

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

Modulio P175B015 „Skaitiniai metodai ir algoritmai“

Laboratorinio darbo ataskaita

Antras laboratorinis darbas

Dėstytojas

Doc. Andrius Kriščiūnas

Doc. Dalia Čalnerytė

Studentas

Rokas Puzonas IF-1/1

KAUNAS, 2023

Turinys

Pirma užduotis	3
Atspindžio metodas.....	4
Gauti rezultatai.....	4
Pasitikrinimas.....	5
Kodas	6
Paprastųjų iteracijų metodas	7
Rezultatai	7
Kodas	7
LU Sklaida	8
Rezultatai	8
Pasitikrinimas.....	9
Kodas	9
Antra užduotis.....	10
Paviršių grafikai	11
Grafinis sprendimas	12
Pradinių artinių tinklelis.....	12
Rezultatai	13
Pasitikrinimas.....	13
Kodas	13
Trečia užduotis.....	18
Skaičiavimo metodas	19
Rezultatai	20
Kodas	20

Pirma užduotis

Tiesinių lygčių sistemų sprendimas

- Lentelėje 1 duotos tiesinės lygčių sistemos, 2 lentelėje nurodyti metodai ir lygčių sistemų numeriai (iš 1 lentelės). Reikia suprogramuoti nurodytus metodus ir jais išspręsti pateiktas lygčių sistemas.
- Lentelėje 3 duotos tiesinės lygčių sistemos, laisvųjų narių vektoriai ir nurodytas skaidos metodas. Reikia suprogramuoti nurodytą metodą ir juo išspręsti pateiktas lygčių sistemas.

Sprendžiant lygčių sistemas (a ir b punktuose), turi būti:

- Programoje turi būti įvertinti atvejai:
 - kai lygčių sistema turi vieną sprendinį;
 - kai lygčių sistema sprendinių neturi;
 - kai lygčių sistema turi be galo daug sprendinių.
- Patikrinkite gautus sprendinius ir skaidas, įrašydami juos į pradinę lygčių sistemą.
- Gautą sprendinį patikrinkite naudodami išorinius išteklius (pvz., standartines Python funkcijas)

Lentelė 2:

Užduoties Nr.	Metodas	Lygčių sistemų Nr.
10	Atsipindžio	8, 13, 20
	Paprastųjų iteracijų	8

Lentelė 1:

Nr.	Lygčių sistema
8	$\begin{cases} 4x_1 + 3x_2 - x_3 + x_4 = 12 \\ 3x_1 + 9x_2 - 2x_3 - 2x_4 = 10 \\ -x_1 - 2x_2 + 11x_3 - x_4 = -28 \\ x_1 - 2x_2 - x_3 + 5x_4 = 16 \end{cases}$
13	$\begin{cases} x_1 - 2x_2 + 3x_3 + 4x_4 = 11 \\ x_1 - x_3 + x_4 = -4 \\ 2x_1 - 2x_2 + 2x_3 + 5x_4 = 7 \\ -7x_2 + 3x_3 + x_4 = 2 \end{cases}$
20	$\begin{cases} 2x_1 + 4x_2 + 6x_3 - 2x_4 = 2 \\ x_1 + 3x_2 + x_3 - 3x_4 = 1 \\ x_1 + x_2 + 5x_3 + x_4 = 7 \\ 2x_1 + 3x_2 - 3x_3 - 2x_4 = 2 \end{cases}$

Lentelė 3:

Užduoties Nr.	Lygčių sistema	b1	b2	b3	Metodas
10	$\begin{cases} 6x_1 + x_2 + 3x_3 - 2x_4 = \dots \\ 6x_1 + 8x_2 + x_3 - x_4 = \dots \\ 12x_1 - 2x_2 + 4x_3 - x_4 = \dots \\ 8x_1 + x_2 + x_3 + 5x_4 = \dots \end{cases}$	$\begin{cases} \dots = 8 \\ \dots = 14 \\ \dots = 13 \\ \dots = 15 \end{cases}$	$\begin{cases} \dots = 67 \\ \dots = 77 \\ \dots = 126 \\ \dots = 95 \end{cases}$	$\begin{cases} \dots = -5.25 \\ \dots = 0 \\ \dots = -13.5 \\ \dots = -7.25 \end{cases}$	LU

Atspindžio metodas

Gauti rezultatai

Lygčių sistema 8:

$$x_1 = 1$$

$$x_2 = 1$$

$$x_3 = -2$$

$$x_4 = 3$$

$$\text{Sklaida} = 1.9539925233402755e-14$$

Lygčių sistema 13:


Be galo daug sprendinių

Lygčių sistema 20:

Nėra sprendinių

Pasitikrinimas

Lygčių sistema 8:

 **WolframAlpha** computational intelligence.

system of equations

[NATURAL LANGUAGE](#) [MATH INPUT](#) [EXTENDED KEYBOARD](#) [EXAMPLES](#) [UPLOAD](#) [RANDOM](#)

Assuming "system of equations" refers to a computation | Use as referring to a mathematical definition or a general topic instead

Computational Inputs:

Assuming a system of four equations | Use a system of three equations or [more](#) instead

» equation 1: $4x_1 + 3x_2 - x_3 + x_4 = 12$

» equation 2: $3x_1 + 9x_2 - 2x_3 - 2x_4 = 10$

» equation 3: $-x_1 - 2x_2 + 11x_3 - x_4 = -28$

» equation 4: $x_1 - 2x_2 - x_3 + 5x_4 = 16$

[Compute](#)

Input interpretation

$4x_1 + 3x_2 - x_3 + x_4 = 12$
$3x_1 + 9x_2 - 2x_3 - 2x_4 = 10$
$-x_1 - 2x_2 + 11x_3 - x_4 = -28$
$x_1 - 2x_2 - x_3 + 5x_4 = 16$


[solve](#)

Result [Step-by-step solution](#)

$x_1 = 1$ and $x_2 = 1$ and $x_3 = -2$ and $x_4 = 3$

[Download Page](#) POWERED BY THE WOLFRAM LANGUAGE

Lygčių sistema 13:

 **WolframAlpha** computational intelligence.

system of equations

[NATURAL LANGUAGE](#) [MATH INPUT](#) [EXTENDED KEYBOARD](#) [EXAMPLES](#) [UPLOAD](#) [RANDOM](#)

Assuming "system of equations" refers to a computation | Use as referring to a mathematical definition or a general topic instead

Computational Inputs:

Assuming a system of four equations | Use a system of three equations or [more](#) instead

» equation 1: $x_1 - 2x_2 + 3x_3 + 4x_4 = 11$

» equation 2: $x_1 - x_3 + x_4 = -4$

» equation 3: $2x_1 - 2x_2 + 2x_3 + 5x_4 = 7$

» equation 4: $-7x_2 + 3x_3 + x_4 = 2$

[Compute](#)

