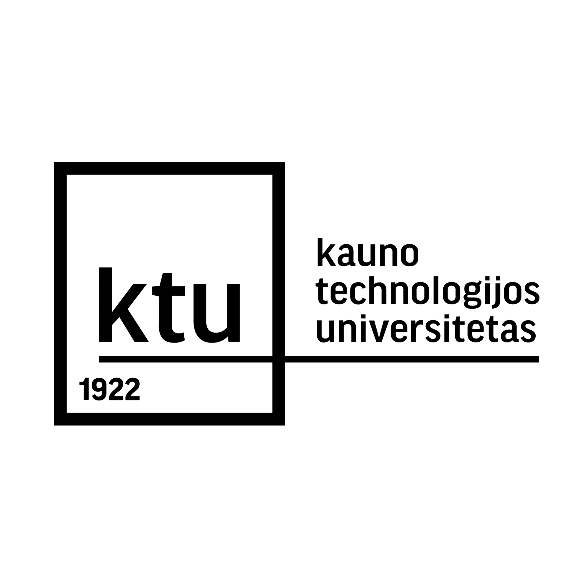
**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

INFORMATIKOS FAKULTETAS

TAIKOMOSIOS INFORMATIKOS KATEDRA

**SKAITINIAI METODAI IR ALGORITMAI(P170B115)**

**2 LABORATORINIS DARBAS**

Varianto Nr. 5

**Atliko:**

IFF-1/8 gr. studentas

Matas Palujanskas**Priėmė:**

Prof. Rimantas Barauskas

Doc. Andrius Kriščiūnas

KAUNAS

2023

**Turinys**

[1. Pirma užduoties dalis 3](#_Toc149491448)

[1.1 Užduoties sąlygos 3](#_Toc149491449)

[1.2 Tiesinių lygčių sprendimas Gauso metodu 4](#_Toc149491450)

[1.2.1 Sprendimo kodas 4](#_Toc149491451)

[1.2.2 2 lygčių sistema 5](#_Toc149491452)

[1.2.3 12 lygčių sistema 7](#_Toc149491453)

[1.2.4 17 lygčių sistema 9](#_Toc149491454)

[1.3 Tiesinės lygties sprendimas paprastųjų iteracijų metodu 11](#_Toc149491455)

[1.3.1 Sprendimo kodas: 11](#_Toc149491456)

[1.3.2 Programiškai gauti rezultatai 12](#_Toc149491457)

[1.3.3 Sprendimo rezultatų tikrinimas 13](#_Toc149491458)

[1.4 Tiesinių lygčių sprendimas sklaidos metodu 14](#_Toc149491459)

[1.4.1 Sprendimo kodas 14](#_Toc149491460)

[1.4.1 B1 rezultatai 15](#_Toc149491461)

[1.4.2 B2 rezultatai 16](#_Toc149491462)

[1.4.3 B3 rezultatai 17](#_Toc149491463)

[2. Antra užduoties dalis 19](#_Toc149491464)

[2.1. Užduoties sąlygos 19](#_Toc149491465)

[2.2. Sprendimo kodas 19](#_Toc149491466)

[2.3. Gauti rezultatai 23](#_Toc149491467)

[2.4. Tikrinimas 25](#_Toc149491468)

[3. Trečia dalis 26](#_Toc149491469)

[3.1 Užduoties sąlyga 26](#_Toc149491470)

[3.2 Sprendimo kodas 26](#_Toc149491471)

[3.3 Aprašymas 28](#_Toc149491472)

[3.4 Gauti rezultatai 29](#_Toc149491473)

[4. Literatūros sąrašas 30](#_Toc149491474)

# **Pirma** **užduoties dalis**

## Užduoties sąlygos

Paveikslėlis, kuriame yra tekstas, ekrano kopija, Šriftas, skaičius

Automatiškai sugeneruotas aprašymas

pav. 1 Antrojo laboratorinio pirmos dalies užduotys

**Užduoties variantas: 5**

**2 lentelėje nurodyti metodai:**



pav. 2 Nurodyti metodai

*Paveikslėlis, kuriame yra tekstas, Šriftas, ekrano kopija, linija

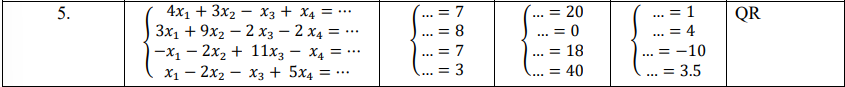
Automatiškai sugeneruotas aprašymas*

*Paveikslėlis, kuriame yra Šriftas, tekstas, rankraštis, linija

Automatiškai sugeneruotas aprašymas*

pav. 3 Gautos tiesinių lygčių sistemos

**2, 12 ir 17 lygčių sistemas reikia išspręsti Gauso metodu, 2 taip pat ir paprastųjų iteracijų metodu.**

****

pav. 4 3 lentelės tiesinės lygčių sistemos su laisvaisiais nariais ir nurodytas sklaidos metodas

