**Shape

Description automatically generated with medium confidence**

**KAUNO TECHOLOGIJOS UNIVERSITETAS**

**INFORMATIKOS FAKULTETAS**

**KOMPIUTERIŲ KATEDRA**

**Skaitiniai metodai ir algoritmai (P170B115)**

Laboratorinis darbas nr. 2

Varianto nr. 9

Atliko:

IFF 8/3 gr. studentas

Dovydas Zamas

Priėmė:

doc. Čalnerytė Dalia

# Contents

[Contents 2](#_Toc85648393)

[1. Uždavinys 3](#_Toc85648394)

[1.1. Tiesinių lygčių sistemų sprendimas 3](#_Toc85648395)

[1.2. Netiesinių lygčių sistemų sprendimas 3](#_Toc85648396)

[2. Uždavinio sprendimas 4](#_Toc85648397)

[2.1. Tiesinių lygčių sistemų sprendimas 4](#_Toc85648398)

[2.1.1. Gauso metodas 4](#_Toc85648399)

[2.1.2. Paprastųjų iteracijų metodas 8](#_Toc85648400)

[2.2. Netiesinių lygčių sprendimas 11](#_Toc85648401)

[2.2.1. Lygtis 11](#_Toc85648402)

[2.2.2. Metodas 11](#_Toc85648403)

[2.2.3. Niutono metodo programos kodo dalis: 12](#_Toc85648404)

[2.2.4. Rezultatai 13](#_Toc85648405)

# Uždavinys

## Tiesinių lygčių sistemų sprendimas

Sukurkite programą, kuri nurodytais metodais spręstų tiesinių 4 nežinomųjų lygčių sistemą. Pateikite programos veikimo rezultatus su duotomis lygtimis.

* Jei metodas leidžia, programoje turi būti įvertinti atvejai:

1. kai lygčių sistema turi vieną sprendinį;
2. kai lygčių sistema sprendinių neturi;
3. kai lygčių sistema turi be galo daug sprendinių.

* Jei metodas paremtas matricos pertvarkymu, programa turi turėti galimybę pateikti tarpines matricų išraiškas kiekviename žingsnyje. Jei metodas iteracinis, grafiškai pavaizduokite, kaip atliekant iteracijas kinta santykinis sprendinio tikslumas esant kelioms skirtingoms konvergavimo daugiklio reikšmėms.
* Patikrinkite gautus sprendinius įrašydami juos į pradinę lygčių sistemą. Pateikite atsakymo ir laisvųjų narių stulpelio skirtumo elementus bent 12 skaitmenų po kablelio tikslumu.
* Gautus sprendinius patikrinkite naudodami išorinius išteklius (pvz., standartines Python bibliotekų funkcijas) ir pateikite patikrinimo rezultatus.

## Netiesinių lygčių sistemų sprendimas

Duota netiesinių lygčių sistema:

* Skirtinguose grafikuose pavaizduokite paviršius 𝑍1 (𝑥1, 𝑥2 ) ir 𝑍2 (𝑥1, 𝑥2 ).
* Užduotyje pateiktą netiesinių lygčių sistemą išspręskite grafiniu būdu.
* Užduotyje pateiktą netiesinių lygčių sistemą išspręskite naudodami užduotyje nurodytą metodą su laisvai pasirinktu pradiniu artiniu (išbandykite bent keturis pradinius artinius). Nurodykite iteracijų pabaigos sąlygas. Lentelėje pateikite pradinį artinį, tikslumą, iteracijų skaičių.
* Gautus sprendinius patikrinkite naudodami išorinius išteklius (pvz., standartines Python bibliotekų funkcijas) ir pateikite patikrinimo rezultatus.

# Uždavinio sprendimas

## Tiesinių lygčių sistemų sprendimas

Pirmoji lygtis:

Antroji lygtis:

### Gauso metodas

* Programa įvertina atvejus:

1. kai lygčių sistema turi vieną sprendinį;
2. kai lygčių sistema sprendinių neturi;
3. kai lygčių sistema turi be galo daug sprendinių.