Input interpretation

$x_1 - 2x_2 + 3x_3 + 4x_4 = 11$
$x_1 - x_3 + x_4 = -4$
$2x_1 - 2x_2 + 2x_3 + 5x_4 = 7$
$-7x_2 + 3x_3 + x_4 = 2$

[solve](#)

Result [Approximate form](#)

$x_2 = \frac{5x_1}{41} + \frac{66}{41} \wedge x_3 = \frac{19x_1}{41} + \frac{177}{41} \wedge x_4 = \frac{13}{41} - \frac{22x_1}{41}$

$e_1 \wedge e_2 \wedge \dots$ is the logical AND function

[Download Page](#) POWERED BY THE WOLFRAM LANGUAGE

Lygčių sistema 20:

The screenshot shows the WolframAlpha interface. At the top, the search bar contains "system of equations". Below the search bar, there are tabs for "NATURAL LANGUAGE" and "MATH INPUT". The main content area shows a message: "Assuming 'system of equations' refers to a computation | Use as referring to a mathematical definition or a general topic instead". Below this, there is a section titled "Computational Inputs:" with a message: "Assuming a system of four equations | Use a system of three equations or more instead". There are four equations listed: equation 1: $2x_1 + 4x_2 + 6x_3 - 2x_4 =$, equation 2: $x_1 + 3x_2 + x_3 - 3x_4 = 1$, equation 3: $x_1 + x_2 + 5x_3 + x_4 = 7$, and equation 4: $2x_1 + 3x_2 - 3x_3 - 2x_4 =$. A "Compute" button is visible. Below the equations, there is a section titled "Input interpretation" showing the equations in a table format. The table has two columns: "solve" and the equations. The equations are: $2x_1 + 4x_2 + 6x_3 - 2x_4 = 2$, $x_1 + 3x_2 + x_3 - 3x_4 = 1$, $x_1 + x_2 + 5x_3 + x_4 = 7$, and $2x_1 + 3x_2 - 3x_3 - 2x_4 = 2$. Below the table, there is a "Result" section showing "(no solutions exist)". At the bottom, there is a "Download Page" button and the text "POWERED BY THE WOLFRAM LANGUAGE".

Kodas

```
import numpy as np
from dataclasses import dataclass
from typing import List

@dataclass
class Lygtis:
    koeficientai: List[float]
    rezultatas: List[float]

@dataclass
class LygciuSistema:
    lygtys: List[Lygtis]

def gauti_sklaida(lygciu_sistema: LygciuSistema, x, rezultato_idx=0):
    n = lygciu_sistema.lygtys[0].koeficientai
    sklaida = 0
    for lygtis in lygciu_sistema.lygtys:
        issistatyta = sum(x[i]*lygtis.koeficientai[i] for i in range(len(n)))
        sklaida += abs(issistatyta - lygtis.rezultatas[rezultato_idx])
    return sklaida

def gauso_atispindzio_metodas(lygciu_sistema: LygciuSistema, epsilon=1e-7):
    koeficientai = list(lygtis.koeficientai for lygtis in lygciu_sistema.lygtys)
    A = np.matrix(koeficientai).astype(float)
    rezultatai = list(lygtis.rezultatas for lygtis in lygciu_sistema.lygtys)
    b = np.matrix(rezultatai).astype(float)
```

```

n = (np.shape(A))[0]
nb = (np.shape(b))[1]
A1 = np.hstack((A, b))

# tiesioginis etapas(atspindziai):
for i in range(0, n - 1):
    z = A1[i:n, i]
    zp = np.zeros(np.shape(z))
    zp[0] = np.linalg.norm(z)
    omega = z - zp
    omega = omega / np.linalg.norm(omega)
    Q = np.identity(n - i) - 2 * omega * omega.transpose()
    A1[i:n, :] = Q.dot(A1[i:n, :])

if np.sum(np.abs(A1[n-1, 0:n-nb+1])) < epsilon:
    if abs(A1[n-1, nb]) < epsilon:
        print("Be galo daug sprendiniu")
    else:
        print("Nera sprendiniu")
    return

# atgalinis etapas:
x = np.zeros(shape=(n, 1))
for i in range(n - 1, -1, -1):
    x[i, :] = (A1[i, n:n + nb] - A1[i, i + 1:n] * x[i + 1:n, :]) / A1[i, i]

print("Sprendinys:")
for i in range(0, n):
    print(f"x{i} = {x[i, 0]:.5f}")

print("Sklaida:", gauti_sklaida(lygciu_sistema, list(x.flat)))

```

Paprastųjų iteracijų metodas

Rezultatai

Reikiamų iteracijų skaičius: 73

$$x_1 = 1$$

$$x_2 = 2$$

$$x_3 = -2$$

$$x_4 = 3$$

Sklaida: 6.569500499153946e-11

Kodas

```

import numpy as np
from dataclasses import dataclass
from typing import List

@dataclass
class Lygtis:
    koeficientai: List[float]
    rezultatas: List[float]

@dataclass
class LygciuSistema:
    lygtys: List[Lygtis]

def gauti_sklaida(lygciu_sistema: LygciuSistema, x, rezultato_idx=0):
    n = lygciu_sistema.lygtys[0].koeficientai
    sklaida = 0

```

```

for lygtis in lygciu_sistema.lygtys:
    issistatyta = sum(x[i]*lygtis.koeficientai[i] for i in range(len(n)))
    sklaida += abs(issistatyta - lygtis.rezultatas[rezultato_idx])
return sklaida

def paprastu_iteraciju_metodas(lygciu_sistema: LygciuSistema, epsilon=1e-12, alpha=1,
iteraciju_kiekis=1000):
    koeficientai = list(lygtis.koeficientai for lygtis in lygciu_sistema.lygtys)
    A = np.matrix(koeficientai).astype(float)
    rezultatai = list(lygtis.rezultatas for lygtis in lygciu_sistema.lygtys)
    b = np.matrix(rezultatai).astype(float)

    n = np.shape(A)[0]
    alpha_mat = np.array([alpha, alpha, alpha, alpha]) # laisvai parinkti metodo parametrai
    Atld = np.diag(1.0/np.diag(A)).dot(A) - np.diag(alpha_mat)
    btld = np.diag(1.0/np.diag(A)).dot(b)

    x = np.zeros(shape = (n,1))
    x1 = np.zeros(shape = (n,1))
    for iteracija in range(iteraciju_kiekis):
        x1 = ((btld - Atld.dot(x)).transpose()/alpha_mat).transpose()
        prec = np.linalg.norm(x1 - x)/(np.linalg.norm(x) + np.linalg.norm(x1))
        if prec < epsilon: break
        x[:]=x1[:]

    print("Reikiamas iteraciju skaicius", iteracija)
    print("Sprendinys:")
    for i in range(0, n):
        print(f"x{i} = {x[i, 0]:.5f}")

    print("Sklaida:", gauti_sklaida(lygciu_sistema, list(x.flat)))

