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

Vizualizacijų sąrašas

1. 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).

1.1. Duoti pradiniai duomenys

Užduoties nr: 8

Taikomas metodas: QR

Lygčių sistemos: 1, 14, 20

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

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

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

1.2. 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

1.3. 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.
```

Duota netiesinių lygčių sistema (3 lentelė):

$$\begin{cases} Z_1(x_1, x_2) = 0 \\ Z_2(x_1, x_2) = 0 \end{cases}$$

a. Skirtinguose grafikuose pavaizduokite paviršius $Z_1(x_1, x_2)$ ir $Z_2(x_1, x_2)$.
b. Užduotyje pateiktą netiesinių lygčių sistemą išspręskite grafiniu būdu.
c. Nagrinėjamoje srityje sudarykite stačiakampį tinklelį (x_1, x_2 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.

d. Gautus sprendinius patikrinkite naudodami išorinius išteklius (pvz., standartines Python funkcijas).

2.1.Duoti pradiniai duomenys

Užduoties variantas: 8

Duota lygčių sistema:

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

Sprendimo metodas:

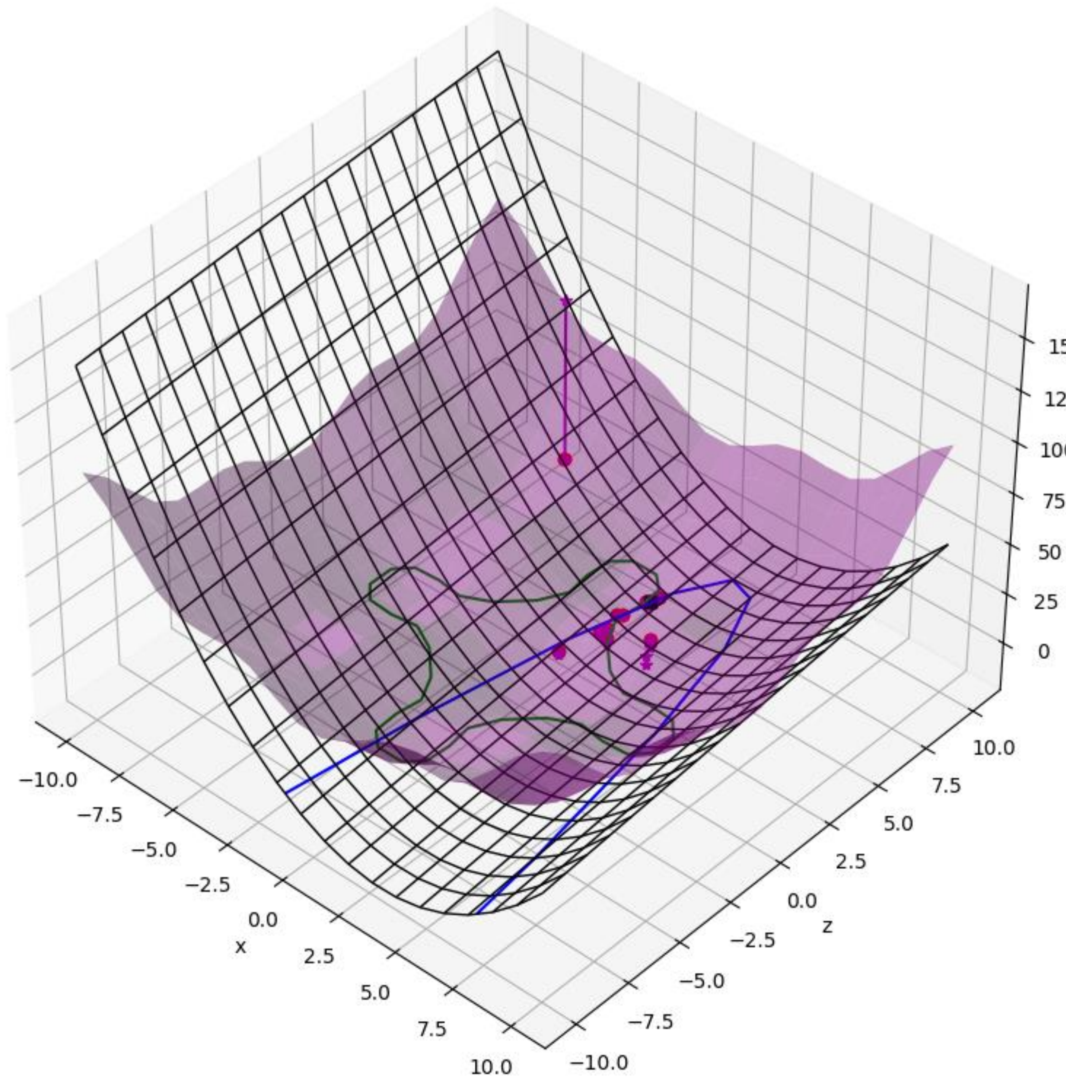
Broideno

2.2. 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 reikšme")
        print(tiksl,"Galutinis tikslumas")
```

