KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

Skaitiniai metodai ir algoritmai (P170B115)

Laboratorinių darbų ataskaita

Atliko:

IFF-1/4 gr. studentas

Mildaras Karvelis

2023 m. lapkričio 30 d.

Priėmė:

Prof. Barauskas Rimantas

KAUNAS 2023

TURINYS

1. Tiesinių lygčių sistemų sprendimas

1.1. Tiesinės lygtys, 1 ir 2 lentelės

1.1.1 Atspindžio metodas (1, 14, 20 lygtys)

1	$\begin{cases} 3x_1 + 7x_2 + x_3 + 3x_4 = 37 \\ x_1 - 6x_2 + 6x_3 + 9x_4 = 11 \\ 4x_1 + 4x_2 - 7x_3 + x_4 = 38 \\ -3x_1 + 8x_2 + 2x_3 + x_4 = 0 \end{cases}$

Gautos x reikmės sprendžiant metodu: [5.55, 1.97, -0.78, 2.44]
Gautos x reikmės patikrinant su Python metodais: [5.55, 1.97, -0.78, 2.44]
Gautos B reikmės įstačius x: [37, 11, 38, 0]

14	$\begin{cases} 2x_1 + 4x_2 + 6x_3 - 2x_4 = 4 \\ x_1 + 3x_2 + x_3 - 3x_4 = -7 \\ x_1 + x_2 + 5x_3 + x_4 = 11 \\ 2x_1 + 3x_2 - 3x_3 - 2x_4 = -4 \end{cases}$

Lygčių sistema turi begalo daug sprendinių

20	$\begin{cases} 2x_1 + 4x_2 + 6x_3 - 2x_4 = 2 \\ x_1 + 3x_2 + x_3 - 3x_4 = 1 \\ x_1 + x_2 + 5x_3 + x_4 = 7 \\ 2x_1 + 3x_2 - 3x_3 - 2x_4 = 2 \end{cases}$

Lygčių sistema turi begalo daug sprendinių

Kodo fragmentai

```
import numpy as np
import matplotlib.pyplot as plt
import math
import random
from scipy.optimize import fsolve
from scipy.linalg import inv
from scipy.linalg import lu

def check_matrix(A1, x, row, n):
  b = A1[row, n]
  b = b - A1[row, row+1:n]*x[row+1:n,:]
  eps = 1e-4
  if b > eps:
    print("Sprendinių nėra")
  else:
    print("Sprendinių begalo daug")

def reflection(A, b):
    n=(np.shape(A))[0] # lygciu skaicius nustatomas pagal ivesta matrica A
    nb=(np.shape(b))[1] # laisvuju nariu vektoriu skaicius nustatomas pagal ivesta
    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,:])
       # atgalinis etapas:
    x=np.zeros(shape=(n,1))
    eps = 1e-4
    flag = True
    for i in range (n-1,-1,-1):
      if abs(A1[i,i]) < eps:
        flag = False
        check_matrix(A1, x, i, n)
        break
      x[i,:]=(A1[i,n:n+n]-A1[i,i+1:n]*x[i+1:n,:])/A1[i,i]
    if flag:
      print("Gautos x reikšmės naudojant Atspindžio algoritmą:", np.round(x, 2))

      Ax = np.dot(A, x)
      print("Gautos B reikšmės įsistacius x: ", np.round(Ax, 2))

      solution = np.linalg.solve(A, b)
      print("Gautos x reikšmės naudojant Python metodus:", np.round(solution, 2))

print("1 sistema")
A1 = np.matrix([[3, 7, 1, 3],
              [1, -6, 6, 9],
              [4, 4, -7, 1],
              [-3, 8, 2, 1]]).astype(np.float64)
B1 = (np.matrix([37, 11, 38, 0])).transpose().astype(np.float64)
reflection(A1, B1)

print(" ")
print("14 sistema")
A2 = np.matrix([[2, 4, 6, -2],
              [1, 3, 1, -3],
              [1, 1, 5, 1],
              [2, 3, -3, -2]]).astype(np.float64)
B2 = (np.matrix([4, -7, 11, -4])).transpose().astype(np.float64)
reflection(A2, B2)


print(" ")
print("20 sistema")
A3 = np.matrix([[2, 4, 6, -2],
              [1, 3, 1, -3],
              [1, 1, 5, 1],
              [2, 3, -3, -2]]).astype(np.float64)
B3 = (np.matrix([2, 1, 7, 2])).transpose().astype(np.float64)
reflection(A3, B3)
```

1.1.2 Gauso-Zeidelio metodas(1 lygtis)

1	$\begin{cases} 3x_1 + 7x_2 + x_3 + 3x_4 = 37 \\ x_1 - 6x_2 + 6x_3 + 9x_4 = 11 \\ 4x_1 + 4x_2 - 7x_3 + x_4 = 38 \\ -3x_1 + 8x_2 + 2x_3 + x_4 = 0 \end{cases}$

Gautos x reikmės sprendžiant Gauso-Zeidelio metodu:

5.55173256	1.97200467	-0.78065453	2.44046936

Gautos x reikmės patikrinant su Python metodais:

5.55173636	1.97200567	-0.78065202	2.44046775

Gautos B reikmės įstačius x:

36.99998394	11.00000164	37.99999998	0

Kodo fragmentai

```
import numpy as np
import matplotlib.pyplot as plt
import math
import random
from scipy.optimize import fsolve
from scipy.linalg import inv
from scipy.linalg import lu

def check_matrix(A1, x, row, n):
  b = A1[row, n]
  b = b - A1[row, row+1:n]*x[row+1:n,:]
  eps = 1e-4
  if b > eps:
    print("Sprendinių nėra")
  else:
    print("Sprendinių begalo daug")

def gaussSeidel(A,b):
  n=np.shape(A)[0]
  alpha = np.array([100, 20, 1, 1])
  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(1, nitmax + 1):
    for i in range(n):
      x1[i]=(btld[i]-Atld[i,:].dot(x1))/alpha[i];
    prec_val=(np.linalg.norm(x1-x)/(np.linalg.norm(x)+np.linalg.norm(x1)))
    if prec_val < eps:
      break
    x[:]=x1[:]
  return x
```

```
print("1 sistema")
A = np.array([[3, 7, 1, 3],
              [1, -6, 6, 9],
              [4, 4, -7, 1],
              [-3, 8, 2, 1]]).astype(np.float64)
B = np.array([37, 11, 38, 0]).astype(np.float64)
x0 = np.array([0, 0, 0, 0]).astype(np.float64)
solutionp = np.linalg.solve(A, B)
print("Gautos x reiksmes naudojant python metodus:", solutionp)

solution = gaussSeidel(A, B)
print("Gautos x reiksmes naudojant gauso-zeidelio algoritma:", solution)
Ax = np.dot(A, solution)
print("Gautos B reiksmes isistacius x: ", Ax)
```

1.2. 3 lentelė

8.	$\begin{cases} 2x_1 + 3x_2 + x_3 + x_4 = \cdots \\ 2x_2 + x_3 + 3x_4 = \cdots \\ 7x_1 - 4x_2 + x_3 + x_4 = \cdots \\ x_1 - 12x_2 + x_3 + x_4 = \cdots \end{cases}$	$\begin{cases} \ldots = 7 \\ \ldots = 6 \\ \ldots = 5 \\ \ldots = -9 \end{cases}$	$\begin{cases} \ldots = 38 \\ \ldots = 50 \\ \ldots = 1 \\ \ldots = -53 \end{cases}$	$\begin{cases} \ldots = -3.5 \\ \ldots = -4 \\ \ldots = -8.5 \\ \ldots = -1.25 \end{cases}$	LU

Lygčių sistemos sprendinys naudojant B1 reikšmes: [1, 1, 1, 1]
Sprendinio patikrinimas įstačius reikšmes: [7, 6, 5, -9]
Patikrinimas naudojant Python funkcijas: [1, 1, 1, 1]

Lygčių sistemos sprendinys naudojant B2 reikšmes: [1, 6, 8, 10]
Sprendinio patikrinimas įstačius reikšmes: [38, 50, 1, -53]
Patikrinimas naudojant Python funkcijas: [1, 6, 8, 10]

Lygčių sistemos sprendinys naudojant B3 reikšmes: [-1.107, -0.076, 0.337, -1,395]
Sprendinio patikrinimas įstačius reikšmes: [-3.5, -4, -8.5, -1.25]
Patikrinimas naudojant Python funkcijas: [-1.107, -0.076, 0.337, -1,395]

1.2.1 Kodo fragmentai

```
# LU skaidos algoritmas
def lu_decomposition(A):
    n = len(A)
    L = np.zeros((n, n))
    U = np.zeros((n, n))

    for i in range(n):
        L[i][i] = 1
        for j in range(i, n):
            U[i][j] = A[i][j]
            for k in range(i):
                U[i][j] -= L[i][k] * U[k][j]
        for j in range(i + 1, n):
            L[j][i] = A[j][i] / U[i][i]
            for k in range(i):
                L[j][i] -= L[j][k] * U[k][i] / U[i][i]
```

