Σειρά προβλημάτων 2

Κυρίτσης Βασίλειος 02999 και Σταμούλος Αλέξανδρος 02954

Νευρο-Ασαφής Υπολογιστική 2023-24
Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών
Πανεπιστήμιο Θεσσαλίας, Βόλος
{vakyritsis, astamoulos}@e-ce.uth.gr

1 Problem-01

$$F(w) = w_1^2 + w_2^2 + (0.5w_1 + w_2)^2 + (0.5w_1 + w_2)^4$$

Conjugate Gradient Method-FLETCHER-REEVES METHOD
Υπολογίζουμε το gradient την F σε κάθε σημείο

$$\nabla F(w) = \begin{pmatrix} \dfrac{\partial F}{\partial w_1} \\ \dfrac{\partial F}{\partial w_2} \end{pmatrix} = \begin{pmatrix} 0.25w_1^3 + 2w_2^3 + 1.5w_1^2 w_2 + 3w_1 w_2^2 + 2.5w_1 + w_2 \\ 0.5w_1^3 + 4w_2^3 + 3w_1^2 w_2 + 6w_1 w_2^2 + w_1 + 4w_2 \end{pmatrix}$$

και τον Hessian Matrix

$$H = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x_1^2} & \dfrac{\partial^2 f}{\partial x_1 \partial x_2} \\ \dfrac{\partial^2 f}{\partial x_2 \partial x_1} & \dfrac{\partial^2 f}{\partial x_2^2} \end{bmatrix} = \begin{bmatrix} 3\left(0.5w_1 + w_2\right)^2 + 0.5 & 6\left(0.5w_1 + w_2\right)^2 + 1 \\ 6\left(0.5w_1 + w_2\right)^2 + 1 & 12\left(0.5w_1 + w_2\right)^2 + 2 \end{bmatrix}$$

Iteration 1

$$w(0) = [3, 3]^T$$

Gradient, διεύθυνση ελαχιστοποίησης και Hessian στο w(0)

$$\nabla F(w(0)) = \begin{bmatrix} 0.25 \cdot 3^3 + 2 \cdot 3^3 + 1.5 \cdot 3^2 \cdot 3 + 3 \cdot 3 \cdot 3^2 + 2.5 \cdot 3 + 3 \\ 0.5 \cdot 3^3 + 4 \cdot 3^3 + 3 \cdot 3^2 \cdot 3 + 6 \cdot 3 \cdot 3^2 + \cdot 3 + 4 \cdot 3 \end{bmatrix} = \begin{bmatrix} 192.75 \\ 379.5 \end{bmatrix}$$

$$s_0 = -\nabla F(w(0)) = \begin{bmatrix} -192.75 \\ -379.5 \end{bmatrix}$$

$$H_0 = \begin{bmatrix} 3\left(0.5w \cdot 3 + 3\right)^2 + 0.5 & 6\left(0.5 \cdot 3 + 3\right)^2 + 1 \\ 6\left(0.5 \cdot 3 + 3\right)^2 + 1 & 12\left(0.5 \cdot 3 + 3\right)^2 + 2 \end{bmatrix} = \begin{bmatrix} 61.25 & 122.5 \\ 122.5 & 245 \end{bmatrix}$$

Έπειτα υπολογίζουμε το μέγεθος βήματος λ_0

$$\lambda_0 = \frac{\nabla^T F(w(0)) \cdot \nabla F(w(0))}{s_0^T \cdot H_0 \cdot s_0} = \frac{\begin{bmatrix}192.75 & 379.5\end{bmatrix}\begin{bmatrix}192.75\\379.5\end{bmatrix}}{\begin{bmatrix}-192.75 & -379.5\end{bmatrix}\begin{bmatrix}61.25 & 122.5\\122.5 & 245\end{bmatrix}\begin{bmatrix}-192.75\\-379.5\end{bmatrix}} = \frac{181172.812}{55481968.828}$$

$\lambda_0 = 0.003$

Συνεπώς το καινούργιο σημείο είναι:

$$w(1) = w(0) + \lambda_0 s_0 = \begin{bmatrix}3\\3\end{bmatrix} + 0.003\begin{bmatrix}-192.75\\-379.5\end{bmatrix} = \begin{bmatrix}3\\3\end{bmatrix} - \begin{bmatrix}0.578\\1.138\end{bmatrix} = \begin{bmatrix}2.422\\1.862\end{bmatrix}$$

Iteration 2

$$w(1) = [2.422 \quad 1.862]$$

Gradient, διεύθυνση ελαχιστοποίησης και Hessian στο w(1)

$$\nabla F(w(1)) = \begin{bmatrix}0.25 \cdot 2.422^3 + 2 \cdot 1.862^3 + 1.5 \cdot 2.422^2 \cdot 1.862 + 3 \cdot 2.422 \cdot 1.862^2 + 2.5 \cdot 2.422 + \cdot 1.862\\ 0.5 \cdot 2.422^3 + 4 \cdot 1.862^3 + 3 \cdot 2.422^2 \cdot 1.862 + 6 \cdot 2.422 \cdot 1.862^2 + \cdot 2.422 + 4 \cdot 1.862\end{bmatrix} = \begin{bmatrix}65.956\\125.947\end{bmatrix}$$

$$s_1 = -\nabla F(w(1)) + \omega_1 s_0$$

$$\omega_1 = \frac{\nabla^T F(w(1)) \cdot \nabla F(w(1))}{\nabla^T F(w(0)) \cdot \nabla F(w(0))} = \frac{\begin{bmatrix}65.956 & 125.947\end{bmatrix}\begin{bmatrix}65.956\\125.947\end{bmatrix}}{\begin{bmatrix}192.75 & 379.5\end{bmatrix}\begin{bmatrix}192.75\\379.5\end{bmatrix}} = \frac{(65.956)^2 + (125.947)^2}{(192.75)^2 + (379.5)^2}$$

$$\omega_1 = \frac{20212.84}{181172.8125} = 0.112$$

$$s_1 = \begin{bmatrix}-65.956\\-125.947\end{bmatrix} + 0.112\begin{bmatrix}-192.75\\-379.5\end{bmatrix} = \begin{bmatrix}-65.956\\-125.947\end{bmatrix} + \begin{bmatrix}-21.588\\-42.504\end{bmatrix} = \begin{bmatrix}-87.544\\-168.451\end{bmatrix}$$

$$H_1 = \begin{bmatrix}3\left(0.5 \cdot 2.422 + \cdot 1.862\right)^2 + 0.5 & 6\left(0.5 \cdot 2.422 + \cdot 1.862\right)^2 + 1\\ 6\left(0.5 \cdot 2.422 + \cdot 1.862\right)^2 + 1 & 12\left(0.5 \cdot 2.422 + \cdot 1.862\right)^2 + 2\end{bmatrix} = \begin{bmatrix}28.83 & 57.66\\57.66 & 115.32\end{bmatrix}$$

$$\lambda_1 = \frac{\nabla^T F(w(1)) \cdot \nabla F(w(1))}{s_1^T \cdot H_1 \cdot s_1} = \frac{\begin{bmatrix}65.956 & 125.947\end{bmatrix}\begin{bmatrix}65.956\\125.947\end{bmatrix}}{= \begin{bmatrix}-87.544 & -168.451\end{bmatrix}\begin{bmatrix}28.83 & 57.66\\57.66 & 115.32\end{bmatrix}\begin{bmatrix}-87.544\\-168.451\end{bmatrix}}$$

$$\lambda_1 = \frac{20212.84}{5193851.551} = 0.004$$

Συνεπώς το καινούργιο σημείο είναι:

$$w(2) = w(1)+\lambda_1 s_1 = \begin{bmatrix}2.422\\1.862\end{bmatrix}+0.004\begin{bmatrix}-87.544\\-168.451\end{bmatrix} = \begin{bmatrix}2.422\\1.862\end{bmatrix}+\begin{bmatrix}-0.350\\-0.674\end{bmatrix} = \begin{bmatrix}2.072\\1.188\end{bmatrix}$$

Iteration 3

$$w(2) = [2.072 \quad 1.188]$$

Gradient, διεύθυνση ελαχιστοποίησης και Hessian στο w(2)

$$\nabla F(w(2)) = \begin{bmatrix}0.25\cdot 2.072^3 + 2\cdot 1.188^3 + 1.5\cdot 2.072^2\cdot 1.188 + 3\cdot 2.072\cdot 1.188^2 + 2.5\cdot 2.072 + \cdot 1.188\\ 0.5\cdot 2.072^3 + 4\cdot 1.188^3 + 3\cdot 2.072^2\cdot 1.188 + 6\cdot 2.072\cdot 1.188^2 + \cdot 2.072 + 4\cdot 1.188\end{bmatrix} = \begin{bmatrix}28.368\\50.825\end{bmatrix}$$

$$s_2 = -\nabla F(w(2)) + \omega_2 s_1$$

$$\omega_2 = \frac{\nabla^T F(w(2))\cdot\nabla F(w(2))}{\nabla^T F(w(1))\cdot\nabla F(w(1))} = \frac{\begin{bmatrix}28.368 & 50.825\end{bmatrix}\begin{bmatrix}28.368\\50.825\end{bmatrix}}{\begin{bmatrix}65.956 & 125.947\end{bmatrix}\begin{bmatrix}65.956\\125.947\end{bmatrix}} = \frac{(28.368)^2+(50.825)^2}{(65.956)^2+(125.947)^2}$$

$$\omega_2 = \frac{3387.924}{20212.84} = 0.168$$

$$s_2 = \begin{bmatrix}-28.368\\-50.825\end{bmatrix} + 0.168\begin{bmatrix}-87.544\\-168.451\end{bmatrix} = \begin{bmatrix}-28.368\\-50.825\end{bmatrix} + \begin{bmatrix}-14.708\\-28.3\end{bmatrix} = \begin{bmatrix}-43.076\\-79.125\end{bmatrix}$$

$$H_2 = \begin{bmatrix}3\left(0.5\cdot 2.072 + \cdot 1.188\right)^2 + 0.5 & 6\left(0.5\cdot 2.072 + \cdot 1.188\right)^2 + 1\\ 6\left(0.5\cdot 2.072 + \cdot 1.188\right)^2 + 1 & 12\left(0.5\cdot 2.072 + \cdot 1.188\right)^2 + 2\end{bmatrix} = \begin{bmatrix}15.338 & 30.677\\30.677 & 61.354\end{bmatrix}$$

$$\lambda_2 = \frac{\nabla^T F(w(2))\cdot\nabla F(w(2))}{s_2^T\cdot H_2\cdot s_2} = \frac{\begin{bmatrix}28.368 & 50.825\end{bmatrix}\begin{bmatrix}28.368\\50.825\end{bmatrix}}{= \begin{bmatrix}-43.076 & -79.125\end{bmatrix}\begin{bmatrix}15.338 & 30.677\\30.677 & 61.354\end{bmatrix}\begin{bmatrix}-43.076\\-79.125\end{bmatrix}}$$

$$\lambda_2 = \frac{3387.924}{621701.582} = 0.005$$

Συνεπώς το καινούργιο σημείο είναι:

$$w(3) = w(2)+\lambda_2 s_2 = \begin{bmatrix}2.072\\1.188\end{bmatrix}+0.005\begin{bmatrix}-43.076\\-79.125\end{bmatrix} = \begin{bmatrix}2.072\\1.188\end{bmatrix}+\begin{bmatrix}-0.21538\\-0.395625\end{bmatrix} = \begin{pmatrix}1.857\\0.792\end{pmatrix}$$

Iterration 4

$$w(3) = [1.857 \quad 0.792]$$

Gradient, διεύθυνση ελαχιστοποίησης και Hessian στο w(3)

$$\nabla F(w(3)) = \begin{bmatrix} 0.25 \cdot 1.857^3 + 2 \cdot 0.792^3 + 1.5 \cdot 1.857^2 \cdot 0.792 + 3 \cdot 1.857 \cdot 0.792^2 + 2.5 \cdot 1.857 + \cdot 0.792 \\ 0.5 \cdot 1.857^3 + 4 \cdot 0.792^3 + 3 \cdot 1.857^2 \cdot 0.792 + 6 \cdot 1.857 \cdot 0.792^2 + \cdot 1.857 + 4 \cdot 0.792 \end{bmatrix} = \begin{bmatrix} 15.628 \\ 25.396 \end{bmatrix}$$

$$s_3 = -\nabla F(w(3)) + \omega_3 s_2$$

$$\omega_3 = \frac{\nabla^T F(w(3)) \cdot \nabla F(w(3))}{\nabla^T F(w(2)) \cdot \nabla F(w(2))} = \frac{\begin{bmatrix} 15.628 & 25.396 \end{bmatrix} \begin{bmatrix} 15.628 \\ 25.396 \end{bmatrix}}{\begin{bmatrix} 28.368 & 50.825 \end{bmatrix} \begin{bmatrix} 28.368 \\ 50.825 \end{bmatrix}} = \frac{(15.628)^2 + (25.396)^2}{(28.368)^2 + (50.825)^2}$$

$$\omega_3 = \frac{889.191}{3387.924} = 0.262$$

$$s_3 = \begin{bmatrix} -15.628 \\ -25.396 \end{bmatrix} + 0.262 \begin{bmatrix} -43.076 \\ -79.125 \end{bmatrix} = \begin{bmatrix} -15.628 \\ -25.396 \end{bmatrix} + \begin{bmatrix} -11.286 \\ -20.730 \end{bmatrix} = \begin{bmatrix} -26.914 \\ -46.126 \end{bmatrix}$$

$$H_3 = \begin{bmatrix} 3\left(0.5 \cdot 1.857 + \cdot 0.792\right)^2 + 0.5 & 6\left(0.5 \cdot 1.857 + \cdot 0.792\right)^2 + 1 \\ 6\left(0.5 \cdot 1.857 + \cdot 0.792\right)^2 + 1 & 12\left(0.5 \cdot 1.99 + \cdot 0.792\right)^2 + 2 \end{bmatrix} = \begin{bmatrix} 9.38 & 18.76 \\ 18.76 & 40.32 \end{bmatrix}$$

$$\lambda_3 = \frac{\nabla^T F(w(3)) \cdot \nabla F(w(3))}{s_3^T \cdot H_3 \cdot s_3} = \frac{\begin{bmatrix} 15.628 & 25.396 \end{bmatrix} \begin{bmatrix} 15.628 \\ 25.396 \end{bmatrix}}{\begin{bmatrix} -26.914 & -46.126 \end{bmatrix} \begin{bmatrix} 9.38 & 18.76 \\ 18.76 & 40.32 \end{bmatrix} \begin{bmatrix} -26.914 \\ -46.126 \end{bmatrix}}$$

$$\lambda_3 = \frac{889.191}{139158.325} = 0.006$$

Συνεπώς το καινούργιο σημείο είναι:

$$w(4) = w(3) + \lambda_3 s_3 = \begin{bmatrix} 1.857 \\ 0.792 \end{bmatrix} + 0.006 \begin{bmatrix} -26.914 \\ -46.126 \end{bmatrix} = \begin{bmatrix} 1.857 \\ 0.792 \end{bmatrix} + \begin{bmatrix} -0.161 \\ -0.277 \end{bmatrix} = \begin{bmatrix} 1.696 \\ 0.515 \end{bmatrix}$$