```

LU Sklaida

Rezultatai

Nr.	x1	x2	x3	x4	Sklaida
1	1	1	1	1	3.5527e-15
2	10	2	3	2	0
3	-1	0.75	0	0	0

Pasitikrinimas

system of equations

NATURAL LANGUAGE

MATH INPUT

EXTENDED KEYBOARD

EXAMPLES

UPLOAD

RANDOM

Assuming "system of equations" refers to a computation | Use as referring to a mathematical definition or a general topic instead

Computational Inputs:

Assuming a system of four equations | Use a system of three equations or

more

 instead

» equation 1:

6x₁ + x₂ + 3x₃ - 2x₄ = 6

» equation 2:

6x₁ + 8x₂ + x₃ - x₄ = 14

» equation 3:

12x₁ - 2x₂ + 4x₃ - x₄ =

» equation 4:

8x₁ + x₂ + x₃ + 5x₄ = 1

Compute

Input interpretation

solve

6x₁ + x₂ + 3x₃ - 2x₄ = 6

6x₁ + 8x₂ + x₃ - x₄ = 14

12x₁ - 2x₂ + 4x₃ - x₄ = 13

8x₁ + x₂ + x₃ + 5x₄ = 15

Result

Step-by-step solution

x₁ = 1 and x₂ = 1 and x₃ = 1 and x₄ = 1

Download Page

POWERED BY THE WOLFRAM LANGUAGE

system of equations

NATURAL LANGUAGE

MATH INPUT

EXTENDED KEYBOARD

EXAMPLES

UPLOAD

RANDOM

Assuming "system of equations" refers to a computation | Use as referring to a mathematical definition or a general topic instead

Computational Inputs:

Assuming a system of four equations | Use a system of three equations or

more

 instead

» equation 1:

6x₁ + x₂ + 3x₃ - 2x₄ = 6

» equation 2:

6x₁ + 8x₂ + x₃ - x₄ = 7

» equation 3:

12x₁ - 2x₂ + 4x₃ - x₄ =

» equation 4:

8x₁ + x₂ + x₃ + 5x₄ = 9

Compute

Input interpretation

solve

6x₁ + x₂ + 3x₃ - 2x₄ = 67

6x₁ + 8x₂ + x₃ - x₄ = 77

12x₁ - 2x₂ + 4x₃ - x₄ = 126

8x₁ + x₂ + x₃ + 5x₄ = 95

Result

Step-by-step solution

x₁ = 10 and x₂ = 2 and x₃ = 3 and x₄ = 2

Download Page

POWERED BY THE WOLFRAM LANGUAGE

system of equations

NATURAL LANGUAGE

MATH INPUT

EXTENDED KEYBOARD

EXAMPLES

UPLOAD

RANDOM

Assuming "system of equations" refers to a computation | Use as referring to a mathematical definition or a general topic instead

Computational Inputs:

Assuming a system of four equations | Use a system of three equations or

more

 instead

» equation 1:

6x₁ + x₂ + 3x₃ - 2x₄ = -

» equation 2:

6x₁ + 8x₂ + x₃ - x₄ = 0

» equation 3:

12x₁ - 2x₂ + 4x₃ - x₄ =

» equation 4:

8x₁ + x₂ + x₃ + 5x₄ = -

Compute

Input interpretation

solve

6x₁ + x₂ + 3x₃ - 2x₄ = -5.25

6x₁ + 8x₂ + x₃ - x₄ = 0

12x₁ - 2x₂ + 4x₃ - x₄ = -13.5

8x₁ + x₂ + x₃ + 5x₄ = -7.25

Result

Decimal form

Step-by-step solution

x₁ = -1 and 4x₂ = 3 and x₃ = 0 and x₄ = 0

Download Page

POWERED BY THE WOLFRAM LANGUAGE

Kodas

```
import numpy as np
from dataclasses import dataclass
from typing import List

@dataclass
class Lygtis:
    koeficientai: List[float]
```

```

rezultatas: List[float]

@dataclass
class LygciuSistema:
    lygtys: list[Lygtis]

def gauti_sklaida(lygciu_sistema: LygciuSistema, x, rezultato_idx=0):
    n = lygciu_sistema.lygtys[0].koeficientai
    sklaida = 0
    for lygtis in lygciu_sistema.lygtys:
        issistatyta = sum(x[i]*lygtis.koeficientai[i] for i in range(len(n)))
        sklaida += abs(issistatyta - lygtis.rezultatas[rezultato_idx])
    return sklaida

def lu_sklaidos_metodas(lygciu_sistema: LygciuSistema):
    koeficientai = list(lygtis.koeficientai for lygtis in lygciu_sistema.lygtys)
    A = np.matrix(koeficientai).astype(float)
    rezultatai = list(lygtis.rezultatas for lygtis in lygciu_sistema.lygtys)
    b = np.matrix(rezultatai).astype(float)

    n = np.shape(A)[0]
    P = np.arange(0, n)

    # tiesioginis etapas:
    for i in range(n-1):
        iii = abs(A[i:n,i]).argmax()
        A[[i,i+iii],:] = A[[i+iii,i],:] # sukeiciamos eilutes
        P[[i,i+iii]] = P[[i+iii,i]] # sukeiciami eiluciu numeriai

        for j in range(i+1,n):
            r = A[j,i]/A[i,i]
            A[j,i:n+1] = A[j,i:n+1] - A[i,i:n+1]*r;
            A[j,i] = r
    b = b[P]

    # 1-as atgalinis etapas, sprendžiama Ly=b, y->b
    for i in range(1, n):
        b[i] = b[i] - A[i,0:i]*b[0:i]

    # 2-as atgalinis etapas , sprendžiama Ux=b, x->b
    for i in range(n-1,-1,-1):
        b[i] = (b[i] - A[i,i+1:n]*b[i+1:n])/A[i,i]

    print("Sprendiniai:")
    for i, column in enumerate(b.transpose()):
        print(f"{i+1}.")
        if not np.isfinite(column).all():
            print(" Nera arba be galo daug sprendiniu")
            continue

    for j in range(0, n):
        print(f" x{j} = {column[0, j]:.5f}")

    print(f" Sklaida = {gauti_sklaida(lygciu_sistema, list(column.flat), i)}")

```

Antra užduotis

Netiesinių lygčių sistemų sprendimas

Duota netiesinių lygčių sistema (4 lentelė):

$$\begin{cases} Z_1(x_1, x_2) = 0 \\ Z_2(x_1, x_2) = 0 \end{cases}$$

- Skirtinguose grafikuose pavaizduokite paviršius $Z_1(x_1, x_2)$ ir $Z_2(x_1, x_2)$.
- Užduotyje pateiktą netiesinių lygčių sistemą išspręskite grafiniu būdu.