pav. 4 – tinklelio kodas

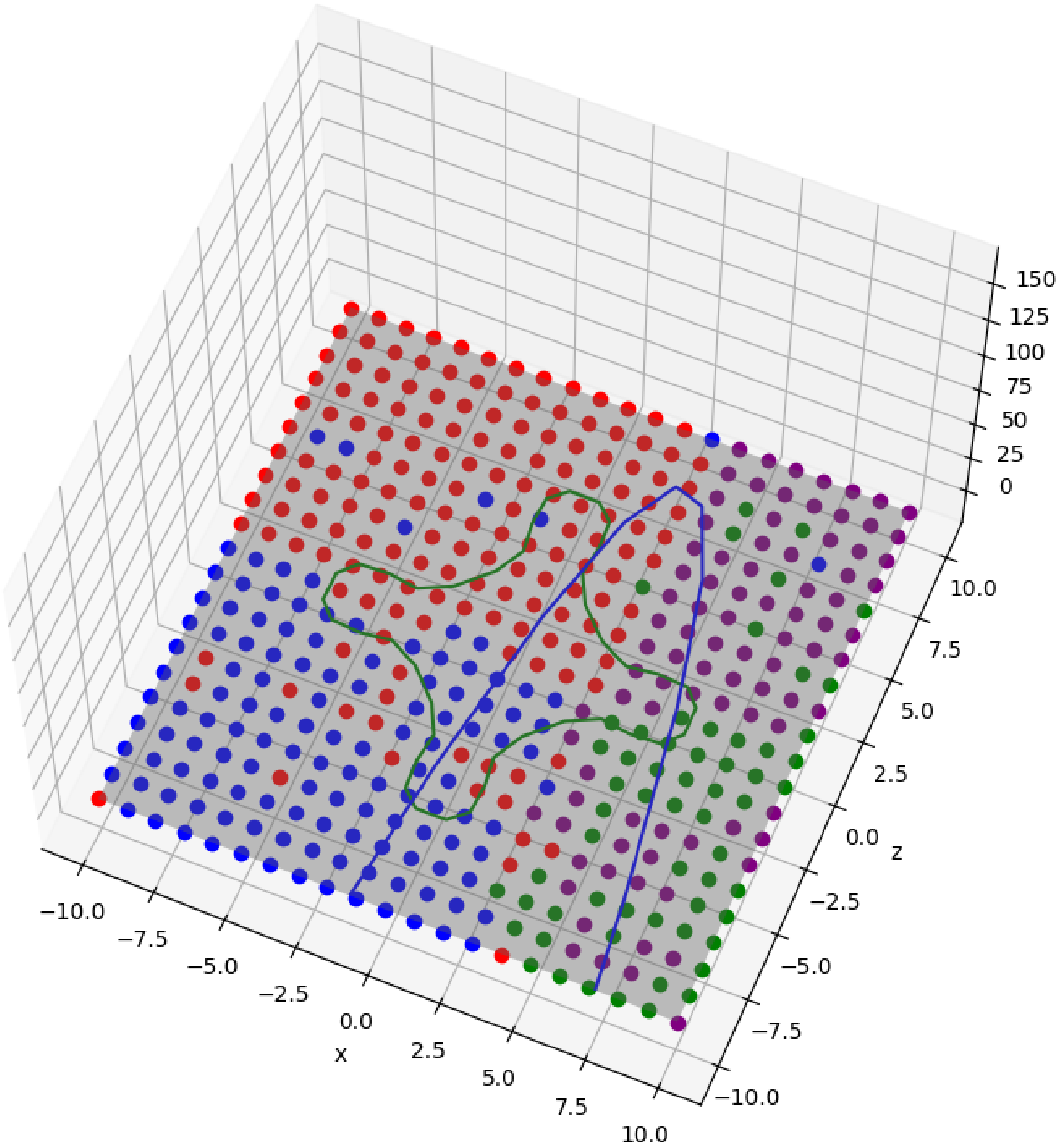
2.3. Rezultatai



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)