## Tiesinių lygčių sprendimas Gauso metodu

### Sprendimo kodas

**Gauso.ipynb:**

**import** numpy **as** np  
  
*# -------------------iseities duomnenys:*A=np.matrix([[3, 10, 1, 5],  
 [-2, 6, 12, 14],  
 [3, 12, 5, 1],  
 [-3, -9, 5, 0]]).astype(float) *# koeficientu matrica*b=(np.matrix([83,178,37,-26])).transpose() *#laisvuju nariu vektorius-stulpelis*n=(np.shape(A))[0] *# lygciu skaicius nustatomas pagal ivesta matrica A*nb=(np.shape(b))[1] *# laisvuju nariu vektoriu skaicius nustatomas pagal ivesta matrica b*A1=np.hstack((A,b)) *#isplestoji matrica*print(A);print(b);print(n);print(nb);  
  
n = A.shape[1] *# Tiesinių lygčių skaičius*rangas\_A = np.linalg.matrix\_rank(A)  
rangas\_AB = np.linalg.matrix\_rank(np.hstack((A, b)))  
  
*# tiesioginis etapas:***for** i **in** range (0,n-1): *# range pradeda 0 ir baigia n-2 (!)* **for** j **in** range (i+1,n): *# range pradeda i+1 ir baigia n-1* A1[j,i:n+nb]=A1[j,i:n+nb]-A1[i,i:n+nb]\*A1[j,i]/A1[i,i];  
 A1[j,i]=0;  
 print(A1)  
  
*# atvirkstinis etapas:*x=np.zeros(shape=(n,nb))  
**for** i **in** range (n-1,-1,-1): *# range pradeda n-1 ir baigia 0 (trecias parametras yra zingsnis)* x[i,:]=(A1[i,n:n+nb]-A1[i,i+1:n]\*x[i+1:n,:])/A1[i,i]  
 print(x,**"x"**)  
  
**if** rangas\_A == rangas\_AB == n:  
 *# Tiesinių lygčių sistema turi vieną sprendinį* print(**"Tiesinių lygčių sistema turi vieną sprendinį:"**)  
 print(x)  
 liekana=A.dot(x)-b;  
 print(**"Bendra santykine paklaida:"**, np.linalg.norm(liekana)/ np.linalg.norm(x))  
**elif** rangas\_A == rangas\_AB:  
 *# Tiesinių lygčių sistema turi daugybę sprendinių* print(**"Tiesinių lygčių sistema turi daugybę sprendinių."**)  
**elif** rangas\_A == rangas\_AB < n:  
 *# Tiesinių lygčių sistema neturi sprendinių* print(**"Tiesinių lygčių sistema neturi sprendinių."**)

### 2 lygčių sistema

*Paveikslėlis, kuriame yra tekstas, Šriftas, ekrano kopija, linija

Automatiškai sugeneruotas aprašymas*

pav. 5 2 lygčių sistema

Galima įsistatyti į matricą.

3 10 1 5 83

-2 6 12 14 178

3 12 5 1 37

-3 -9 5 0 -26

Programiškai gauti rezultatai:

Paveikslėlis, kuriame yra tekstas, ekrano kopija, skaičius, Šriftas

Automatiškai sugeneruotas aprašymasGautas vienas sprendinys.

Tikriname ar rezultatai yra teisingi:

Tikrinimui naudojame: <https://onlinemschool.com/math/assistance/equation/gaus/>

Paveikslėlis, kuriame yra tekstas, ekrano kopija, Šriftas, skaičius

Automatiškai sugeneruotas aprašymas

pav. 6 Tikrinimas naudojant išorinius šaltinius

Įstatome sprendinius į pradinę lygtį:

Paveikslėlis, kuriame yra tekstas, ekrano kopija, Šriftas, skaičius

Automatiškai sugeneruotas aprašymas

pav. 7 Gauti rezultatai

**Išvada**: gautas vienas sprendinys, naudojant išorinius šaltinius rezultatai sutapo.

### **12 lygčių sistema**

*Paveikslėlis, kuriame yra tekstas, Šriftas, ekrano kopija, linija

Automatiškai sugeneruotas aprašymas*

pav. 8 12 lygčių sistema

Galima įsistatyti į matricą.

3 1 -1 5 20

-3 4 -8 -1 -36

1 -3 7 6 41

0 5 -9 4 -16

Programiškai gauti rezultatai:

Paveikslėlis, kuriame yra tekstas, ekrano kopija, skaičius, linija

Automatiškai sugeneruotas aprašymas

pav. 9 Programiškai gauti rezultatai

Tikriname ar rezultatai yra teisingi:

Tikrinimui naudojame: <https://onlinemschool.com/math/assistance/equation/gaus/>

Paveikslėlis, kuriame yra tekstas, ekrano kopija, skaičius, Šriftas

Automatiškai sugeneruotas aprašymas

pav. 10 Tikrinimas naudojant išorinius šaltinius

Paveikslėlis, kuriame yra tekstas, ekrano kopija, monitorius, programinė įranga

Automatiškai sugeneruotas aprašymas

pav. 11 Gauti rezultatai

**Išvada**: lygčių sistema turi be galo daug sprendinių, naudojant išorinius šaltinius rezultatai sutapo.

### **17 lygčių sistema**

*Paveikslėlis, kuriame yra Šriftas, tekstas, rankraštis, linija

Automatiškai sugeneruotas aprašymas*

pav. 12 17 lygčių sistema

Galima įsistatyti į matricą.