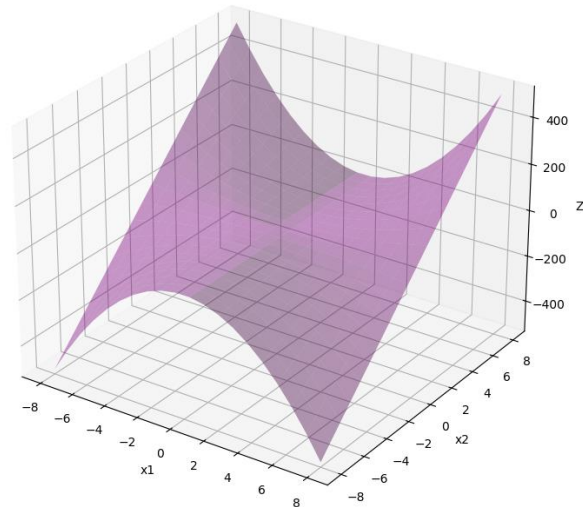
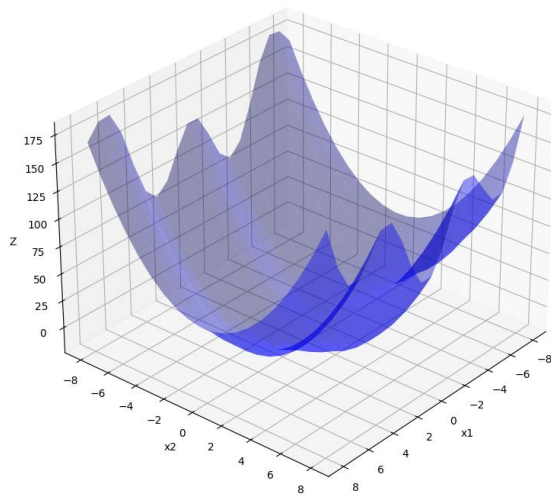
- c. Nagrinėjamoje srityje sudarykite stačiakampį tinklą (x_1, x_2 poros). Naudodami užduotyje nurodytą metodą apskaičiuokite netiesinių lygčių sistemos sprendinius, kai pradinis artinys įgyja tinklo koordinatų reikšmes. Tinklelyje vienodai pažymėkite taškus, kuriuos naudojant kaip pradinius artinius gaunamas tas pats sprendinys. Lentelėje pateikite apskaičiuotus skirtingus sistemos sprendinius ir bent po vieną jam atitinkantį pradinį artinį.
- d. Gautus sprendinius patikrinkite naudodami išorinius išteklius (pvz., standartines Python funkcijas).

Lentelė 4:

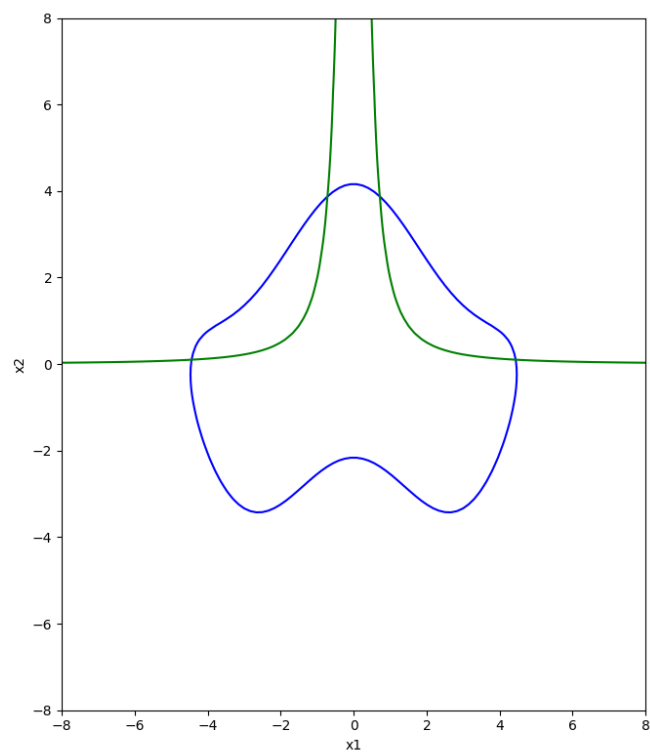
Metodas: Broideno

$$\begin{cases} x_1^2 + 2(x_2 - \cos(x_1))^2 - 20 = 0 \\ x_1^2 x_2 - 2 = 0 \end{cases}$$