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

2. 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 ($\omega \ni W$ 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:

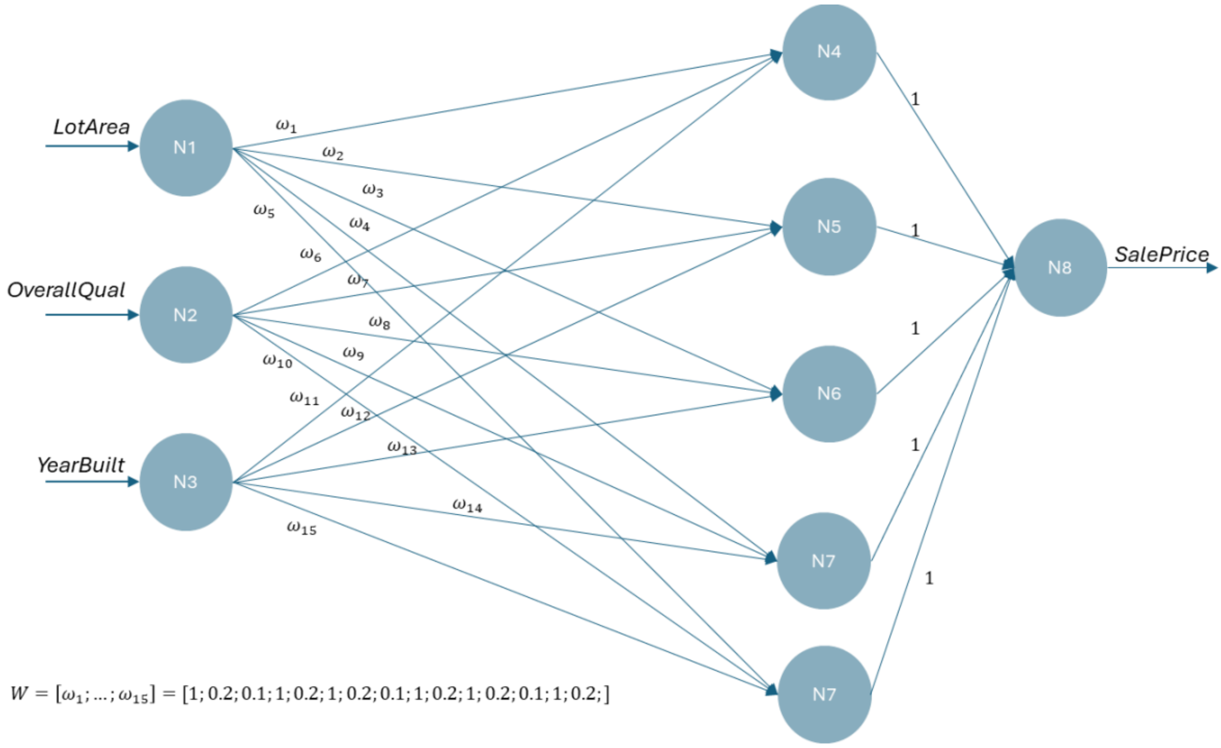
$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

kur n – duomenų kiekis, y_i – tikrosios reikšmės (nurodytos kainos), $\hat{y}_i$ – prognozuotos reikšmės (prognozuojamos kainos).