Iteration 5

$$w(4) = [1.696 \quad 0.515]$$

Gradient, διεύθυνση ελαχιστοποίησης και Hessian στο w(4)

$$\nabla F(w(4)) = \begin{bmatrix} 0.25 \cdot 1.696^3 + 2 \cdot 0.515^3 + 1.5 \cdot 1.696^2 \cdot 0.515 + 3 \cdot 1.696 \cdot 0.515^2 + 2.5 \cdot 1.696 + \cdot 0.515 \\ 0.5 \cdot 1.696^3 + 4 \cdot 0.515^3 + 3 \cdot 1.696^2 \cdot 0.515 + 6 \cdot 1.696 \cdot 0.515^2 + \cdot 1.696 + 4 \cdot 0.515 \end{bmatrix}$$
$$= \begin{bmatrix} 9.82 \\ 13.884 \end{bmatrix}$$

$$s_4 = -\nabla F(w(4)) + \omega_4 s_3$$

$$\omega_4 = \frac{\nabla^T F(w(4)) \cdot \nabla F(w(4))}{\nabla^T F(w(3)) \cdot \nabla F(w(3))} = \frac{\begin{bmatrix} 9.82 & 13.884 \end{bmatrix} \begin{bmatrix} 9.82 \\ 13.884 \end{bmatrix}}{\begin{bmatrix} 15.628 & 25.396 \end{bmatrix} \begin{bmatrix} 15.628 \\ 25.396 \end{bmatrix}} = \frac{(9.82)^2 + (13.884)^2}{(15.628)^2 + (25.396)^2}$$

$$\omega_4 = \frac{289.198}{889.191} = 0.325$$

$$s_4 = \begin{bmatrix} -9.82 \\ -13.884 \end{bmatrix} + 0.325 \begin{bmatrix} -26.914 \\ -46.126 \end{bmatrix} = \begin{bmatrix} -9.82 \\ -13.884 \end{bmatrix} + \begin{bmatrix} -8.74705 \\ -14.99095 \end{bmatrix} = \begin{bmatrix} -18.56705 \\ -28.87495 \end{bmatrix}$$

$$H_4 = \begin{bmatrix} 3\left(0.5 \cdot 1.696 + \cdot 0.515\right)^2 + 0.5 & 6\left(0.5 \cdot 1.696 + \cdot 0.515\right)^2 + 1 \\ 6\left(0.5 \cdot 1.696 + \cdot 0.515\right)^2 + 1 & 12\left(0.5 \cdot 1.99 + \cdot 0.515\right)^2 + 2 \end{bmatrix} = \begin{bmatrix} 6.073 & 12.147 \\ 12.147 & 29.361 \end{bmatrix}$$

$$\lambda_4 = \frac{\nabla^T F(w(4)) \cdot \nabla F(w(4))}{s_4^T \cdot H_4 \cdot s_4} = \frac{\begin{bmatrix} 9.82 & 13.884 \end{bmatrix} \begin{bmatrix} 9.82 \\ 13.884 \end{bmatrix}}{= \begin{bmatrix} -18.56705 & -28.87495 \end{bmatrix} \begin{bmatrix} 6.073 & 12.147 \\ 12.147 & 29.361 \end{bmatrix} \begin{bmatrix} -18.56705 \\ -28.87495 \end{bmatrix}}$$

$$\lambda_4 = \frac{289.197}{39598.249} = 0.007$$

Συνεπώς το καινούργιο σημείο είναι:

$$w(5) = w(4) + \lambda_4 s_4 = \begin{bmatrix} 1.696 \\ 0.515 \end{bmatrix} + 0.007 \begin{bmatrix} -18.56705 \\ -28.87495 \end{bmatrix} = \begin{bmatrix} 1.696 \\ 0.515 \end{bmatrix} + \begin{bmatrix} -0.123 \\ -0.202 \end{bmatrix} = \begin{bmatrix} 1.573 \\ 0.313 \end{bmatrix}$$

Gradient Descent method

$$\nabla F(w) = \begin{pmatrix} \dfrac{\partial F}{\partial w_1} \\ \dfrac{\partial F}{\partial w_2} \end{pmatrix} = \begin{pmatrix} 0.25w_1^3 + 2w_2^3 + 1.5w_1^2 w_2 + 3w_1 w_2^2 + 2.5w_1 + w_2 \\ 0.5w_1^3 + 4w_2^3 + 3w_1^2 w_2 + 6w_1 w_2^2 + w_1 + 4w_2 \end{pmatrix}$$

$$||\nabla f(w)|| = \sqrt{\nabla^T f(w_1, w_2) \nabla f(w_1, w_2)}$$

$$= \sqrt{(0.25w_1^3 + 2w_2^3 + 1.5w_1^2w_2 + 3w_1w_2^2 + 2.5w_1 + w_2)^2 + (0.5w_1^3 + 4w_2^3 + 3w_1^2w_2 + 6w_1w_2^2 + w_1 + 4w_2)^2}$$

Iteration 1

$$w(0) = [3 \quad 3]^T$$

$$\nabla F(w(0) = \begin{pmatrix} 0.25 \cdot 3^3 + 2 \cdot 3^3 + 1.5 \cdot 3^2 \cdot 3 + 3 \cdot 3 \cdot 3^2 + 2.5 \cdot 3 + 3 \\ 0.5 \cdot 3^3 + 4 \cdot 3^3 + 3 \cdot 3^2 \cdot 3 + 6 \cdot 3 \cdot 3^2 + \cdot 3 + 4 \cdot 3 \end{pmatrix} = \begin{pmatrix} 192.75 \\ 379.5 \end{pmatrix}$$

$$||\nabla F(w(0))|| = \sqrt{192.75^2 + 379.5^2} = \sqrt{37152.5625 + 144020.25} = \sqrt{181.172,8125} = 425.644$$

Άρα η κατεύθυνση καθόδου:

$$\frac{-\nabla F(w(0))}{||\nabla F(w(0)||} = -\frac{1}{425.644} \begin{pmatrix} 192.75 \\ 379.5 \end{pmatrix} = \begin{pmatrix} -0.453 \\ -0.891 \end{pmatrix}$$

Συνεπώς

$$w(1) = \begin{pmatrix} 3 \\ 3 \end{pmatrix} - \lambda_0 \begin{pmatrix} -0.453 \\ -0.891 \end{pmatrix} = \begin{pmatrix} 3 + 0.453\lambda_0 \\ 3 + 0.891\lambda_0 \end{pmatrix}$$

Αντικαθιστούμε στην F το w_1:

$$\begin{aligned} F(w(1)) &= (3 + 0.453\lambda_0)^2 + (3 + 0.891\lambda_0)^2 \\ &\quad + (1.5 + 0.2265\lambda_0 + 3 + 0.891\lambda_0)^2 + (1.5 + 0.2265\lambda_0 + 3 + 0.891\lambda_0)^4 \\ &= (3 + 0.453\lambda_0)^2 + (3 + 0.891\lambda_0)^2 + (1.118\lambda_0 + 4.5)^2 + (1.118\lambda_0 + 4.5)^4 \end{aligned}$$

Για να βρούμε το βρούμε το optimal λ_0, θα βρούμε την παράγωγο την προηγούμενης συνάρτησης ως προς λ_0 και θα την θέσουμε ίση με το 0.

$$\frac{dF(w(1))}{d\lambda_0} = 6.249\lambda_0^3 + 74.46\lambda_0^2 + 308.23\lambda_0 + 425.637$$

$$\frac{dF(w(1))}{d\lambda_0} = 0$$
$$6.249\lambda_0^3 + 74.46\lambda_0^2 + 308.23\lambda_0 + 425.637 = 0$$
$$\lambda_0 = -4.03$$

Αντικαθιστούμε το λ0 στο w1 και για να βρούμε το καινούργιο σημείο.

$$w(1) = \begin{pmatrix} 3 + 0.453(-4.03) \\ 3 + 0.891(-4.03) \end{pmatrix} = \begin{pmatrix} 3 - 1.83 \\ 3 - 3.59 \end{pmatrix} = \begin{pmatrix} 1.17 \\ -0.59 \end{pmatrix}$$

Iteration 2

$$w(1) = [1.17 \quad -0.59]^T$$

$$\nabla F(w(1)) = \begin{pmatrix} 0.25 \cdot 1.17^3 + 2 \cdot (-0.59)^3 + 1.5 \cdot 1.17^2 \cdot (-0.59) + 3 \cdot 1.17 \cdot (-0.59)^2 + 2.5 \cdot 1.17 + \cdot(-0.59) \\ 0.5 \cdot 1.17^3 + 4 \cdot (-0.59)^3 + 3 \cdot 1.17^2 \cdot (-0.59) + 6 \cdot 1.17 \cdot (-0.59)^2 + \cdot 1.17 + 4 \cdot (-0.59) \end{pmatrix}$$
$$= \begin{pmatrix} 2.334 \\ -1.19 \end{pmatrix}$$

$$||\nabla F(w(1))|| = \sqrt{2.334^2 + (-1.19)^2} = \sqrt{5.447 + 1.416} = \sqrt{6.863} = 2.62$$

Άρα η κατεύθυνση καθόδου:

$$\frac{-\nabla F(w(1))}{||\nabla F(w(1)||} = -\frac{1}{2.62} \begin{pmatrix} 2.334 \\ -1.19 \end{pmatrix} = \begin{pmatrix} -0.89 \\ 0.454 \end{pmatrix}$$

Συνεπώς

$$w(2) = \begin{pmatrix} 1.17 \\ -0.59 \end{pmatrix} - \lambda_1 \begin{pmatrix} -0.89 \\ 0.454 \end{pmatrix} = \begin{pmatrix} 1.17 + 0.89\lambda_1 \\ -0.59 - 0.454\lambda_1 \end{pmatrix}$$

Αντικαθιστούμε στην F το w_2:

$$\begin{aligned} F(w(2)) &= (1.17 + 0.89\lambda_1)^2 + (-0.59 - 0.454\lambda_1)^2 \\ &\quad + (0.585 + 0.445\lambda_1 - 0.59 - 0.454\lambda_1)^2 + (0.585 + 0.445\lambda_1 - 0.59 - 0.454\lambda_1)^4 \\ &= (1.17 + 0.89\lambda_1)^2 + (-0.59 - 0.454\lambda_1)^2 + (-0.009\lambda_1 - 0.005)^2 + (-0.009\lambda_1 - 0.005)^4 \end{aligned}$$

Για να βρούμε το βρούμε το optimal λ_1, θα βρούμε την παράγωγο την προηγούμενης συνάρτησης ως προς λ_1 και θα την θέσουμε ίση με το 0.

$$\frac{dF(w(2))}{d\lambda_1} = 2.6244 \cdot 10^{-8}\lambda_1^3 + 4.374 \cdot 10^{-8}\lambda_1^2 + 1.997\lambda_1 + 2.618$$

$$\frac{dF(w(2))}{d\lambda_1} = 0$$
$$2.6244 \cdot 10^{-8}\lambda_1^3 + 4.374 \cdot 10^{-8}\lambda_1^2 + 1.997\lambda_1 + 2.618 = 0$$
$$\lambda_1 = -1.311$$

Αντικαθιστούμε το λ1 στο w2 και για να βρούμε το καινούργιο σημείο.

$$w(2) = \begin{pmatrix} 1.17 - 0.89 \cdot (-1.311) \\ -0.59 + 0.454 \cdot (-1.311) \end{pmatrix} = \begin{pmatrix} 1.17 + 1.167 \\ -0.59 - 0.595 \end{pmatrix} = \begin{pmatrix} 2.237 \\ -1.185 \end{pmatrix}$$

Iteration 3

$$w(2) = [2.237 \quad -1.185]^T$$

$$\nabla F(w(2) = \begin{pmatrix} 0.25 \cdot 2.237^3 + 2 \cdot (-1.185)^3 + 1.5 \cdot 2.237^2 \cdot (-1.185) + 3 \cdot 2.237 \cdot (-1.185)^2 + 2.5 \cdot 2.237 + \cdot(-1.185) \\ 0.5 \cdot 2.237^3 + 4 \cdot (-1.185)^3 + 3 \cdot 2.237^2 \cdot (-1.185) + 6 \cdot 2.237 \cdot (-1.185)^2 + \cdot 2.237 + 4 \cdot (-1.185) \end{pmatrix} = \begin{pmatrix} 4.406 \\ -2.504 \end{pmatrix}$$

$$||\nabla F(w(2))|| = \sqrt{4.406^2 + (-2.504)^2} = \sqrt{19.412 + 6.27} = \sqrt{25.682} = 5.068$$

Άρα η κατεύθυνση καθόδου:

$$\frac{-\nabla F(w(2))}{||\nabla F(w(2)||} = -\frac{1}{5.068} \begin{pmatrix} 4.406 \\ -2.504 \end{pmatrix} = \begin{pmatrix} -0.869 \\ 0.494 \end{pmatrix}$$

Συνεπώς

$$w(3) = \begin{pmatrix} 2.237 \\ -1.185 \end{pmatrix} - \lambda_2 \begin{pmatrix} -0.869 \\ 0.494 \end{pmatrix} = \begin{pmatrix} 2.237 + 0.869\lambda_2 \\ -1.185 - 0.494\lambda_2 \end{pmatrix}$$

Αντικαθιστούμε στην F το w_3:

$$\begin{aligned} F(w(3)) &= (2.237 + 0.869\lambda_2)^2 + (-1.185 - 0.494\lambda_2)^2 \\ &\quad + (1.118 + 0.434\lambda_2 - 1.185 - 0.494\lambda_2)^2 + (1.118 + 0.434\lambda_2 - 1.185 - 0.494\lambda_2)^4 \\ &= (2.237 + 0.869\lambda_2)^2 + (-1.185 - 0.494\lambda_2)^2 + (-0.06\lambda_2 - 0.067)^2 + (-0.06\lambda_2 - 0.067)^4 \end{aligned}$$

Για να βρούμε το βρούμε το optimal λ_2, θα βρούμε την παράγωγο την προηγούμενης συνάρτησης ως προς λ_2 και θα την θέσουμε ίση με το 0.

$$\frac{dF(w(3))}{d\lambda_2} = 0.00005184\lambda_2^3 + 0.000173664\lambda_2^2 + 2.00578\lambda_2 + 5.06679$$

$$\begin{aligned} \frac{dF(w(3))}{d\lambda_2} &= 0 \\ 0.00005184\lambda_2^3 + 0.000173664\lambda_2^2 + 2.00578\lambda_2 + 5.06679 &= 0 \\ \lambda_2 &= -2.526 \end{aligned}$$

Αντικαθιστούμε το λ2 στο w3 και για να βρούμε το καινούργιο σημείο.

$$w(3) = \begin{pmatrix} 2.237 + 0.869(-2.526) \\ -1.185 - 0.494(-2.526) \end{pmatrix} = \begin{pmatrix} 2.237 - 2.195 \\ -1.185 + 1.247 \end{pmatrix} = \begin{pmatrix} 0.042 \\ 0.062 \end{pmatrix}$$