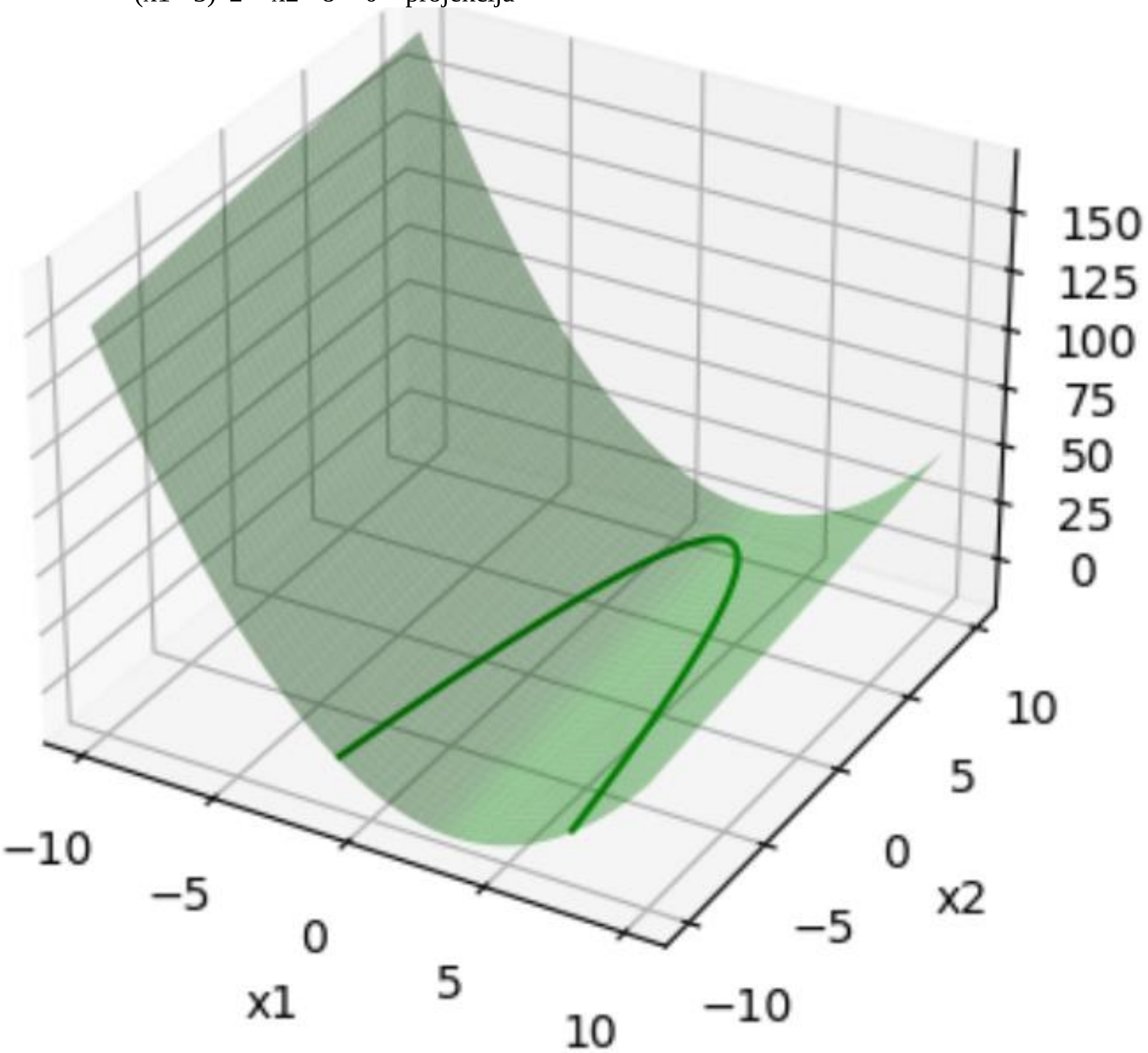
```
    return L, U

# Lygčių sistemos sprendimas naudojant LU
def solve_LU(A, b):
    L, U = lu_decomposition(A)
    n = len(A)
    y = np.zeros(n)
    x = np.zeros(n)

    # Lygties L*y = b sprendimas
    for i in range(n):
        y[i] = b[i]
        for j in range(i):
            y[i] -= L[i][j] * y[j]

    # Lygties U*x = y sprendimas
    for i in range(n - 1, -1, -1):
        x[i] = y[i]
        for j in range(i + 1, n):
            x[i] -= U[i][j] * x[j]
        x[i] /= U[i][i]

    return x

# Užduoties duomenys
A = np.array([[2, 3, 1, 1],
              [0, 2, 1, 3],
              [7, -4, 1, 1],
              [1, -12, 1, 1]])
b1 = np.array([7, 6, 5, -9])
b2 = np.array([38, 50, 1, -53])
b3 = np.array([-3.5, -4, -8.5, -1.25])

x1 = solve_LU(A, b1)
x2 = solve_LU(A, b2)
x3 = solve_LU(A, b3)

print(f"Sprendinys naudojant b1: {x1}")
print(f"Sprendinys naudojant b2: {x2}")
print(f"Sprendinys naudojant b3: {np.round(x3, 3)}")
def check_solution(matrix, solution, expected):
    result = np.dot(matrix, solution) #sudauginant gaunamas vektorius
    return np.allclose(result, expected) #palyginama - true, false
print(f"Sprendinio su b1 patikrinimas: {check_solution(A, x1, b1)}")
print(f"Sprendinio su b2 patikrinimas: {check_solution(A, x2, b2)}")
print(f"Sprendinio su b3 patikrinimas: {check_solution(A, x3, b3)}")

x1_check = np.linalg.solve(A, b1)
print(f"Patikrinimas naudojant numpy funkciją b1: {x1_check}")

x2_check = np.linalg.solve(A, b2)
print(f"Patikrinimas naudojant numpy funkciją b1: {x2_check}")

x3_check = np.linalg.solve(A, b3)
print(f"Patikrinimas naudojant numpy funkciją b1: {np.round(x3_check, 3)}")
```

