****

**Kauno technologijos universitetas**

Informatikos fakultetas

Laboratorinio darbo ataskaita

**Antras laboratorinis darbas**

P170B115 Skaitiniai metodai ir algoritmai

|  |
| --- |
|  |
| **IFF-2/1 Kristupas Kondratavičius** |
| Studentas |
| **Andrius Kriščiūnas**  Dėstytojas |
|  |

**Kaunas, 2024**

**Turinys**

[1. Tiesinių lygčių sistemų sprendimas 4](#_Toc180673163)

[1.1. Duoti pradiniai duomenys 4](#_Toc180673164)

[1.2. Programos kodas 5](#_Toc180673165)

[1.3. Rezultatai 6](#_Toc180673166)

[2.1. Duoti pradiniai duomenys 7](#_Toc180673167)

[2.2. Programos kodas 8](#_Toc180673168)

[2.3. Rezultatai 9](#_Toc180673169)

[2. Netiesinių lygčių sistemų sprendimas 11](#_Toc180673170)

[3.1. Duoti pradiniai duomenys 12](#_Toc180673172)

[3.2. Programos kodas 13](#_Toc180673173)

[3.3. Rezultatai 14](#_Toc180673174)

Vizualizacijų sąrašas

[pav. 1 – QR skaidos algoritmas 5](#_Toc180673175)

[pav. 2 – Singuliarumo tikrinimo kodas 5](#_Toc180673176)

[pav. 3 Atgalinio etapo kodas 5](#_Toc180673177)

[pav. 4 – tinklelio kodas 8](#_Toc180673178)

[pav. 5 – 3D erdvėje vaizduojamos funkcijos, bei paryškinta dalis kurioje jos yra lygios nuliui. Raudoni taškai yra iteracijos o juodas taškas – gaunamas atsakymas kai pradinis artinys – (3;3) 9](#_Toc180673179)

[pav. 6 – Gaunamas sprendinys ir jo tikslumas kai pradinis artinys – (3;3) 9](#_Toc180673180)

[pav. 7 – Grafikas vaizduojantis funkcijų galimų sprendinius, bei tinklelį taškų - liestinių, kurie nuspalvinti pagal priklausomybę su jais gaunamam sprendiniui 10](#_Toc180673181)

[pav. 8 – vaizduojami visi gaunami sprendiniai su jiems priskiriama spalva 10](#_Toc180673182)

[pav. 9 – Duoto neuroninio tinklo vizualizacija 12](#_Toc180673183)

[pav. 10 – Greičiausio nusileidimo algoritmas 13](#_Toc180673184)

[pav. 11 – Gaunamas grafikas vaizduojantis Mean Squared Error priklausomybę nuo epochų 14](#_Toc180673185)

[pav. 12 – Programos išspausdinami duomenys vaizduojantys skirtumą tarp pradinių ir galutinių ryšių svorių, pradinės ir galutinės numatomos kainos, bei pradiniai bei galutiniai MSE ir MAE 14](#_Toc180673186)

## Tiesinių lygčių sistemų sprendimas

Lentelėje 1 duotos tiesinės lygčių sistemos, 2 lentelėje nurodytas metodas ir lygčių sistemų numeriai (iš 1 lentelės). Reikia suprogramuoti nurodytą metodą ir išspręsti pateiktas lygčių sistemas. Programoje sprendžiant lygčių sistemas turi būti įvertinti atvejai:

* kai lygčių sistema turi vieną sprendinį;
* kai lygčių sistema sprendinių neturi;
* kai lygčių sistema turi be gali daug sprendinių.

Patikrinkite gautus sprendinius įrašydami juos į pradinę lygčių sistemą. Gautą sprendinį patikrinkite

naudodami išorinius išteklius (pvz., standartines Python funkcijas).