Iteration 4

$$w(3) = [0.042 \quad 0.062]^T$$

$$\nabla F(w(3)) = \begin{pmatrix} 0.25 \cdot 0.042^3 + 2 \cdot (0.062)^3 + 1.5 \cdot 0.042^2 \cdot (0.062) + 3 \cdot 0.042 \cdot (0.062)^2 + 2.5 \cdot 0.042 + \cdot(0.062) \\ 0.5 \cdot 0.042^3 + 4 \cdot (0.062)^3 + 3 \cdot 0.042^2 \cdot (0.062) + 6 \cdot 0.042 \cdot (0.062)^2 + \cdot 0.042 + 4 \cdot (0.062) \end{pmatrix}$$
$$= \begin{pmatrix} 0.168 \\ 0.292 \end{pmatrix}$$

$$||\nabla F(w(3))|| = \sqrt{0.168^2 + 0.292^2} = \sqrt{0.028 + 0.085} = \sqrt{0.108} = 0.329$$

Άρα η κατεύθυνση καθόδου:

$$\frac{-\nabla F(w(3))}{||\nabla F(w(3))||} = -\frac{1}{0.329} \begin{pmatrix} 0.168 \\ 0.292 \end{pmatrix} = \begin{pmatrix} -0.51 \\ -0.888 \end{pmatrix}$$

Συνεπώς

$$w(4) = \begin{pmatrix} 0.042 \\ 0.062 \end{pmatrix} - \lambda_3 \begin{pmatrix} -0.51 \\ -0.888 \end{pmatrix} = \begin{pmatrix} 0.042 + 0.51\lambda_3 \\ 0.062 + 0.888\lambda_3 \end{pmatrix}$$

Αντικαθιστούμε στην F το w_3:

$$\begin{aligned} F(w(4)) &= (0.042 + 0.51\lambda_3)^2 + (0.062 + 0.888\lambda_3)^2 \\ &\quad + (0.021 + 0.255\lambda_3 + 0.062 + 0.888\lambda_3)^2 + (0.021 + 0.255\lambda_3 + 0.062 + 0.888\lambda_3)^4 \\ &= (0.042 + 0.51\lambda_3)^2 + (0.062 + 0.888\lambda_3)^2 + (0.021 + 0.90381\lambda_3)^2 + (0.021 + 0.90381\lambda_3)^4 \end{aligned}$$

Για να βρούμε το βρούμε το optimal λ_3, θα βρούμε την παράγωγο την προηγούμενης συνάρτησης ως προς λ_3 και θα την θέσουμε ίση με το 0.

$$\frac{dF(w(4))}{d\lambda_3} = 2.66912\lambda_3^3 + 0.18605\lambda_3^2 + 3.73535\lambda_3 + 0.19094$$

$$\frac{dF(w(4))}{d\lambda_3} = 0$$
$$2.66912\lambda_3^3 + 0.18605\lambda_3^2 + 3.73535\lambda_3 + 0.19094 = 0$$
$$\lambda_3 = -0.03$$

Αντικαθιστούμε το λ3 στο w4 και για να βρούμε το καινούργιο σημείο.

$$w(4) = \begin{pmatrix} 0.042 + 0.51(-0.03) \\ 0.062 + 0.888(-0.03) \end{pmatrix} = \begin{pmatrix} 0.042 - 0.015 \\ 0.062 - 0.027 \end{pmatrix} = \begin{pmatrix} 0.027 \\ 0.035 \end{pmatrix}$$

Iteration 5

$$w(4) = [0.027 \quad 0.035]^T$$

$$\nabla F(w(4)) = \begin{pmatrix} 0.25 \cdot 0.027^3 + 2 \cdot (0.035)^3 + 1.5 \cdot 0.027^2 \cdot (0.035) + 3 \cdot 0.027 \cdot (0.035)^2 + 2.5 \cdot 0.027 + \cdot(0.035) \\ 0.5 \cdot 0.027^3 + 4 \cdot (0.035)^3 + 3 \cdot 0.027^2 \cdot (0.035) + 6 \cdot 0.027 \cdot (0.035)^2 + \cdot 0.027 + 4 \cdot (0.035) \end{pmatrix}$$
$$= \begin{pmatrix} 0.103 \\ 0.167 \end{pmatrix}$$

$$||\nabla F(w(4))|| = \sqrt{0.103^2 + 0.167^2} = 0.038$$

Άρα η κατεύθυνση καθόδου:

$$\frac{-\nabla F(w(4))}{||\nabla F(w(4)||} = -\frac{1}{0.038} \begin{pmatrix} 0.103 \\ 0.167 \end{pmatrix} = \begin{pmatrix} -2.71 \\ -4.394 \end{pmatrix}$$

Συνεπώς

$$w(5) = \begin{pmatrix} 0.027 \\ 0.035 \end{pmatrix} - \lambda_4 \begin{pmatrix} -2.71 \\ -4.394 \end{pmatrix} = \begin{pmatrix} 0.027 + 2.71\lambda_4 \\ 0.035 + 4.394\lambda_4 \end{pmatrix}$$

Αντικαθιστούμε στην F το w_5:

$$\begin{aligned} F(w(5)) &= (0.027 + 2.71\lambda_4)^2 + (0.035 + 4.394\lambda_4)^2 \\ &+ (0.014 + 1.355\lambda_4 + 0.035 + 4.394\lambda_4)^2 + (0.014 + 1.355\lambda_4 + 0.035 + 4.394\lambda_4)^4 \\ &= (0.027 + 2.71\lambda_4)^2 + (0.035 + 4.394\lambda_4)^2 + (5.749\lambda_4 + 0.049)^2 + (5.749\lambda_4 + 0.049)^4 \end{aligned}$$

Για να βρούμε το βρούμε το optimal λ_4, θα βρούμε την παράγωγο την προηγούμενης συνάρτησης ως προς λ_4 και θα την θέσουμε ίση με το 0.

$$\frac{dF(w(5))}{d\lambda_4} = 4369.47466\lambda_4^3 + 111.72600\lambda_4^2 + 120.35693\lambda_4 + 1.02002$$

$$\frac{dF(w(5))}{d\lambda_4} = 0$$
$$4369.47466\lambda_4^3 + 111.72600\lambda_4^2 + 120.35693\lambda_4 + 1.02002 = 0$$
$$\lambda_4 = -0.0002$$

Αντικαθιστούμε το λ4 στο w5 και για να βρούμε το καινούργιο σημείο.

$$w(5) = \begin{pmatrix} 0.027 + 2.71(-0.0002) \\ 0.035 + 4.394(-0.0002) \end{pmatrix} = \begin{pmatrix} 0.027 - 0.0005 \\ 0.035 - 0.0009 \end{pmatrix} = \begin{pmatrix} 0.265 \\ 0.034 \end{pmatrix}$$

Iteration 6

$$w(5) = [0.265 \quad 0.034]$$

$$\nabla F(w(5)) =$$
$$\begin{pmatrix} 0.25 \cdot 0.0265^3 + 2 \cdot (0.034)^3 + 1.5 \cdot 0.0265^2 \cdot (0.034) + 3 \cdot 0.0265 \cdot (0.034)^2 + 2.5 \cdot 0.0265 + \cdot(0.034) \\ 0.5 \cdot 0.0265^3 + 4 \cdot (0.034)^3 + 3 \cdot 0.0265^2 \cdot (0.034) + 6 \cdot 0.0265 \cdot (0.034)^2 + \cdot 0.0265 + 4 \cdot (0.034) \end{pmatrix}$$
$$= \begin{pmatrix} 0.1 \\ 0.162 \end{pmatrix}$$

$$||\nabla F(w(5))|| = \sqrt{0.1^2 + 0.162^2} = 0.036$$

Άρα η κατεύθυνση καθόδου:

$$\frac{-\nabla F(w(5))}{||\nabla F(w(5)||} = -\frac{1}{0.036} \begin{pmatrix} 0.1 \\ 0.162 \end{pmatrix} = \begin{pmatrix} -2.78 \\ -4.5 \end{pmatrix}$$

Συνεπώς

$$w(6) = \begin{pmatrix} 0.0265 \\ 0.034 \end{pmatrix} - \lambda_5 \begin{pmatrix} -2.78 \\ -4.5 \end{pmatrix} = \begin{pmatrix} 0.0265 + 2.78\lambda_5 \\ 0.034 + 4.5\lambda_5 \end{pmatrix}$$

Αντικαθιστούμε στην F το w_6:

$$\begin{aligned} F(w(6)) &= (0.0265 + 2.78\lambda_5)^2 + (0.034 + 4.5\lambda_5)^2 \\ &\quad + (0.01325 + 1.355\lambda_5 + 0.034 + 4.5\lambda_5)^2 + (0.01325 + 1.355\lambda_5 + 0.034 + 4.5\lambda_5)^4 \\ &= (0.0265 + 2.78\lambda_5)^2 + (0.034 + 4.5\lambda_5)^2 + (5.855\lambda_5 + 0.048)^2 + (5.855\lambda_5 + 0.048)^4 \end{aligned}$$

Για να βρούμε το βρούμε το optimal λ_5, θα βρούμε την παράγωγο την προηγούμενης συνάρτησης ως προς λ_5 και θα την θέσουμε ίση με το 0.

$$\frac{dF(w(6))}{d\lambda_5} = 4700.75470\lambda_5^3 + 115.61207\lambda_5^2 + 125.46665\lambda_5 + 1.02701$$

$$\frac{dF(w(6))}{d\lambda_5} = 0$$
$$4700.75470\lambda_5^3 + 115.61207\lambda_5^2 + 125.46665\lambda_5 + 1.02701 = 0$$
$$\lambda_5 = -0.0002$$

Αντικαθιστούμε το λ5 στο w6 και για να βρούμε το καινούργιο σημείο.

$$w(6) = \begin{pmatrix} 0.0265 + 2.78(-0.0002) \\ 0.034 + 4.35(-0.0002) \end{pmatrix} = \begin{pmatrix} 0.0265 - 0.0005 \\ 0.034 - 0.0009 \end{pmatrix} = \begin{pmatrix} 0.26 \\ 0.033 \end{pmatrix}$$

Iteration 7

$$w(6) = [0.26 \quad 0.033]$$

$$\nabla F(w(6)) = \begin{pmatrix} 0.25 \cdot 0.026^3 + 2 \cdot (0.033)^3 + 1.5 \cdot 0.026^2 \cdot (0.033) + 3 \cdot 0.026 \cdot (0.033)^2 + 2.5 \cdot 0.026 + \cdot(0.033) \\ 0.5 \cdot 0.026^3 + 4 \cdot (0.033)^3 + 3 \cdot 0.026^2 \cdot (0.033) + 6 \cdot 0.026 \cdot (0.033)^2 + \cdot 0.026 + 4 \cdot (0.033) \end{pmatrix} = \begin{pmatrix} 0.098 \\ 0.158 \end{pmatrix}$$

$$||\nabla F(w(6))|| = \sqrt{0.098^2 + 0.158^2} = 0.034$$

Άρα η κατεύθυνση καθόδου:

$$\frac{-\nabla F(w(6))}{||\nabla F(w(6)||} = -\frac{1}{0.034} \begin{pmatrix} 0.098 \\ 0.158 \end{pmatrix} = \begin{pmatrix} -2.882 \\ -4.647 \end{pmatrix}$$

Συνεπώς

$$w(7) = \begin{pmatrix} 0.026 \\ 0.033 \end{pmatrix} - \lambda_6 \begin{pmatrix} -2.882 \\ -4.647 \end{pmatrix} = \begin{pmatrix} 0.026 + 2.882\lambda_6 \\ 0.033 + 4.647\lambda_6 \end{pmatrix}$$

Αντικαθιστούμε στην F το w_7:

$$\begin{aligned} F(w(6)) &= (0.026 + 2.882\lambda_6)^2 + (0.033 + 4.647\lambda_6)^2 \\ &+ (0.013 + 1.441\lambda_6 + 0.033 + 4.647\lambda_6)^2 + (0.013 + 1.441\lambda_6 + 0.033 + 4.647\lambda_6)^4 \\ &= (0.026 + 2.882\lambda_6)^2 + (0.033 + 4.647\lambda_6)^2 + (6.088\lambda_6 + 0.048)^2 + (6.088\lambda_6 + 0.048)^4 \end{aligned}$$

Για να βρούμε το βρούμε το optimal λ_6, θα βρούμε την παράγωγο την προηγούμενης συνάρτησης ως προς λ_6 και θα την θέσουμε ίση με το 0.

$$\frac{dF(w(7))}{d\lambda_6} = 5494.88447\lambda_6^3 + 129.97098\lambda_6^2 + 134.95329\lambda_6 + 1.04370$$

$$\begin{gathered} \frac{dF(w(7))}{d\lambda_6} = 0 \\ 4814.16720\lambda_6^3 + 112.79373\lambda_6^2 + 126.22190\lambda_6 + 0.98573 = 0 \\ \lambda_6 = -0.00018 \end{gathered}$$

Αντικαθιστούμε το λ6 στο w7 και για να βρούμε το καινούργιο σημείο.

$$w(7) = \begin{pmatrix} 0.026 + 2.882(-0.00018) \\ 0.033 + 4.647(-0.00018) \end{pmatrix} = \begin{pmatrix} 0.026 - 0.0005 \\ 0.033 - 0.0008 \end{pmatrix} = \begin{pmatrix} 0.0255 \\ 0.0322 \end{pmatrix}$$

Iteration 8

$$w(7) = [0.0255 \quad 0.0.322]$$

$$\nabla F(w(7)) =$$
$$\begin{pmatrix} 0.25 \cdot 0.0255^3 + 2 \cdot (0.0322)^3 + 1.5 \cdot 0.0255^2 \cdot (0.0322) + 3 \cdot 0.0255 \cdot (0.0322)^2 + 2.5 \cdot 0.0255 + \cdot(0.0322) \\ 0.5 \cdot 0.0255^3 + 4 \cdot (0.0322)^3 + 3 \cdot 0.0255^2 \cdot (0.0322) + 6 \cdot 0.0255 \cdot (0.0322)^2 + \cdot 0.0255 + 4 \cdot (0.0322) \end{pmatrix}$$
$$= \begin{pmatrix} 0.096 \\ 0.154 \end{pmatrix}$$

$$||\nabla F(w(7))|| = \sqrt{0.096^2 + 0.154^2} = 0.033$$

Άρα η κατεύθυνση καθόδου:

$$\frac{-\nabla F(w(7))}{||\nabla F(w(7)||} = -\frac{1}{0.033} \begin{pmatrix} 0.096 \\ 0.154 \end{pmatrix} = \begin{pmatrix} -2.909 \\ -4.667 \end{pmatrix}$$

Συνεπώς

$$w(8) = \begin{pmatrix} 0.0255 \\ 0.0322 \end{pmatrix} - \lambda_7 \begin{pmatrix} -2.909 \\ -4.667 \end{pmatrix} = \begin{pmatrix} 0.0255 + 2.909\lambda_7 \\ 0.0322 + 4.667\lambda_7 \end{pmatrix}$$