2. Netiesinių lygčių sistemų sprendimas

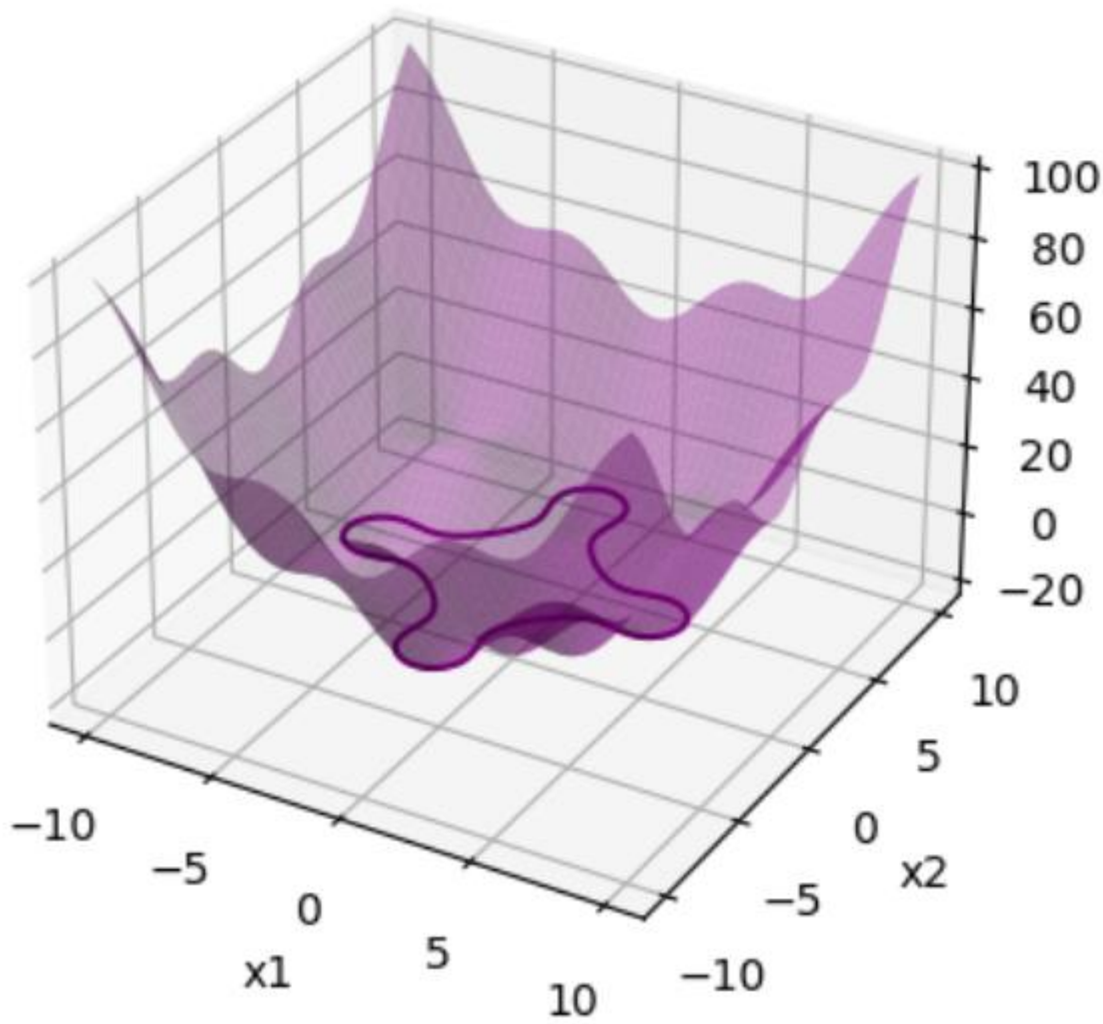
8	$\begin{cases} (x_1 - 3)^2 + x_2 - 8 = 0 \\ \frac{x_1^2 + x_2^2}{2} - 6(\cos(x_1) + \cos(x_2)) - 10 = 0 \end{cases}$	Broideno

