ECE/CS 559 - Fall 2017 - Homework #3
Due: 10/06/2017, the end of class.

Erdem Koyuncu

Note: All notes in the beginning of Homework #1 apply. Only a subset of the problems may be graded.

1. **Theorem:** *Let X be a real $m \times q$ matrix. There exists a unique $q \times m$ matrix X^+ with the following properties: (i) $XX^+X = X$, (ii) $X^+XX^+ = X^+$, and (iii) X^+X and XX^+ are symmetric matrices. The matrix X^+ is called the pseudo-inverse, or the Moore-Penrose pseudo-inverse of X.*

   Show that if X has linearly independent columns, then X^TX is invertible, and $X^+ = (X^TX)^{-1}X^T$. You may use the theorem above.

   Show that if X has linearly independent rows, then XX^T is invertible, and $X^+ = X^T(XX^T)^{-1}$. You may use the theorem above.

2. In this computer experiment, we will implement the gradient descent method and Newton's method. Let $f(x, y) = -\log(1 - x - y) - \log x - \log y$ with domain $\mathcal{D} = \{(x, y) : x + y < 1,\ x > 0,\ y > 0\}$.

   (a) Find the gradient and the Hessian of f on paper.

   (b) Begin with an initial point in $w_0 \in \mathcal{D}$ with $\eta = 1$ and estimate the global minimum of f using the Gradient descent method, which will provide you with points $w_1, w_2, \ldots,$. Report your initial point w_0 and η of your choice. Draw a graph that shows the trajectory followed by the points at each iteration. Also, plot the energies $f(w_0), f(w_1), \ldots$, achieved by the points at each iteration. Note: During the iterations, your point may "jump" out of $\mathcal{D}$ where f is undefined. If that happens, change your initial starting point and/or η.

   (c) Repeat part (b) using Newton's method.

   (d) Compare the speed of convergence of gradient descent and Newton's method, i.e. how fast does each method approach the estimated global minimum?

3. Perform the following steps: **Steps (e)-(g) are optional and will not be graded, although I highly recommend you do them**.

   (a) Let $x_i = i$, $i = 1, \ldots, 50$.

   (b) Let $y_i = i + u_i$, $i = 1, \ldots, 50$, where each u_i should be chosen to be an arbitrary real number between -1 and 1.

   (c) Find the linear least squares fit to (x_i, y_i), $i = 1, \ldots, 50$. Note that the linear least squares fit is the line $y = w_0 + w_1x$, where w_0 and w_1 should be chosen to minimize $\sum_{i=1}^{50}(y_i - (w_0 + w_1x_i))^2$.

   (d) Plot the points (x_i, y_i), $i = 1, \ldots, 50$ together with their linear least squares fit.

   (e) Find (on paper) the gradient of $\sum_{i=1}^{50}(y_i - (w_0 + w_1x_i))^2$ (derivatives with respect to w_0 and w_1).

   (f) (Re)find the linear least squares fit using the gradient descent algorithm. Compare with (c).

   (g) Show (on paper) that a single iteration of Newton's method with $\eta = 1$ provides the globally optimal solution (the solution in (c)) regardless of the initial point.

OLIVO IACOPO - HW 3

q7

1) a) A matrix is invertible iff

- Square
- $\det(A) \neq 0$

$\Rightarrow X \in \mathbb{R}^{n \times q}$, $X^T \in \mathbb{R}^{q \times n}$ $\Rightarrow X^T X \in \mathbb{R}^{q \times q}$ $\Rightarrow$ Square

- X has l.i. columns $\iff N(X) = \{\bar{0}\}$

$\bar{a} \in N(X^T X)$

$\Rightarrow X^T X \bar{a} = \bar{0}$

$\bar{a}^T X^T X \bar{a} = \bar{0}^T \bar{0} = 0$

$(X\bar{a})^T X\bar{a} = 0$

$\Rightarrow X\bar{a} \cdot X\bar{a} = 0$

$\Rightarrow \|X\bar{a}\|^2 = 0$

$\Rightarrow X\bar{a} = \bar{0}$ hence $\bar{a} \in N(X)$

$\Rightarrow N(X^T X) = N(X) = \{\bar{0}\}$

$\Rightarrow X^T X$ has l.i. columns.

Every l.i. column matrix can be reduced to I.