Αντικαθιστούμε στην F το w_8:

$$\begin{aligned} F(w(8)) &= (0.0255 + 2.909\lambda_7)^2 + (0.0322 + 4.647\lambda_7)^2 \\ &\quad + (0.0127 + 1.4545\lambda_7 + 0.0322 + 4.647\lambda_7)^2 + (0.0127 + 1.4545\lambda_7 + 0.0322 + 4.647\lambda_7)^4 \\ &= (0.0255 + 2.909\lambda_7)^2 + (0.0322 + 4.647\lambda_7)^2 + (6.1015\lambda_7 + 0.0449)^2 + (6.1015\lambda_7 + 0.0449)^4 \end{aligned}$$

Για να βρούμε το βρούμε το optimal λ_7, θα βρούμε την παράγωγο την προηγούμενης συνάρτησης ως προς λ_7 και θα την θέσουμε ίση με το 0.

$$\frac{dF(w(8))}{d\lambda_7} = 5543.78595\lambda_7^3 + 122.38760\lambda_7^2 + 135.4710\lambda_7 + 0.99774$$

$$\frac{dF(w(8))}{d\lambda_7} = 0$$
$$5543.78595\lambda_7^3 + 122.38760\lambda_7^2 + 135.4710\lambda_7 + 0.99774 = 0$$
$$\lambda_7 = -0.00017$$

Αντικαθιστούμε το λ7 στο w8 και για να βρούμε το καινούργιο σημείο.

$$w(8) = \begin{pmatrix} 0.0255 + 2.909(-0.00017) \\ 0.0322 + 4.667(-0.00017) \end{pmatrix} = \begin{pmatrix} 0.0255 - 0.0005 \\ 0.0322 - 0.0008 \end{pmatrix} = \begin{pmatrix} 0.025 \\ 0.0314 \end{pmatrix}$$

Iteration 9

$$w(8) = [0.025 \quad 0.0314$$

$$\nabla F(w(8)) =$$
$$\begin{pmatrix} 0.25 \cdot 0.025^3 + 2 \cdot (0.0314)^3 + 1.5 \cdot 0.025^2 \cdot (0.0314) + 3 \cdot 0.025 \cdot (0.0314)^2 + 2.5 \cdot 0.025 + \cdot(0.0314) \\ 0.5 \cdot 0.025^3 + 4 \cdot (0.0314)^3 + 3 \cdot 0.025^2 \cdot (0.0314) + 6 \cdot 0.025 \cdot (0.0314)^2 + \cdot 0.025 + 4 \cdot (0.0314) \end{pmatrix}$$
$$= \begin{pmatrix} 0.094 \\ 0.15 \end{pmatrix}$$

$$||\nabla F(w(8))|| = \sqrt{0.096^2 + 0.154^2} = 0.031$$

Άρα η κατεύθυνση καθόδου:

$$\frac{-\nabla F(w(8))}{||\nabla F(w(8))||} = -\frac{1}{0.031} \begin{pmatrix} 0.094 \\ 0.15 \end{pmatrix} = \begin{pmatrix} -3.032 \\ -4.839 \end{pmatrix}$$

Συνεπώς

$$w(9) = \begin{pmatrix} 0.025 \\ 0.0314 \end{pmatrix} - \lambda_8 \begin{pmatrix} -3.032 \\ -4.839 \end{pmatrix} = \begin{pmatrix} 0.025 + 3.032\lambda_8 \\ 0.0314 + 4.839\lambda_8 \end{pmatrix}$$

Αντικαθιστούμε στην F το w_9:

$$\begin{aligned} F(w(9)) &= (0.025 + 3.032\lambda_8)^2 + (0.0314 + 4.839\lambda_8)^2 \\ &\quad + (0.0125 + 1.516\lambda_8 + 0.0314 + 4.839\lambda_8)^2 + (0.0125 + 1.516\lambda_8 + 0.0314 + 4.839\lambda_8)^4 \\ &= (0.025 + 3.032\lambda_8)^2 + (0.0314 + 4.839\lambda_8)^2 + (6.355\lambda_8 + 0.0439)^2 + (6.355\lambda_8 + 0.0439)^4 \end{aligned}$$

$$\frac{dF(w(9))}{d\lambda_8} = 6524.12406\lambda_8^3 + 135.20489\lambda_8^2 + 146.92392\lambda_8 + 1.01560$$

$$\frac{dF(w(9))}{d\lambda_8} = 0$$
$$6524.12406\lambda_8^3 + 135.20489\lambda_8^2 + 146.92392\lambda_8 + 1.01560 = 0$$
$$\lambda_8 = -0.00014$$

Αντικαθιστούμε το λ8 στο w9 και για να βρούμε το καινούργιο σημείο.

$$w(9) = \begin{pmatrix} 0.025 + 3.032(-0.00014) \\ 0.0314 + 4.839(-0.00014) \end{pmatrix} = \begin{pmatrix} 0.025 - 0.0004 \\ 0.0314 - 0.0007 \end{pmatrix} = \begin{pmatrix} 0.021 \\ 0.0307 \end{pmatrix}$$

Iteration 10

$$w(9) = [0.021 \quad 0.0307]$$

$$\nabla F(w(9)) =$$
$$\begin{pmatrix} 0.25 \cdot 0.021^3 + 2 \cdot (0.0307)^3 + 1.5 \cdot 0.021^2 \cdot (0.0307) + 3 \cdot 0.021 \cdot (0.0307)^2 + 2.5 \cdot 0.021 + \cdot(0.0307) \\ 0.5 \cdot 0.021^3 + 4 \cdot (0.0307)^3 + 3 \cdot 0.021^2 \cdot (0.0307) + 6 \cdot 0.021 \cdot (0.0307)^2 + \cdot 0.021 + 4 \cdot (0.0307) \end{pmatrix}$$
$$= \begin{pmatrix} 0.083 \\ 0.143 \end{pmatrix}$$

$$||\nabla F(w(9))|| = \sqrt{0.083^2 + 0.143^2} = 0.027$$

Άρα η κατεύθυνση καθόδου:

$$\frac{-\nabla F(w(9))}{||\nabla F(w(9)||} = -\frac{1}{0.027}\begin{pmatrix}0.083\\0.143\end{pmatrix} = \begin{pmatrix}-3.074\\-5.296\end{pmatrix}$$

Συνεπώς

$$w(10) = \begin{pmatrix}0.021\\0.0307\end{pmatrix} - \lambda_9\begin{pmatrix}-3.074\\-5.296\end{pmatrix} = \begin{pmatrix}0.021 + 3.074\lambda_9\\0.0307 + 5.296\lambda_9\end{pmatrix}$$

Αντικαθιστούμε στην F το w_{10}:

$$\begin{aligned}F(w(10)) &= (0.021 + 3.074\lambda_9)^2 + (0.0307 + 5.296\lambda_9)^2 \\ &+ (0.0105 + 1.537\lambda_9 + 0.0307 + 5.296\lambda_9)^2 + (0.0105 + 1.537\lambda_9 + 0.0307 + 5.296\lambda_9)^4 \\ &= (0.021 + 3.074\lambda_9)^2 + (0.0307 + 5.296\lambda_9)^2 + (6.833\lambda_9 + 0.0412)^2 + (6.833\lambda_9 + 0.0412)^4\end{aligned}$$

$$\frac{dF(w(10))}{d\lambda_9} = 8719.78293\lambda_9^3 + 157.72942\lambda_9^2 + 169.32500\lambda_9 + 1.01923$$

$$\begin{aligned}\frac{dF(w(10))}{d\lambda_9} &= 0 \\ 8719.78293\lambda_9^3 + 157.72942\lambda_9^2 + 169.32500\lambda_9 + 1.01923 &= 0 \\ \lambda_9 &= -0.0001\end{aligned}$$

Αντικαθιστούμε το λ9 στο w10 και για να βρούμε το καινούργιο σημείο.

$$w(10) = \begin{pmatrix}0.021 + 3.074(-0.0001)\\0.0307 + 5.296(-0.0001)\end{pmatrix} = \begin{pmatrix}0.021 - 0.0003\\0.0307 - 0.0005\end{pmatrix} = \begin{pmatrix}0.0207\\0.0302\end{pmatrix}$$

2 Problem-02

```
import numpy as np
import matplotlib.pyplot as plt
from sympy import *

def newton_multi_var(grad_F, H, x_init, fixed_step):
    x = x_init
    iter = 0
```

```
    while True:
        iter = iter + 1
        H_inv = np.linalg.inv(np.float64(H(x)))
        x_move = np.dot(H_inv, np.float64(grad_F(x)))
        x = x - fixed_step*x_move
        if np.linalg.norm(x_move) < 1e-8:
            break

    return x, iter

w1, w2 = symbols("w1 w2")

f = w1**2 + w2**2 + (0.5*w1 + w2)**2 + (0.5*w1 + w2)**4
x_init = np.array([3, 3])
fixed_step = 1
grad_w1 = f.diff(w1)
grad_w2 = f.diff(w2)
calculate_grad_f = lambda val: np.array(
    [
        grad_w1.subs([(w1, val[0]), (w2, val[1])]),
        grad_w2.subs([(w1, val[0]), (w2, val[1])]),
    ]
)
H = lambda val: np.array(
    [
        [
            f.diff(w1, w1).subs([(w1, val[0]), (w2, val[1])]),
            f.diff(w1, w2).subs([(w1, val[0]), (w2, val[1])]),
        ],
        [
            f.diff(w1, w2).subs([(w1, val[0]), (w2, val[1])]),
            f.diff(w2, w2).subs([(w1, val[0]), (w2, val[1])]),
        ],
    ]
)

local_min, iterations = newton_multi_var(calculate_grad_f, H, x_init, fixed_step)
round_local_min_pos = np.array([round(local_min[0], 3), round(local_min[1], 3)])

print("Number of iteration: ", iterations)
print("Local min: ({0},{1})".format(round_local_min_pos[0]
```

```
, round_local_min_pos[1]))
f = lambda val: val[0]**2 + val[1]**2 + (0.5*val[0] + val[1])**2
    + (0.5*val[0] + val[1])**4

x = np.linspace(-10, 10, 10)
y = np.linspace(-10, 10, 10)
X, Y = np.meshgrid(x, y)
Z = f(np.array([X, Y]))
ax = plt.axes(projection="3d")
ax.set_title("f = w1**2 + w2**2 + (0.5*w1 + w2)**2 + (0.5*w1 + w2)**4")
ax.contour3D(X, Y, Z, 30)
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("z")
ax.scatter([local_min[0]], [local_min[1]], [f(local_min)], color="red")
label = "Local min: ({0},{1})".format(
round_local_min_pos[0], round_local_min_pos[1]
)
ax.text(round_local_min_pos[0], round_local_min_pos[1], f(local_min), label, None)
plt.show()
```

Output:
Number of iteration: 9
Local min: (-0.0,0.0)

3 Problem-03

Αρχικά υπολογίζουμε τις παραγώγους των activation functions σε κάθε layer διότι θα τις χρειαστούμε για την εύρεση των sensitivities.

$$\text{Για το layer 1: } f^1(x) = \text{Swish} = \frac{x}{1+e^{-x}}$$

$$\text{Για το layer 2: } f^2(x) = \text{LReLU με α=0.001} = \begin{cases} x & x > 0 \\ 0.001x & x \leq 0 \end{cases}$$

$$f'^1(x) = \frac{xe^{-x} + e^{-x} + 1}{(1+e^{-x})^2}$$

$$f'^2(x) = \begin{cases} 1 & x > 0 \\ 0.001 & x \leq 0 \end{cases}$$

ITERATION 1
Ξεκινάμε το propagate του input:

$$n^1 = w^1 p + b^1 = -3 \cdot 1 + 2 = -3 + 2 = -1$$

$$a^1 = swish(n^1) = swish(-1) = -\frac{1}{1+e}$$

$$n^2 = w^2 a^1 + b^2 = (-1)\cdot\frac{-1}{1+e} - 1 = \frac{1-(1+e)}{1+e} = \frac{-e}{1+e}$$

$$a^2 = LReLU(n^2) = 0.001\cdot\frac{-e}{1+e} = -\frac{0.001\cdot e}{1+e}$$

$$error = t - a^2 = 0 - (-\frac{0.001\cdot e}{1+e}) = \frac{0.001\cdot e}{1+e}$$

Έπειτα κάνουμε backpropage τα sensitivities:

$$s^2 = -2f'^2(n^2)(t-a^2) = -2(0.001)(error) = \frac{-2e(0.001)^2}{1+e}$$

$$s^1 = f'^1(n^1)(W^2)^T s^2 = \frac{(-1)e^{-(-1)} + e^{-(-1)} + 1}{(1+e^{-(-1)})^2} w^2 s2 = \frac{-e+e+1}{(1+e)^2}(-1)\frac{-2e(0.001)^2}{1+e}$$
$$= \frac{-e+e+1}{(1+e)^2}\cdot\frac{2e(0.001)^2}{1+e} = \frac{2e(0.001)^2}{(1+e)^3}$$

Τέλος ενημερώνουμε τα weights και τα biases:

$$w^2(1) = w^2(0) - as^2(a^1)^T = -1 - 1\cdot\frac{-2e(0.001)^2}{1+e}\cdot\frac{-1}{1+e} = -1 - \frac{2e(0.001)^2}{(1+e)^2}$$

$$b^2(1) = b^2(0) - as^2 = -1 - 1\frac{-2e(0.001)^2}{1+e} = -1 + \frac{2e(0.001)^2}{1+e}$$

$$w^1(1) = w^1(0) - as^1(a^0)^T = -3 - 1\cdot\frac{2e(0.001)^2}{(1+e)^3}\cdot 1 = -3 - \frac{2e(0.001)^2}{(1+e)^3}$$

$$b^1(1) = b^1(0) - as^1 = 2 - 1\cdot\frac{2e(0.001)^2}{(1+e)^3} = 2 - \frac{2e(0.001)^2}{(1+e)^3}$$

ITERATION 2

Ξεκινάμε το propagate του input:

$$n^1 = w^1 p + b^1 = (-3 - \frac{2e(0.001)^2}{(1+e)^3})\cdot 1 + 2 - \frac{2e(0.001)^2}{(1+e)^3} = -1 - \frac{4e(0.001)^2}{(1+e)^3} \approx -1$$

Το κλάσμα μπορεί να θεωρηθεί αμελητέο αφού αν κάνουμε τον υπολογισμό:
-0.000000211508371

$$a^1 = swish(n^1) = swish(-1) = -\frac{1}{1+e}$$

$$n^2 = w^2a^1+b^2 = (-1-\frac{2e(0.001)^2}{(1+e)^2})\cdot\frac{-1}{1+e}-1+\frac{2e(0.001)^2}{1+e} = \frac{1}{1+e}+\frac{2e(0.001)^2}{(1+e)^3}-1+\frac{2e(0.001)^2}{1+e}$$
$$= \frac{2e(0.001)^2+1}{1+e} + \frac{2e(0.001)^2}{(1+e)^3} - 1 = -0.731$$

$$a^2 = LReLU(n^2) = 0.001 \cdot (-0.731) = -0.000731$$
$$error = t - a^2 = 0 - (-0.000731) = 0.000731$$

