

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) 3

1.1. A dalies sprendimas 3

1.1.1. Užduotis 3

1.1.2. Gauso metodu spręstos lygtys 3

1.1.3. Gauso-Zeidelio metodu spręsta lygtis 8

1.2. B dalies sprendimas 9

1.2.1. Užduotis 9

1.2.2. QR sklaidos metodu spręstos lygtys 9

2. Antra dalis (Netiesinių lygčių sprendimas) 14

2.1. Užduotis 14

2.2. A dalies sprendimas 14

2.3. B dalies sprendimas 16

2.4. C dalies sprendimas 17

2.5. D dalies sprendimas (patikrinimas) 21

3. Trečia dalis (optimizavimas) 22

3.1. Užduotis 22

3.2. Tikslo funkcijos aprašymas 23

3.3. Taikyto metodo pavadinimas 23

3.4. Funkcijos priklausomybės nuo iteracijų skaičiaus grafikas 24

3.5. Programos kodas 24

# Pirma dalis (tiesinių lygčių sistemų sprendimas)

1. 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.
2. Lentelėje 3 duotos tiesinės lygčių sistemos, laisvųjų narių vektoriai ir nurodytas skaidos metodas. Reikia suprogramuoti nurodytą metodą ir juo išspręsti pateiktas lygčių sistemas.

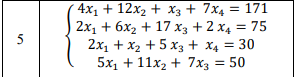
Sprendžiant lygčių sistemas (a ir b punktuose), turi būti:

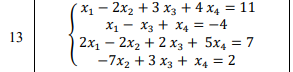
1. Programoje turi būti įvertinti atvejai:
   * kai lygčių sistema turi vieną sprendinį;
   * kai lygčių sistema sprendinių neturi;
   * kai lygčių sistema turi be gali daug sprendinių.
2. Patikrinkite gautus sprendinius ir skaidas, įrašydami juos į pradinę lygčių sistemą.
3. Gautą sprendinį patikrinkite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

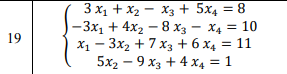
## A dalies sprendimas

### Užduotis









### 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)))

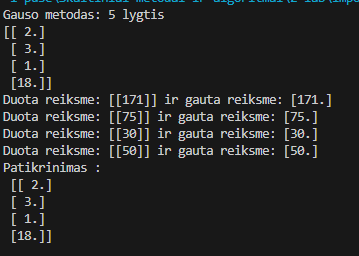
    else:

        x, residuals, rank, s = np.linalg.lstsq(A, b, rcond=None)

        print('Patikrinimas :\n {0}'.format(x))

print()

**Rezultatai:**

****

**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)))

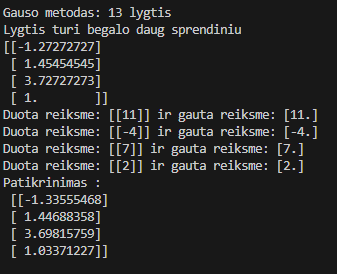
    else:

        x, residuals, rank, s = np.linalg.lstsq(A, b, rcond=None)

        print('Patikrinimas :\n {0}'.format(x))

print()

**Rezultatai:**

****

**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-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):

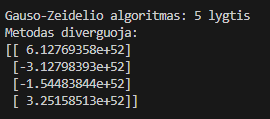
    print(x)

    patikr(A, b, x)

    print('Patikrinimas :\n {0}'.format(np.dot(A, x)))

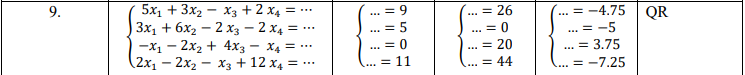
print()

**Rezultatai:**

****

## B dalies sprendimas

### Užduotis



### QR sklaidos metodu spręstos lygtys

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()

x = QRAtgalinis(Q, A, b3)

if (x is not None):

    print("Nezinomieji: ")

    print(x)

    print("b3 laisvieji nariai: ")

    patikr(Ap, b3, x)