### Duoti pradiniai duomenys

Užduoties nr: 8

Taikomas metodas: QR

Lygčių sistemos: 1, 14, 20

1:

14:

20:

### Programos kodas

|  |
| --- |
| # tiesioginis etapas(QR skaida):  Q=np.identity(n)  for i in range (0,n-1):  print("\n---------------")  print("ciklo iteracija: " + str(i))  print("---------------")  #sukame cikla kiekvienai matricos eilutei  #z nustatytas taip, kad jame butu dabartines eilutes elementai  z=A[i:n,i]; SpM(z, "z"); print("")  #vektorius nuliu  zp=np.zeros(np.shape(z));  #nustatome pirmaji zp elementa kaip z ilgi  zp[0]=np.linalg.norm(z); SpM(zp, "zp"); print("")  #skaiciuojame omega kuri yra skirtumas tarp z ir zp  omega=z-zp;  #pabaigiame realizuoti formule  omega=omega/np.linalg.norm(omega); SpM(omega, "omega"); print("")  #sudarome householder matrica Qi (identity matrix dydzio (n-i) pakeista omegos)  Qi=np.identity(n-i)-2\*omega\*omega.transpose(); SpM(Qi, "Qi"); print("")  #atnaujinti A nuo dabartines eilutes tolyn dauginant su Qi  A[i:n,i:n]=Qi.dot(A[i:n,i:n]); SpM(A, "A"); print("")  #naujiname Q taikydami Qi nuo dabartinio stulpelio tolyn  Q[:,i:n]=Q[:,i:n].dot(Qi); SpM(Q, "Q"); print("")  # po ciklo mes turime virsutine trikampe matrica A(R) ir matrica Q kuri sukaupe ortogonalias transformacijas  print("\nmatrica Q ir trikampe matrica A(R) sekmingai sumontuota") |

pav. 1 – QR skaidos algoritmas

|  |
| --- |
| if np.any(np.abs(np.diag(A)) < 1e-10): # Tikrinimas ar istrizai einantys matricos R elementai yra netoli nulio  rank\_A = np.linalg.matrix\_rank(A)  rank\_Ab = np.linalg.matrix\_rank(np.hstack((Ap, b)))  if rank\_A < rank\_Ab:  print("sistema neturi sprendiniu.\n")  elif rank\_A < n:  print("sistema turi daug sprendiniu.\n")  return  SpM(A) |

pav. 2 – Singuliarumo tikrinimo kodas

|  |
| --- |
| # 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];  SpM(x, "x") |

pav. 3 Atgalinio etapo kodas

### Rezultatai

|  |
| --- |
| -----------------------  sprendziama matrica: A1  ----------------------- sprendinys: =  [[ 5.55173636]  [ 1.97200567]  [-0.78065202]  [ 2.44046775]]  ------------ sprendinio patikrinimas ----------------  liekana =  [[-7.10542736e-15]  [-5.32907052e-15]  [ 0.00000000e+00]  [-7.10542736e-15]]  bendra santykine paklaida: = 1.770410401522562e-15  -----------------------  sprendziama matrica: A2  -----------------------  sistema turi daug sprendiniu.  -----------------------  sprendziama matrica: A3  -----------------------  sistema neturi sprendiniu. |

1. **Netiesinių lygčių sistemų sprendimas**

Duota netiesinių lygčių sistema (3 lentelė):

1. Skirtinguose grafikuose pavaizduokite paviršius ir .
2. Užduotyje pateiktą netiesinių lygčių sistemą išspręskite grafiniu būdu.
3. Nagrinėjamoje srityje sudarykite stačiakampį tinklelį ( 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).