Έπειτα κάνουμε backpropage τα sensitivities:

$$s^2 = -2f'^2(n^2)(t-a^2) = -2(0.001)(error) = -2\cdot(0.001)\cdot 0.000731 = -0.00146$$

$$s^1 = f'^1(n^1)(W^2)^T s^2 = \frac{(-1)e^{-(-1)}+e^{-(-1)}+1}{(1+e^{-(-1)})^2}w^2 s2 = \frac{1}{(1+e)^2}\cdot(-1-\frac{2e(0.001)^2}{(1+e)^2})\cdot(-0.00146)$$
$$= \frac{0.00146}{(1+e)^2} + \frac{2e(0.001)^2\cdot 0.00146}{(1+e)^4} = 0.00010$$

Τέλος ενημερώνουμε τα weights και τα biases:

$$w^2(2) = w^2(1)-as^2(a^1)^T = -1-\frac{2e(0.001)^2}{(1+e)^2}-1\cdot(-0.00146)\cdot\frac{-1}{1+e} = -1-\frac{2e(0.001)^2}{(1+e)^2}-\frac{0.00146}{1+e}$$

$$b^2(2) = b^2(1) - as^2 = -1 + \frac{2e(0.001)^2}{1+e} - 1\cdot(-0.00146) = \frac{2e(0.001)^2}{1+e} - 0.99854$$

$$w^1(2) = w^1(1)-as^1(a^0)^T = -3-\frac{2e(0.001)^2}{(1+e)^3}-1\cdot 0.00010\cdot 1 = -3.00010-\frac{2e(0.001)^2}{(1+e)^3}$$

$$b^1(1) = b^1(0) - as^1 = 2 - \frac{2e(0.001)^2}{(1+e)^3} - 1\cdot 0.00010 = 1.9999 - \frac{2e(0.001)^2}{(1+e)^3}$$

4 Problem-04

Ο κώδικας αναφέρεται στην περίπτωση για τους 12 νευρώνες

```
    import numpy as np
import numpy as np
import random
import matplotlib.pyplot as plt
```

```
def g(x):
    return 1 + np.sin(x * (3 * np.pi) / 8)

def logsig(x):
    return 1/(1 + np.exp(-x))

def relu(x):
    return max(x,0)

def derivative_logsig(x):
    return logsig(x) * (1 - logsig(x))

def derivative_relu(x):
    if x > 0:
        return 1
    else:
        return 0

def evaluate(p, W1, b1, W2, b2):
    n1 = np.dot(W1, p)+ b1
    a1 = logsig(n1)
    n2 = np.dot(a1, W2) + b2[0]
    a2 = relu(n2)
    actual = g(p)
    error = actual - a2

    return error
# init

s = 12 # number of neuros in hidden layer 2,8,12
max_limit = 0.5
min_limit = -0.5
a = 0.1 # learning rate

# weights and biases for first layer
W1 = [random.uniform(min_limit,max_limit) for _ in range(s)]
b1 = [random.uniform(min_limit,max_limit) for _ in range(s)]
#weights and bias for second layer
W2 = [random.uniform(min_limit,max_limit) for _ in range(s)]
b2 = [random.uniform(min_limit,max_limit)] # 1 neuron so 1 bias
```

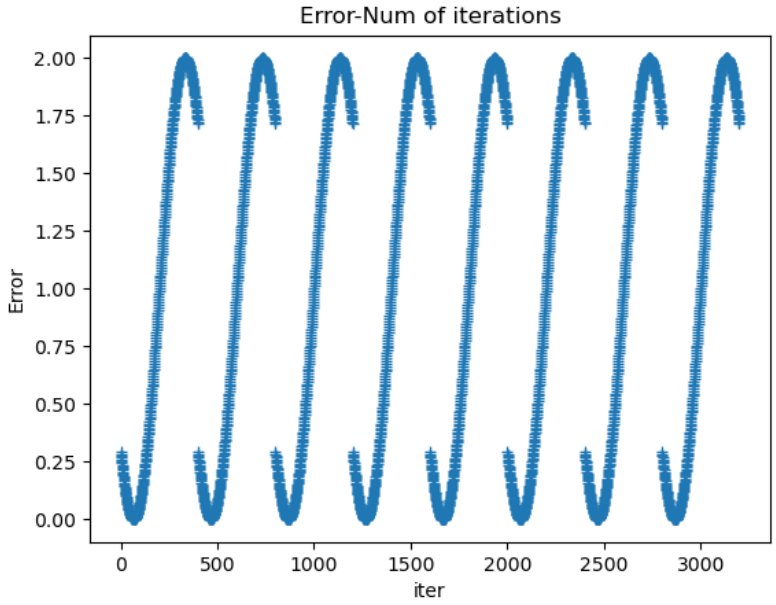
```
# lists for plotting
error_to_plot = []
input_list = []
actual_values = []
predicted_values = []
# Initialize lists corresponding to each value of W2
w2_lists = [[] for _ in range(s)]

# Assign values to corresponding w2 lists
for i in range(s):
    w2_lists[i] = [W2[i]]

# Unpack the w2_lists into individual lists w2_1_list, w2_2_list, ..., w2_12_list
(w2_1_list, w2_2_list, w2_3_list, w2_4_list, w2_5_list,
 w2_6_list, w2_7_list, w2_8_list, w2_9_list, w2_10_list,
 w2_11_list, w2_12_list) = w2_lists

i=0
while (i < 4000):
    for p in np.arange(-2,2,0.01):

        # Feed forward for single p
        n1 = np.dot(W1, p)+ b1
        a1 = logsig(n1)
        n2 = np.dot(a1, W2) + b2[0]
        a2 = relu(n2)

        # Compute actual value
        actual = g(p)

        # Compute error
        error = actual - a2

        # Compute sensitivities:
        s2 = -2 * derivative_relu(n2) * error
        s1 = np.dot(derivative_logsig(n1), W2) * s2

        # Update weigths and biases
        W1 = np.array(W1) - a * s1 * p
        b1 = np.array(b1) - a * s1
        W2 = np.array(W2) - a * s2 * a1
```

```
        b2 = np.array(b2) - a * s2

        # Store values for plotting
        input_list.append(p)
        predicted_values.append(a2)
        actual_values.append(actual)
        error_to_plot.append(error)

        w2_1_list.append(W2[0])
        w2_2_list.append(W2[1])
        w2_3_list.append(W2[2])
        w2_4_list.append(W2[3])
        w2_5_list.append(W2[4])
        w2_6_list.append(W2[5])
        w2_7_list.append(W2[6])
        w2_8_list.append(W2[7])
        w2_9_list.append(W2[8])
        w2_10_list.append(W2[9])
        w2_11_list.append(W2[10])
        w2_12_list.append(W2[11])

        # Update i
        i = i + 1

# MAIN
print("-----EVALUATION-----")
point_not_int_dataset = 0.005
point_in_dataset = 0.1
print('error(%.3f) = ' % point_not_int_dataset,
              evaluate(point_not_int_dataset, W1, b1, W2, b2))
print('error(%.2f) = '% point_in_dataset,
              evaluate(point_in_dataset, W1, b1, W2, b2))


plt.title("Actual-Predicted values")
plt.xlabel('iter')
plt.ylabel('g(p)')
plt.plot( actual_values, 'o', label='actual values')
plt.plot( predicted_values, '+', label='Predicted values')
plt.legend()
plt.show()
```

```
plt.title("Error over iteration")
plt.xlabel('iter')
plt.ylabel('Error')
plt.plot(error_to_plot, '+', label='Error')
plt.legend()
plt.show()


plt.title("W2 Convergence over iteration")
plt.xlabel('iter')
plt.ylabel('w2 values')
plt.plot(w2_1_list, 'o', label='W2_1 Convergence')
plt.plot(w2_2_list, 'o', label='W2_2 Convergence')
plt.plot(w2_3_list, 'o', label='W2_3 Convergence')
plt.plot(w2_4_list, 'o', label='W2_4 Convergence')
plt.plot(w2_5_list, 'o', label='W2_5 Convergence')
plt.plot(w2_6_list, 'o', label='W2_6 Convergence')
plt.plot(w2_7_list, 'o', label='W2_7 Convergence')
plt.plot(w2_8_list, 'o', label='W2_8 Convergence')
plt.plot(w2_9_list, 'o', label='W2_9 Convergence')
plt.plot(w2_10_list, 'o', label='W2_10 Convergence')
plt.plot(w2_11_list, 'o', label='W2_11 Convergence')
plt.plot(w2_12_list, 'o', label='W2_12 Convergence')
plt.legend()
plt.show()
```

Σχόλια:

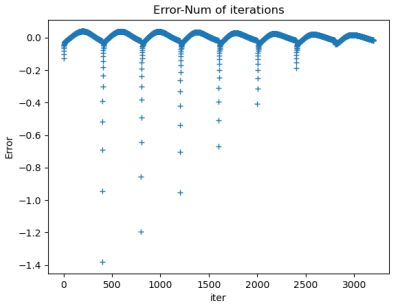
– Παρατηρήσαμε ότι δεν είναι δεκτές όλες οι πιθανές αρχικές τιμές για τα weigths και τα biases (οι αριθμοί από -0.5 έως 0.5). Για τους συνδυασμούς που δίνουν $n^2 < 0$ η παράγωγος της ReLu είναι 0, άρα το s2 = 0 και συνεπώς το s1 = 0, ενώ και η ίδια η ReLU για αρνητικές τιμές είναι μηδέν και έτσι η έξοδος του νευρωνικού είναι μηδέν. Αφού τα sensitivities είναι μηδέν τα weights και τα biases δεν μπορούν να γίνουν update και έτσι η εκπαίδευση καταλήγει σε ένα loop που τα parameters δεν αλλάζουν. Το παρακάτω διάγραμμα του error:

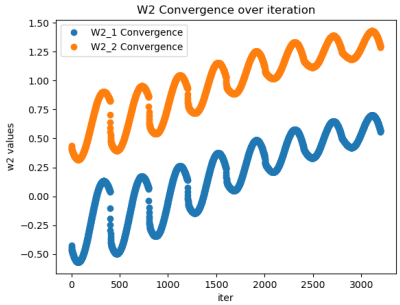


iteration: Μια επανάληψη feed-forward και backpropagation.
Βλέπουμε ότι το error δεν συγκλίνει και είναι περιοδικό, αυτό είναι λογικό αφού πάντα error = g(p)-0. Δηλαδή το error προσομοιώνει την συνάρτηση.

– Ως trainning set χρησιμοποιήσαμε: $-2 \leq p \leq 2$ με βήμα 0.01
– Ως accuracy θα μετρήσουμε το error του νευρωνικού για μια τιμή που δεν υπήρχε στο dataset και για μια που υπήρχε για να δούμε πως θα τις προσομοιώσει (το 0.005 και το 0.1).

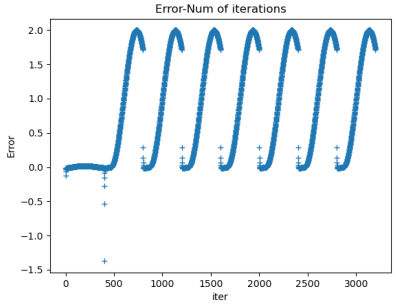
- $S^1 = 2, a = 0.1$
error(0.005)=-0.058
error(0.1) = 0.0016

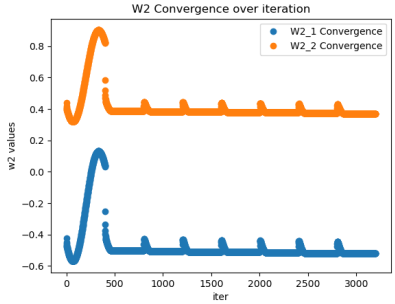




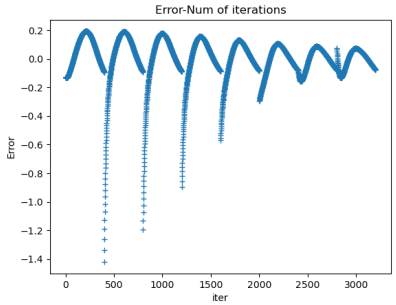
- $S^1 = 2, a = 0.2$
error(0.005)=1.0058

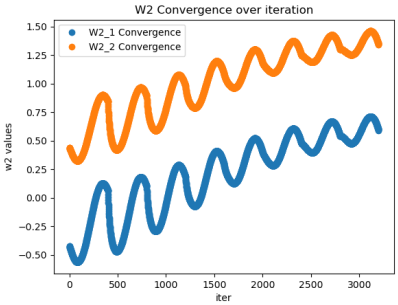
error(0.1)=1.1175





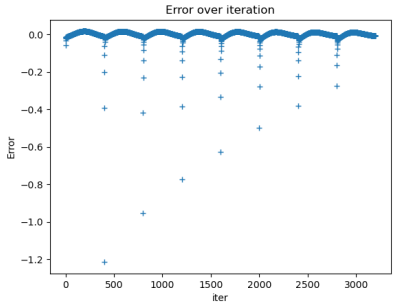
- $S^1 = 2, a = 0.02$
  error(0.005)=-0.051
  error(0.1)=0.000023

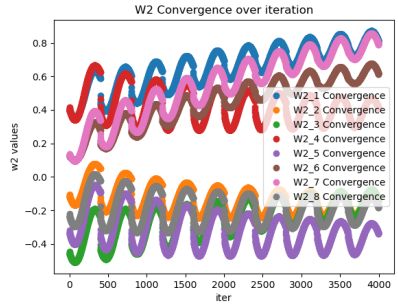




Βάση πειραματισμών παρατηρούμε ότι για α>0.2 δεν υπάρχει σύγκλιση του error και των parameters, ενώ για μικρή τιμή παίρνουμε καλύτερα αποτελέσματα. Η τιμή 0.1 είναι κατάλληλη άλλα όχι απαραίτητα η καλύτερη

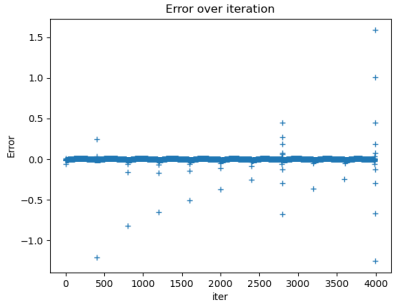
- $S^1 = 8, a = 0.1$
  error(0.005) = -0.107
  error(0.1) = -0.0321

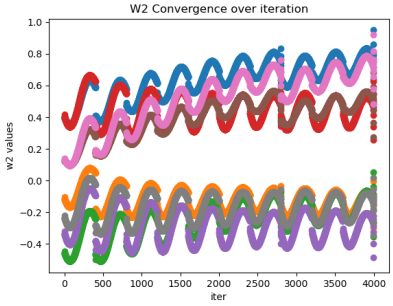




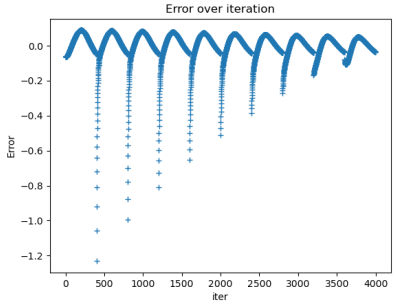
- $S^1 = 8, a = 0.2$
  error(0.005) = -0.810