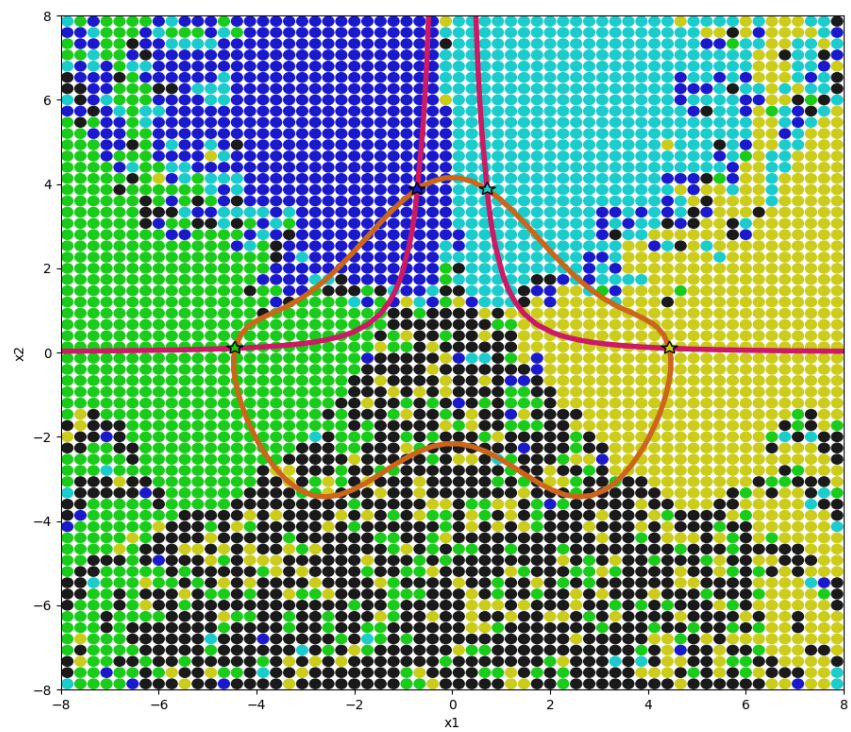
Paviršių grafikai



Grafinis sprendimas



Pradinių artinių tinklelis

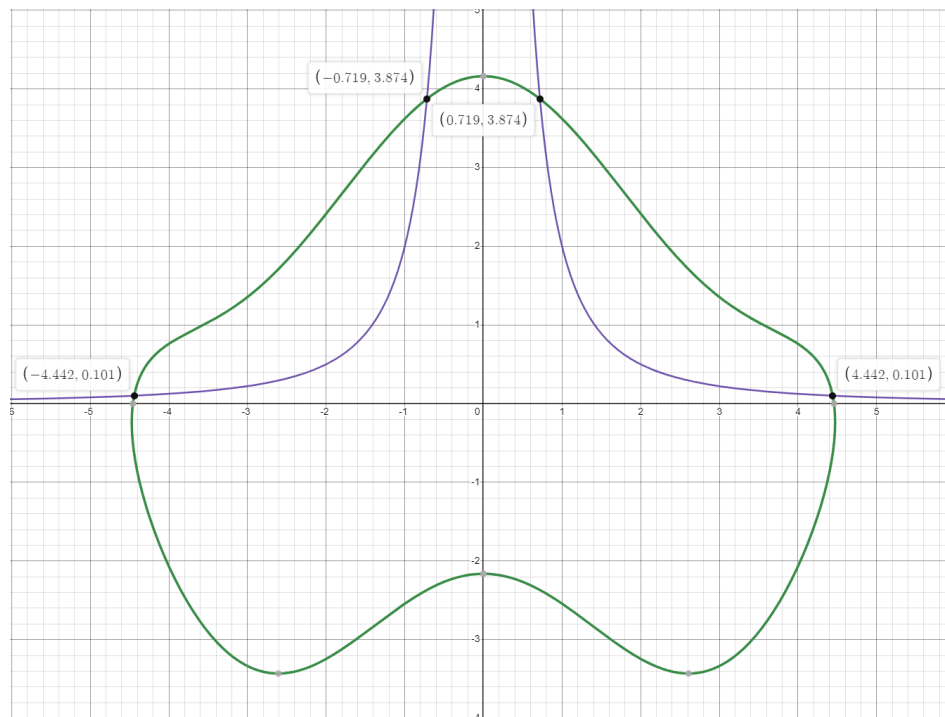


Rezultatai

Nr.	Artinys	Sprendinys	Iteracijos	Tikslumas
1	-4, 0	-4.442, 0.101	9	1.1435e-14
2	4, 0	4.442, 0.101	9	1.1435e-14
3	-2, 4	-0.718, 3.874	11	1.407e-16
4	2, 4	0.718, 3.874	11	1.087e-16

Pasitikrinimas

Naudota <https://www.desmos.com/calculator>



Kodas

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from dataclasses import dataclass
from typing import Iterable
import tkinter as tk

EPSILON = 1e-12

def ScrollTextBox(w,h):
    """
    sukuria TextBox su scrolais pagal w ir pagal h
    w ir h _ plotis ir aukstis
    Grazina textBoxa T
    Programoje textBox sukuriamas: T=ScrollTextBox(140,20)
```

```

    Irasoma komandomis : T.insert(END,str+"\n"); T.yview(END)
    """
    root = tk.Tk(); root.title("Scroll Text Box")
    frame1=tk.Frame(root); frame1.pack()
    #T = ScrolledText(root, height=h, width=w,wrap=WORD) # jeigu reikia scrolinti tik pagal h
    scrollbarY = tk.Scrollbar(frame1, orient='vertical'); scrollbarY.pack(side=tk.RIGHT, fill=tk.Y)
    scrollbarX = tk.Scrollbar(frame1, orient='horizontal'); scrollbarX.pack(side=tk.BOTTOM, fill=tk.X)
    T = tk.Text(frame1, width = w, height = h, wrap = tk.NONE,yscrollcommand = scrollbarY.set,xscrollcommand
= scrollbarX.set)
    T.pack();
    scrollbarX.config(command = T.xview); scrollbarY.config(command = T.yview)
    return T

def SpausdintiMatrica(*args):
    """
    A - matrica

    T - TextBox

    str - eilute pradzioje, str=, neprivaloma
    """
    A=args[0]; T=args[1]; str="";
    if len(args) == 3: str=args[2];
    siz=np.shape(A)
    T.insert(tk.END,"\n"+str+"=")
    if len(siz) > 0:
        for i in range (0,siz[0]):
            T.insert(tk.END,"\n")
            if len(siz) > 1:
                for j in range (0,siz[1]): T.insert(tk.END,"%12g  "%A[i,j]);
            else: T.insert(tk.END,"%12g  "%A[i])
    else : T.insert(tk.END,"%12g  "%A);
    T.yview(tk.END)
    T.update()

def Pavirsius(X, Y,LFF):
    """
    X,Y - meshgrid masyvai

    LF - dviejų kintamųjų vektorinės funkcijos vardas, argumentas paduodamas vektoriumi, išejimas vektorius
    ilgio 2

    rezultatas - dvigubas meshgrid masyvas Z[:,:][0:1]
    """
    siz=np.shape(X)
    Z=np.zeros(shape=(siz[0],siz[1],2))
    for i in range (0,siz[0]):
        for j in range (0,siz[1]): Z[i,j,:]=LFF([X[i][j],Y[i][j]]).transpose();
    return Z

@dataclass
class BroidenoRezultatas:
    x1: float
    x2: float
    y1: float
    y2: float
    tikslumas: float
    iteracija: int

@dataclass
class Tinklelis:
    from_x1: float
    to_x1: float
    from_x2: float
    to_x2: float
    density: int

def gauti_bendra_lygti(Z1, Z2):
    def LF(x): # grazina reiksmiu stulpeli
        s = np.array([
            Z1(x[0], x[1]),
            Z2(x[0], x[1]),
        ])
        s.shape=(2,1)

```

```

        return np.matrix(s)

    return LF

def gauti_tiksluma(x1, x2, f1, f2):
    x_abs_diff = np.abs(x1-x2)
    x_abs_sum = x1+x2
    f1_abs = np.abs(f1)
    f2_abs = np.abs(f2)
    if not np.isscalar(x1):
        x_abs_diff = sum(x_abs_diff)
        x_abs_sum = sum(x_abs_sum)
        f1_abs = sum(f1_abs)
        f2_abs = sum(f2_abs)

    if x_abs_sum > EPSILON:
        return x_abs_diff/(x_abs_sum + f1_abs + f2_abs)
    else:
        return x_abs_diff + f1_abs + f2_abs

def broideno_metodas_iter(
    init_x1: float,
    init_x2: float,
    Z1,
    Z2,
) -> Iterable[BroidenoRezultatas]:

    MAX_ITERATIONS = 80 # didžiausias leistinas iteraciju skaicius
    JACOB_DX = 0.1 # dx pradiniam Jakobio matricos inverciui

    if init_x1 == 0 and init_x2 == 0:
        return

    n = 2 # lygciu skaicius
    x = np.matrix(np.zeros(shape=(n,1)))
    x[0] = init_x1
    x[1] = init_x2

    LF = gauti_bendra_lygti(Z1, Z2)

    A = np.matrix(np.zeros(shape=(n,n)))
    x1 = np.zeros(shape=(n,1))
    for i in range(0,n):
        x1 = np.matrix(x)
        x1[i] += JACOB_DX
        A[:,i] = (LF(x1) - LF(x))/JACOB_DX

    ff = LF(x)

    for i in range(1, MAX_ITERATIONS+1):
        deltax = -np.linalg.solve(A,ff)
        x1 = np.matrix(x + deltax)
        ff1 = LF(x1)
        A += (ff1 - ff - A*deltax)*deltax.transpose()/(deltax.transpose() * deltax)
        tiksl = gauti_tiksluma(x, x1, ff, ff1)
        ff = ff1
        x = x1

        yield BroidenoRezultatas(
            x1[0, 0],
            x1[1, 0],
            ff1[0, 0],
            ff1[1, 0],
            tiksl[0, 0],
            i
        )

def broideno_metodas(
    init_x1: float,
    init_x2: float,
    Z1,
    Z2,
):
    for rez in broideno_metodas_iter(init_x1, init_x2, Z1, Z2):
        if rez.tikslumas < EPSILON:

```

```

        return rez

def viz_broideno_metodas(
    init_x1: float,
    init_x2: float,
    Z1,
    Z2,
    view_x1 = (-8, 8),
    view_x2 = (-8, 8)
):
    T = ScrollTextBox(100,20) # sukurti teksto isvedimo langa
    T.insert(tk.END,"Broideno metodas")

    #----- Grafika: funkciju LF paviršiai -----
    fig1 = plt.figure(1, figsize=plt.figaspect(0.5))

    ax1 = fig1.add_subplot(1, 2, 1, projection='3d')
    ax1.set_xlabel('x1')
    ax1.set_ylabel('x2')
    ax1.set_zlabel('Z')

    ax2 = fig1.add_subplot(1, 2, 2, projection='3d')
    ax2.set_xlabel('x')
    ax2.set_ylabel('y')
    ax2.set_zlabel('z')

    plt.draw() #plt.pause(1);
    xx = np.linspace(view_x1[0], view_x1[1], 20)
    yy = np.linspace(view_x2[0], view_x2[1], 20)
    X, Y = np.meshgrid(xx, yy)
    Z = Pavirsius(X, Y, gauti_bendra_lygti(Z1, Z2))

    # surf1 = ax1.plot_surface(X, Y, Z[:, :, 0], color='blue', alpha=0.4, linewidth=0.1, antialiased=True)
    wire1 = ax1.plot_wireframe(X, Y, Z[:, :, 0], color='black', alpha=1, linewidth=1, antialiased=True)
    surf2 = ax1.plot_surface(X, Y, Z[:, :, 1], color='purple', alpha=0.4, linewidth=0.1, antialiased=True)

    CS11 = ax1.contour(X, Y, Z[:, :, 0], [0], colors='b')
    CS12 = ax1.contour(X, Y, Z[:, :, 1], [0], colors='g')
    CS1 = ax2.contour(X, Y, Z[:, :, 0], [0], colors='b')
    CS2 = ax2.contour(X, Y, Z[:, :, 1], [0], colors='g')

    XX = np.linspace(-5,5,2)
    YY = XX
    XX, YY = np.meshgrid(XX, YY)
    ZZ = XX*0
    zeroplane = ax2.plot_surface(XX, YY, ZZ, color='gray', alpha=0.4, linewidth=0, antialiased=True)
    #-----

    init_y1 = Z1(init_x1, init_x2)
    init_y2 = Z2(init_x1, init_x2)
    ax1.plot3D([init_x1, init_x1], [init_x2, init_x2], [0, init_y1], "m*-")
    plt.draw()
    plt.pause(1)

    prev_x = (init_x1, init_x2)
    prev_y = (init_y1, init_y2)
    final_rez = None
    for rez in broideno_metodas_iter(init_x1, init_x2, Z1, Z2):
        SpausdintiMatrica(rez.tikslumas, T, "tiksl")
        if rez.tikslumas < EPSILON:
            final_rez = rez
            break

    SpausdintiMatrica(np.matrix([[rez.x1], [rez.x2]]), T, "x1")

    #----- Grafika: -----
    ax1.plot3D([rez.x1, rez.x1], [rez.x2, rez.x2], [0, 0], "ro-") # reikia prideti antra
    indeksa, kadangi x yra matrica
    ax1.plot3D([rez.x1, rez.x1], [rez.x2, rez.x2], [rez.y1, rez.y1], "c-")
    ax1.plot3D([rez.x1, rez.x1], [rez.x2, rez.x2], [0, rez.y1], "m*-")
    ax2.plot3D([rez.x1, rez.x1], [rez.x2, rez.x2], [0, 0], "ro-")

```



```

plt.draw()
plt.pause(2)
#-----
prev_x = (rez.x1, rez.x2)
prev_y = (rez.y1, rez.y2)

#----- Grafika: -----
ax1.plot3D([prev_x[0],prev_x[0]], [prev_y[1],prev_y[1]], [0,0], "ks")
ax2.plot3D([prev_x[0],prev_x[0]], [prev_y[1],prev_y[1]], [0,0], "ks")
plt.draw()
plt.pause(1)
#-----

assert final_rez
SpausdintiMatrica(np.matrix([[final_rez.x1], [final_rez.x2]]), T, "Sprendinys")
SpausdintiMatrica(final_rez.tikslumas, T, "Galutinis tikslumas")