$\det(I) = 1 \Rightarrow \det(X^T X) \neq 0 \Rightarrow X^T X$ invertible

Moreover:

$X = XX^+X$ $\quad$ $X = XX^+X$

$X^T = (X X^T X)^T$ $\quad$ $X^T = (X X^+ X)^T$

$X^T = (X^+ X)^T X^T$ $\quad$ $X^T = X^T (X X^+)^T$

$X^T = X^+ X X^T$ $\quad$ $X^T = X^T X X^+$

$X^T (X X^T)^{-1} = X^+$ $\quad$ $(X^T X)^{-1} X^T = X^+$

b) Similarly as a)

- $X \in R^{m \times q}$, $X^T \in R^{q \times m} \Rightarrow XX^T \in R^{m \times m}$ Square

- X^T les l.i. columns $N(X^T) = \{\bar{0}\}$

$\bar{0} \in N(XX^T)$

$XX^T \bar{0} = \bar{0}$

$\bar{0}^T XX^T \bar{0} = \bar{0}^T \bar{0} = 0$

$(X^T \bar{0})^T X^T \bar{0} = 0$

$\Rightarrow X^T \bar{0} \cdot X^T \bar{0} = 0$

$\Rightarrow \| X^T \bar{0} \|^2 = 0$

$\Rightarrow X^T \bar{0} = \bar{0} \quad \Rightarrow 0 \in N(X^T)$

$\Rightarrow N(XX^T) = N(X^T) = \{\bar{0}\}$

$\Rightarrow XX^T$ l.i. columns $\Rightarrow$ invertible

Moreover

$X \; XX^+X$

$X^T = (XX^+X)^T$

$X^T = (X^+X)^T X^T$

$X^T = X^+X \; X^T$

$X^T (XX^T)^{-1} = X^+$

3) $f(x,y) = -\log(1-x-y) - \log(x) - \log(y)$

$$\frac{\partial f}{\partial x} = \frac{-1}{x+y-1} - \frac{1}{x} \quad \checkmark$$

$$\frac{\partial^2 f}{\partial x^2} = \frac{1}{(x+y-1)^2} + \frac{1}{x^2} \quad \checkmark$$

$$\frac{\partial f}{\partial y} = \frac{-1}{x+y-1} - \frac{1}{y} \quad \checkmark$$

$$\frac{\partial^2 f}{\partial y^2} = \frac{1}{(x+y-1)^2} + \frac{1}{y^2} \quad \checkmark$$

$$\frac{\partial^2 f}{\partial x \partial y} = \frac{\partial^2 f}{\partial y \partial x} = \frac{1}{(x+y-1)^2} \quad \checkmark$$

$$\Rightarrow \nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

Olivo Iacopo – Neural Networks HW3

Question 2:

In order to solve the problem, I developed a Matlab script made by different functions:

Function:

```
function [z] = func(x)
    z = -log(1-x(1)-x(2)) -log(x(1)) -log(x(2));
end
```

initial weights 2, 2

It is used in order to calculate the energy relative to a coordinate (x,y)

Gradient:

```
function g = gradient(x)

    xp = 1 / (1 -x(1) -x(2)) - 1/x(1);
    yp = 1 / (1 -x(1) -x(2)) - 1/x(2);
    g = [xp, yp];
end
```

Calculates the gradient relative to a coordinate (x,y)

Hessian:

```
function h = hessian (x)
    xx = 1 / ((1 -x(1) -x(2))^2) + 1/(x(1)^2);
    xy = 1 / ((1 -x(1) -x(2))^2);
    yx = 1 / ((1 -x(1) -x(2))^2);
    yy = 1 / ((1 -x(1) -x(2))^2) + 1 / (x(2)^2);

    h = [xx, xy; yx, yy];
end
```

Calculate the Hessian matrix relative to a coordinate(x,y)

Gradient Descent:

```
function s = gdescent(w, eta)
% s = [x, y, z]
    W = w;
    s = [W, func(W)];
    epsilon = 0.001;
   while (norm(gradient(W)) > epsilon)
        g = gradient(W);
        W = W - eta * g;
        s = [s ; W, func(W)];
    end
end
```

Implements the gradient descent method. In order to stop the loop, we use the norm of the gradient. We know that in the given function the local minimum will have slope=0 for each partial derivative, hence we use the norm in order to get the magnitude. Epsilon is chosen after a series of tries.