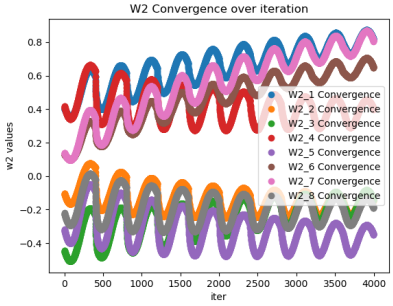
error(0.1) = -0.717



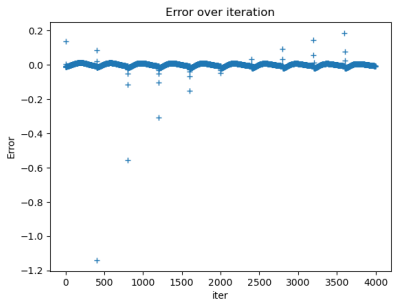


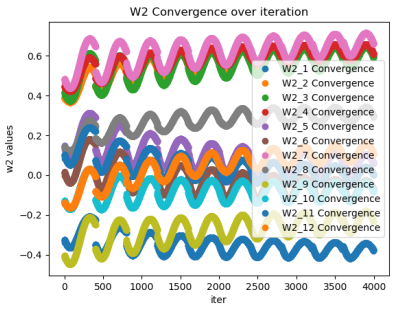
- $S^1 = 8, a = 0.02$
  error(0.005) = -0.0625
  error(0.1) = 0.00422





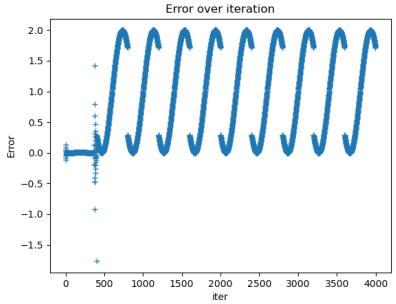
- $S^1 = 12, a = 0.1$
  error(0.005) = 0.0585
  error(0.1) = 0.1211

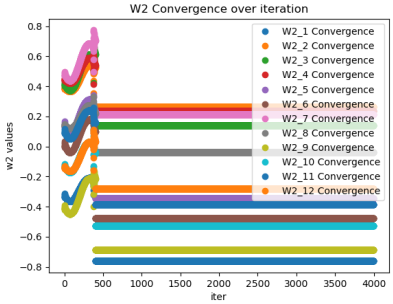




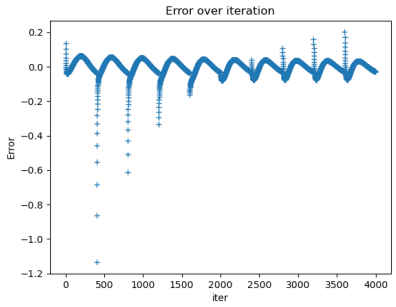
- $S^1 = 12, a = 0.2$
  error(0.005) = 1.0058

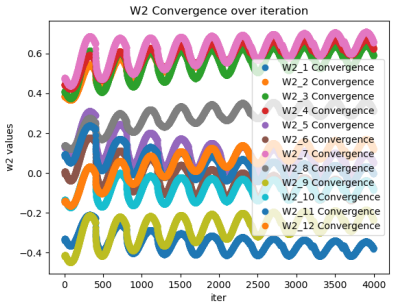
error(0.1) = 1.1175





- $S^1 = 12, a = 0.02$
  error(0.005) = 0.0533
  error(0.1) = 0.1146





Και για τις τρεις αρχιτεκτονικές τα καλύτερα αποτελέσματα και την μεγαλύτερη σύγκλιση την παίρνουμε για α=0.02, ενώ όταν το α μεγαλώσει

5 Problem-05

Ο κώδικας αναφέρεται στην περίπτωση για τους 12 νευρώνες

```
import numpy as np
import random
import matplotlib.pyplot as plt

def g(x):
    return 1 + np.sin(x * (3 * np.pi) / 8)

def logsig(x):
    return 1/(1 + np.exp(-x))
```

```
def relu(x):
    return max(x,0)

def derivative_logsig(x):
    return logsig(x) * (1 - logsig(x))

def derivative_relu(x):
    if x > 0:
        return 1
    else:
        return 0

def evaluate(p, W1, b1, W2, b2):
    n1 = np.dot(W1, p)+ b1
    a1 = logsig(n1)
    n2 = np.dot(a1, W2) + b2[0]
    a2 = relu(n2)
    actual = g(p)
    error = actual - a2

    return error

# init

s = 12 # number of neuros in hidden layer 2,8,12
max_limit = 0.5
min_limit = -0.5
a = 0.1 # learning rate
theta = 0.35 # 0.15 || 0.25 || 0.35

# weights and biases for first layer
W1 = [random.uniform(min_limit,max_limit) for _ in range(s)]
b1 = [random.uniform(min_limit,max_limit) for _ in range(s)]
#weights and bias for second layer
W2 = [random.uniform(min_limit,max_limit) for _ in range(s)]
b2 = [random.uniform(min_limit,max_limit)] # 1 neuron so 1 bias

# lists for plotting
error_to_plot = []
input_list = []
```

```
actual_values = []
predicted_values = []
# Initialize lists corresponding to each value of W2
w2_lists = [[] for _ in range(s)]

# Assign values to corresponding w2 lists
for i in range(s):
    w2_lists[i] = [W2[i]]

# Unpack the w2_lists into individual lists w2_1_list, w2_2_list, ..., w2_12_list
(w2_1_list, w2_2_list, w2_3_list, w2_4_list, w2_5_list,
 w2_6_list, w2_7_list, w2_8_list, w2_9_list, w2_10_list,
 w2_11_list, w2_12_list) = w2_lists

i=0
while (i < 8000):
    for p in np.arange(-2,2,0.01):

        # Feed forward
        n1 = np.dot(W1, p)+ b1
        a1 = logsig(n1)
        # Creates random probability for each neuro and converts to TRUE OR FALSE
        d1 = np.random.rand(a1.shape[0]) < theta
        # Drops neuros and FALSE
        a1 = a1*d1
        n2 = np.dot(a1, W2) + b2[0]
        a2 = relu(n2)

       # Compute actual value
        actual = g(p)

        # Compute error
        error = actual - a2

        # Compute sensitivities:
        s2 = -2 * derivative_relu(n2) * error
        s1 = np.dot(derivative_logsig(n1), W2) * s2

        # Update weights and biases
        W1 = np.array(W1) - a * s1 * p
        b1 = np.array(b1) - a * s1
```

```
        W2 = np.array(W2) - a * s2 * a1
        b2 = np.array(b2) - a * s2

        # Store values for plotting
        input_list.append(p)
        predicted_values.append(a2)
        actual_values.append(actual)
        error_to_plot.append(error)

        w2_1_list.append(W2[0])
        w2_2_list.append(W2[1])
        w2_3_list.append(W2[2])
        w2_4_list.append(W2[3])
        w2_5_list.append(W2[4])
        w2_6_list.append(W2[5])
        w2_7_list.append(W2[6])
        w2_8_list.append(W2[7])
        w2_9_list.append(W2[8])
        w2_10_list.append(W2[9])
        w2_11_list.append(W2[10])
        w2_12_list.append(W2[11])

        # Update i
        i = i + 1


#printing evaluation

print("-----EVALUATION-----")
point_not_int_dataset = 0.005
point_in_dataset = 0.1
print('error(%.3f) = ' % point_not_int_dataset,
                    evaluate(point_not_int_dataset, W1, b1, W2, b2))
print('error(%.1f) = '% point_in_dataset,
                    evaluate(point_in_dataset, W1, b1, W2, b2))


plt.title("Actual-Predicted values")
plt.xlabel('iter')
plt.ylabel('g(p)')
plt.plot( actual_values, 'o', label='actual values')
```

```
plt.plot( predicted_values, '+', label='Predicted values')
plt.legend()
plt.show()

plt.title("Error over iteration")
plt.xlabel('iter')
plt.ylabel('Error')
plt.plot(error_to_plot, '+', label='Error')
plt.legend()
plt.show()


plt.title("W2 Convergence over iteration")
plt.xlabel('iter')
plt.ylabel('w2 values')
plt.plot(w2_1_list, 'o', label='W2_1 Convergence')
plt.plot(w2_2_list, 'o', label='W2_2 Convergence')
plt.plot(w2_3_list, 'o', label='W2_3 Convergence')
plt.plot(w2_4_list, 'o', label='W2_4 Convergence')
plt.plot(w2_5_list, 'o', label='W2_5 Convergence')
plt.plot(w2_6_list, 'o', label='W2_6 Convergence')
plt.plot(w2_7_list, 'o', label='W2_7 Convergence')
plt.plot(w2_8_list, 'o', label='W2_8 Convergence')
plt.plot(w2_9_list, 'o', label='W2_9 Convergence')
plt.plot(w2_10_list, 'o', label='W2_10 Convergence')
plt.plot(w2_11_list, 'o', label='W2_11 Convergence')
plt.plot(w2_12_list, 'o', label='W2_12 Convergence')
plt.legend()
plt.show()
```

Output:

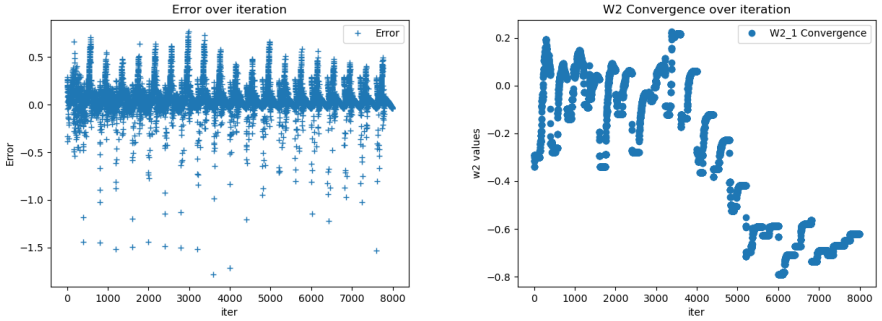
- Για θ = 0.15:
  error(0.005) = -1.0213
  error(0.1) = -0.8935

- Για θ = 0.25:
  error(0.005) = -0.6008
  error(0.1) = -0.5374

- Για θ = 0.35:
  error(0.005) = -0.5011

error(0.1) = -0.4692

Παρατηρούμε ότι όσο μεγαλώνει το θ παίρνουμε καλύτερα αποτελέσματα, ωστόσο όταν τα συγκρίνουμε αυτά με την αντίστοιχη υλοποίηση χωρίς dropout είναι αρκετά πιο χαμηλά. Για την περίπτωση που το θ=0.35 παρατίθενται τα παρακάτω διαγράμματά:



Το error ταλαντώνεται μεταξύ -0.5 και 0.5 χωρίς να υπάρχει σταθερή μείωση, συνεπώς θα μπορούσαμε να πούμε ότι το δίκτυο με τη χρήση dropout δεν παράγει καλύτερα αποτελέσματα και ότι η σύγκλιση των παραμέτρων δεν είναι ομαλή (NOTE: Στο διάγραμμα σύγκλισης παρουσιάζουμε μόνο ένα βάρος)

6 Problem-06

A.

Αρχικά ορίζουμε το performance index, θα χρησιμοποιήσουμε το MSE:

$$F(x) = (t-a)^2 = e^2$$

Για την ενημέρωση των weigths και του bias θα χρησιμοποιήσουμε τον αλγόριθμο steepest descent:

$$\Delta w_i = -\alpha \frac{\partial}{\partial w_i} F(x)$$

$$\Delta b = -\alpha \frac{\partial}{\partial b} F(x)$$

Οι μερικές παράγωγοι ορίζονται ως εξής:

$$\frac{\partial}{\partial w_i} F(x) = \frac{\partial}{\partial w_i} (a-t)^2 = 2(t-a)\left\{-\frac{\partial a}{\partial w_i}\right\}$$

$$\frac{\partial}{\partial b}F(x) = \frac{\partial}{\partial b}(a-t)^2 = 2(t-a)\left\{-\frac{\partial a}{\partial b}\right\}$$

Συνεπώς πρέπει να υπολογίσουμε τις μερικές παραγώγους:

$$\frac{\partial a}{\partial w_i}, \frac{\partial a}{\partial b}$$

Η εξίσωση του δικτύου είναι η εξής:

$$a = tanh(w_1 \cdot p_1 + w_2 \cdot p_2 + w_{1,2} \cdot p_1 \cdot p_2 + b)$$

Βρίσκουμε τις μερικές παραγώγους

$$\frac{\partial a}{\partial w_1} = (1 - tanh^2(a)(p_1)$$

$$\frac{\partial a}{\partial w_2} = (1 - tanh^2(a)(p_2)$$

$$\frac{\partial a}{\partial w_{1,2}} = (1 - tanh^2(a)(p_1 \cdot p_2)$$

$$\frac{\partial a}{\partial b} = (1 - tanh^2(a))$$

Αντικαθιστούμε τις μερικές της α προς τα βάρη και bias στις μερικές της συνάρτησης σφάλματος

$$\frac{\partial F(x)}{\partial w_1} = -2(t-a)(1-tanh^2(a)(p_1) => \Delta w_1 = -\alpha(-2)(t-a)(1-tanh^2(a)(p_1)$$

$$\frac{\partial F(x)}{\partial w_2} = -2(t-a)(1-tanh^2(a)(p_2)\Delta w_2 = -\alpha(-2)(t-a)(1-tanh^2(a)(p_2)$$

$$\frac{\partial F(x)}{\partial w_{1,2}} = -2(t-a)(1-tanh^2(a)(p_1 \cdot p_2)\Delta w_{1,2} = -\alpha(-2)(t-a)(1-tanh^2(a)(p_1 \cdot p_2)$$

$$\frac{\partial F(x)}{\partial b} = -2(t-a)(1-tanh^2(a))\Delta b = -\alpha(-2)(t-a)(1-tanh^2(a)$$

Άρα η αντικατάσταση των παραμέτρων του δικτύου θα είναι

$$w_{1_{new}} = w_{1_{old}} + \Delta w_1 => w_{1_{new}} = w_{1_{old}} + 2\alpha(t-a)(1-tanh^2(a)(p_1)$$

$$w_{2_{new}} = w_{2_{old}} + \Delta w_2 => w_{2_{new}} = w_{2_{old}} + 2\alpha(t-a)(1-tanh^2(a)(p_2)$$

$$w_{1,2_{new}} = w_{1,2_{old}} + \Delta w_{1,2} => w_{1,2_{new}} = w_{1,2_{old}} + 2\alpha(t-a)(1-tanh^2(a)(p_1 \cdot p2)$$

$$b_{new} = b_{old} + \Delta b => b_{new} = b_{old} + 2\alpha(t-a)(1-tanh^2(a)$$

Β.

```
import numpy as np
# initial conditions of network

w1 = 1
w2 = -1
w12 = 0.5
b = 1
p1 = 0
p2 = 1
t = 0.75
# learning rate
learning_rate = 1

# feed forward phase

n = w1 * p1 + w2 * p2 + w12 * p1 * p2 + b
a = np.tanh(n)
# compute the error

error = (t - a)**2

# update the parameters (weights and bias) using the learning rule we have presented

w1 = w1 + 2 * learning_rate * (t - a) * (1 - np.tanh(a)**2) * p1
w2 = w2 + 2 * learning_rate * (t - a) * (1 - np.tanh(a)**2) * p2
w12 = w12 + 2 * learning_rate * (t - a) * (1 - np.tanh(a)**2) *p1 * p2
b = b + 2 * learning_rate * (t - a) * (1 - np.tanh(a)**2)
```

w1 new = 1.0 w2 new = 0.5 w1,2 new = 0.5 b new = 2.5

7 Problem-07

To MLP με ένα hidden layer και activation function την ReLu $\sigma(x) = \max(0, x)$:

$$\mathbf{H} = \sigma(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}),$$
$$\mathbf{O} = \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}.$$

οπου έχουμε

$$\mathbf{O} = \begin{cases} \mathbf{b}^{(2)}, & \mathbf{H} \leq 0 \\ \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}, & \mathbf{H} > 0 \end{cases}$$

To output ειναι piecewise linear αφου το H ειναι piecewise linear.
με περισσότερα layers δεν αλλάζει κάτι αφού θα έχουμε

$$\mathbf{H}^{(1)} = \sigma_1(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}) \text{ και } \mathbf{H}^{(2)} = \sigma_2(\mathbf{H}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)})$$

Το output ειναι piecewise linear αφού το $\mathbf{H}^{(2)}$ είναι piecewise linear.