    #Python tikrinimas

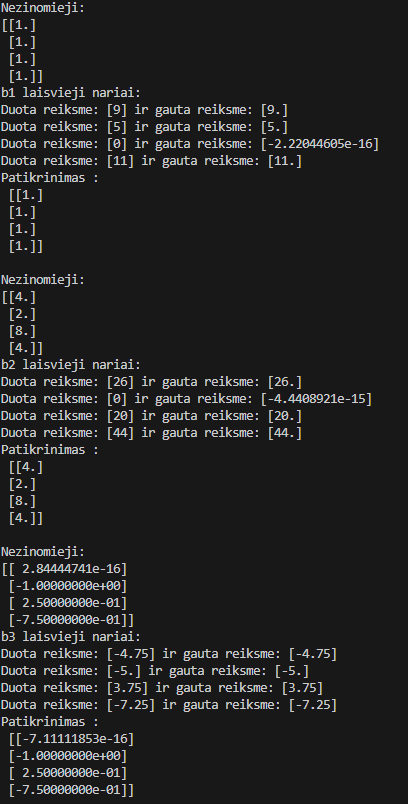
    Qi, Ri = np.linalg.qr(Ap)

    y = np.dot(Qi.transpose(), b3)

    x = np.linalg.solve(Ri, y)

    print('Patikrinimas :\n {0}'.format(x))

    print()



# Antra dalis (Netiesinių lygčių sprendimas)

Duota netiesinių lygčių sistema (4 lentelė):

{ 𝑍1 (𝑥1, 𝑥2 ) = 0

{𝑍2 (𝑥1, 𝑥2 ) = 0

1. Skirtinguose grafikuose pavaizduokite paviršius 𝑍1 (𝑥1, 𝑥2 ) ir 𝑍2 (𝑥1, 𝑥2 ).
2. Užduotyje pateiktą netiesinių lygčių sistemą išspręskite grafiniu būdu.
3. 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.
4. Gautus sprendinius patikrinkite naudodami išorinius išteklius (pvz., standartines Python funkcijas).

## Užduotis



## 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')

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()

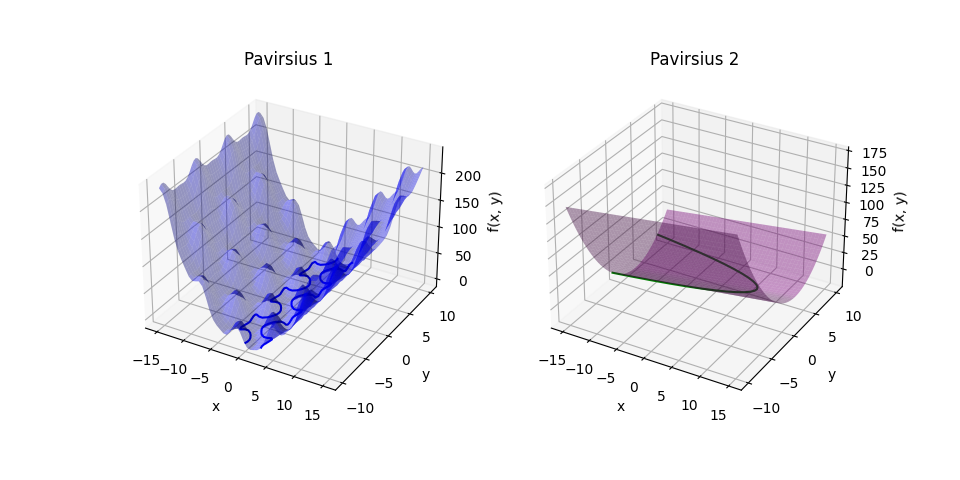
surf1 = ax1.plot\_surface(X, Y, Z[:,:,0], color='blue', alpha=0.4)