Newton Method:

```
function s = nmethod(w, eta)
% s = [x, y, z]
    W = w;
    s = [W, func(W)];
    epsilon = 0.001;
  while (norm(gradient(W)) > epsilon)
        g = gradient(W);
        h = hessian(W);
        W = W - eta *(h\g')';
        s = [s ; W, func(W)];
    end
end
```

Implements the newton method. It uses the value of the norm as a stopping condition like it is done in the gradient descent algorithm.

Program:

Uses all the previous functions in order to satisfy the exercise requirements.

```
close all;
clear all;
clc;


W0 = [rand(),  rand()];
while ( (W0(1)+W0(2)>= 1 || W0(1)<=0  || W0(2)<=0 ) )
    W0 = [rand(),  rand()];
end

eta  = 0.01;

gradient = gdescent(W0, eta);
newton = nmethod(W0, eta);
```

Results:

Below are reported the produced graphs which shows the position of the points in the space and their relative energies during the various iterations of the algorithm.

Figure 1 and *Figure 2* show the path followed by the points.

We note how the path followed are different between Gradient descent and Newton method.

In fact, we note how the gradient descent path present an inversion of the curvature of the path: it is as first concave, then convex. The Newton methods points follow a convex curve. This difference is due to the application of different rules in the point calculation. The two algorithms need different settings, hence by applying the same values for both of them, they will perform differently.

Figure 3 and *Figure 4* show the change in energies among the points in different iterations (defined as Epoch) of the algorithm. The shape of the two graphs is very similar and the energy decrease monotonically to an asymptote (3.3).

We can generally note how the newton method needs more iterations in order to obtain the same precision as the gradient method (roughly ten times more). This is due to ?