2.1. a dalis

(x1 - 3)^2 + x2 - 8 = 0 – projekcija

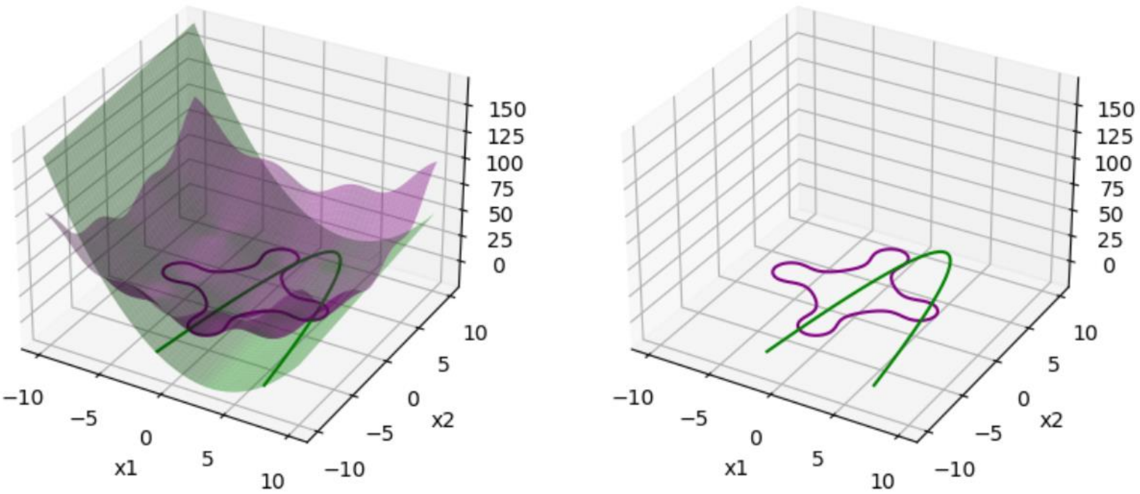


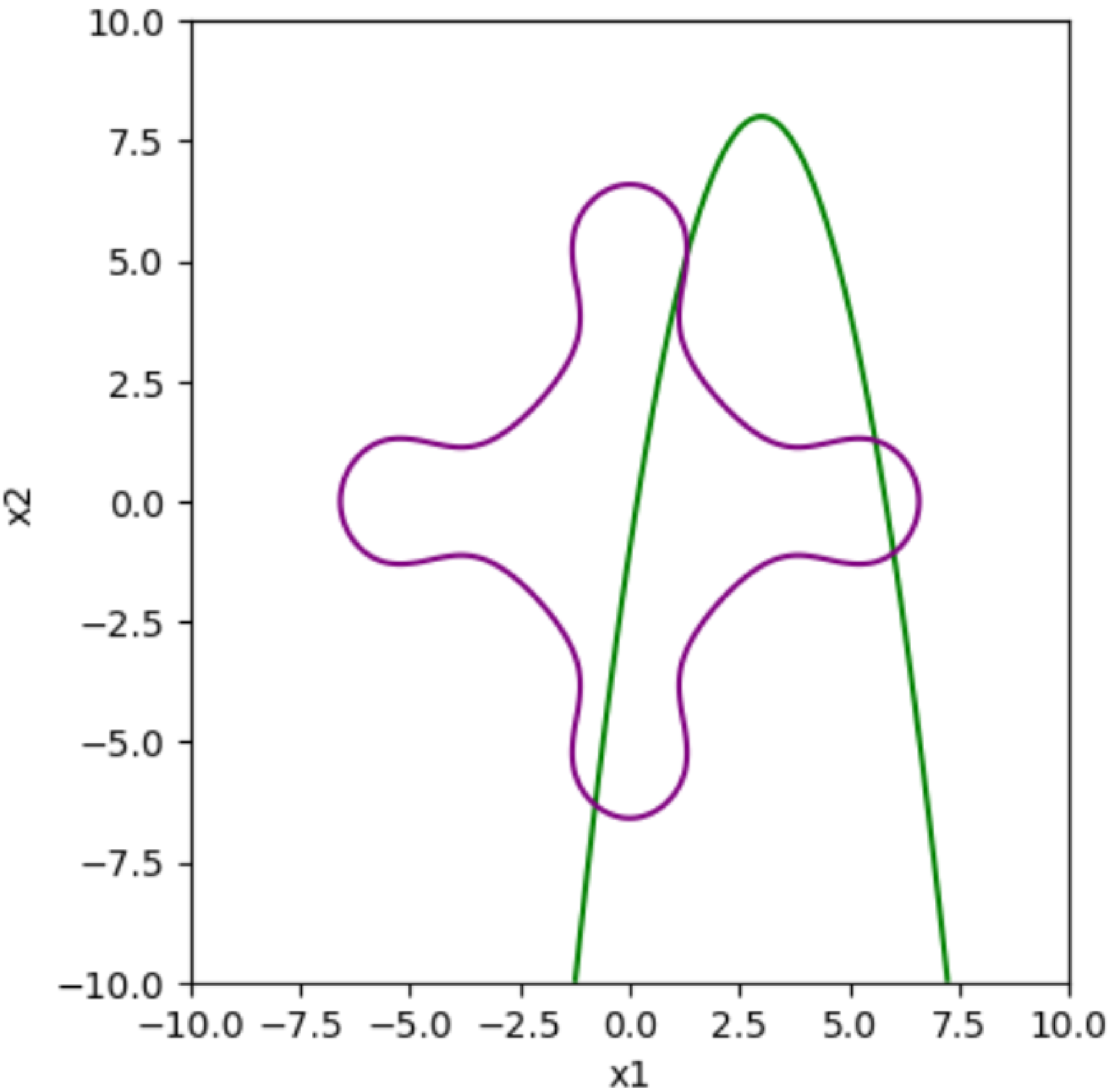
$$\frac{x_1^2 + x_2^2}{2} - 6(\cos x_1 + \cos x_2) - 10 = 0 - projekcija$$