### Duoti pradiniai duomenys

Užduoties variantas: 8

Duota lygčių sistema:

Sprendimo metodas:

Broideno

### Programos kodas

|  |
| --- |
| # Define the colors and initialize plot for marking solutions  colors = ['red', 'blue', 'green', 'purple', 'orange', 'cyan', 'magenta', 'yellow', 'black']  solutions = []  color\_index = 0  # Create a grid ranging from -10 to 10 with a step of 1  xx = np.arange(-10, 11, 1)  yy = np.arange(-10, 11, 1)  X, Y = np.meshgrid(xx, yy)  # Iterate over the grid points  for i in range(len(xx)):  for j in range(len(yy)):  xs = xx[i]  ys = yy[j]  print (str(xs) + " ; " + str(ys))  x = np.matrix([xs, ys], dtype=float).transpose()    # Reset Jacobian approximation  for k in range(n):  x1 = np.matrix(x, dtype=float)  x1[k] += dx  A[:, k] = (LF(x1) - LF(x)) / dx  ff = LF(x)    for k in range(maxiter):  deltax = -np.linalg.solve(A, ff)  x1 = np.matrix(x + deltax, dtype=float)  ff1 = LF(x1)  A += (ff1 - ff - A \* deltax) \* deltax.transpose() / (deltax.transpose() \* deltax)  tiksl = np.linalg.norm(deltax) + np.linalg.norm(ff1)  ff = ff1  x = x1  if tiksl < eps:  break  #ax2.plot3D([x[0,0],x1[0,0]],[x[1,0],x1[1,0]],[0,0],"ro-")  plt.draw();    # Check if the solution is close to an existing one  solution\_found = False  for idx, sol in enumerate(solutions):  if np.allclose(sol, x, atol=1e-6):  solution\_found = True  color\_idx = idx  break    if not solution\_found:  print("================================naujas===================================")  color\_index = len(solutions) % len(colors)  solutions.append(x)  print(color\_index)  color\_idx = color\_index    ax2.plot(xs, ys, 'o', color=colors[color\_idx])  print(x1.transpose(),"Sprendinys")  print(ff1,"funkcijos reiksme")  print(tiksl,"Galutinis tikslumas") |

pav. 4 – tinklelio kodas

### Rezultatai

A graph with lines and dots

Description automatically generated

pav. 5 – 3D erdvėje vaizduojamos funkcijos, bei paryškinta dalis kurioje jos yra lygios nuliui. Raudoni taškai yra iteracijos o juodas taškas – gaunamas atsakymas kai pradinis artinys – (3;3)

|  |
| --- |
| [[1.30474092 5.12609667]] Sprendinys  [[-7.53992424e-11]  [-3.59751340e-10]] funkcijos reiksme  4.396925633960575e-08 Galutinis tikslumas |

pav. 6 – Gaunamas sprendinys ir jo tikslumas kai pradinis artinys – (3;3)

A graph of a graph of a diagram

Description automatically generated with medium confidence

pav. 7 – Grafikas vaizduojantis funkcijų galimų sprendinius, bei tinklelį taškų - liestinių, kurie nuspalvinti pagal priklausomybę su jais gaunamam sprendiniui

|  |
| --- |
| jei su is grido paimta liestine gautas sprendinys: [[1.30474092]]; [[5.12609667]]  jo spalva yra: red  jei su is grido paimta liestine gautas sprendinys: [[-0.78353562]]; [[-6.3151418]]  jo spalva yra: blue  jei su is grido paimta liestine gautas sprendinys: [[6.01207886]]; [[-1.07261906]]  jo spalva yra: green  jei su is grido paimta liestine gautas sprendinys: [[5.59552862]]; [[1.26323116]]  jo spalva yra: purple |

pav. 8 – vaizduojami visi gaunami sprendiniai su jiems priskiriama spalva

## Netiesinių lygčių sistemų sprendimas

Pagal 7 lentelėje nurodytą schemą realizuokite dirbtinio neuroninio tinklo architektūrą nekilnojamojo turto (butų) kainos prognozei pagal istorinius duomenis atlikti (duomenys pateikti TSV formatu faile „data\_for\_task3.tsv“), kai įvesties parametrai yra: buto plotas (LotArea), kokybės įvertis (OveralQUal) ir pastatymo metai (YearBuilt). Taikydami 6 lentelėje nurodytą optimizavimo metodą raskite tokias 𝑊 vektoriaus reikšmes, kai atliekant kainos prognozę su nurodyta tinklo struktūra vidutinė kvadratinė paklaida (mean-square-error) faile „data\_for\_task4.tsv“ pateiktiems duomenims būtų kaip galima mažesnė.

