Kauno technologijos universitetas
Informatikos fakultetas

Skaitiniai metodai ir algoritmai

Lygčių sistemų sprendimas ir optimizavimas

Vytenis Kriščiūnas IFF-1/1

Studentas

doc. Kriščiūnas Andrius

Dėstytojas

Kaunas 2023

TURINYS

1. Pirma dalis (tiesinių lygčių sistemų sprendimas)

a) 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.
b) 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:

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

1.1. A dalies sprendimas

1.1.1. Užduotis

9	Gauso	5, 13, 19
	Gauso-Zeidelio	5

5	$\begin{cases} 4x_1 + 12x_2 + x_3 + 7x_4 = 171 \\ 2x_1 + 6x_2 + 17x_3 + 2x_4 = 75 \\ 2x_1 + x_2 + 5x_3 + x_4 = 30 \\ 5x_1 + 11x_2 + 7x_3 = 50 \end{cases}$
13	$\begin{cases} x_1 - 2x_2 + 3x_3 + 4x_4 = 11 \\ x_1 - x_3 + x_4 = -4 \\ 2x_1 - 2x_2 + 2x_3 + 5x_4 = 7 \\ -7x_2 + 3x_3 + x_4 = 2 \end{cases}$
19	$\begin{cases} 3x_1 + x_2 - x_3 + 5x_4 = 8 \\ -3x_1 + 4x_2 - 8x_3 - x_4 = 10 \\ x_1 - 3x_2 + 7x_3 + 6x_4 = 11 \\ 5x_2 - 9x_3 + 4x_4 = 1 \end{cases}$

1.1.2. Gauso metodu spręstos lygtys

Gauso metodas:

```
def Gous(A1):
    ar = "Veina" #Skirtas singuliarumo salygai
    for i in range (0,n-1):
        a, iii = np.max(np.abs(A1[i:n, i])), np.argmax(np.abs(A1[i:n, i])) + i

        if a == 0:
            continue

        if iii > i:
            A1[[i, iii], :] = A1[[iii, i], :]

        for j in range (i+1,n):
            A1[j,i:n+1]=A1[j,i:n+1]-A1[i,i:n+1]*A1[j,i]/A1[i,i]
            A1[j,i]=0


    #Grizimas atgal
    x=np.zeros(shape=(n,1))
    for i in range (n-1,-1,-1):
        if (A1[i,i] == 0 and A1[i,n:n+1] == 0):
            print("Lygtis turi begalo daug sprendiniu")
            ar = "Daug"
            x[i,:] = 1
        elif (A1[i,i] == 0 and A1[i,n:n+1] != 0):
            print("Lygtis neturi sprendiniu")
            return None, ar
        else:
            x[i,:]=(A1[i,n:n+1]-A1[i,i+1:n]*x[i+1:n,:])/A1[i,i]

    print(x)
    return x, ar
```

Gautų sprendinių patikrinimas:

```
def patikr(A, b, x):
    ats = 0
    for i in range(0, n):
        for j in range(0, n):
            ats = ats + A[i, j] * x[j]

        print('Duota reiksme: {0} ir gauta reiksme: {1}'.format(b[i], ats))
        ats = 0
```

5 Lygtis:

```
#5
print("Gauso metodas: 5 lygtis")
A=np.matrix([[4 , 12,  1,  7],
             [2, 6, 17,  2],
             [2,  1, 5,  1],
             [5,  11,  7, 0]]).astype(float)
b=(np.matrix([171,75,30,50])).transpose()

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

x, ar = Gous(A1)
if (x is not None):
    patikr(A, b, x)
    if (ar == "Viena"):
        print('Patikrinimas :\n {0}'.format(np.linalg.solve(A, b)))
    else:
        x, residuals, rank, s = np.linalg.lstsq(A, b, rcond=None)
        print('Patikrinimas :\n {0}'.format(x))
print()
```

Rezultatai:

```
Gauso metodas: 5 lygtis
[[ 2.]
 [ 3.]
 [ 1.]
 [18.]]
Duota reiksme: [[171]] ir gauta reiksme: [171.]
Duota reiksme: [[75]] ir gauta reiksme: [75.]
Duota reiksme: [[30]] ir gauta reiksme: [30.]
Duota reiksme: [[50]] ir gauta reiksme: [50.]
Patikrinimas :
 [[ 2.]
 [ 3.]
 [ 1.]
 [18.]]
```

13 Lygtis:

```
#13
ar = "Viena"
print("Gauso metodas: 13 lygtis")
A=np.matrix([[1 , -2,  3,  4],
             [1, 0, -1,  1],
             [2,  -2, 2,  5],
             [0,  -7,  3, 1]]).astype(float)
b=(np.matrix([11,-4,7,2])).transpose()

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

x, ar = Gous(A1)
if (x is not None):
    patikr(A, b, x)
    if (ar == "Viena"):
        print('Patikrinimas :\n {0}'.format(np.linalg.solve(A, b)))
    else:
        x, residuals, rank, s = np.linalg.lstsq(A, b, rcond=None)
        print('Patikrinimas :\n {0}'.format(x))
print()
```