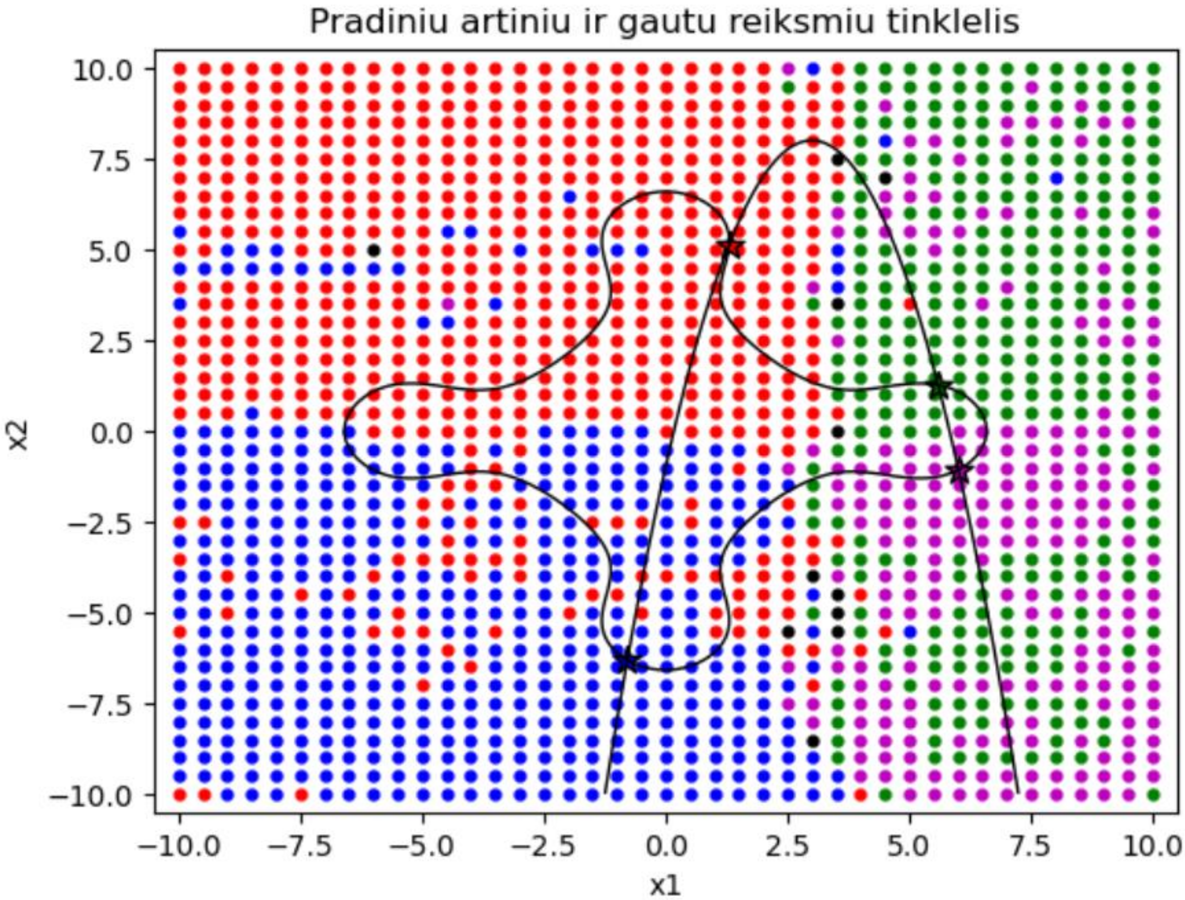
2.2. *b dalis*

Grafinis sprendinys:





2.3. c dalis



Pradinis artinys	Gauta reikšmė
[-1, -6]	[-0.78353562 -6.3151418]
[1, 5]	[1.30474092 5.12609666]
[6, 2]	[5.59552862 1.26323116]
[7, -1]	[6.01207886 -1.07261907]

Pirmas sprendinys naudojant fsolve: [-0.78353562 -6.3151418]
Antras sprendinys naudojant fsolve: [1.30474092 5.12609667]
Trecias sprendinys naudojant fsolve: [5.59552862 1.26323116]
Ketvirtas sprendinys naudojant fsolve: [6.01207886 -1.07261906]