# print("Plotting pre-finished")
# plt.show()
# print("Plotting finished")

def iter_tinklelis(tinklelis: Tinklelis):
    for x1_idx in range(tinklelis.density):
        x1 = lerp(tinklelis.from_x1, tinklelis.to_x1, (x1_idx + 0.5)/tinklelis.density)
        for x2_idx in range(tinklelis.density):
            x2 = lerp(tinklelis.from_x2, tinklelis.to_x2, (x2_idx + 0.5)/tinklelis.density)

            yield (x1, x2)

def viz_broideno_tinklelis(
    Z1,
    Z2,
    tinklelis: Tinklelis,
    precision = 5,
    circle_colors = [
        (0.1, 0.1, 0.1),
        (0.1, 0.8, 0.8),
        (0.1, 0.8, 0.1),
        (0.1, 0.1, 0.8),
        (0.8, 0.8, 0.1),
        (0.8, 0.1, 0.8),
        (0.4, 0.4, 0.4)
    ]
):
    tinklelio_reiksmes = {}
    sprendiniai = []
    for x1, x2 in iter_tinklelis(tinklelis):
        broideno_rez = broideno_metodas(x1, x2, Z1, Z2)
        if broideno_rez:
            rez_x1 = round(broideno_rez.x1, precision)
            rez_x2 = round(broideno_rez.x2, precision)

            tinklelio_reiksmes[(x1, x2)] = (rez_x1, rez_x2)
            if (rez_x1, rez_x2) not in sprendiniai:
                sprendiniai.append((rez_x1, rez_x2))
        else:
            tinklelio_reiksmes[(x1, x2)] = None

    print(sprendiniai)

    fig1 = plt.figure(1)

    ax = fig1.add_subplot(1, 1, 1)
    ax.set_xlabel('x1')
    ax.set_ylabel('x2')
    ax.set_xlim(tinklelis.from_x1, tinklelis.to_x1)
    ax.set_ylim(tinklelis.from_x2, tinklelis.to_x2)
    plt.draw()

    xx = np.linspace(tinklelis.from_x1, tinklelis.to_x1, tinklelis.density)
    yy = np.linspace(tinklelis.from_x2, tinklelis.to_x2, tinklelis.density)
    X, Y = np.meshgrid(xx, yy)
    Z = Pavirsius(X, Y, gauti_bendra_lygti(Z1, Z2))

```

```

ax.contour(X, Y, Z[:, :, 0], [0], colors=[(0.8, 0.4, 0.1, 1)], linewidths=4.0)
ax.contour(X, Y, Z[:, :, 1], [0], colors=[(0.8, 0.1, 0.4, 1)], linewidths=4.0)

assert (len(sprendiniai)+1) <= len(circle_colors)

circle_width = (tinklelis.to_x1 - tinklelis.from_x1) / tinklelis.density / 2
circle_height = (tinklelis.to_x2 - tinklelis.from_x2) / tinklelis.density / 2
circle_radius = min(circle_width, circle_height) * 0.85

for i in range(len(sprendiniai)):
    x1, x2 = sprendiniai[i]
    pos = np.array([x1, x2])
    star_vertices = np.array([
        [0, 1], [0.3, 0.3], [1, 0.3], [0.45, 0], [0.6, -0.7],
        [0, -0.35], [-0.6, -0.7], [-0.45, 0], [-1, 0.3], [-0.3, 0.3]
    ]) * (circle_radius*1.5)
    star = patches.Polygon(star_vertices + pos, closed=True, edgecolor='black',
        facecolor=circle_colors[i+1], zorder=5)
    ax.add_patch(star)

for x1, x2 in iter_tinklelis(tinklelis):
    sprendinys_idx = 0
    if tinklelio_reiksmes[(x1, x2)] is not None:
        sprendinys_idx = sprendiniai.index(tinklelio_reiksmes[(x1, x2)]) + 1

    circle_color = circle_colors[sprendinys_idx]

    ax.add_patch(patches.Circle((x1, x2), circle_radius, edgecolor=circle_color,
        facecolor=circle_color))

plt.show()

def lerp(min_x, max_x, percent):
    return min_x + (max_x - min_x) * percent

def main(Z1, Z2, tinklelis: Tinklelis):
    viz_broideno_metodas(2.5, 0.3, Z1, Z2)

    viz_broideno_tinklelis(Z1, Z2, tinklelis)

    print(broideno_metodas(-4, 0, Z1, Z2))
    print(broideno_metodas( 4, 0, Z1, Z2))
    print(broideno_metodas(-2, 4, Z1, Z2))
    print(broideno_metodas( 2, 4, Z1, Z2))

# Variantas: 10
main(
    lambda x1, x2: x1**2 + 2*(x2 - np.cos(x1))**2 - 20,
    lambda x1, x2: x1**2 * x2 - 2,
    tinklelis = Tinklelis(
        from_x1=-8, to_x1=8,
        from_x2=-8, to_x2=8,
        density=60
    )
)

```

Trečia užduotis

Optimizavimas

Miestas išsidėstęs kvadrate, kurio koordinatės ($-10 \leq x \leq 10$, $-10 \leq y \leq 10$). Mieste yra n ($n \geq 3$) vieno tinklo parduotuvių, kurių koordinatės yra žinomos (Koordinatės gali būti generuojamos atsitiktinai, negali būti kelios parduotuvės toje pačioje vietoje). Planuojama pastatyti dar m ($m \geq$

3) šio tinklo parduotuvių. Parduotuvės pastatymo kaina (vietos netinkamumas) vertinama pagal atstumus iki kitų parduotuvių ir poziciją (koordinates). Reikia parinkti naujų parduotuvių vietas (koordinates) taip, kad parduotuvių pastatymo kainų suma būtų kuo mažesnė (naujos parduotuvės gali būti statomos ir už miesto ribos). Atstumo tarp dviejų parduotuvių, kurių koordinatės (x_1, y_1) ir (x_2, y_2) , kaina apskaičiuojama pagal formulę:

$$C(x_1, y_1, x_2, y_2) = \exp(-0.3((x_1 - x_2)^2 + (y_1 - y_2)^2))$$

Parduotuvės, kurios koordinatės (x_1, y_1) , vietos kaina apskaičiuojama pagal formulę:

$$C^p(x_1, y_1) = \frac{x_1^4 + y_1^4}{1000} + \frac{\sin(x_1) + \cos(y_1)}{5} + 0.4$$

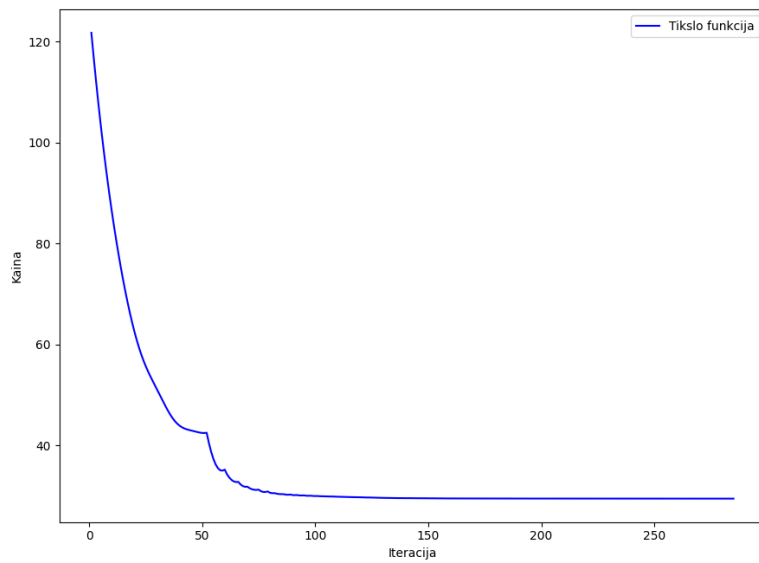
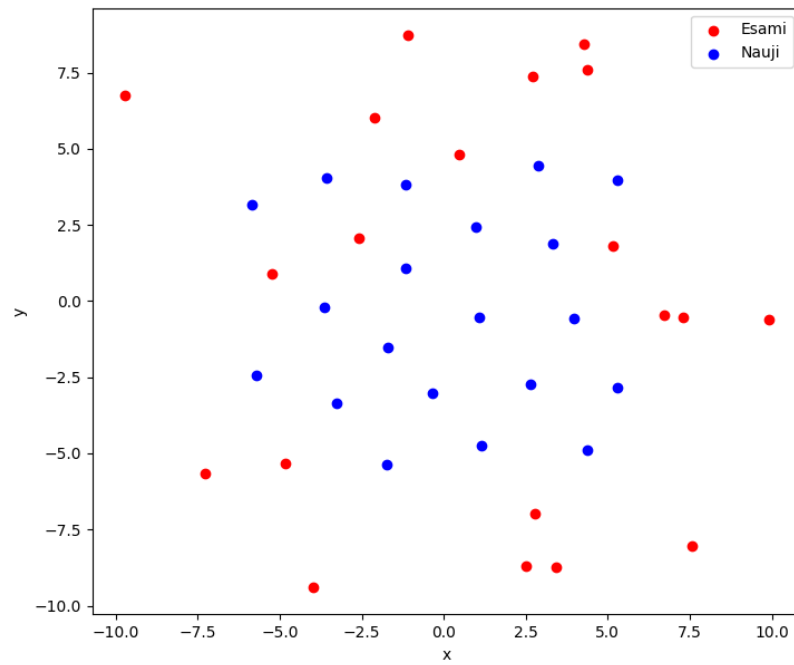
Skaičiavimo metodas

Naudotas gradientų metodas su tokiais parametrais:

- Maksimalus iteracijų skaičius = 1000
- Iteracijos žingsnio dydis = 0.5
- Esamų parduotuvių kiekis = 20
- Naujų parduotuvių kiekis = 20
- Nutraukimo sąlygos epsilon = 1e-6
- Gradientų žingsnio dydis = 0.1

Tikslo funkcija sumuoja: parduotuvės vietos kainą, parduotuvės atstumo kainą su visomis kitomis parduotuvėmis.

Rezultatai



Kodas