Rezultatai:

```
Gauso metodas: 13 lygtis
Lygtis turi begalo daug sprendiniu
[[-1.27272727]
 [ 1.45454545]
 [ 3.72727273]
 [ 1.        ]]
Duota reiksme: [[11]] ir gauta reiksme: [11.]
Duota reiksme: [[-4]] ir gauta reiksme: [-4.]
Duota reiksme: [[7]] ir gauta reiksme: [7.]
Duota reiksme: [[2]] ir gauta reiksme: [2.]
Patikrinimas :
 [[-1.33555468]
 [ 1.44688358]
 [ 3.69815759]
 [ 1.03371227]]
```

19 Lygtis

```
#19
print("Gauso metodas: 19 lygtis")
A=np.matrix([[3 , 1,  -1,  5],
             [-3, 4, -8,  -1],
             [1,  -3, 7,  6],
             [0,  5,  -9, 4]]).astype(float)
b=(np.matrix([8,10,11,1])).transpose()

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

x, ar = Gous(A1)
if (x is not None):
    patikr(A, b, x)
    if (ar == "Viena"):
        print('Patikrinimas :\n {0}'.format(np.linalg.solve(A, b)))
    else:
        x, residuals, rank, s = np.linalg.lstsq(A, b, rcond=None)
        print('Patikrinimas :\n {0}'.format(x))
print()
```

Rezultatai:

```
Gauso metodas: 19 lygtis
Lygtis neturi sprendiniu
```

1.1.3. Gauso-Zeidelio metodu spręsta lygtis

5 Lygtis

```
def GZeid(A, b, n):
    P=np.arange(0,n)
    for i in range (0,n):

        if (np.diag(A)[i] == 0): #Negali vykti dalyba is 0
            iii = np.argmax(np.abs(A[0:n, i]))

            A[[i, iii], :] = A[[iii, i], :]
            P[[i,iii]]=P[[iii,i]]
    b=b[P]

    alpha=np.array([2, 2, 2, 2])
    Atld=np.diag(1./np.diag(A)).dot(A)-np.diag(alpha)
    btld=np.diag(1./np.diag(A)).dot(b)

    nitmax=1000; eps=1e-12

    x=np.zeros(shape=(n,1)); x1=np.zeros(shape=(n,1))

    for it in range (0,nitmax):
        for i in range (0,n):
            x1[i]=(btld[i]-Atld[i,:].dot(x1))/alpha[i]

        if (math.isinf(np.linalg.norm(x)+np.linalg.norm(x1))):
            print("Lygtis neturi sprendiniu arba pasirinkta netinkama alpha
reiksme:")
            print(x)
            return None, b

        prec=(np.linalg.norm(x1-x)/(np.linalg.norm(x)+np.linalg.norm(x1)))
        if (prec < eps):
            return x, b
        x[:]=x1[:]
    print("Metodas diverguoja:")
    print(x)
    return None, b
```

```
#5 Zeidelio
print("Gauso-Zeidelio algoritmas: 5 lygtis")
A=np.matrix([[4 , 12,  1,  7],
             [2, 6, 17,  2],
             [2,  1, 5,  1],
             [5,  11,  7, 0]]).astype(float)
b=np.array([[171],[75],[30],[50]])
n=(np.shape(A))[0]

x, b = GZeid(A, b, n)
if (x is not None):
    print(x)
    patikr(A, b, x)
    print('Patikrinimas :\n {0}'.format(np.dot(A, x)))
print()
```

Rezultatai:

```
Gauso-Zeidelio algoritmas: 5 lygtis
Metodas diverguoja:
[[ 6.12769358e+52]
 [-3.12798393e+52]
 [-1.54483844e+52]
 [ 3.25158513e+52]]
```

1.2. B dalies sprendimas

1.2.1. Užduotis

9.	$\begin{cases} 5x_1 + 3x_2 - x_3 + 2x_4 = \cdots \\ 3x_1 + 6x_2 - 2x_3 - 2x_4 = \cdots \\ -x_1 - 2x_2 + 4x_3 - x_4 = \cdots \\ 2x_1 - 2x_2 - x_3 + 12x_4 = \cdots \end{cases}$	$\begin{cases} \ldots = 9 \\ \ldots = 5 \\ \ldots = 0 \\ \ldots = 11 \end{cases}$	$\begin{cases} \ldots = 26 \\ \ldots = 0 \\ \ldots = 20 \\ \ldots = 44 \end{cases}$	$\begin{cases} \ldots = -4.75 \\ \ldots = -5 \\ \ldots = 3.75 \\ \ldots = -7.25 \end{cases}$	QR