2.4. Kodo fragmentai

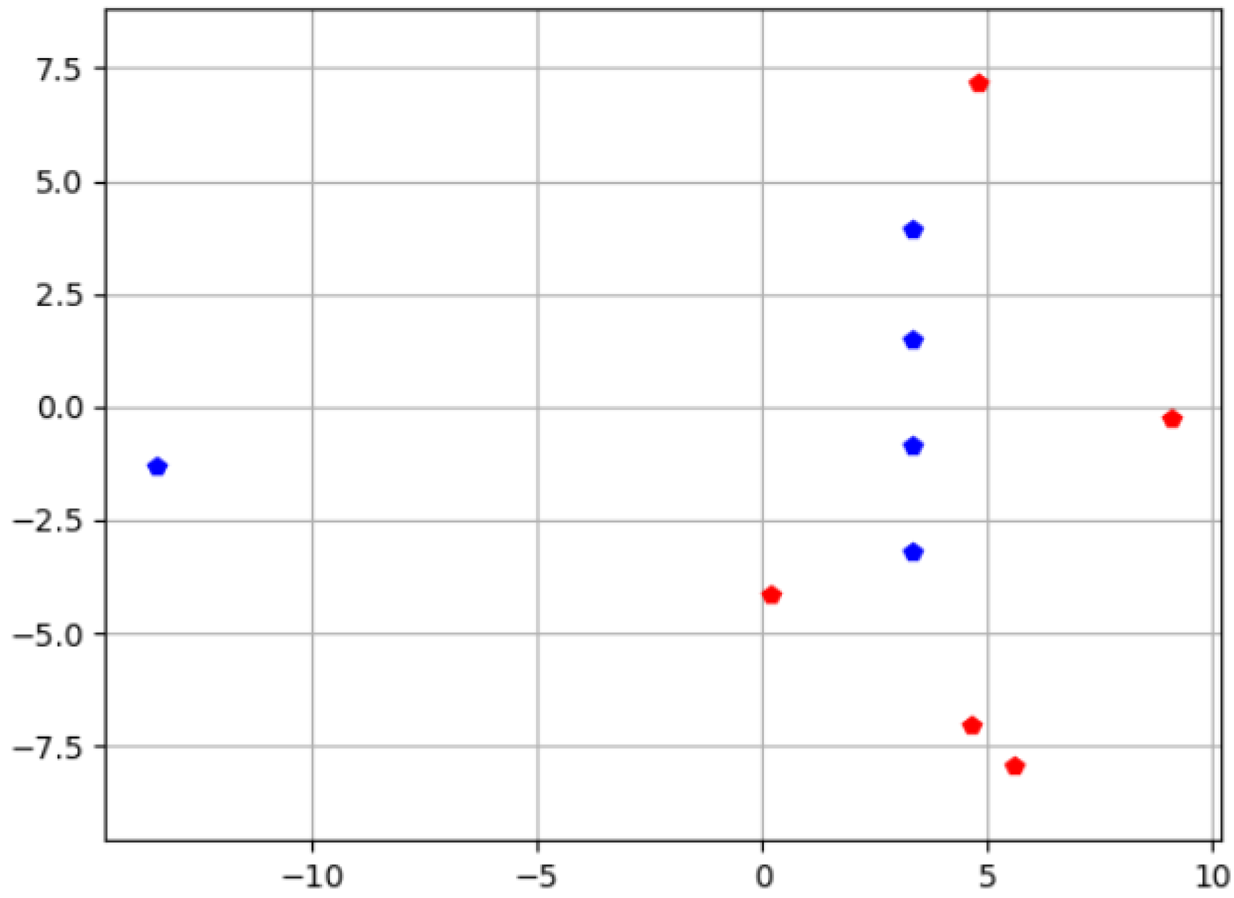
```
def nonlinearSystem(x):
  eq1 = (x[0]-3)**2 + x[1] - 8
  eq2 = (x[0]**2+x[1]**2)/2-6*(np.cos(x[0])+np.cos(x[1]))-10
  return np.array([eq1, eq2])

def createFigures():
  fig1 = plt.figure(1, figsize=plt.figaspect(0.5))
  fig2 = plt.figure(2, figsize=plt.figaspect(0.5))
  fig3 = plt.figure(3, figsize=plt.figaspect(0.5))
  ax1 = fig2.add_subplot(1, 2, 1, projection='3d')
  ax2 = fig2.add_subplot(1, 2, 2, projection='3d')
  ax3 = fig1.add_subplot(1, 2, 1, projection='3d')
  ax4 = fig1.add_subplot(1, 2, 2, projection='3d')
  ax5 = fig3.add_subplot(1, 2, 1)
  for ax in [ax1, ax2, ax3, ax4, ax5]:
    ax.set_xlabel('x1')
```

```
    ax.set_ylabel('x2')
  return fig1, fig2, fig3, ax1, ax2, ax3, ax4, ax5
def createGrid():
  xx = np.linspace(-10, 10, 200)
  yy = np.linspace(-10, 10, 200)
  X, Y = np.meshgrid(xx, yy)
  return X, Y
def createSurfaceAndContour(ax, X, Y, Z, color):
  surf = ax.plot_surface(X, Y, Z, color=color, alpha=0.4, linewidth=0.1, antialiased=True)
  CS = ax.contour(X, Y, Z, [0], colors=color)
  return surf, CS

X, Y = createGrid()
Z = np.zeros(shape=(len(X), len(Y), 2))
for i in range(len(X)):
  for j in range(len(Y)):
    Z[i, j, :] = nonlinearSystem([X[i, j], Y[i, j]]).transpose()
fig1, fig2, fig3, ax1, ax2, ax3, ax4, ax5 = createFigures()
surf1, CS11 = createSurfaceAndContour(ax1, X, Y, Z[:, :, 0], 'green')
surf2, CS12 = createSurfaceAndContour(ax1, X, Y, Z[:, :, 1], 'purple')
CS1 = ax2.contour(X, Y, Z[:, :, 0], [0], colors='green')
CS2 = ax2.contour(X, Y, Z[:, :, 1], [0], colors='purple')
surf3, CS3 = createSurfaceAndContour(ax3, X, Y, Z[:, :, 0], 'green')
surf4, CS4 = createSurfaceAndContour(ax4, X, Y, Z[:, :, 1], 'purple')
CS51 = ax5.contour(X, Y, Z[:, :, 0], [0], colors='green')
CS52 = ax5.contour(X, Y, Z[:, :, 1], [0], colors='purple')
plt.show()

def broyden(f, x0, maxiter=1000, eps=1e-6):
  n = len(x0)
  dx = 0.1
  A = np.zeros(shape=(n, n))
  x = x0
  for i in range(n):
    x1 = x.copy()
    x1[i] += dx
    A[:, i] = (f(x1) - f(x)) / dx
  ff = f(x)
  for i in range(1, maxiter):
    deltax = -np.linalg.solve(A, ff)
    x1 = x + deltax
    ff1 = f(x1)
    A += np.outer(ff1 - ff - np.dot(A, deltax), deltax) / np.dot(deltax, deltax)
    s = 0
    if np.sum(np.abs(x + x1)) > eps:
      s = np.sum(np.abs(x - x1)) / np.sum(np.abs(x + x1) + np.abs(ff) + np.abs(ff1))
    else:
      s = np.sum(np.abs(x - x1) + np.abs(ff) + np.abs(ff1))
    ff = ff1
    x = x1
    if s < eps:
      break
  return x

def findSolutions():
  x1_values = np.arange(-10, 10.5, 0.5)
  x2_values = np.arange(-10, 10.5, 0.5)
```

```
  solutions = []
  for x1 in x1_values:
    for x2 in x2_values:
      initial_guess = np.array([x1, x2])
      solution = broyden(nonlinearSystem, initial_guess, 100)
      solutions.append((initial_guess, solution))
  return solutions

def plotSolutions(solutions, X1, X2, X3, X4, eps=1e-6):
  x1_initial, x2_initial = zip(*[initial for initial, _ in solutions])
  x1_solution, x2_solution = zip(*[solution for _, solution in solutions])
  plt.figure()
  plt.xlabel('x1')
  plt.ylabel('x2')
  plt.contour(X, Y, Z[:, :, 0], [0], colors='black', linewidths=1)
  plt.contour(X, Y, Z[:, :, 1], [0], colors='black', linewidths=1)
  plt.title('Pradiniu artiniu ir gautu reiksmiu tinklelis')
  for solution in solutions:
    colors = ['b', 'r', 'g', 'm']
    markers = ['.', '.', '.', '.']
    labels = [X1, X2, X3, X4]
    for i, label in enumerate(labels):
      if np.allclose(solution[1], label, atol=eps):
        plt.scatter(solution[0][0], solution[0][1], marker=markers[i], color=colors[i], s=50)
        break
    else:
      plt.scatter(solution[0][0], solution[0][1], marker='.', color='black', s=50)
  for i, label in enumerate(labels):
    plt.scatter(label[0], label[1], marker='*', color=colors[i], s=100, edgecolors='black')
  plt.xlim(-10.5, 10.5)
  plt.ylim(-10.5, 10.5)
  plt.show()

X1 = broyden(nonlinearSystem, np.array([-1, -6]).astype(np.float64))
X2 = broyden(nonlinearSystem, np.array([1, 5]).astype(np.float64))
X3 = broyden(nonlinearSystem, np.array([6, 2]).astype(np.float64))
X4 = broyden(nonlinearSystem, np.array([7, -1]).astype(np.float64))
solutions = findSolutions()
plotSolutions(solutions, X1, X2, X3, X4)
print('{0: >0}'.format('Pradiniu artiniu ir gautu reiksmiu lentele'))
print("----------------------------------------------------")
print('|' + '{0: >20} {1: >30}'.format('Pradinis Artinys |', 'Gauta Reiksme |'))
print("----------------------------------------------------")
print('|' + '{0: >20} {1: >30}'.format('[-1, -6]' + ' |', str(X1) + ' |'))
print('|' + '{0: >20} {1: >30}'.format('[1, 5]' + ' |', str(X2) + ' |'))
print('|' + '{0: >20} {1: >30}'.format('[6, 2]' + ' |', str(X3) + ' |'))
print('|' + '{0: >20} {1: >30}'.format('[7, -1]' + ' |', str(X4) + ' |'))
print("----------------------------------------------------")

solution1 = fsolve(nonlinearSystem, [-1, -6])
print("Pirmas sprendinys:", solution1)
solution2 = fsolve(nonlinearSystem, [1, 5])
print("Antras sprendinys:", solution2)
solution3 = fsolve(nonlinearSystem, [6, 2])
print("Trecias sprendinys:", solution3)
solution4 = fsolve(nonlinearSystem, [7, -1])
print("Ketvirtas sprendinys:", solution4)
```