Γενικά ένα MLP που χρησιμοποιεί μόνο συναρτήσεις ενεργοποίησης ReLU κατασκευάζει μια piece wise linear συνάρτηση επειδή οι πράξεις που εκτελούνται από αυτές τις συναρτήσεις ενεργοποίησης είναι γραμμικές και η σύνθεση αυτών των πράξεων σε όλα τα επίπεδα του MLP έχει ως αποτέλεσμα μια continuous piecewise linear συνάρτηση.

8 Problem-08

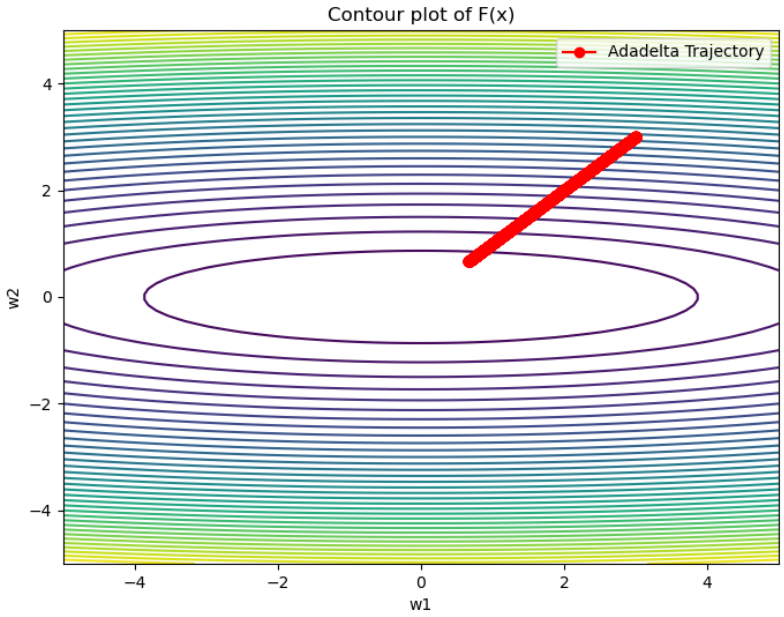
A.

```python
import numpy as np
import matplotlib.pyplot as plt

# Define the function F
def F(w):
    return 0.1*w[0]**2 + 2*w[1]**2

# Define the gradient of F
def grad_F(w):
    return np.array([0.2*w[0], 4*w[1]])

# Adadelta optimizer
def adadelta_optimizer(initial_w, iterations=10, rho=0.99, epsilon=1e-8, a=1):
    w = initial_w
    delta_w_sq = np.zeros_like(w)
    acc_delta_w_sq = np.zeros_like(w)
    trajectory = [w.copy()]

    for _ in range(iterations):
        gradient = grad_F(w)
        acc_delta_w_sq = rho * acc_delta_w_sq + (1 - rho) * gradient**2
        rms_delta_w = np.sqrt(acc_delta_w_sq + epsilon)
        delta_w = -np.sqrt(delta_w_sq + epsilon) / rms_delta_w * gradient
        w += a * delta_w
        delta_w_sq = rho * delta_w_sq + (1 - rho) * delta_w**2
```

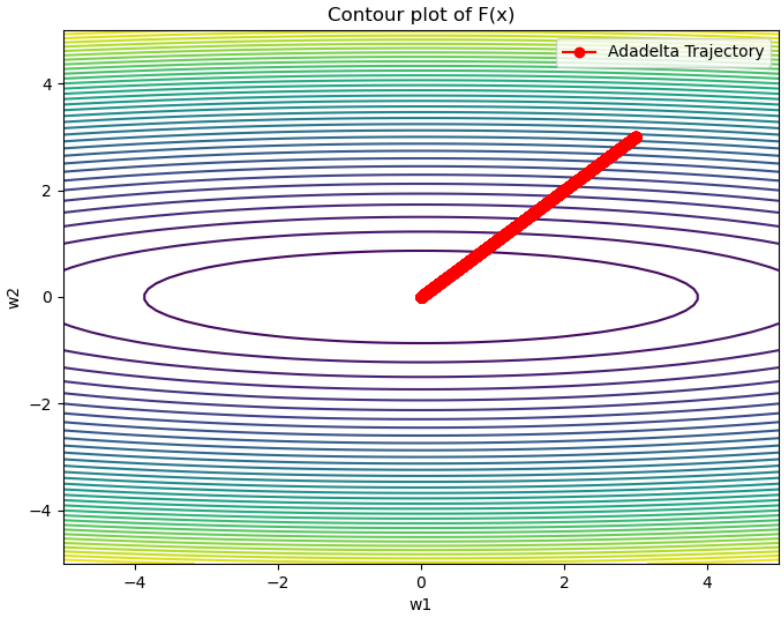
```
        trajectory.append(w.copy())

    return np.array(trajectory)

# Plotting
def plot_contour():
    x = np.linspace(-5, 5, 100)
    y = np.linspace(-5, 5, 100)
    X, Y = np.meshgrid(x, y)
    Z = F([X, Y])

    plt.figure(figsize=(8, 6))
    plt.contour(X, Y, Z, levels=50, cmap='viridis')
    plt.title("Contour plot of F(x)")
    plt.xlabel("w1")
    plt.ylabel("w2")

# Run Adadelta optimizer and plot trajectory
a = 0.3
initial_weights = np.array([3.0, 3.0])
trajectory = adadelta_optimizer(initial_weights, iterations=100000, a=a)

plot_contour()
plt.plot(trajectory[:, 0], trajectory[:, 1], marker='o', color='red',
                         label='Adadelta Trajectory')
plt.legend()
plt.show()
```

Για learning rate = 0.3

Χρησιμοποιώντας 10000 iterations:



Χρησιμοποιώντας 100000 iterations:

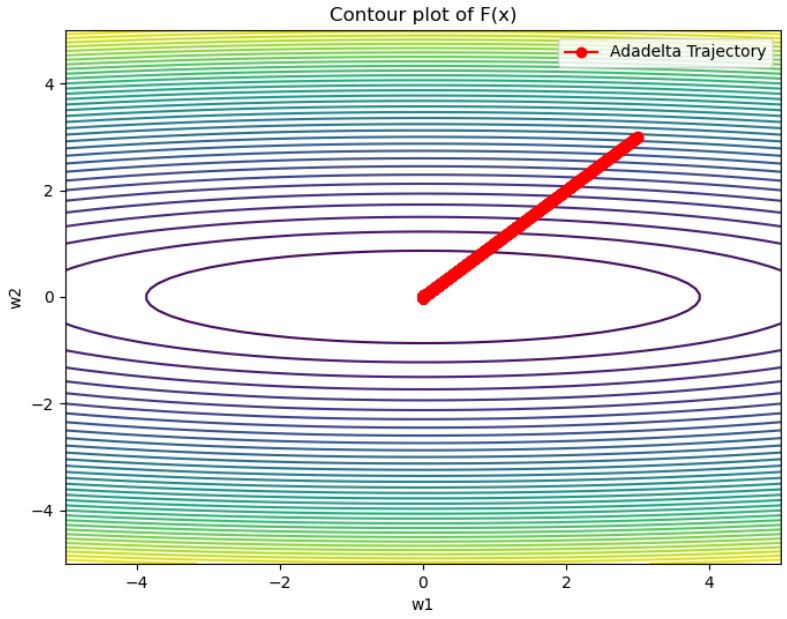


B.

Για learning rate = 3

Χρησιμοποιώντας 10000 iterations:

Παρατηρούμε ότι για μεγαλύτερο learning rate έχουμε γρηγορότερη σύγκλιση.



C.

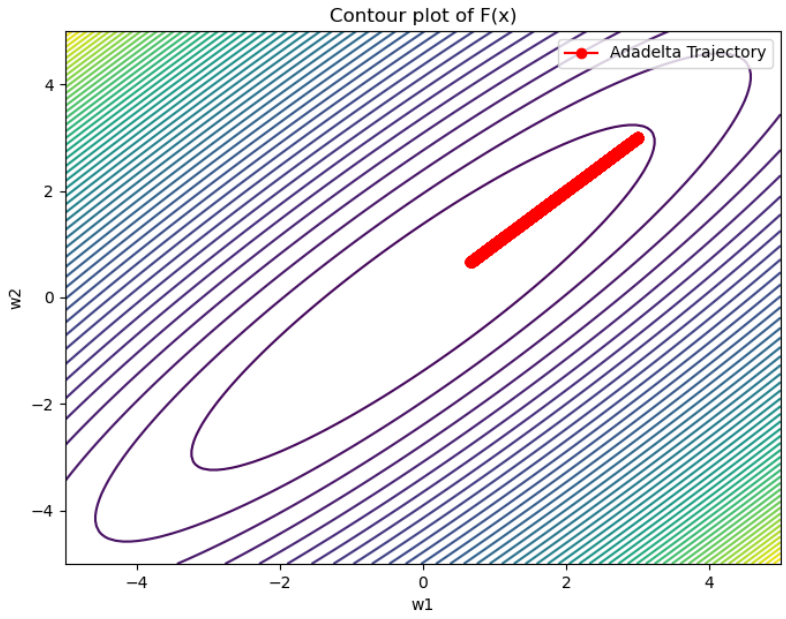
```
import numpy as np
import matplotlib.pyplot as plt

def F_45(w):
    return 0.1*(w[0]+w[1])**2 + 2*(w[0]-w[1])**2

def grad_F_45(w):
    return np.array([4.2*w[0]-3.8*w[1], 4.2*w[1]-3.8*w[0]])
```

```
# Adadelta optimizer
def adadelta_optimizer(initial_w, iterations=10, rho=0.99, epsilon=1e-8, a=1):
    w = initial_w
    delta_w_sq = np.zeros_like(w)
    acc_delta_w_sq = np.zeros_like(w)
    trajectory = [w.copy()]

    for _ in range(iterations):
        gradient = grad_F_45(w)
        acc_delta_w_sq = rho * acc_delta_w_sq + (1 - rho) * gradient**2
        rms_delta_w = np.sqrt(acc_delta_w_sq + epsilon)
        delta_w = -np.sqrt(delta_w_sq + epsilon) / rms_delta_w * gradient
        w += a * delta_w
        delta_w_sq = rho * delta_w_sq + (1 - rho) * delta_w**2

        trajectory.append(w.copy())

    return np.array(trajectory)

# Plotting
def plot_contour():
    x = np.linspace(-5, 5, 100)
    y = np.linspace(-5, 5, 100)
    X, Y = np.meshgrid(x, y)
    Z = F_45([X, Y])

    plt.figure(figsize=(8, 6))
    plt.contour(X, Y, Z, levels=50, cmap='viridis')
    plt.title("Contour plot of F(x)")
    plt.xlabel("w1")
    plt.ylabel("w2")

# Run Adadelta optimizer and plot trajectory
a = 3
initial_weights = np.array([3.0, 3.0])
trajectory = adadelta_optimizer(initial_weights, iterations=100000, a=a)

plot_contour()
plt.plot(trajectory[:, 0], trajectory[:, 1], marker='o', color='red',
                            label='Adadelta Trajectory')
```

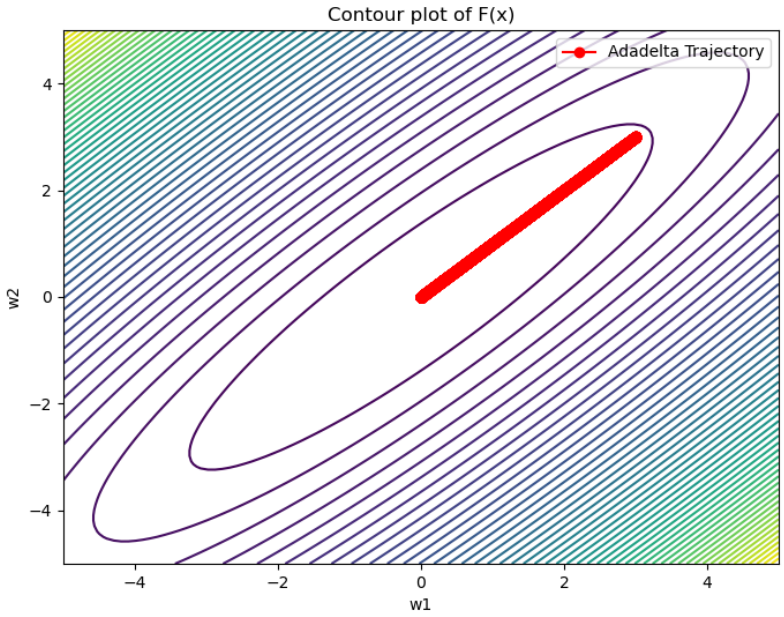
```
plt.legend()
plt.show()
```

Για learning rate = 0.3

Χρησιμοποιώντας 10000 iterations:

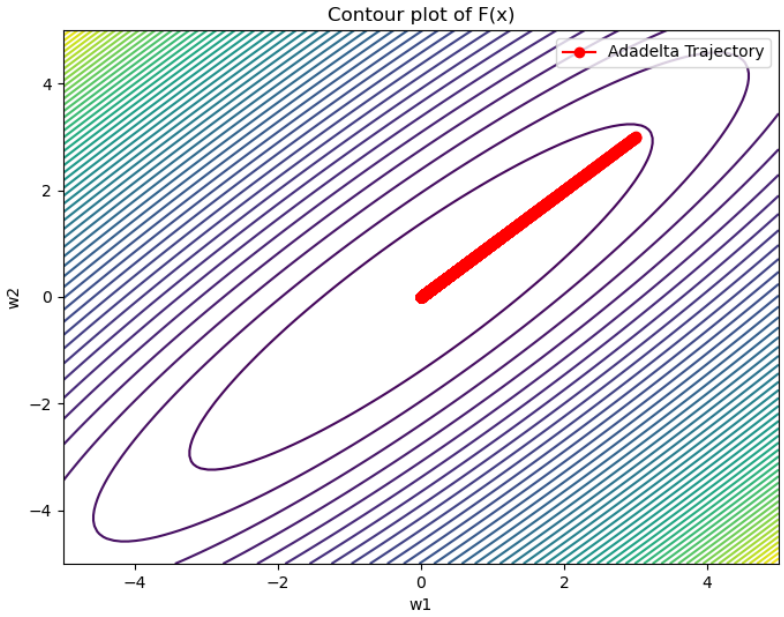


Χρησιμοποιώντας 100000 iterations:



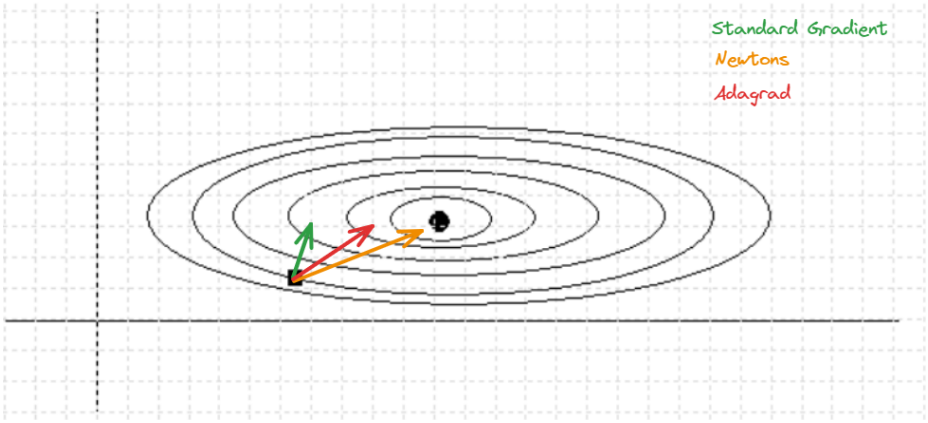
Για learning rate = 3

Χρησιμοποιώντας 10000 iterations:



Συνεπώς παρατηρούμε ότι δεν έχει διαφορά εάν στρέψουμε την συνάρτηση κατά 45°. Δοκιμάζοντας τα ίδια learning rate, στα ίδια iterations και με ίδια αρχική τιμή βλέπουμε ότι η σύγκλιση γίνεται με ίδιο τρόπο.