1.2.2. QR sklaidos metodu spręstos lygtys

```python
def QGavimas(A):

    Q=np.identity(n)
    for i in range (0,n-1):
        z=A[i:n,i]
        zp=np.zeros(np.shape(z))
        zp[0]=np.linalg.norm(z)

        omega=z-zp
        omega=omega/np.linalg.norm(omega)
        Qi=np.identity(n-i)-2*omega*omega.transpose()
        A[i:n,:]=Qi.dot(A[i:n,:]) #Trikampe matrica

        Q[:,i:n]= Q[:,i:n].dot(Qi) #Ortogonalioji matrica

    return Q



def QRAtgalinis(Q, A, b):

    b1=Q.transpose().dot(b)
    x=np.zeros(shape=(n,1))

    for i in range (n-1,-1,-1):
        if  (A[i,i] == 0 and b1[i,:] == 0):
            print("Lygtis turi begalo daug sprendiniu")
            x[i,:] = 1
        elif (A[i,i] == 0 and b1[i,:] != 0):
            print("Lygtis neturi sprendiniu")
            return
        else:
            x[i,:]=(b1[i,:]-A[i,i+1:n]*x[i+1:n,:])/A[i,i]

    return x
```

```
#QR sklaida
print("QR skaidos algoritmas")
A=np.matrix([[5 , 3,  -1,  2],
             [3, 6, -2,  -2],
             [-1,  -2, 4,  -1],
             [2,  -2,  -1, 12]]).astype(float)
Ap = np.copy(A)

#Laisvuju nariu vektoriai
b1 = np.array([[9],[5],[0],[11]])
b2 = np.array([[26],[0],[20],[44]])
b3 = np.array([[-4.75],[-5],[3.75],[-7.25]])

Q = QGavimas(A)
```

```
x = QRAtgalinis(Q, A, b1)
if (x is not None):
    print("Nezinomieji: ")
    print(x)

    print("b1 laisvieji nariai: ")
    patikr(Ap, b1, x)
    #Python tikrinimas
    Qi, Ri = np.linalg.qr(Ap)
    y = np.dot(Qi.transpose(), b1)
    x = np.linalg.solve(Ri, y)
    print('Patikrinimas :\n {0}'.format(x))
    print()

x = QRAtgalinis(Q, A, b2)
if (x is not None):
    print("Nezinomieji: ")
    print(x)

    print("b2 laisvieji nariai: ")
    patikr(Ap, b2, x)
    #Python tikrinimas
    Qi, Ri = np.linalg.qr(Ap)
    y = np.dot(Qi.transpose(), b2)
    x = np.linalg.solve(Ri, y)
    print('Patikrinimas :\n {0}'.format(x))
    print()
```

```python
x = QRAtgalinis(Q, A, b3)
if (x is not None):
    print("Nezinomieji: ")
    print(x)

    print("b3 laisvieji nariai: ")
    patikr(Ap, b3, x)
    #Python tikrinimas
    Qi, Ri = np.linalg.qr(Ap)
    y = np.dot(Qi.transpose(), b3)
    x = np.linalg.solve(Ri, y)
    print('Patikrinimas :\n {0}'.format(x))
    print()
```

```
Nezinomieji:
[[1.]
 [1.]
 [1.]
 [1.]]
b1 laisvieji nariai:
Duota reiksme: [9] ir gauta reiksme: [9.]
Duota reiksme: [5] ir gauta reiksme: [5.]
Duota reiksme: [0] ir gauta reiksme: [-2.22044605e-16]
Duota reiksme: [11] ir gauta reiksme: [11.]
Patikrinimas :
 [[1.]
 [1.]
 [1.]
 [1.]]

Nezinomieji:
[[4.]
 [2.]
 [8.]
 [4.]]
b2 laisvieji nariai:
Duota reiksme: [26] ir gauta reiksme: [26.]
Duota reiksme: [0] ir gauta reiksme: [-4.4408921e-15]
Duota reiksme: [20] ir gauta reiksme: [20.]
Duota reiksme: [44] ir gauta reiksme: [44.]
Patikrinimas :
 [[4.]
 [2.]
 [8.]
 [4.]]

Nezinomieji:
[[ 2.84444741e-16]
 [-1.00000000e+00]
 [ 2.50000000e-01]
 [-7.50000000e-01]]
b3 laisvieji nariai:
Duota reiksme: [-4.75] ir gauta reiksme: [-4.75]
Duota reiksme: [-5.] ir gauta reiksme: [-5.]
Duota reiksme: [3.75] ir gauta reiksme: [3.75]
Duota reiksme: [-7.25] ir gauta reiksme: [-7.25]
Patikrinimas :
 [[-7.11111853e-16]
 [-1.00000000e+00]
 [ 2.50000000e-01]
 [-7.50000000e-01]]
```

2. Antra dalis (Netiesinių lygčių sprendimas)

Duota netiesinių lygčių sistema (4 lentelė):

{ $Z1$ ($x1$, $x2$) = 0

{$Z2$ ($x1$, $x2$) = 0

a) Skirtinguose grafikuose pavaizduokite paviršius $Z1$ ($x1$, $x2$) ir $Z2$ ($x1$, $x2$).
b) Užduotyje pateiktą netiesinių lygčių sistemą išspręskite grafiniu būdu.
c) Nagrinėjamoje srityje sudarykite stačiakampį tinklelį ($x1$, $x2$ poras). Naudodami užduotyje nurodytą metodą apskaičiuokite netiesinių lygčių sistemos sprendinius, kai pradinis artinys įgyja tinklelio koordinačių 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).