3. Optimizavimas

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



3.1. Kodo fragmentai

```
def Target(X, Y, dist):  # Tikslo funkciją
    n = len(X)  # Skaičiuojame parduotuvių kiekį
    full_distance = 0  # Inicializuojame bendrą atstumo kintamajį

    for i in range(n):
        for j in range(i + 1, n):
            # Skaičiuojame atstumą tarp parduotuvių naudodami eksponentinę funkciją
            computation_distance = np.exp(-0.3 * ((X[i] - X[j]) ** 2 + (Y[i] - Y[j]) ** 2))
            # Skaičiuojame kainą pagal duotus formules
```

```
        compuation_price = ((X[i] ** 4 + Y[i] ** 4) / 1000) + ((np.sin(X[i]) + np.cos(X[i])) / 5) + 0.4
        # Pridedame atstumą ir kainą prie bendro atstumo
        full_distance += computation_distance + compuation_price

    # Pridedame vidurkį kvadratų prie bendro atstumo
    full_distance = full_distance + np.average(X) ** 2 + np.average(Y) ** 2
    return full_distance

# Gradientas
def NumericalGradient(X, Y, dist, h):
    n = len(X)

    # Kopijuojame X ir Y masyvus
    xx = np.array(X)
    yy = np.array(Y)

    # Sukuriame tuščius gradientų masyvus
    Gx = np.zeros(n)
    Gy = np.zeros(n)

    for i in range(n):
        # Atnaujiname xx su h pakeitimu ir skaičiuojame dalinį išvestinę
        xx[i] = xx[i] + h
        Gx[i] = (Target(xx, Y, dist) - Target(X, Y, dist)) / h
        xx[i] = X[i]

        # Atnaujiname yy su h pakeitimu ir skaičiuojame dalinį išvestinę
        yy[i] = yy[i] + h
        Gy[i] = (Target(X, yy, dist) - Target(X, Y, dist)) / h
        yy[i] = Y[i]

    # Normalizuojame gradientus
    aa = np.linalg.norm(np.hstack((Gx, Gy)))
    Gx0 = Gx / aa
    Gy0 = Gy / aa

    return Gx0, Gy0

n = 5  # Senų parduotuvių kiekis
X = np.zeros(n)
Y = np.zeros(n)

# Sugeneruojame pradines parduotuvių x ir y koordinates
for i in range(n): X[i] = random.uniform(-10., 10.)
for i in range(n): Y[i] = random.uniform(-10., 10.)

# Braižome pradines parduotuves raudonais taškais
plt.plot(X, Y, 'rp')
plt.grid()
plt.axis('equal')

m = 5
X = np.zeros(m)
Y = np.zeros(m)

# Sugeneruojame naujas parduotuvių reikšmes
for i in range(m): X[i] = random.uniform(-20., 20.)
```

```
for i in range(m): Y[i] = random.uniform(-20., 20.)

itmax = 1000;
step = 0.2;
h = 0.00001
dist = 3
fff = Target(X, Y, dist);

# Optimizacijos ciklas
for iii in range(itmax):
    Gx0, Gy0 = NumericalGradient(X, Y, dist, h)
    X = X - step * Gx0
    Y = Y - step * Gy0
    fff1 = Target(X, Y, dist)

    # Jeigu optimizacija bloga, atgalinė nuostolių mažinimo procedūra
    if fff1 > fff:
        X = X + step * Gx0
        Y = Y + step * Gy0
        step = step / 3
        print('step=', step)
    else:
        fff = fff1

    # Jeigu žingsnio dydis mažesnis nei slenkstis, nutraukiame optimizaciją
    if step < 1e-16:
        print('optimizavimas baigtas fff=', fff, "iteracijų skaičius=", iii)
        break

# Braižome galutines parduotuves mėlynais taškais
plt.plot(X, Y, 'bp')
plt.show()
```