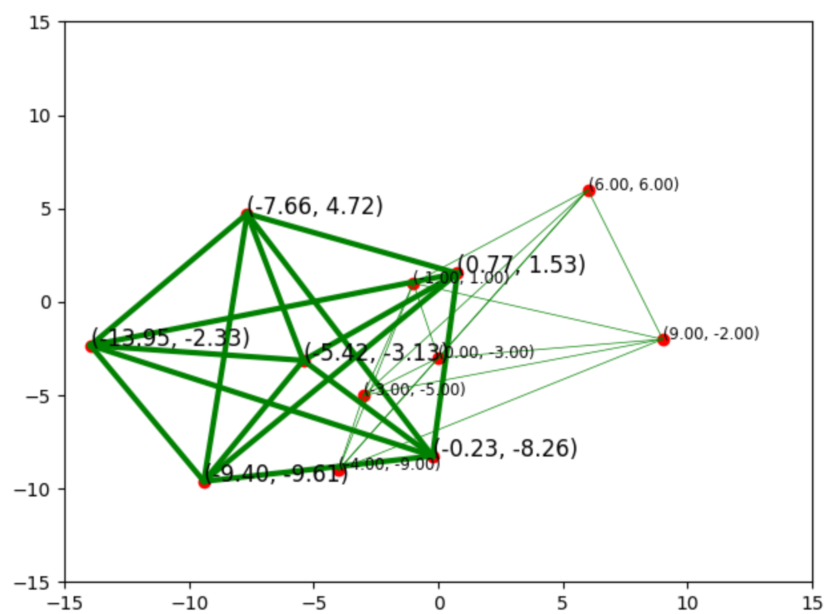
Pridedame n taškų, tada pridedam m taškų, tada m taškų kordinates koreguojam taip kad atstumai tarp taškų būtų artimesni vidutiniam atstumui. Apskaičiuojame vidutinį atstumą(naudojam formule atstumui tarp taškų skaičiuoti(sqrt((x1-x2)^2+(y1-y2)^2)), tada kol negaunam tinkamo tikslumo, skaičiuojam gradienta,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

Taškai ir atstumai prieš pertvarkyma:



Pav. 10

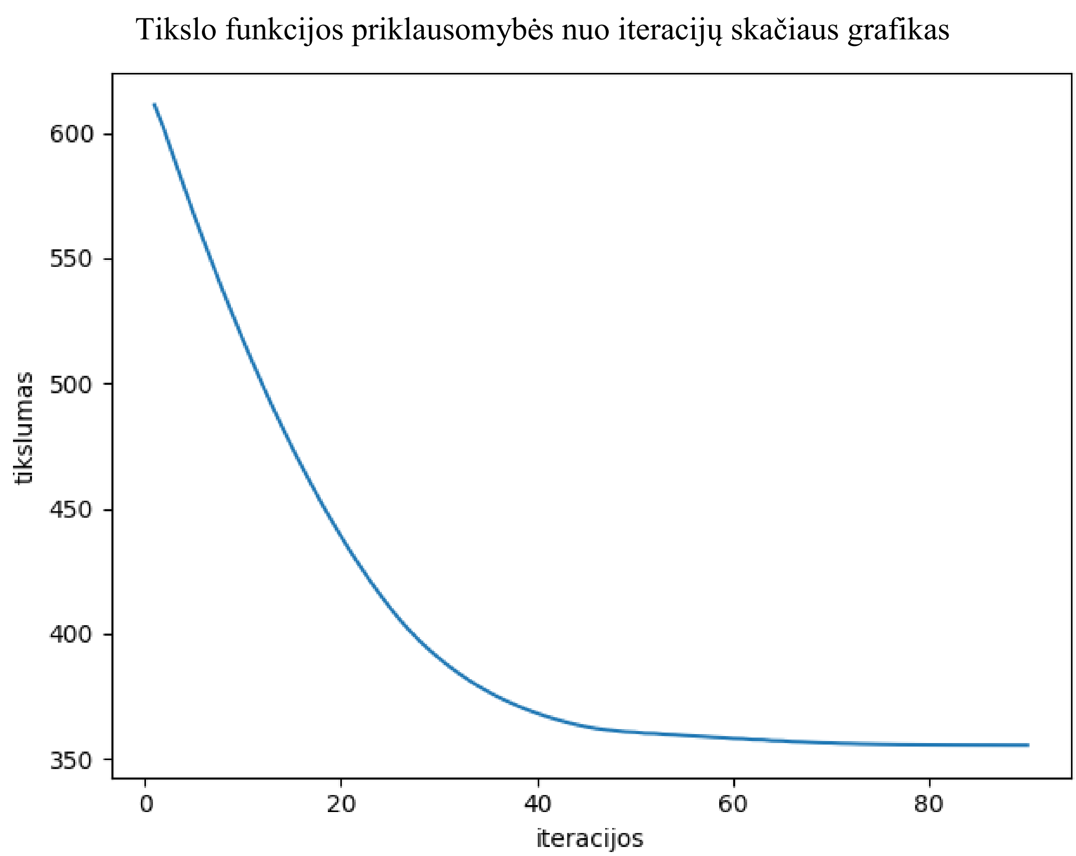
Taškai po pertvarkymo:



Pav. 11

Metodas	Atstumų vidurkis	Iteracijos
Gradientinis greičiausio nusileidimo	11.817279422100027	90

Lentele. 3



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.