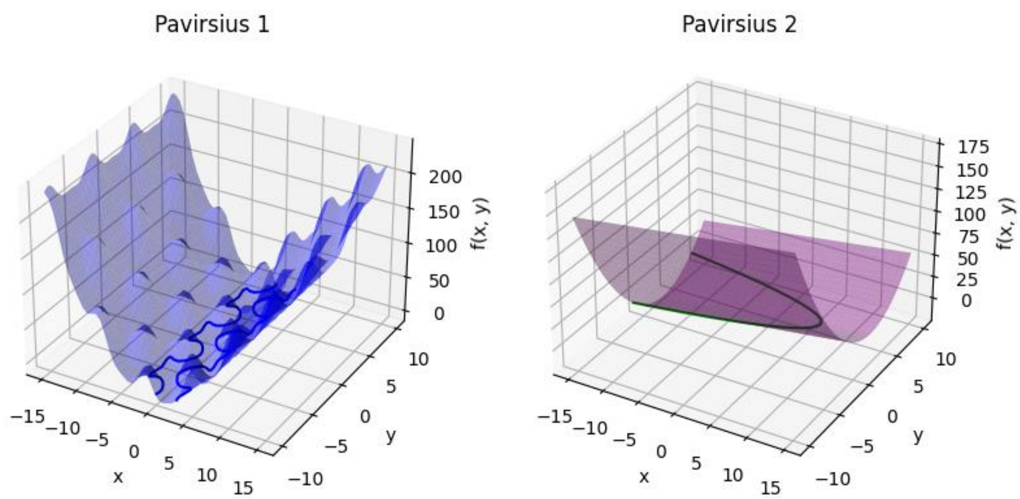
2.1. Užduotis

9	$\begin{cases} x_1^2 + 10(\sin(x_1) + \cos(x_2))^2 - 10 = 0 \\ (x_2 - 3)^2 + x_1 - 8 = 0 \end{cases}$	Niutono

2.2. A dalies sprendimas

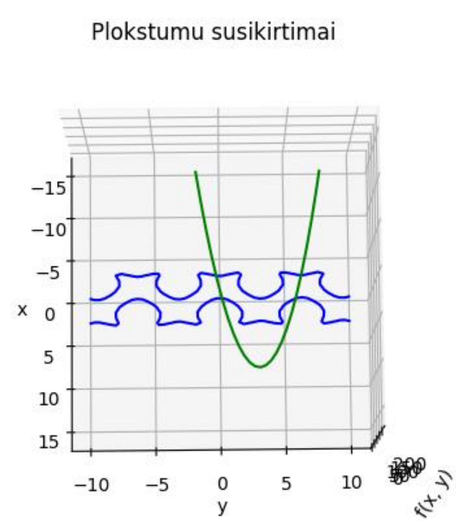
```
def LF(x):
    s=np.matrix( [[x[0]**2 + 10*(np.sin(x[0]) + np.cos(x[1]))**2 - 10], [(x[1] -
3)**2 + x[0] - 8]])
    return s

fig1=plt.figure(1,figsize=plt.figaspect(0.5)) #Abeji pavirseiai
fig2=plt.figure(2,figsize=plt.figaspect(0.5)) #Vienas pavirsius

ax1 = fig1.add_subplot(1, 2, 1, projection='3d')
ax2 = fig1.add_subplot(1, 2, 2, projection='3d')
ax1.set_title('Pavirsius 1')
ax2.set_title('Pavirsius 2')

ax3 = fig2.add_subplot(1, 1, 1, projection='3d')
ax3.set_title('Plokstumu susikirtimai')

ax1.set_xlabel('x')
ax1.set_ylabel('y')
```

```python
ax1.set_zlabel('f(x, y)')

ax2.set_xlabel('x')
ax2.set_ylabel('y')
ax2.set_zlabel('f(x, y)')

ax3.set_xlabel('x')
ax3.set_ylabel('y')
ax3.set_zlabel('f(x, y)')

plt.draw()

xx=np.linspace(-15,15,50)
yy=np.linspace(-10,10,50)
X, Y = np.meshgrid(xx, yy)
Z=np.zeros(shape=(len(xx),len(yy),2))


for i in range (0,len(xx)):
    for j in range (0,len(yy)):
        Z[i,j,:]=LF([X[i][j],Y[i][j]]).transpose()

surf1 = ax1.plot_surface(X, Y, Z[:,:,0], color='blue', alpha=0.4)
CS11 = ax1.contour(X, Y, Z[:,:,0],[0],colors='b')

surf2 = ax2.plot_surface(X, Y, Z[:,:,1], color='purple',alpha=0.4)
CS12 = ax2.contour(X, Y, Z[:,:,1],[0],colors='g')
plt.show()
```



2.3. B dalies sprendimas

```
CS1 = ax3.contour(X, Y, Z[:,:,0],[0],colors='b')
CS2 = ax3.contour(X, Y, Z[:,:,1],[0],colors='g')

plt.show()
```