2 5 1 2 -1

-2 0 3 5 7

1 0 -1 1 3

0 5 4 7 4

Programiškai gauti rezultatai:

Paveikslėlis, kuriame yra tekstas, ekrano kopija, Šriftas, dokumentas

Automatiškai sugeneruotas aprašymas

pav. 13 Programiškai gauti rezultatai

Tikriname ar rezultatai yra teisingi:

Tikrinimui naudojame: <https://onlinemschool.com/math/assistance/equation/gaus/>

Paveikslėlis, kuriame yra tekstas, ekrano kopija, skaičius, Šriftas

Automatiškai sugeneruotas aprašymas

pav. 14 Tikrinimas naudojant išorinius šaltinius

Paveikslėlis, kuriame yra tekstas, Šriftas, linija, skaičius

Automatiškai sugeneruotas aprašymas

pav. 15 Gauti rezultatai

**Išvada**: lygčių sistema neturi sprendinių, naudojant išorinius šaltinius rezultatai sutapo.

## Tiesinės lygties sprendimas paprastųjų iteracijų metodu

### Sprendimo kodas:

**Paprastųjų iteracijų 2.ipynb:**

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
  
A = np.array([  
 [3, 10, 1, 5],  
 [-2, 6, 12, 14],  
 [3, 12, 5, 1],  
 [-3, -9, 5, 0]  
], dtype=float)  
  
b = np.array([83,178,37,-26], dtype=float)  
epsilon = 1e-6  
A = A + epsilon \* np.eye(n)  
  
n = A.shape[0]  
Aprad = A.copy()  
  
method = **'simple\_iterations'***# method = 'Gauss-Seidel\_iterations'*alpha = np.array([100, 20, 1, 1], dtype=float) *# Laisvai pasirenkami metodo parametrai*Atld = np.diag(1. / np.diag(A)).dot(A) - np.diag(alpha)  
btld = np.diag(1. / np.diag(A)).dot(b)  
  
nitmax = 59  
eps = 1e-12  
x = np.zeros(n)  
x1 = np.zeros(n)  
prec = []  
  
print(**'\nSprendimas iteracijomis:'**)  
**for** it **in** range(nitmax):  
 **if** method == **'Gauss-Seidel\_iterations'**:  
 **for** i **in** range(n):  
 x1[i] = (btld[i] - np.dot(Atld[i, :], x1)) / alpha[i]  
 **elif** method == **'simple\_iterations'**:  
 x1 = (btld - np.dot(Atld, x)) / alpha  
 **else**:  
 print(**'Neapibrėžtas metodas'**)  
 **break** prec.append(np.linalg.norm(x1 - x) / (np.linalg.norm(x) + np.linalg.norm(x1)))  
 print(**f'Iteracija Nr. {**it + 1**}, Tikslumas: {**prec[it]**}'**)  
  
 **if** prec[it] < eps:  
 **break** x = x1  
  
x\_solution = x  
print(**'Sprendinys:'**)  
print(x\_solution)  
print(**'Patikrinimas:'**)  
print(np.dot(Aprad, x\_solution) - b)  
  
plt.semilogy(range(1, len(prec) + 1), prec, **'r-'**)  
plt.grid(**True**)  
plt.xlabel(**'Iteracija'**)  
plt.ylabel(**'Tikslumas'**)  
plt.show()

1. Programiškai gauti rezultatai:

Paveikslėlis, kuriame yra tekstas, ekrano kopija, linija, diagrama

Automatiškai sugeneruotas aprašymas

pav. 16 Programiškai gauti rezultatai

Nustatyta, kad didžiausia iteracija yra 59.

1. Sprendimo rezultatų tikrinimas:

Tikriname ar rezultatai yra teisingi:

Tikrinimui naudojame: <https://onlinemschool.com/math/assistance/equation/gaus/>

Paveikslėlis, kuriame yra tekstas, ekrano kopija, Šriftas, skaičius

Automatiškai sugeneruotas aprašymas

pav. 17 Gauti rezultatai

**Išvada**: gauti tokie pat rezultatai kaip ir tikrinant kitais šaltiniais.

## Tiesinių lygčių sprendimas sklaidos metodu

### Sprendimo kodas

**1b.ipynb:**

**import** numpy **as** np  
  
*# -------------------iseities duomnenys:*A=np.matrix([[4 , 3, -1, 1],  
 [3, 9, -2, -2],  
 [-1, -2, 11, -1],  
 [1, -2, -1, 5]]).astype(float) *# koeficientu matrica*Ap=A *# bus naudojama patikrinimui*b=(np.matrix([1,4,-10,3.5])).transpose().astype(float) *#laisvuju nariu vektorius-stulpelis*n=(np.shape(A))[0] *# lygciu skaicius nustatomas pagal ivesta matrica A*nb=(np.shape(b))[1] *# laisvuju nariu vektoriu skaicius nustatomas pagal ivesta matrica b*print(A,**'A matrica'**);print(b,**'b'**);print(n,**'n'**);print(nb,**'nb'**);  
  