9 Problem-09



– Το standard gradient θα είναι κάθετο στο contour line
– Από το σχήμα παρατηρούμε ότι η συνάρτηση είναι quadratic ή ένα πολύ καλό quadratic approximation έτσι η μέθοδος Newton's βρίσκει το ελά-

χιστο την συνάρτησης σε ένα step. Αυτό συμβαίνει επειδή η μέθοδος του Newton σχεδιάστηκε για να προσεγγίσει μια συνάρτηση ως τετραγωνική και στη συνέχεια να εντοπίζει το σταθερό σημείο της τετραγωνικής προσέγγισης. Αν η αρχική συνάρτηση είναι τετραγωνική (με ένα ισχυρό ελάχιστο), τότε θα ελαχιστοποιηθεί σε ένα βήμα.

– Η μέθοδος Adagrad παρακολουθεί το άθροισμα των τετραγώνων των gradient και χρησιμοποιεί αυτό το άθροισμα για να προσαρμόσει το gradient σε διάφορες κατευθύνσεις. Έτσι όσο περισσότερο έχει ενημερωθεί ήδη ένα feature, τόσο λιγότερο θα ενημερωθεί στο μέλλον, προσφέροντας έτσι μια ευκαιρία στα πιο sparse features. Γεωμετρικά η Adagrad θα πάει πιο άμεσα στο ελάχιστο από ότι η standard gradient που θα πάρει την μεγαλύτερη κλίση προς τα πάνω και θα πάει μετά δεξιά.

10 Problem-10

$$g_{t+1} \leftarrow bg_t + (1-b)\nabla \hat{L}_t(\theta_t)$$

$$\theta_{t+1} \leftarrow \theta_t - a\left[(1-v)\nabla \hat{L}_t(\theta_t) + vg_{t+1}\right]$$

Για v = 1 εχουμε SGD with momentum:

$$g_{t+1} \leftarrow bg_t + (1-b)\nabla \hat{L}_t(\theta_t)$$

$$\theta_{t+1} \leftarrow \theta_t - a\left[g_{t+1}\right]$$

Για v = 0 εχουμε standard gradient descent:

$$g_{t+1} \leftarrow bg_t + (1-b)\nabla \hat{L}_t(\theta_t)$$

$$\theta_{t+1} \leftarrow \theta_t - a\nabla \hat{L}_t(\theta_t)$$

11 Problem-11

```python
import numpy as np

def convolution(input_array, kernel_array, stride=(1, 1)):
    input_height, input_width = input_array.shape
    kernel_height, kernel_width = kernel_array.shape
    stride_height, stride_width = stride

    # Calculate output dims
    output_height = int((input_height - kernel_height) / stride_height) + 1
    output_width = int((input_width - kernel_width) / stride_width) + 1
```

```
    # Initialize an array to store the result of the convolution
    output_array = np.zeros((output_height, output_width))

    # Iterate through the input array with the specified stride and perform the convolution
    for i in range(0, output_height * stride_height, stride_height):
        for j in range(0, output_width * stride_width, stride_width):
            # Compute the convolution at the current position
            output_array[i // stride_height, j // stride_width] = np.sum(
                input_array[i:i+kernel_height, j:j+kernel_width] * kernel_array
            )

    return output_array

def max_pooling(input_array, window_size, stride):
    input_shape = input_array.shape
    pool_size = ((input_shape[0] - window_size[0]) // stride[0]) + 1

    # Initialize output
    output_array = np.zeros((pool_size, pool_size))

    # Iterate through the input array with the specified stride and perform max pooling
    for i in range(pool_size):
        for j in range(pool_size):
            # Extract the window at the current position
            window = input_array[i * stride[0]:i * stride[0] + window_size[0],
                                 j * stride[1]:j * stride[1] + window_size[1]]
            # Find the maximum value in the window and store it in the output array
            output_array[i, j] = np.max(window)

    return output_array

input = np.array([[20, 35, 35, 35, 35, 20],
                  [29, 46, 44, 42, 42, 27],
                  [16, 25, 21, 19, 19, 12],
                  [66, 120, 116, 154, 114, 62],
                  [74, 216, 174, 252, 172, 112],
                  [70, 210, 174, 252, 172, 112]])

kernel = np.array([[1, 1, 1],
                   [1, 0, 1],
```

```
                [1, 1, 1]])

stride = (1, 1)

output = convolution(input, kernel, stride)
print("Input Array:")
print(input)
print("\nKernel Array:")
print(kernel)
print("\nOutput Array after Convolution:")
print(output)

window_size = (2, 2)
stride = (2, 1)

pooling_output = max_pooling(output, window_size, stride)
print("\nMaxPooling Array:")
print(pooling_output)
```

A.

Από το convolution παίρνουμε:

$$\begin{pmatrix} 225 & 258 & 250 & 209 \\ 458 & 566 & 552 & 472 \\ 708 & 981 & 887 & 802 \\ 1004 & 1494 & 1328 & 1230 \end{pmatrix}$$

B.

Από το maxpooling παίρνουμε:

$$\begin{pmatrix} 566 & 566 \\ 1494 & 1494 \end{pmatrix}$$

C.

Ο πρώτος kernel φαίνεται να έχει ένα μοτίβο όπου υπάρχει μια κεντρική περιοχή με υψηλότερη ένταση που περιβάλλεται από μια περιοχή χαμηλότερης έντασης. Αυτός ο τύπος πυρήνα μπορεί να χρησιμοποιηθεί για horizontal edge detection.

Ο kernel μοιάζει με ένα Laplace of Gaussian kernel.Τα Laplacian filters είναι derivative filters used to find areas of rapid change (edges) in images. Οπότε το kernel κάνει edge detection.

Το τρίτο kenel Θα μπορούσε να είναι ένα διαγώνιο sobel φίλτρο για edge detection

12 Problem-12

Σε ένα multi channel convolution έχουμε weights $= c_{\mathrm{o}} \times c_{\mathrm{i}} \times k_{\mathrm{h}} \times k_{\mathrm{w}}$ και ο αριθμός τον biases εξαρταται απο το πληθος των output channels άρα:

- Layer 1: $c_{\mathrm{i}} = 3$ απο την εισοδο, $c_{\mathrm{o}} = 4$, $k_{\mathrm{h}} = k_{\mathrm{w}} = 3$.
  Άρα έχουμε $3 \times 4 \times 3 \times 3 = 108$ weights και 4 biases
- Layer 2: $c_{\mathrm{i}} = 4$ απο το layer 1, $c_{\mathrm{o}} = 10$,, $k_{\mathrm{h}} = k_{\mathrm{w}} = 5$.
  Άρα έχουμε $4 \times 10 \times 5 \times 5 = 1000$ weights και 10 biases

13 Problem-13

A.

Θέλουμε να εκφράσουμε το max(a. b) με την relu. Έχουμε

$$ReLu(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$

και

$$max(a,b) = \begin{cases} b, & a < b \\ a, & a \geq b \end{cases} \iff max(a,b) = \begin{cases} b, & a - b < 0 \\ a, & a - b \geq 0 \end{cases}$$

Άρα δοκιμάζουμε για x = a - b :

$$ReLu(a-b) = \begin{cases} 0, & x < 0 \\ a - b, & a - b \geq 0 \end{cases}$$

Προσθέτουμε και στα δυο μέλη b :

$$ReLu(a-b)+b = \begin{cases} 0 + b, & a - b < 0 \\ a - b + b, & a - b \geq 0 \end{cases} \iff ReLu(a-b)+b = \begin{cases} b, & a < b \\ a, & a \geq b \end{cases} = max(a,b)$$

B.

Η λογική είναι να κάνουμε convolution οπού θα παίρνουμε κάθε φορά ένα στοιχείο από το κάθε patch έπειτα θα χρησιμοποιήσουμε την relu όπως δείξαμε παραπάνω για να βρούμε το max του output και του αποτελέσματος του convolution έπειτα κάνουμε convolution για να πάρουμε τα επόμενα στοιχεία π.χ.

Έστω ότι έχουμε ένα πίνακα 4x4 και θέλουμε να κάνουμε maxpooling με window 2x2 αρχικοποιούνται την έξοδο σε ένα πίνακα 2x2 με -inf

$$input = \begin{matrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{matrix}, \quad output = \begin{matrix} -inf & -inf \\ -inf & -inf \end{matrix}$$

Κάνουμε convolution με kernel που παίρνει το πρώτο στοιχείο από κάθε patch και stride 2x2 και παίρνουμε

$$kernel = \begin{matrix} 1 & 0 \\ 0 & 0 \end{matrix}, \quad \text{convolution output } = \begin{matrix} 1 & 3 \\ 9 & 11 \end{matrix}$$

Έπειτα χρησιμοποιούμε ένα relu layer για να βρούμε το max του convolution output και του output: output = relu(output - convolution_output) + convolution_output οπότε το output γίνεται

$$output = \begin{matrix} 1 & 3 \\ 9 & 11 \end{matrix}$$

Έπειτα κάνουμε convolution με τον επόμενο kernel

$$kernel = \begin{matrix} 0 & 1 \\ 0 & 0 \end{matrix}, \quad \text{convolution output } = \begin{matrix} 2 & 4 \\ 10 & 12 \end{matrix}$$

Relu για να πάρουμε το max από το output και το convolution_output και το αποθηκεύουμε στο output. Επαναλαμβάνουμε την παραπάνω διαδικασία άλλες δυο φορές με kernels:

$$kernel = \begin{matrix} 0 & 0 \\ 1 & 0 \end{matrix} \quad \text{και } kernel = \begin{matrix} 0 & 0 \\ 0 & 1 \end{matrix}$$

το τελικό αποτέλεσμα είναι ισοδύναμο με ένα max-pooling layer. Συνοπτικός κώδικάς:

```
function custom_max_pooling(input_tensor, pool_size):
    # Get the dimensions of the input tensor
    batch_size, channels, height, width = input_tensor.size()

    # Print the number of channels
    print(channels)

    # Define kernel size and stride for pooling
    kernel_size = (pool_size, pool_size)
    stride = (pool_size, pool_size)
```

```
    # Calculate the output shape after convolution
    output_shape = convolution_output_shape(input_tensor, channels, pool_size)

    # Initialize output tensor with negative infinity values
    output_tensor = initialize_output_tensor(output_shape)

    # Iterate over each element of the pooling kernel
    for i in range(pool_size * pool_size):
        # Create a kernel with all zeros except for the i-th element
        kernel = create_pooling_kernel(pool_size, i)

        # Apply convolution with the created kernel
        temp = apply_convolution(input_tensor, kernel, stride)

        # Update the output tensor using max pooling with ReLU
        output_tensor = update_output_tensor(output_tensor, temp)

    # Return the final output tensor
    return output_tensor
```

Ουσιαστικά χρειαζόμαστε όσα layers όσα ο αριθμός των στοιχείων του pooling window κάθε φορά χρησιμοποιούμε ένα kernel που έχει έναν άσο στη ith θέση και στις υπόλοιπες μηδέν και κάνουμε ανά δυο συγκρίσεις χρησιμοποιώντας την relu

14 Problem-14

- LAYER 1: Convolutional layer with 100 5x5 convolutional filters.
  100 * (5 * 5) = 2,500
- LAYER 2: Convolutional layer with 100 5x5 convolutional filters.
  100 * (5 * 5 * 100) = 250,000
  The filters at this layer are 5 x 5 x 100, since they need to be applied across all of the filter results from the previous layer.
- LAYER 4: Dense layer with 100 units
  (50 * 50 * 100) * 100 = 25,000,000
  There are (100*100*100) output values from Layer 2, this becomes (50 * 50 * 100) after the downsampling in Layer 3. Each of those units is connected to each of the 100 Layer 4 units.
- LAYER 5: Dense layer with 100 units
  100 * 100 = 10,000
- LAYER 6: Single output unit
  100

Total: 2,500 + 250,000 + 25,000,000 + 10,000 + 100 = 25,262,600

15 Problem-15

A.

Data Reuse and Memory Access Efficiency: Μεγαλύτερες τιμές του Δ επιτρέπουν καλύτερη εκμετάλλευση των δεδομένων που έχουν ήδη φορτωθεί στη μνήμη ή την κρυφή μνήμη. Κατά την επεξεργασία μιας λωρίδας πλάτους k, μπορεί να χρειαστεί να φορτώσετε τα ίδια δεδομένα πολλές φορές για συνεχόμενες τιμές εξόδου. Σε αντίθεση, με μια πιο ευρεία λωρίδα, μπορείτε να επαναχρησιμοποιήσει τα φορτωμένα δεδομένα για πολλαπλές υπολογιστικές εξόδους, βελτιώνοντας την αποτελεσματικότητα της πρόσβασης στη μνήμη.

Ωστόσο, υπάρχει όριο στο πόσο μεγάλο θα πρέπει να είναι το Δ, και αυτό εξαρτάται από διάφορους παράγοντες, συμπεριλαμβανομένων της διαθέσιμης μνήμης, του μεγέθους της κρυφής μνήμης το μέγεθος kernel και τα λοιπά.