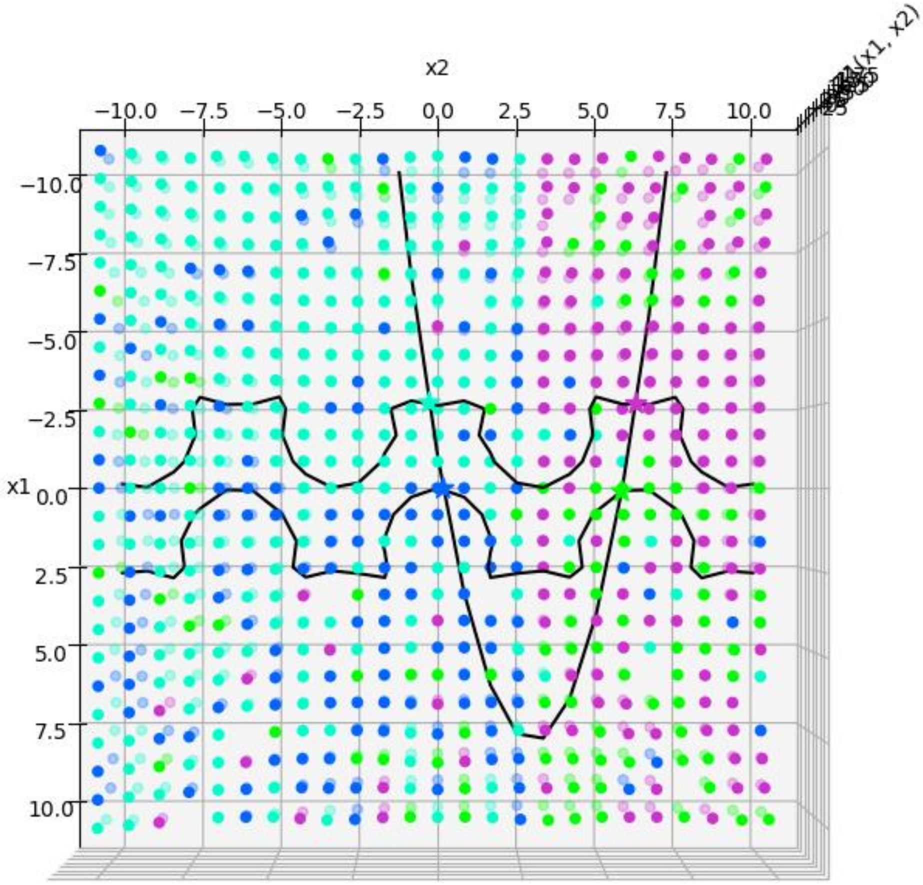
2.4. C dalies sprendimas

```
from tkinter import *
import numpy as np
from numpy import *
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import math
from scipy.optimize import fsolve

#--------Randama RGB spalva
def jetColormapValueToRGB(value) :
            N = 5;   # jet colormap vartoja 5 spalvu kubo virsunes
            #jetColors = [[0, 0, 1 ],[0, 1, 1 ], [0, 1, 0 ],[1, 1, 0 ],[1, 0, 
0]];
            jetColors = [[0, 0, 1 ],[0, 1, 1 ], [0, 1, 0 ],[1, 0, 1 ],[1, 0, 0]];
            if (value < 0) | (value > 1):
                #print("***** jetColormapValueToRGB:   value not in range [0,1]")
                return None

            for  i in range (0, N-1) :
                a = 1.0*i / (N - 1);  b = 1.0*(i+1) / (N - 1);  #print(a); 
print(b);
                if (value >= a) & (value <= b) :
                      rr = (value - a) / (b - a);  rgb=[];
                      for j in range (0,3) :  rgb.append(double(jetColors[i][j] * 
(1 - rr) + jetColors[i + 1][j] * rr));
                      break
            return rgb


#------Jakobino matrica
def numerical_jacobi(f,x,dx):
    n=np.size(f(x))
    m=np.size(x)
    J=np.matrix(np.zeros((n,m),dtype=float))
    x1=np.matrix(x)
    for j in range (m):
        x1[j]=x[j]+dx
        J[:,j]=(f(x1)-f(x))/dx
        x1[j]=x[j]
    return J

#--------Pradine funkcija
```

```python
def f(x):
    return np.vstack((x[0]**2 + 10*(np.sin(x[0]) + np.cos(x[1]))**2 - 10, (x[1] -
3)**2 + x[0] - 8))

#----------Netuno metodas
def Newton(xx):

    alpha = 0.5
    eps = 1e-5
    itmax = 200
    x=np.matrix([[xx[0]],[xx[1]]],dtype=float) #Pradiniai artiniai
    ff = f(x)
    dff = numerical_jacobi(f,x,eps)

    for iii in range(itmax):
        dff = numerical_jacobi(f,x,eps)

        deltax = -np.linalg.solve(dff, ff)
        x1 = x + alpha * deltax
        ff1 = f(x1)

        tikslumas = np.linalg.norm(deltax) / (np.linalg.norm(x) +
np.linalg.norm(deltax))

        #print(f'\n iteracija {iii+1}  tikslumas {tikslumas}')
        if tikslumas < eps:
            #print(f'\n sprendinys x = {x}')
            return x

        elif iii == itmax - 1:
            #print(f'\n ****tikslumas nepasiektas. Paskutinis artinys x = {x}')
            return None

        x = x1
        ff = ff1


xx=np.linspace(-10,10,25)
yy=np.linspace(-10,10,25)
X, Y = np.meshgrid(xx, yy)
Z=np.zeros(shape=(len(xx),len(yy),2))
Z1=np.zeros(shape=(len(xx),len(yy),2))

fig1=plt.figure(1,figsize=plt.figaspect(0.5))
ax = fig1.add_subplot(1, 1, 1, projection='3d')
```

```
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('Z1(x1, x2)')


solutions = []
AllRoots = []
pradArt = []

#-------------Skirtas nupaisyti funkcijas
for i in range (0,len(xx)):
    for j in range (0,len(yy)):
        initial_guess = [X[i][j], Y[i][j]]
        Z1[i,j,:]=f(initial_guess).transpose()


#------------Skirtas tasku sudejimui tinklelyje
for i in range (0,len(xx)):
    for j in range (0,len(yy)):

        initial_guess = [X[i][j], Y[i][j]]
        solution = Newton(initial_guess)

        if (solution is None):
            continue


        is_close = False
        for existing_solution in solutions:
            if np.linalg.norm(existing_solution - solution) < math.exp(-4):
                is_close = True
                break

        if not is_close:
            AllRoots.append(solution)
            pradArt.append(initial_guess)
            print('Sprendinyas: {0}'.format(solution))
            print('Pradinis artinys: {0}'.format(initial_guess))


        l = 0.1
        for s in AllRoots:
            if np.linalg.norm(s - solution) < math.exp(-2):
                ax.scatter(initial_guess[0], initial_guess[1],
f(initial_guess).transpose(), c=jetColormapValueToRGB(l))
```

```
            l = l + 0.2

        solutions.append(solution)

#------------Rastu saknu pavaizdavimas
i = 0.1
for sol in AllRoots:
    ax.scatter(sol[0], sol[1], f(sol).transpose(), c=jetColormapValueToRGB(i),
marker='*', s=100)
    i = i + 0.2


CS1 = ax.contour(X, Y, Z1[:,:,0],[0],colors='black')
CS2 = ax.contour(X, Y, Z1[:,:,1],[0],colors='black')

ax.view_init(elev=90, azim=0)
plt.show()
```