*# tiesioginis etapas(QR skaida):*Q=np.identity(n)  
**for** i **in** range (0,n-1):  
 z=A[i:n,i] *#vektorius* zp=np.zeros(np.shape(z)); zp[0]=np.linalg.norm(z)  
 print(zp,**'zp'**)  
 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,:])  
 print(A,**'A matrica'**)  
 Q[:,i:n]=Q[:,i:n].dot(Qi)  
 print(Q,**'Q reikšmė'**)  
 print(A,**'R reikšmė'**)  
  
 *# atgalinis etapas:*b1=Q.transpose().dot(b);  
x=np.zeros(shape=(n,nb));  
**for** i **in** range (n-1,-1,-1): *# range pradeda n-1 ir baigia 0 (trecias parametras yra zingsnis)* x[i,:]=(b1[i,:]-A[i,i+1:n]\*x[i+1:n,:])/A[i,i];  
  
 print(x,**"x"**)  
  
print(**'X lygus:'**);  
print(x);  
print(**"Naudotos Q ir R reiksmes:"**)  
print(**'Q -'**, Q)  
print(**'R -'**, A)  
print(**"------------ sprendinio patikrinimas ----------------"**);  
print(**'liekana:'**)  
liekana=Ap.dot(x)-b1;print(liekana);  
print(**'bendra santykine paklaida:'**)  
print(np.linalg.norm(liekana)/ np.linalg.norm(x))

### B1 rezultatai

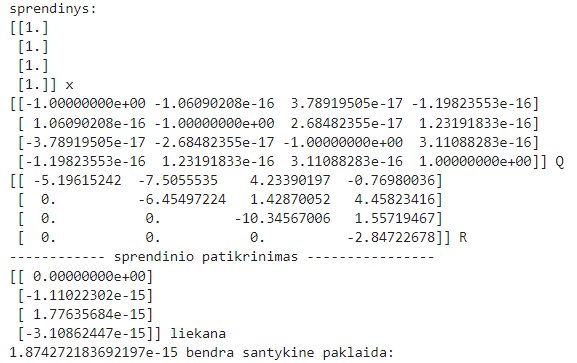
Reikšmės:

Paveikslėlis, kuriame yra tekstas, Šriftas, ekrano kopija, linija

Automatiškai sugeneruotas aprašymas

pav. 18 B1 reikšmės

Programos išvesti rezultatai:



pav. 19 Programiškai gauti rezultatai

Tikrinimas:

Tikrinimui naudojame: <https://onlinemschool.com/math/assistance/equation/gaus/>

Įstatome sprendinius į pradinę lygtį:

Paveikslėlis, kuriame yra tekstas, ekrano kopija, Šriftas, skaičius

Automatiškai sugeneruotas aprašymas

pav. 20 Tikrinimas naudojant išorinius šaltinius

**Išvada**: lygčių sistema turi vieną sprendinį, rezultatai sutampa ir yra teisingi.

### B2 rezultatai

Reikšmės:

Paveikslėlis, kuriame yra ekrano kopija, tekstas, Šriftas, diagrama

Automatiškai sugeneruotas aprašymas

pav. 21 B2 reikšmės

Programos išvesti rezultatai:

Paveikslėlis, kuriame yra tekstas, ekrano kopija, Šriftas, skaičius

Automatiškai sugeneruotas aprašymas

pav. 22 Programiškai gauti rezultatai

Tikrinimas:

Tikrinimui naudojame: <https://onlinemschool.com/math/assistance/equation/gaus/>

Įstatome sprendinius į pradinę lygtį:

Paveikslėlis, kuriame yra tekstas, ekrano kopija, Šriftas, skaičius

Automatiškai sugeneruotas aprašymas

pav. 23 Gauti rezultatai

**Išvada**: lygčių sistema turi vieną sprendinį, rezultatai sutampa ir yra teisingi.

### B3 rezultatai

Reikšmės:

Paveikslėlis, kuriame yra ekrano kopija, Šriftas, diagrama, linija

Automatiškai sugeneruotas aprašymas

pav. 24 B3 reikšmės

Programos išvesti rezultatai:

Paveikslėlis, kuriame yra tekstas, ekrano kopija, Šriftas, skaičius

Automatiškai sugeneruotas aprašymas

pav. 25 Programiškai gauti rezultatai

Tikrinimas:

Tikrinimui naudojame: <https://onlinemschool.com/math/assistance/equation/gaus/>

Įstatome sprendinius į pradinę lygtį:

Paveikslėlis, kuriame yra tekstas, ekrano kopija, Šriftas, skaičius

Automatiškai sugeneruotas aprašymas

pav. 26 Gauti rezultatai

**Išvada**: lygčių sistema turi vieną sprendinį, rezultatai sutampa ir yra teisingi.

# Antra užduoties dalis

## Užduoties sąlygos

Paveikslėlis, kuriame yra tekstas, ekrano kopija, Šriftas, skaičius

Automatiškai sugeneruotas aprašymas

pav. 27 Antros dalies sąlygos

5 varianto netiesinės lygčių sistemos:

Paveikslėlis, kuriame yra tekstas, Šriftas, linija, ekrano kopija

Automatiškai sugeneruotas aprašymas

pav. 28 Netiesinės lygčių sistemos

## Sprendimo kodas

**Netiesiniu lygciu sprendimas.ipynb:**

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**import** math  
**from** scipy.optimize **import** fsolve  
  