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()

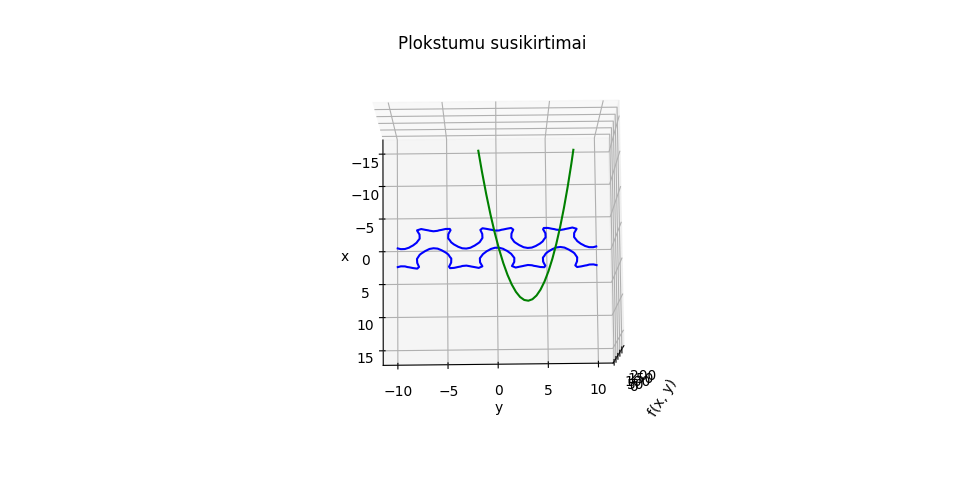


## B dalies sprendimas

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

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

plt.show()



## 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

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)

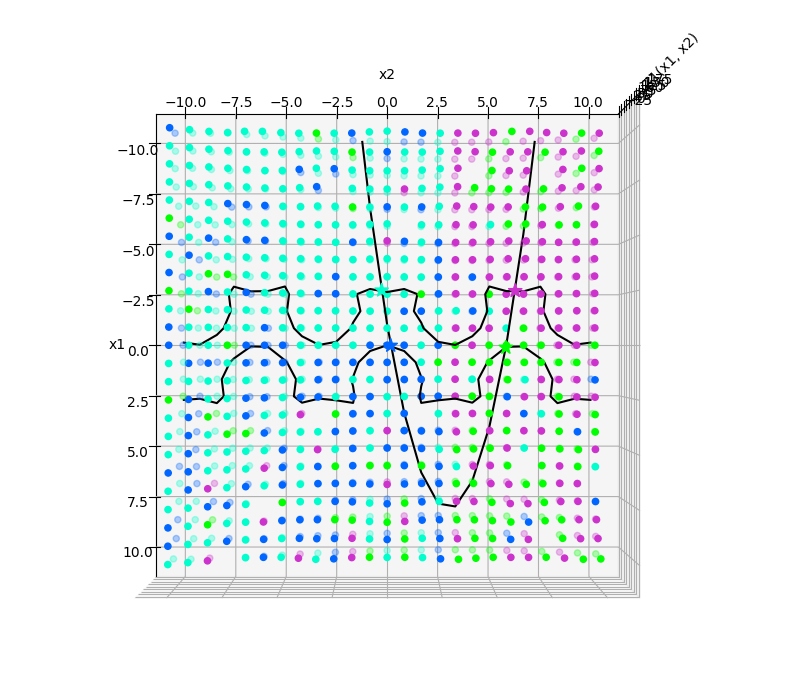
    i = i + 0.2

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

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

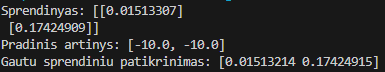
ax.view\_init(elev=90, azim=0)

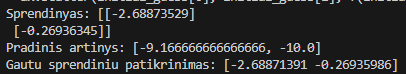
plt.show()