Sprendiniai	Pradiniai artiniai
[[0.01513307]; [0.17424909]]	[-10.0, -10.0]
[[-2.68873529];[-0.26936345]]	[-9.166666666666666, -10.0]
[[0.11004612];[5.8089138]]	[-5.833333333333333, -10.0]
[[-2.6629701];[6.26546211]]	[6.666666666666668, -8.333333333333334]

2.5. D dalies sprendimas (patikrinimas)

```
Sprendinyas: [[0.01513307]
 [0.17424909]]
Pradinis artinys: [-10.0, -10.0]
Gautu sprendiniu patikrinimas: [0.01513214 0.17424915]
```

```
Sprendinyas: [[-2.68873529]
 [-0.26936345]]
Pradinis artinys: [-9.166666666666666, -10.0]
Gautu sprendiniu patikrinimas: [-2.68871391 -0.26935986]
```

```
Sprendinyas: [[0.11004612]
 [5.8089138 ]]
Pradinis artinys: [-5.833333333333333, -10.0]
Gautu sprendiniu patikrinimas: [0.10998982 5.80891619]
```

```
Sprendinyas: [[-2.6629701 ]
 [ 6.26546211]]
Pradinis artinys: [6.666666666666668, -8.333333333333334]
Gautu sprendiniu patikrinimas: [-2.66299049  6.26542348]
```

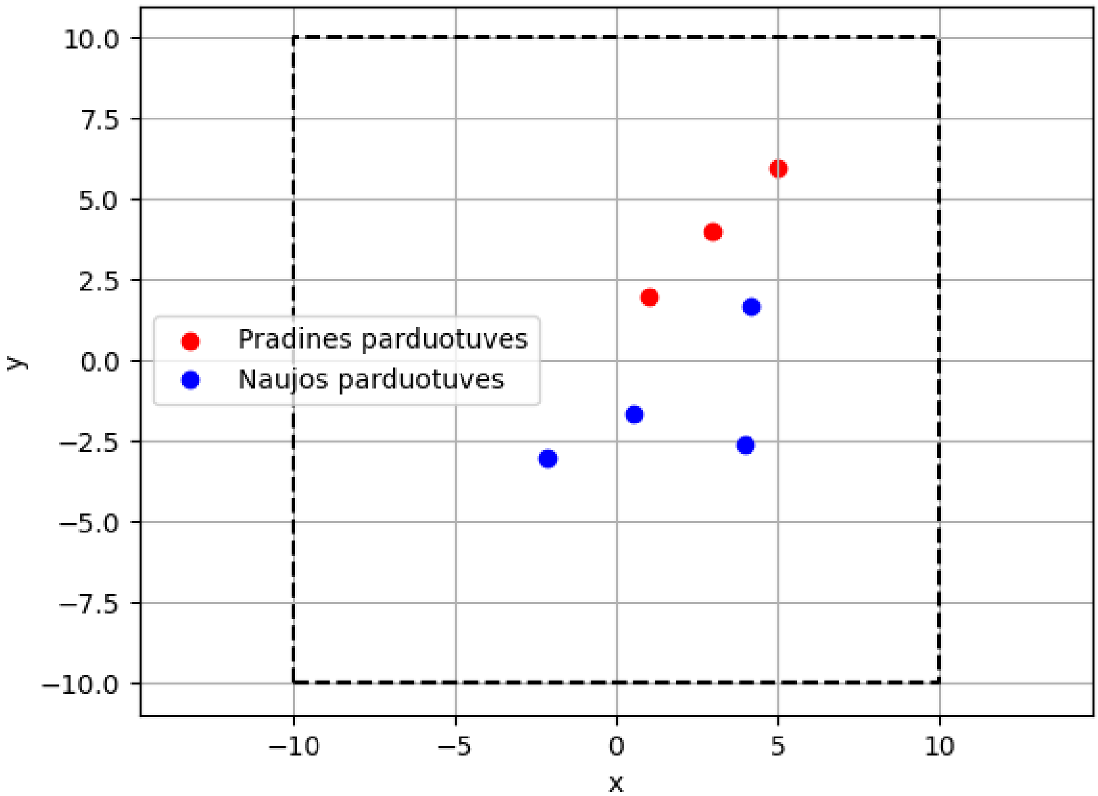
3. Trečia dalis (optimizavimas)