*# Niutono - Rafsono metodo naudojimas nelinearinių lygčių sistemoms spresti.  
# Pagrindinė idėja: Jakobiano matrica \* delta(x) = -f(x) => gauname delta(x) => x = x + delta(x)  
  
# Lygčių sistema ir jos Jakobiano matricos apibrėžimas:***def** LF\_scipy(vars):  
 x1, x2 = vars  
 eq1 = x1\*\*2 / ((x2 + math.cos(x1))\*\*2 + 1) - 2  
 eq2 = (x1/3)\*\*2 + (x2 + math.cos(x1))\*\*2 - 5  
 **return** [eq1, eq2]  
  
**def** LF(x):  
 s = np.array([x[0]\*\*2 / ((x[1] + math.cos(x[0]))\*\*2 + 1) - 2, (x[0]/3)\*\*2 + (x[1] + math.cos(x[0]))\*\*2 - 5])  
 **return** s  
  
  
*#jakobio matrica***def** DLF(x):  
 df1\_dx0 = (2 \* x[0] \* (x[1] + math.cos(x[0]))\*\*2) / (((x[1] + math.cos(x[0]))\*\*2 + 1)\*\*2)  
 df1\_dx1 = -2 \* x[0] \* (x[1] + math.cos(x[0])) \* math.sin(x[0]) / (((x[1] + math.cos(x[0]))\*\*2 + 1)\*\*2)  
 df2\_dx0 = (2/9) \* (x[0]\*\*2 - 9 \* (x[1] + math.cos(x[0]))\*\*2)  
 df2\_dx1 = 2 \* (x[1] + math.cos(x[0]))  
  
 J = np.array([[df1\_dx0, df1\_dx1], [df2\_dx0, df2\_dx1]])  
 **return** J  
  
*# Vizualizacijos funkcijos:***def** braizom\_viena(F, numb,fun\_color, cont\_color, title):  
 fig1 = plt.figure(1, figsize=plt.figaspect(0.5))  
 ax1 = fig1.add\_subplot(1, 1, 1, projection=**'3d'**)  
 plt.title(title)  
 ax1.set\_xlabel(**'x1'**)  
 ax1.set\_ylabel(**'x2'**)  
 ax1.set\_zlabel(**'z'**)  
 xx = np.linspace(-10, 10, 100)  
 yy = np.linspace(-10, 10, 100)  
 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, :] = F([X[i][j], Y[i][j]]).transpose()  
 surf1 = ax1.plot\_surface(X, Y, Z[:, :, numb], color=fun\_color, alpha=0.4)  
 CS11 = ax1.contour(X, Y, Z[:, :, numb], [0], colors=cont\_color)  
 plt.show()  
  
**def** braizom\_sistema(F, title):  
 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'**)  
 ax1.set\_title(title)  
 ax2 = fig1.add\_subplot(1, 2, 2, projection=**'3d'**)  
 ax2.set\_xlabel(**'x1'**)  
 ax2.set\_ylabel(**'x2'**)  
 ax2.set\_zlabel(**'z'**)  
 *#ax2.set\_title(title)  
 #ax2.set\_title("Kontūrais, ties kuriais funkcijos kerta Z=0 plokštumą")* plt.draw()  
 xx = np.linspace(-10, 10, 100)  
 yy = np.linspace(-10, 10, 100)  
 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, :] = F([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 = ax1.plot\_surface(X, Y, Z[:, :, 1], color=**'purple'**, alpha=0.4)  
 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'**)  
 plt.show()  
  
*# Niutono metodo funkcijos:***def** Niutono(DF, F, iter\_max, alpha, x0, eps, print\_var):  
 **for** i **in** range(iter\_max):  
 **try**:  
 **if** np.any(np.abs(x0) > 1e50):  
 **return** [math.inf, math.inf]  
 ff = F(x0)  
 dff = DF(x0)  
 deltax = -np.linalg.solve(dff, ff)  
 x1 = x0 + alpha \* deltax  
 precision = np.linalg.norm(deltax) / (np.linalg.norm(x0) + np.linalg.norm(deltax))  
 **if** precision < eps:  
 **if**(print\_var):  
 print(**f"Konvergavo. sprendinys x = {**x0**}"**)  
 **return** x0  
 **elif** i == iter\_max - 1:  
 **if** (print\_var):  
 print(**f"Sprendinys pasiekė iteracijų limitą. paskutinis x = {**x0**}"**)  
 **return** [math.inf, math.inf]  
 x0 = x1  
 **except**:  
 **return** [math.inf, math.inf]  
  