3.1.Duoti pradiniai duomenys

3 Greičiausio nusileidimo



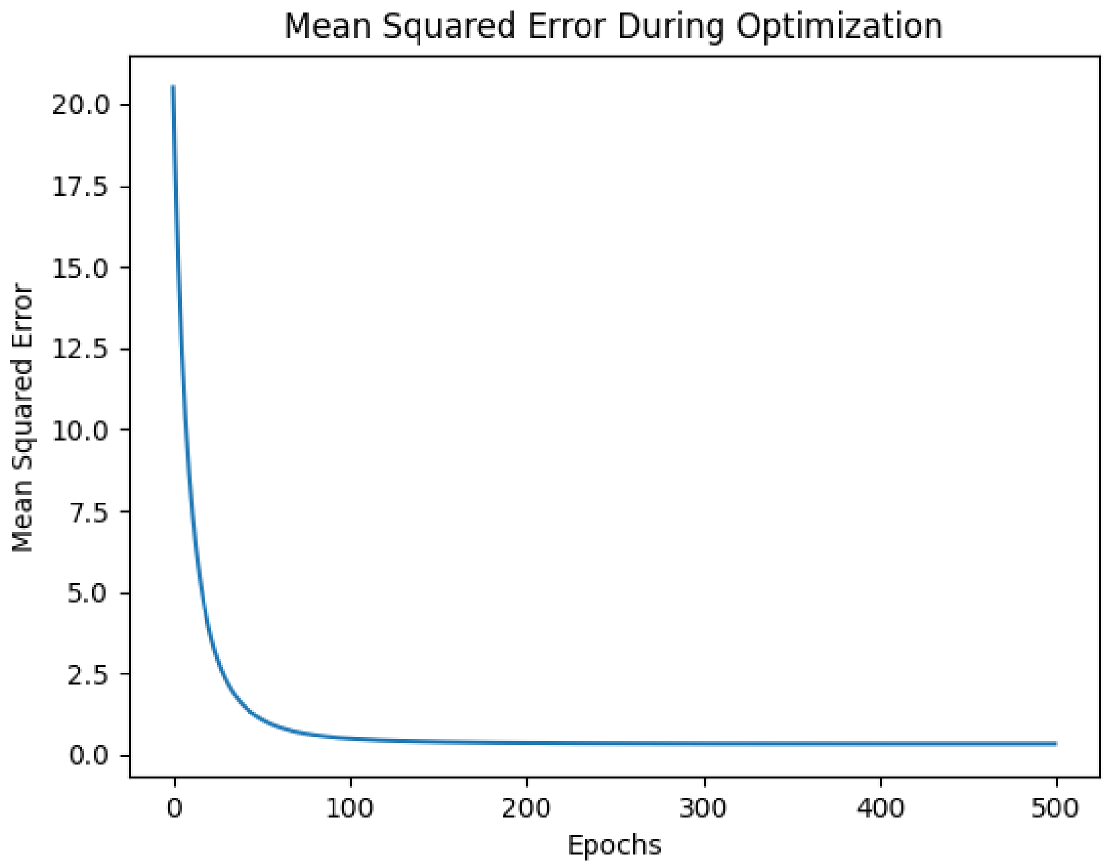
pav. 9 – Duoto neuroninio tinklo vizualizacija

3.2. 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

3.3. Rezultatai



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

```
Initial MSE: 20.51584059425585
Final MSE: 0.3322989062242607
   Weight  Initial  Optimized
0      w1      1.0   0.537538
1      w2      0.2  -0.262462
2      w3      0.1  -0.362462
3      w4      1.0   0.537538
4      w5      0.2  -0.262462
5      w6      1.0   0.640795
6      w7      0.2  -0.159205
7      w8      0.1  -0.259205
8      w9      1.0   0.640795
9     w10      0.2  -0.159205
10    w11      1.0   0.523465
11    w12      0.2  -0.276535
12    w13      0.1  -0.376535
13    w14      1.0   0.523465
14    w15      0.2  -0.276535
   LotArea  OverallQual  YearBuilt  Known SalePrice  Predicted SalePrice (Initial)  Predicted SalePrice (Optimized)
0     8450            7       2003           208500                  477801.629885                    224048.120332
1     9600            6       1976           181500                  179533.656714                    176995.864629
2    11250            7       2001           223500                  520364.295092                    227613.698767
3     9550            7       1915           140000                  -78975.250167                    198534.313329
4    14260            8       2000           250000                  717287.004693                    272239.498872
5    14115            5       1993           143000                  237554.419998                    148548.940730
6    10084            8       2004           307000                  660496.290401                    267235.586421
7    10382            7       1973           200000                  318972.367297                    217676.158239
8     6120            7       1931           129900                  -42013.321950                    198348.047249
9     7420            5       1939           118000                 -250751.676785                    121883.056026
            Error Type    Initial     Final
0   Mean Squared Error  20.515841  0.332299
1  Mean Absolute Error   3.499620  0.390918
```

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