Pagal pateiktą uždavinio sąlygą (5 lentelė) sudarykite tikslo funkciją ir išspręskite jį vienu iš gradientinių metodų (gradientiniu, greičiausio nusileidimo). Gautą taškų konfigūraciją pavaizduokite programoje, skirtingais ženklais pavaizduokite duotus ir pridėtus (jei sąlygoje tokių yra) taškus. Ataskaitoje pateikite pradinę ir gautą taškų konfigūracijas, taikytos tikslo funkcijos aprašymą, taikyto metodo pavadinimą ir parametrus, iteracijų skaičių, iteracijų pabaigos sąlygas ir tikslo funkcijos priklausomybės nuo iteracijų skaičiaus grafiką.

3.1. Užduotis

Uždavinys 7-10 variantams
Miestas išsidėstęs kvadrate, kurio koordinatės ($-10 \le \boldsymbol{x} \le 10$, $-10 \le \boldsymbol{y} \le 10$). Mieste yra **n** ($\boldsymbol{n} \ge \mathbf{3}$) vieno tinklo parduotuvių, kurių koordinatės yra žinomos (*Koordinatės gali būti generuojamos atsitiktinai, negali būti kelios parduotuvės toje pačioje vietoje*). Planuojama pastatyti dar **m** ($\boldsymbol{m} \ge \mathbf{3}$) 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 $(\boldsymbol{x_1}, \boldsymbol{y_1})$ ir $(\boldsymbol{x_2}, \boldsymbol{y_2})$, kaina apskaičiuojama pagal formulę: $$C(x_1, y_1, x_2, y_2) = \exp(-0.3 \cdot ((x_1 - x_2)^2 + (y_1 - y_2)^2))$$ Parduotuvės, kurios koordinatės $(\boldsymbol{x_1}, \boldsymbol{y_1})$, vietos kaina apskaičiuojama pagal formulę: $$C^P(x_1, y_1) = \frac{x_1^4 + y_1^4}{1000} + \frac{\sin(x_1) + \cos(y_1)}{5} + 0.4$$

Gauta taškų konfiguracija




```
Pradine tasku konfiguracija: [(8, 6), (9, 1), (4, 3), (9, 9)]
Gauta tasku konfiguracija x:[ 4.14942378  3.98841759  0.56053848 -2.1205308 ] ir y:[ 1.69602839 -2.6225953  -1.66059109 -3.03883704]
```

3.2. Tikslo funkcijos aprašymas

Tikslo funkcijos prasmė šiame uždavinyje yra rasti pačią mažiausią kainų sumą, kuri priklauso nuo parduotuvės pastatymo vietos ir atstumo tarp parduotuvių.

3.3. Taikyto metodo pavadinimas

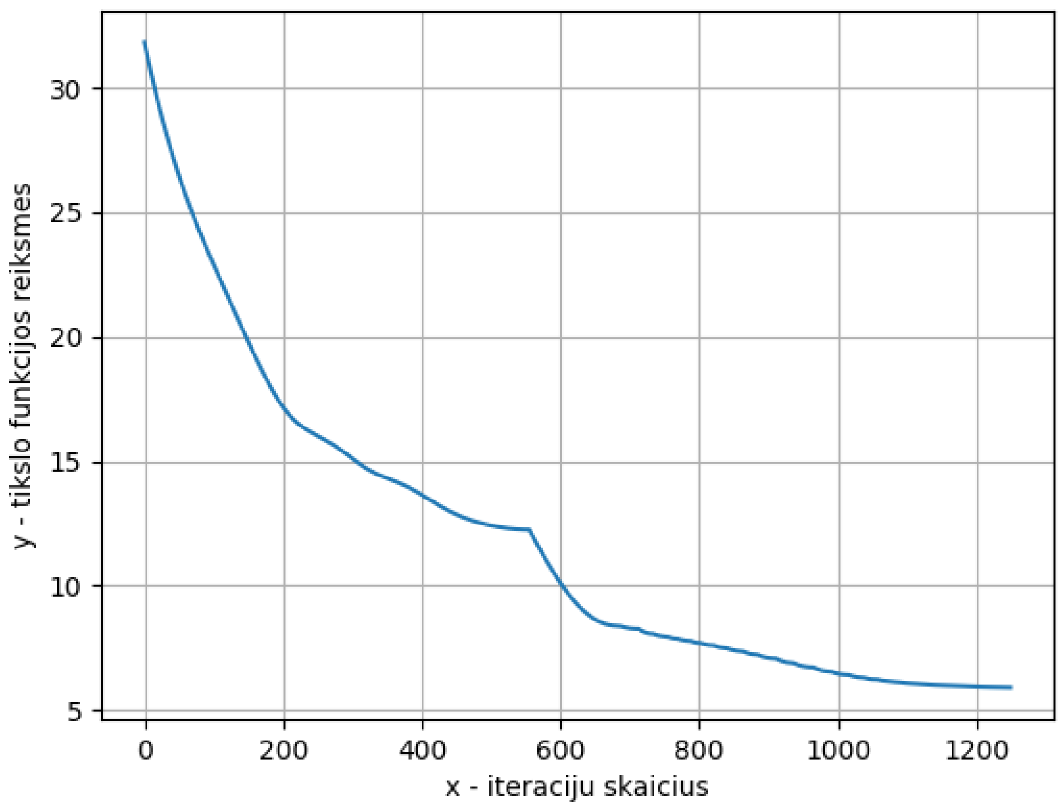
Taikytas greičiausio nusileidimo metodas, gradientas perskaičiuojamas tik kai reikšmė ima augti.