*# Tikrina, ar sprendiniai unikalūs:***def** is\_solution\_unique(solution, solutions\_list):  
 **for** sol\_dict **in** solutions\_list:  
 sol = sol\_dict[**"solution"**]  
 **if** any(np.isclose(x, sol, atol=1e-5)):  
 **return False  
 return True***# Inicializuoja pradinių artinių tinklelį vizualizacijai:***def** init\_paint(sprendiniai):  
 fig1 = plt.figure(1, figsize=plt.figaspect(0.5))  
 ax2 = fig1.add\_subplot(1, 1, 1)  
 ax2.set\_xlabel(**'x1'**)  
 ax2.set\_ylabel(**'x2'**)  
 ax2.set\_title(**"Pradinių artinių tinklelis"**)  
 plt.draw()  
 xx = np.linspace(-10, 10, 100)  
 yy = np.linspace(-10, 10, 100)  
 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()  
 CS1 = ax2.contour(X, Y, Z[:, :, 0], [0], colors=**'lime'**, linewidths=2, zorder=2)  
 CS2 = ax2.contour(X, Y, Z[:, :, 1], [0], colors=**'lime'**, linewidths=2, zorder=2)  
 **for** sol\_dict **in** sprendiniai:  
 sol = sol\_dict[**"solution"**]  
 ax2.scatter(sol[0], sol[1], marker=**"P"**, color=sol\_dict[**"color"**], linewidths=1, s=50, edgecolor=**'black'**,zorder=3)  
 **return** ax2  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 *# Vizualizacija: braižomi funkcijų grafikai ir kontūrai.* braizom\_viena(LF,0,**'blue'**,**'b'**, **"Z1: (x1^2/(x2+(cos(x1))^2+1))-2"**)  
 braizom\_viena(LF,1,**'purple'**,**'g'**, **"Z2: (x1/3)^2+(x2+cos(x1))^2-5"**)  
 print(**"Kontūrais, ties kuriais funkcijos kerta Z=0 plokštumą:"**)  
 braizom\_sistema(LF,**"(x1^2/(x2+(cos(x1))^2+1)-2 ir (x1/3)^2+(x2+cos(x1))^2-5"**)  
  
 *# Parametrai Niutono metodui:* eps = 1e-10  
 alpha = 1  
 itmax = 4  
  
 *#netiesinė lygciu sistema grafiškai išsprendus turi 4 sprendinius. ju artinius apsirašome:* print(**"Netiesinių lygčių sistemos sprendiniai"**)  
 Artiniai = [[-3, 3], [-3, -1], [3, 3], [3, -1]]  
 *#Artiniai = [[0, 0]]* Sprendiniai = []  
 spalvos = [**'b'**, **'g'**, **'r'**, **'c'**, **'m'**, **'y'**]  
 ind=0  
 **for** art **in** Artiniai:  
 x = Niutono(DLF, LF, itmax, alpha, art, eps, **True**)  
 **if** x[0] != math.inf **and** x[1] != math.inf **and** is\_solution\_unique(x, Sprendiniai):  
 Sprendiniai.append({**"solution"**: x, **"color"**: spalvos[ind]})  
 ind=ind+1  
  
 **if**(len(Artiniai) == len(Sprendiniai)):  
 print(**"visi Sprendiniai yra unikalūs"**)  
 **else**:  
 print(**"visi Sprendiniai yra unikalūs"**)  
 *#print("Yra vienodų sprendinių")  
  
 #braizome tinkleli* ax = init\_paint(Sprendiniai)  
  
 **for** i **in** range(-10, 11):  
 **for** j **in** range(-10, 11):  
 x = Niutono(DLF, LF, itmax, alpha, np.array([i, j]), eps, **False**)  
 **if** x[0] != math.inf **and** x[1] != math.inf:  
 **for** sol\_dict **in** Sprendiniai:  
 sol = sol\_dict[**"solution"**]  
 **if** any(np.isclose(x, sol, atol=1e-5)):  
 ax.scatter(i, j, marker=**"o"**, c=sol\_dict[**"color"**], zorder=1, s=100, edgecolor=**'black'**, linewidths=2)  
 **elif** i == 0: *# Tikriname, ar taškas yra per centrą ir yra vertikalias* ax.scatter(i, j, marker=**"o"**, c=**"black"**, zorder=1, s=100, edgecolor=**'black'**, linewidths=2)  
 **else**:  
 ax.scatter(i, j, marker=**"o"**, c=**"purple"**, zorder=1, s=100, edgecolor=**'black'**, linewidths=2)  
  
 plt.show()  
  
 *#Tikrinimas su scipy* scipy\_sprendiniai = []  
 A = Sprendiniai  
 **for** art **in** Artiniai:  
 x1, x2 = fsolve(LF\_scipy, (art[0], art[1]))  
 *#if is\_solution\_unique\_scipy([x1, x2], scipy\_sprendiniai):* scipy\_sprendiniai.append([x1, x2])  
  
 sprend = []  
 **for** sp **in** Sprendiniai:  
 sprend.append([sp[**"solution"**][0], sp[**"solution"**][1]])  
  
 **if** len(sprend) == len(scipy\_sprendiniai):  
 sprend.sort(key=**lambda** x: (x[1], x[0]))  
 scipy\_sprendiniai.sort(key=**lambda** x: (x[1], x[0]))  
 visi\_vienodi = **True  
 for** i **in** range(len(scipy\_sprendiniai)):  
 print(**f"niutono sprendinys - {**sprend[i]**} ir scipy sprendinys - {**scipy\_sprendiniai[i]**}"**)  
 **if not** (np.isclose(sprend[i][0], scipy\_sprendiniai[i][0], atol=1e-5) **and** np.isclose(sprend[i][1], scipy\_sprendiniai[i][1], atol=1e-5)):  
 visi\_vienodi = **False** print(**f"Ar visi vienodi? - {**visi\_vienodi**}"**)  
 **else**:  
 print(**"Sąrašų ilgiai skiriasi. Negalima lyginti."**)