```
from dataclasses import dataclass
from random import Random
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches

@dataclass
class Point:
    x: float
    y: float
```

```

@staticmethod
def rand(rand: Random, from_x: float, to_x: float, from_y: float, to_y: float):
    return Point(
        rand.uniform(from_x, to_x),
        rand.uniform(from_y, to_y)
    )

def gen_points(
    rand: Random,
    n: int,
    x_range = (-10, 10),
    y_range = (-10, 10)
):
    points = []
    while len(points) < n:
        point = Point.rand(rand, x_range[0], x_range[1], y_range[0], y_range[1])
        if point not in points:
            points.append(point)
    return points

def get_total_price(
    shops: list[Point],
    new_shops: list[Point],
    C,
    Cp
):
    total_price = 0
    for i, point in enumerate(new_shops):
        total_price += Cp(point.x, point.y)
        for other_point in shops:
            total_price += C(point.x, point.y, other_point.x, other_point.y)

        for j, other_point in enumerate(new_shops):
            if i == j: continue
            total_price += C(point.x, point.y, other_point.x, other_point.y)
    return total_price

def get_price_gradient(
    shops: list[Point],
    new_shops: list[Point],
    C,
    Cp,
    step = 0.001
):
    grad = []
    current_price = get_total_price(shops, new_shops, C, Cp)
    for i in range(len(new_shops)):
        new_shops[i].x += step
        new_price_x = get_total_price(shops, new_shops, C, Cp)
        new_shops[i].x -= step

        new_shops[i].y += step
        new_price_y = get_total_price(shops, new_shops, C, Cp)
        new_shops[i].y -= step

        grad.append(Point(
            new_price_x - current_price,
            new_price_y - current_price
        ))

    L = 0
    for grad_point in grad:
        L += grad_point.x**2 + grad_point.y**2
    L = L**0.5

    for grad_point in grad:
        grad_point.x /= L
        grad_point.y /= L

    return grad

def apply_gradient(points: list[Point], gradient: list[Point], step_size: float):
    for i, point in enumerate(points):
        point.x += step_size * gradient[i].x

```

```

        point.y += step_size * gradient[i].y

def gradient_descent(
    shops: list[Point],
    new_shops: list[Point],
    C,
    Cp,
    max_iterations = 100,
    step_size = 0.5,
    epsilon = 1e-6
):
    total_price = 1e10
    price_gradient = get_price_gradient(shops, new_shops, C, Cp)
    for iteration_idx in range(max_iterations):
        apply_gradient(new_shops, price_gradient, -step_size)

        new_total_price = get_total_price(shops, new_shops, C, Cp)
        if abs(total_price - new_total_price) < epsilon:
            return iteration_idx+1

        if total_price > new_total_price:
            total_price = new_total_price
        else:
            apply_gradient(new_shops, price_gradient, +step_size)
            price_gradient = get_price_gradient(shops, new_shops, C, Cp)
            step_size *= 0.9

    return -1

def main(
    N: list[Point],
    m: int,
    C,
    Cp,
    rand,
    max_iterations,
    step_size,
):
    shops = N
    new_shops = gen_points(rand, m)

    print("Starting price: ", get_total_price(shops, new_shops, C, Cp))

    iterations_used = gradient_descent(shops, new_shops, C, Cp, max_iterations, step_size)
    if iterations_used == -1:
        print("ERROR: Failed to reach minimum, not enough iterations")
        return

    print("Iterations: ", iterations_used)
    print("Minumum price: ", get_total_price(shops, new_shops, C, Cp))

    plt.scatter(list(p.x for p in shops), list(p.y for p in shops), c="r", label="Esami")

    plt.scatter(list(p.x for p in new_shops), list(p.y for p in new_shops), c="b", label="Nauji")
    plt.legend()

    plt.xlabel("x")
    plt.ylabel("y")

    plt.show()

# Variantas: 10
rand = Random(3)
main(
    rand = rand,
    N = gen_points(rand, 20),
    m = 20,
    C = lambda x1, y1, x2, y2: np.exp(-0.3 * ((x1 - x2)**2 + (y1 - y2)**2)),
    Cp = lambda x1, y1: (x1**4 + y1**4)/1000 + (np.sin(x1) + np.cos(y1))/5 + 0.4,

    max_iterations = 1000,
    step_size=0.5
)

```