Parametrai: x ir y reikšmės, argumento prieaugis: 0.001.

Iteracijų skaičius: 1250.

Iteracijų pabaigos sąlyga: ciklas sukasi tol kol pasieka iteracijų skaičiaus pabaigą.

3.4. Funkcijos priklausomybės nuo iteracijų skaičiaus grafikas



3.5. Programos kodas

```
import numpy as np
import matplotlib.pyplot as plt

existing_stores = [(1, 2), (3, 4), (5, 6)]

# Funkcija apskaičiuojanti kainą tarp dviejų parduotuvių
def C(x1, y1, x2, y2):
    return np.exp(-0.3 * ((x1 - x2)**2 + (y1 - y2)**2))

# Funkcija apskaičiuojanti kainą naujos parduotuvės vietoje
def CP(x1, y1):
    return (x1**4 + y1**4) / 1000 + (np.sin(x1) + np.cos(y1)) / 5 + 0.4


# Tikslo funkcija - suma visų parduotuvių kainų
def objective_function(x, y):
```

```
    total_cost = 0

    # Sąnaudos dėl parduotuvių vietų
    for i in range(len(x)):
        total_cost += CP(x[i], y[i])


    # Sąnaudos dėl atstumų
    for i in range(len(x)):
        for x2, y2 in existing_stores:
            total_cost += C(x[i], y[i], x2, y2)

    for i in range(len(x)):
        for j in range(len(x)):
            total_cost += C(x[i], y[i], x[j], y[j])

    return total_cost

TaskuSkaicius = 3
areaLim = 10
new_stores = [(8, 6), (9, 1), (4, 3), (9, 9)]
x = [float(x) for x, y in new_stores]
y = [float(y) for x, y in new_stores]

fig=plt.figure(0)
ax=fig.add_subplot(1,1,1)
ax.set_xlabel('x')
ax.set_ylabel('y')
plt.xlim([-areaLim-2, areaLim+2])
plt.ylim([-areaLim-2, areaLim+2])
plt.plot([-areaLim, -areaLim, areaLim, areaLim, -areaLim], [-areaLim, areaLim,
areaLim, -areaLim, -areaLim],'--k')
ax.scatter(*zip(*existing_stores), c='red', marker='o', label='Pradines
parduotuves')


#Gradiantas
def dTF(x,y,h):
     n=len(x)
     gradx=np.zeros(n,dtype=float)
     grady=np.zeros(n,dtype=float)

     for i in range (n):

       x1=np.array(x)
```

```
       y1=np.array(y)

       x1[i]=x1[i]+h
       y1[i]=y1[i]+h

       gradx[i]=(objective_function(x1,y)-objective_function(x,y))/h #h -
argumentu prieaugis
       grady[i]=(objective_function(x,y1)-objective_function(x,y))/h

       L=np.linalg.norm([gradx,grady])
       gradx=gradx/L
       grady=grady/L
     return gradx,grady

TFValues = []
Iterations = []

#Vyksta optimizavimas (greičiausias nusileidimas)
step=0.01*TaskuSkaicius
print('pradine funkcijos reiksme',objective_function(x, y))
TFmin=1e10
gradx,grady=dTF(x,y,0.001)
for j in range (1250):
  x=x-step*gradx
  y=y-step*grady
  tf=objective_function(x,y)

  #---Grafikui
  TFValues.append(tf)
  Iterations.append(j)
  #---

  if TFmin  > tf:
    TFmin=tf

  else:
    x=x+step*gradx
    y=y+step*grady
    gradx, grady=dTF(x,y,0.001)


print('minimizuota funkcijos reiksme',objective_function(x,y))

#Pradiniai ir galutiniai taskai
print('Pradine tasku konfiguracija: {0}'.format(new_stores))
```

```
print('Gauta tasku konfiguracija x:{0} ir y:{1}'.format(x, y))

ax.plot(x,y,'bo', label='Naujos parduotuves')
plt.axis('equal')
plt.legend()
plt.grid()

#Iteraciju / tikslo funkcijos grafikas
fig=plt.figure(1)
ax1=fig.add_subplot(1,1,1)
ax1.plot(Iterations, TFValues)
ax1.set_xlabel('x - iteraciju skaicius')
ax1.set_ylabel('y - tikslo funkcijos reiksmes')

plt.grid()
plt.show()
```