* Programos kodo dalis:

**if** aug\_matrix[aug\_matrix.shape[0] - 1, aug\_matrix.shape[1] - 2] == 0:  
 **if** aug\_matrix[aug\_matrix.shape[0] - 1, aug\_matrix.shape[1] - 1] == 0:  
 **return** print(**"sprendinių be galo daug"**)  
 **else**:  
 **return** print(**"sprendinių nėra"**)

* Gauso metodo programos kodo dalis:

**def** GaussMethod(matrix\_A, matrix\_b):  
 n = (np.shape(matrix\_A))[0] *# lygciu skaicius nustatomas pagal ivesta matrica A* nb = (np.shape(matrix\_b))[1] *# laisvuju nariu vektoriu skaicius nustatomas pagal ivesta matrica b* aug\_matrix = np.hstack((matrix\_A, matrix\_b)) *# isplestoji matrica* print(**"A = \n"**, matrix\_A[0:4, 0:4])  
 print(**"b = \n"**, matrix\_b[:, 0])  
  
 *# 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* aug\_matrix[j, i:n + nb] = aug\_matrix[j, i:n + nb] - aug\_matrix[i, i:n + nb] \* aug\_matrix[j, i] / aug\_matrix[  
 i, i]  
 aug\_matrix[j, i] = 0  
 print(i + 1, **"iteration"**)  
 print(aug\_matrix)  
 print()  
  
 *# atvirkstinis etapas:* **if** aug\_matrix[aug\_matrix.shape[0] - 1, aug\_matrix.shape[1] - 2] == 0:  
 **if** aug\_matrix[aug\_matrix.shape[0] - 1, aug\_matrix.shape[1] - 1] == 0:  
 **return** print(**"sprendinių be galo daug"**)  
 **else**:  
 **return** print(**"sprendinių nėra"**)  
 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, :] = (aug\_matrix[i, n:n + nb] - aug\_matrix[i, i + 1:n] \* x[i + 1:n, :]) / aug\_matrix[i, i]  
 **for** i **in** range(len(matrix\_b)):  
 print(**"x"**, i + 1, **"="**, **"{:.2f}"**.format(x[i, 0]))  
 ans = np.zeros(shape=(n, nb))  
 **for** i **in** range(0, n):  
 **for** j **in** range(0, n):  
 ans[i, 0] = ans[i, 0] + x[j, 0] \* matrix\_A[i, j]  
 ansx = np.zeros(shape=(n, nb))  
 *# for i in range(0,n):  
 # for j in range(0,n):  
 # ansx[i,0] = ansx[i,0]+ans[i,j]* ans.transpose()  
 print()  
 print(**"Initial matrix b"**)  
 **for** i **in** range(len(ans)):  
 print(**"{:.2f}"**.format(ans[i, 0]))

* Rezultatai Python konsolėje:

-----------------------------------------------------------------

Gauss Method

-----------------------------------------------------------------

First linear equation:

-----------------------------------------------------------------

A =

[[ 3. 1. -1. 5.]

[-3. 4. -8. -1.]

[ 1. -3. 7. 6.]

[ 0. 5. -9. -4.]]

b =

[[ 20.]

[-36.]

[ 41.]

[-16.]]

1 iteration

[[ 3. 1. -1. 5. 20. ]

[ 0. 5. -9. 4. -16. ]

[ 0. -3.33333333 7.33333333 4.33333333 34.33333333]

[ 0. 5. -9. -4. -16. ]]

2 iteration

[[ 3. 1. -1. 5. 20. ]

[ 0. 5. -9. 4. -16. ]

[ 0. 0. 1.33333333 7. 23.66666667]

[ 0. 0. 0. -8. 0. ]]

3 iteration

[[ 3. 1. -1. 5. 20. ]

[ 0. 5. -9. 4. -16. ]

[ 0. 0. 1.33333333 7. 23.66666667]

[ 0. 0. 0. -8. 0. ]]

x 1 = 3.00

x 2 = 28.75

x 3 = 17.75

x 4 = -0.00

Initial matrix b

20.00

-36.00

41.00

-16.00

-----------------------------------------------------------------

Second linear equation:

-----------------------------------------------------------------

A =

[[ 5. 2. 0. 1.]

[ 2. -5. 0. 2.]

[ 9. -6. -6. 1.]

[ 1. 2. 1. 1.]]

b =

[[19.]

[-5.]

[39.]

[ 5.]]