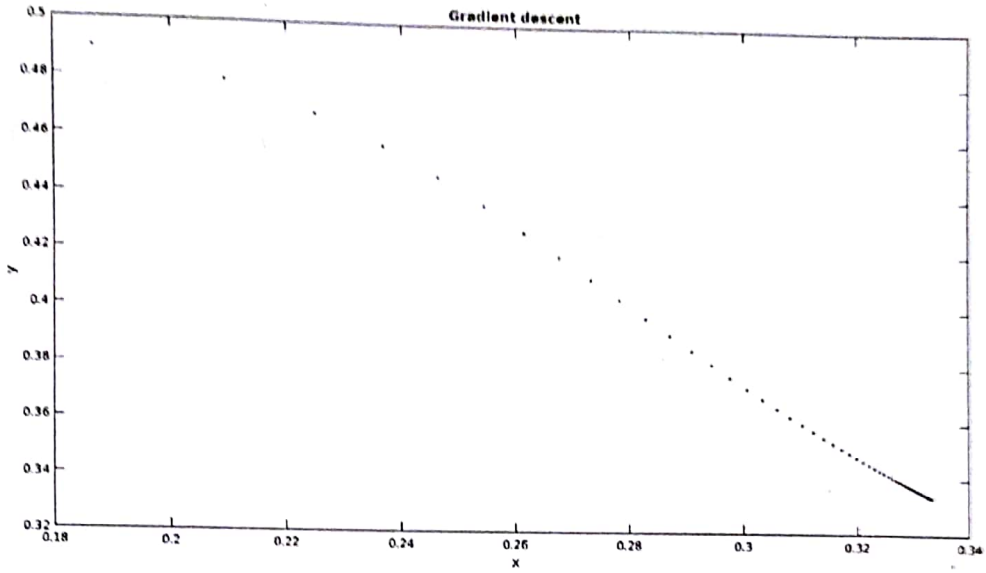

Figure 1: Gradient descent points

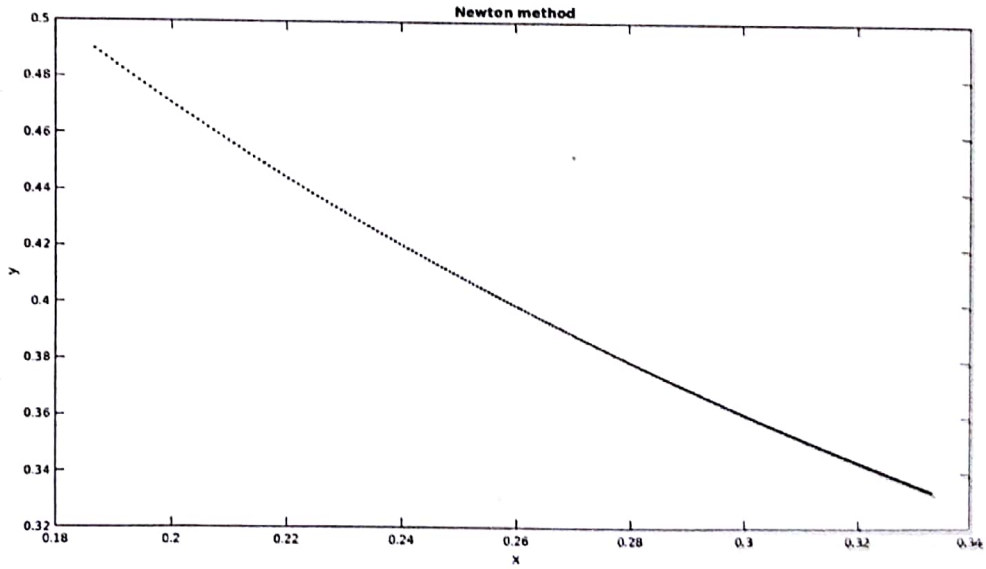

Figure 2: Newton method points

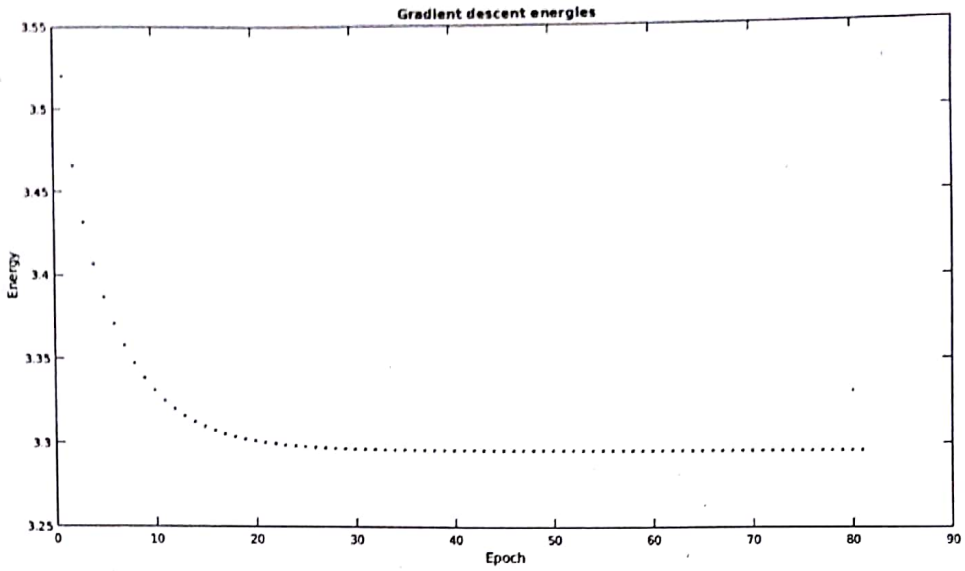

Figure 3: Gradient descent energies

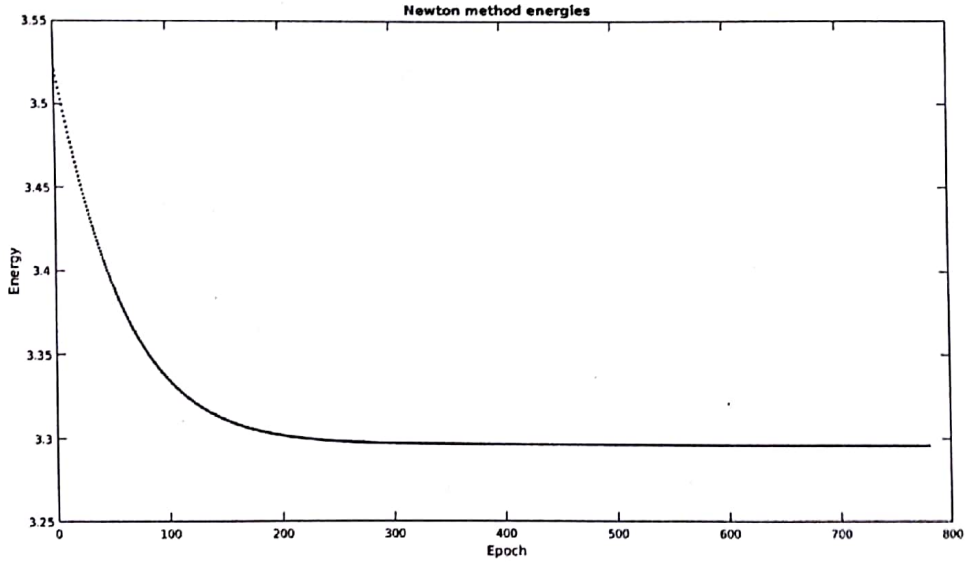

Figure 4: Newton method energies

Question 3:

In order to determine the linear least square fit we can use a Matlab script:

```
xi = (1:1:50)';
yi = (rand(50,1)*2 - 1) + xi;
x = [xi ones(50,1)];

WOptimal = x\yi;

%Try to  get them through gradient descent
W = [rand() rand()];
eta = 0.00001;
Wapprox = gdescent(xi, yi, W, eta);
```

Which uses the following functions:

Gradient

```
function g = gradient(x, y, W)
    w0 = sum(2 .* (W(2) + W(1) .* x - y));
    w1 = sum(2 .* x .* (W(2) + W(1) .* x - y));

    g = [w1 w0];
end
```

Calculate the gradient for a couple of values W0,W1

Func

```
function [z] = func(x, y , W)
    z = sum((y -  (W(2) + W(1).*x)).^2);