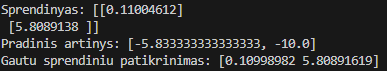


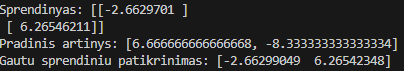
|  |  |
| --- | --- |
| **Sprendiniai** | **Pradiniai artiniai** |
| [[0.01513307]; [0.17424909]] | [-10.0, -10.0] |
| [[-2.68873529];[-0.26936345]] | [-9.166666666666666, -10.0] |
| [[0.11004612];[5.8089138 ]] | [-5.833333333333333, -10.0] |
| [[-2.6629701 ];[ 6.26546211]] | [6.666666666666668, -8.333333333333334]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

## D dalies sprendimas (patikrinimas)

****

****

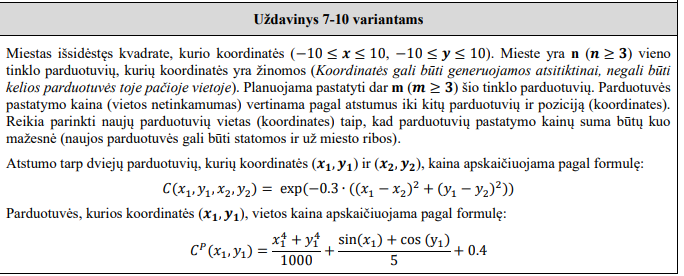
****

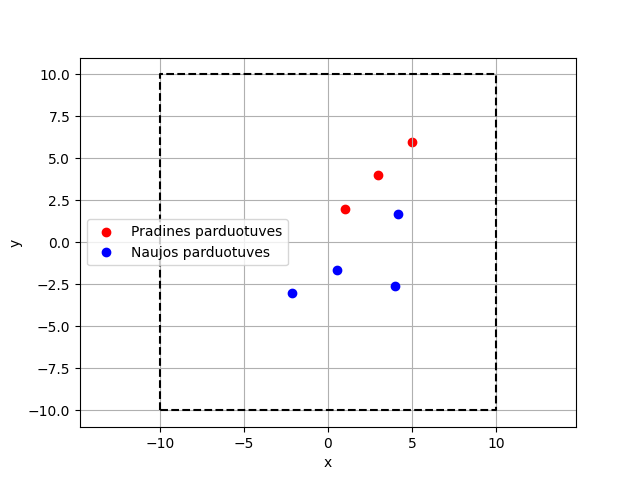
****

# 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.

## Užduotis



Gauta taškų konfiguracija



## Tikslo funkcijos aprašymas

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

## Taikyto metodo pavadinimas

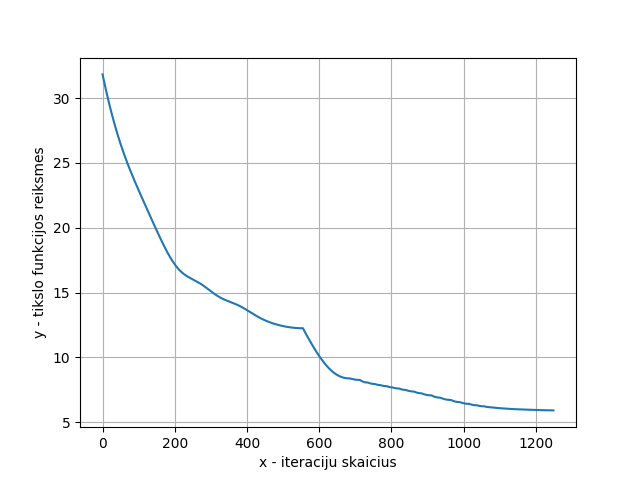
Taikytas greičiausio nusileidimo metodas, gradientas perskaičiuojamas tik kai reikšmė ima augti.

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

Iteracijų skaičius: 1250.

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

## Funkcijos priklausomybės nuo iteracijų skaičiaus grafikas



## 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()