## Gauti rezultatai

Atvaizduoti Z1 ir Z2 paviršiai skirtinguose grafikuose:

*Paveikslėlis, kuriame yra diagrama, origamis, dizainas

Automatiškai sugeneruotas aprašymas*

*Paveikslėlis, kuriame yra diagrama, origamis, dizainas

Automatiškai sugeneruotas aprašymas*

pav. 29 Z1 ir Z2

Abu paviršiai viename grafike:

*Paveikslėlis, kuriame yra eskizas, diagrama, origamis, dizainas

Automatiškai sugeneruotas aprašymas*

pav. 30 Z1 ir Z2 viename grafike

Gauti rezultatai, patikrinta su scipy:

Paveikslėlis, kuriame yra tekstas, Šriftas, ekrano kopija, linija

Automatiškai sugeneruotas aprašymas

pav. 31 Rezultatai

Grafinis sprendimas:

Paveikslėlis, kuriame yra rašas, ekrano kopija, linija

Automatiškai sugeneruotas aprašymas

pav. 32 Grafinis sprendimas

## Tikrinimas

Tikrinimui naudota Wolfram Alpha aplinka.

Paveikslėlis, kuriame yra tekstas, diagrama, Grafikas, linija

Automatiškai sugeneruotas aprašymas

pav. 33 Z2 lygtis

Paveikslėlis, kuriame yra diagrama, linija, Grafikas, tekstas

Automatiškai sugeneruotas aprašymas

pav. 34 Z1 lygtis

**Išvada**: Gauti grafikai sutampa.

# Trečia dalis

## Užduoties sąlyga

Paveikslėlis, kuriame yra tekstas, Šriftas, ekrano kopija

Automatiškai sugeneruotas aprašymas

pav. 35 Trečios dalies sąlyga

Mano variantas yra 5, todėl gautas šis uždavinys:

Paveikslėlis, kuriame yra tekstas, ekrano kopija, Šriftas, skaičius

Automatiškai sugeneruotas aprašymas

pav. 36 Gautas uždavinys

## Sprendimo kodas

**3 uzduotis.ipynb:**

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
  
*# Pradinės sąlygos*n = 5 *# Parduotuvių skaičius*m = 3 *# Planuojamų parduotuvių skaičius*miesto\_ribos = 10 *# Miesto ribos dydis  
  
# Sugeneruojame pradines parduotuvių koordinates*pradines\_parduotuves = np.random.rand(n, 2) \* 20 - 10 *# Koordinatės nuo -10 iki 10  
  
# Sugeneruojame pradines naujų parduotuvių koordinates*naujos\_parduotuves = np.random.rand(m, 2) \* 20 - 10 *# Koordinatės nuo -10 iki 10  
  
# Apskaičiuojame pastatytų parduotuvių vietos netinkamumo kainą***def** pastatytu\_parduotuviu\_kaina(parduotuves):  
 kaina = 0  
 **for** i **in** range(len(parduotuves)):  
 **for** j **in** range(len(parduotuves)):  
 **if** i != j:  
 kaina += np.exp(-0.1 \* ((parduotuves[i][0] - parduotuves[j][0])\*\*2 + (parduotuves[i][1] - parduotuves[j][1])\*\*2))  
 kaina += 0.5 \* ((miesto\_ribos - parduotuves[i][0])\*\*2 + (miesto\_ribos - parduotuves[i][1])\*\*2)  
 **return** kaina  
  
*# Gradientinio nusileidimo metodas***def** gradientinis\_nusileidimas(parduotuves, m, miesto\_ribos, mokymo\_zingsnis, iteraciju\_skaicius):  
 kainos = []  
 **for** \_ **in** range(iteraciju\_skaicius):  
 gradientas = np.zeros((m, 2))  
 **for** i **in** range(m):  
 **for** j **in** range(m):  
 **if** i != j:  
 gradientas[i] += 0.2 \* (parduotuves[i] - parduotuves[j]) \* np.exp(-0.1 \* ((parduotuves[i][0] - parduotuves[j][0])\*\*2 + (parduotuves[i][1] - parduotuves[j][1])\*\*2))  
 gradientas[i] += 0.1 \* (parduotuves[i] - miesto\_ribos) \* np.exp(-0.1 \* ((parduotuves[i][0] - miesto\_ribos)\*\*2 + (parduotuves[i][1] - miesto\_ribos)\*\*2))  
 parduotuves -= mokymo\_zingsnis \* gradientas  
 kainos.append(pastatytu\_parduotuviu\_kaina(parduotuves))  
 **return** parduotuves, kainos  
  
*# Nustatome optimizavimo parametrus*mokymo\_zingsnis = 0.5  
iteraciju\_skaicius = 100  
  
*# Vykdome gradientinio nusileidimo metodą*parduotuves, kainos = gradientinis\_nusileidimas(naujos\_parduotuves, m, miesto\_ribos, mokymo\_zingsnis, iteraciju\_skaicius)  
  