1 iteration

[[ 5. 2. 0. 1. 19. ]

[ 0. -5.8 0. 1.6 -12.6]

[ 0. -9.6 -6. -0.8 4.8]

[ 0. 1.6 1. 0.8 1.2]]

2 iteration

[[ 5. 2. 0. 1. 19. ]

[ 0. -5.8 0. 1.6 -12.6 ]

[ 0. 0. -6. -3.44827586 25.65517241]

[ 0. 0. 1. 1.24137931 -2.27586207]]

3 iteration

[[ 5. 2. 0. 1. 19. ]

[ 0. -5.8 0. 1.6 -12.6 ]

[ 0. 0. -6. -3.44827586 25.65517241]

[ 0. 0. 0. 0.66666667 2. ]]

x 1 = 2.00

x 2 = 3.00

x 3 = -6.00

x 4 = 3.00

Initial matrix b

19.00

-5.00

39.00

5.00

## Paprastųjų iteracijų metodas

Chart, line chart

Description automatically generated

pav. 1 Pirmosios lygties santykinio tikslumo kaita

Chart

Description automatically generated

pav. 2 Antrosios lygties santykinio tikslumo kaita

* Paprastųjų iteracijų programos kodo dalis:

**def** SimpleIterationMethod(A, b, alpha, max\_epochs, precision):  
 **if** np.linalg.det(A) == 0:  
 **return** print(**"LinAlgErr: Singular matrix"**)  
 n = np.shape(A)[0]  
 *# laisvai parinkti metodo parametrai* Atld = np.diag(1. / np.diag(A)).dot(A) - np.diag(alpha)  
 btld = np.diag(1. / np.diag(A)).dot(b)  
 prec = [] *# tuscias sarasas tikslumo reiksmiukaupimui* x = np.zeros(shape=(n, 1))  
 x1 = np.zeros(shape=(n, 1))  
 **for** it **in** range(0, max\_epochs):  
 x1 = ((btld - Atld.dot(x)).transpose() / alpha).transpose()  
 prec.append(  
 np.linalg.norm(x1 - x, ord=np.inf) / (np.linalg.norm(x, ord=np.inf) + np.linalg.norm(x1, ord=np.inf))  
 )  
 **if** prec[it] < precision:  
 **break** x[:] = x1[:]  
 print(x.transpose())  
 **return** prec

* Paprastųjų iteracijų grafinio atvaizdavimo kodo dalis:

**def** plot(y, alpha, equation):  
 x = np.arange(len(y))  
 plt.plot(x, y, label=**'$alpha = {i}$'**.format(i=alpha))  
 plt.ylim(1e-12, 2e-1)  
 plt.yscale(**'logit'**)  
 plt.title(equation)

**def** ExecuteFirstTask():  
 **for** i **in** range(2, 7):  
 CONST\_alpha[:] = i  
 prec = SimpleIterationMethod(A1, b11, CONST\_alpha, CONST\_max\_epochs, CONST\_e)  
 plot(prec, i, **"First linear equation"**)  
 plt.legend(loc=**'best'**)  
 plt.show()

* Rezultatai Python konsolėje:

-----------------------------------------------------------------

Simple Iteration Method

-----------------------------------------------------------------

First linear equation:

-----------------------------------------------------------------

[[3.00000000e+00 2.87500000e+01 1.77500000e+01 6.58033628e-11]]

[[ 3.00000000e+00 2.87500000e+01 1.77500000e+01 -1.17059547e-10]]

[[3.00000000e+00 2.87500000e+01 1.77500000e+01 1.68171255e-10]]

[[ 3.00000000e+00 2.87500000e+01 1.77500000e+01 -1.70778947e-10]]

[[ 3.00000000e+00 2.87500000e+01 1.77500000e+01 -2.52941372e-10]]

-----------------------------------------------------------------

Second linear equation:

-----------------------------------------------------------------

[[ 2. 3. -6. 3.]]

[[ 2. 3. -6. 3.]]

[[ 2. 3. -6. 3.]]

[[ 2. 3. -6. 3.]]

[[ 2. 3. -6. 3.]]

-----------------------------------------------------------------

Python numpy solver

-----------------------------------------------------------------

First linear equation:

-----------------------------------------------------------------

[[ 3. ]

[28.75]

[17.75]

[-0. ]]

-----------------------------------------------------------------

Second linear equation:

-----------------------------------------------------------------

[[ 2.]

[ 3.]

[-6.]

[ 3.]]