a) pateikite grafikus kaip keičiasi vidutinė kvadratinė paklaida ir vidutinė absoliuti paklaida optimizavimo metu (atskirai lentelėje nurodykite minimas paklaidų reikšmes prieš ir po optimizavimo)

b) lentelėje pateikite svorinius koeficientus prieš ir po optimizavimo

c) lentelėje pateikite pirmų 10 objektų (iš failo „data\_for\_task4.tsv“) tinko įvesties duomenis, žinomą kainą ir kainos prognozę su užduotyje nurodytais ir optimizuotais tinklo svoriniais koeficientais

Paaiškinimas: pateiktoje schemoje 𝑁1, 𝑁2, … yra neuronai, kurie pirmame sluoksnyje (kairėje pusėje) yra lygūs įvesties duomenims, o antrame sluoksnyje yra lygūs pirmo sluoksnio neuronų sandaugų iš atitinkamų svorių (𝜔 ∋ 𝑊 pažymėtais ant rodyklės, pradinės 𝑊 reikšmės nurodytos po schema) sumai. Kainos prognozės rezultatas – antro sluoksnio neuronų suma padauginta iš nurodytų koeficientų (trečio sluoksnio neuronas). Vidutinė kvadratinė paklaida (MSE, mean square error) ir vidutinė absoliuti paklaida (MAE, mean absolute error) apskaičiuojamos pagal formules:

kur 𝑛 – duomenų kiekis, – tikrosios reikšmės (nurodytos kainos), – prognozuotos reikšmės (prognozuojamos kainos).



### Duoti pradiniai duomenys

3 Greičiausio nusileidimo

A diagram of a network

Description automatically generated

pav. 9 – Duoto neuroninio tinklo vizualizacija

### Programos kodas

|  |
| --- |
| # Compute Gradients  def compute\_gradients(X, y, W):  y\_pred = forward\_pass(X, W)  error = y\_pred - y  gradients = np.zeros\_like(W)  for i in range(len(W)):  if i < 5:  gradients[i] = 2 \* np.mean(error \* X[:, 0])  elif i < 10:  gradients[i] = 2 \* np.mean(error \* X[:, 1])  else:  gradients[i] = 2 \* np.mean(error \* X[:, 2])    return gradients  # Steepest Descent Optimization  def steepest\_descent(X, y, W, learning\_rate=0.0001, epochs=2000, lambda\_reg=0.01):  mse\_history = []  for epoch in range(epochs):  y\_pred = forward\_pass(X, W)  mse = mean\_squared\_error(y, y\_pred)  mse\_history.append(mse)    gradients = compute\_gradients(X, y, W)  # Calculate step size for steepest descent  XtX = X.T @ X  gradient\_XtX = np.dot(gradients[:3], XtX @ gradients[:3])  step\_size = np.dot(gradients, gradients) / (gradient\_XtX + lambda\_reg \* np.dot(gradients, gradients))    W -= step\_size \* gradients    # Clipping weights to prevent explosion  W = np.clip(W, -1, 1)    return W, mse\_history |

pav. 10 – Greičiausio nusileidimo algoritmas

### Rezultatai

A graph with a blue line

Description automatically generated

pav. 11 – Gaunamas grafikas vaizduojantis Mean Squared Error priklausomybę nuo epochų

A screenshot of a computer screen

Description automatically generated

pav. 12 – Programos išspausdinami duomenys vaizduojantys skirtumą tarp pradinių ir galutinių ryšių svorių, pradinės ir galutinės numatomos kainos, bei pradiniai bei galutiniai MSE ir MAE