*# Spausdiname pradinių parduotuvių koordinates ir jų pastatymo kainas*print(**'Pradinės parduotuves:'**)  
**for** i, (x, y) **in** enumerate(pradines\_parduotuves):  
 kaina = pastatytu\_parduotuviu\_kaina(pradines\_parduotuves)  
 print(**f"Parduotuvė {**i + 1**}: Koordinatės ({**x**:.2f}, {**y**:.2f}), Pastatymo kaina: {**kaina**:.2f}"**)  
  
*# Spausdiname naujų parduotuvių koordinates ir jų pastatymo kainas*print(**'Naujos parduotuves:'**)  
**for** i, (x, y) **in** enumerate(parduotuves):  
 kaina = pastatytu\_parduotuviu\_kaina(parduotuves)  
 print(**f"Parduotuvė {**i + 1**}: Koordinatės ({**x**:.2f}, {**y**:.2f}), Pastatymo kaina: {**kaina**:.2f}"**)  
  
*# Pavaizduojame rezultatus*plt.scatter(pradines\_parduotuves[:, 0], pradines\_parduotuves[:, 1], c=**'r'**, marker=**'x'**, label=**'Pradinės parduotuvės'**)  
plt.scatter(parduotuves[:, 0], parduotuves[:, 1], c=**'g'**, marker=**'s'**, label=**'Naujos optimizuotos parduotuvės'**)  
plt.xlim(-11, 11)  
plt.ylim(-11, 11)  
plt.legend()  
plt.title(**"Parduotuvių vietos"**)  
plt.show()  
  
plt.plot(kainos)  
plt.xlabel(**'Iteracijos'**)  
plt.ylabel(**'Kaina'**)  
plt.title(**'Tikslo funkcijos priklausomybė nuo iteracijų skaičiaus'**)  
plt.show()  
  
*#Tikriname iteracijų pabaigos sąlygas ir spausdiname rezultatą***if** len(kainos) >= 2 **and** abs(kainos[-1] - kainos[-2]) < 0.01:  
 print(**"Optimizacija pasieka stabilumą."**)  
**elif** len(kainos) >= iteraciju\_skaicius:  
 print(**"Pasiektas maksimalus iteracijų skaičius."**)  
**else**:  
 print(**"Optimizacija baigėsi dėl kitos priežasties."**)

## Aprašymas

1. Pradinės sąlygos:
   * Parduotuvių skaičius (n): 5
   * Planuojamų parduotuvių skaičius (m): 3
   * Miesto ribos dydis: 10
2. Pradinės taškų konfigūracijos:
   * Sugeneruotos pradinės parduotuvės: 5 atsitiktinai sugeneruotos parduotuvės su koordinatėmis nuo -10 iki 10.
   * Sugeneruotos naujos parduotuvės: 3 atsitiktinai sugeneruotos parduotuvės su koordinatėmis nuo -10 iki 10.
3. Tikslo funkcijos aprašymas:

Programoje taikoma tikslo funkcija, kuri apskaičiuoja kainą, susijusią su pastatytomis parduotuvėmis. Ši funkcija apima du komponentus:

* Komponentas, apibūdinantis atstumų tarp parduotuvių įtaką kainai. Jis pagrįstas eksponentinio sumažinimo funkcija, kurioje atstumas tarp parduotuvių mažinant apie 0, taip pat užima parduotuvės vietą.
* Komponentas, apibūdinantis atstumą nuo parduotuvės iki miesto ribos ir užimantį parduotuvės vietą. Jis taip pat pagrįstas eksponentinio sumažinimo funkcija.

1. Taikytas metodas:

Naudojamas gradientinio nusileidimo metodas parduotuvių vietos optimizavimui.

1. Optimizacijos parametrai:
   * Mokymo žingsnis: 0.01
   * Iteracijų skaičius: 1000
2. Iteracijos pabaigos sąlygos:
   * Programa vykdo 1000 iteracijų

## Gauti rezultatai

Paveikslėlis, kuriame yra tekstas, ekrano kopija, Šriftas, skaičius

Automatiškai sugeneruotas aprašymas

pav. 37 Programiškai gauti rezultatai

Paveikslėlis, kuriame yra tekstas, ekrano kopija, skaičius, diagrama

Automatiškai sugeneruotas aprašymas

pav. 38 Grafiškai atvaizduotos pradinės ir naujos parduotuvės

Paveikslėlis, kuriame yra tekstas, ekrano kopija, linija, diagrama

Automatiškai sugeneruotas aprašymas

pav. 39 Tikslo funkcijos priklausomybės nuo iteracijų skaičiaus grafikas

**Išvada:** gautos naujų optimizuotų parduotuvių vietos ir kainos, keičiant optimizavimo parametrus skyrėsi ir gauti rezultatai, rezultatai buvo skirtingi ir todėl, nes kiekvieną kartą buvo generuojama su atsitiktinai pasirinktomis pradinėmis koordinatėmis. Geriausi rezultatai buvo pasiekti su 1000 iteracijų skaičiumi.

# Literatūros sąrašas

1. „Skaitiniai metodai ir algoritmai“ „Moodle“ aplinkoje [HTTPS://MOODLE.KTU.EDU/COURSE/VIEW.PHP?ID=7639](https://moodle.ktu.edu/course/view.php?id=7639)