-----------------------------------------------------------------

END

-----------------------------------------------------------------

## Netiesinių lygčių sprendimas

### Lygtis

### Metodas

Šiai užduočiai atlikti buvo panaudotas Niutono metodas. Niutono metodo formulė:

Niutono metodas visuomet konverguoja, kai pradedama skaičiuoti nuo gero pradinio artinio. Kiekvienos iteracijos metu reikia apskaičiuoti funkcijos ir Jakobio matricos reikšmes.

Chart, surface chart

Description automatically generated

pav. 3 Pirmosios ir antrosios lygties grafinis atvaizdavimas

Chart, line chart

Description automatically generated

pav. 4 Lygčių sistemos grafinis sprendimas

### Niutono metodo programos kodo dalis:

**def** function(xy):  
 x, y = xy  
 **return** [x \*\* 2 + 2 \* ((y - np.cos(x)) \*\* 2) - 20,  
 x \*\* 2 \* y - 2]  
  
  
**def** jacobian(xy):  
 x, y = xy  
 **return** [[2 \* x + 4 \* np.sin(x) \* (y - np.cos(x)), 4 \* (y - np.cos(x))],  
 [2 \* x \* y, x \*\* 2]]  
  
  
**def** iterative\_newton(fun, x\_init, jacobian\_fun):  
 max\_epochs = 50  
 epsilon = 1e-12  
  
 x\_last = x\_init  
  
 **for** epoch **in** range(max\_epochs):  
 J = np.array(jacobian\_fun(x\_last))  
 F = np.array(fun(x\_last))  
  
 diff = np.linalg.solve(J, -F)  
 x\_last = x\_last + diff  
  
 *# Stop condition:* **if** np.linalg.norm(diff) < epsilon:  
 print(**'convergence!, epoch:'**, epoch)  
 **break  
  
 else**: *# only if the for loop end 'naturally'* print(**'not converged'**)  
  
 **return** x\_last

### Rezultatai

Iteracijų pabaigos sąlygos:

* Daugiausiai iteracijų: 50
* Tikslumas: 1e-12

|  |  |  |
| --- | --- | --- |
| Pradinis artinys | Tikslumas | Iteracijų skaičius |
| [-5;4] | [-3.552713678800501e-15, -6.661338147750939e-16] | 10 |
| [-2;4] | [0.0, 0.0] | 6 |
| [0.5;2] | [0.0, 8.881784197001252e-16] | 6 |
| [3;0.5] | [-3.552713678800501e-15, -4.440892098500626e-16] | 6 |

Rezultatai Python konsolėje:

Solutions by python

x1 = [-4.44160772 0.10137937]

x2 = [-0.71851577 3.87398007]

x3 = [0.71851577 3.87398007]

x4 = [4.44160772 0.10137937]

convergence!, epoch: 10

solution found at: [-4.44160772 0.10137937]

F(x1) [-3.552713678800501e-15, -6.661338147750939e-16]

convergence!, epoch: 6

solution found at: [-0.71851577 3.87398007]

F(x2) [0.0, 0.0]

convergence!, epoch: 6

solution found at: [0.71851577 3.87398007]

F(x3) [0.0, 8.881784197001252e-16]

convergence!, epoch: 6

solution found at: [4.44160772 0.10137937]

F(x4) [-3.552713678800501e-15, -4.440892098500626e-16]

# Išvados

Pirmosios dvi užduotys buvo sėkmingai realizuotos, tačiau pastebėta, jog paprastųjų iteracijų metodas gali diverguoti netgi esant nesinguliariai lygčiai. Kad konverguotų, gali tekti sukeisti lygčių eilutes lygčių sistemoje, todėl, skaičiuojant antrosios lygties sprendinius, paprastųjų iteracijų metodu, reikėjo sukeisti lygčių eilutes vietomis. Iš **pav. 1** ir **pav. 2** galime padaryti išvadą, jog parinkus skirtingas *alpha* reikšmes, funkcijos sprendinių tikslumą pasiekia per skirting iteracijų skaičių.

# Paveikslėlių sąrašas

[pav. 1 Pirmosios lygties santykinio tikslumo kaita 8](#_Toc85649292)

[pav. 2 Antrosios lygties santykinio tikslumo kaita 9](#_Toc85649293)

[pav. 3 Pirmosios ir antrosios lygties grafinis atvaizdavimas 11](#_Toc85649294)

[pav. 4 Lygčių sistemos grafinis sprendimas 12](#_Toc85649295)