end
```

Calculate the value of the function in the given point

Gdescent

```
function W = gdescent(x, y, w, eta)
% s = [x, y, z]
    W = w;

    epsilon = 0.001;
   while (norm(gradient(x, y, W)) > epsilon)
        g = gradient(x, y, W);
        W = W - eta * g;
    end
end
```

Implements the Gradient descent algorithm applied to the given problem.

The script will return us the optimal slope and offset, which in our random case are

```
WOptimal =

    0.9995
   -0.1773
```

In order to calculate the *WOptimal*, we use the function

X\yi

The \ operator is used in Matlab implements the least square of *XWOptimal=yi*

The resulting line is calculated also by applying the gradient descent method, which leads to *Figure 5*. There we can note how the difference between the optimal and the approximated lines is nearly nonexistent.

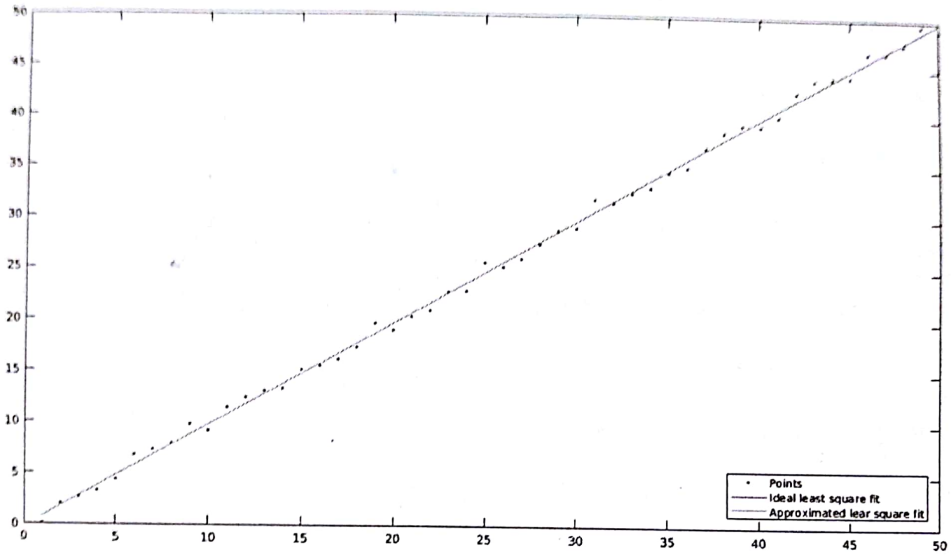

Figure 5: Optimal and Approximated lines