Project "Hessian Matrices and : "
Title
IB3702 Mathematics for Machine Learning

Evertjan Karman John Stegink

15 November, 2025

1 Introduction

The basis of a machine learning algorithm that it tries to predict the right output using a certain input. First the algorithm will have to be trained using correct data. During the training process, the difference between the predicted value and the actual value must be minimized. A cost function is used to quantize the difference, this difference must be minimized. To find the minimum value, the machine learning algorithm iterates until it has found the minimum value. The methods for this iteration are numerous, the most well known algorithm is gradient descent (sections 3.1 and 3.4). For the training to be as effective as possible it is necessary to find the minimum in little iterations. In this report we compare two methods, Gradient descent and Newton's method. For Newton's method the Hessian matrix with second partial derivatives is required.
First a description of gradient descent and Newton's method will be given for functions using one variable, this is to make the principle clear. Normally for machine learning, 1 variable is not sufficient, the loss function mostly contains multiple variables.
Newton's method can find the minimum in far less iterations than Gradient descent. The problem of Newton's method, is that calculating a Hessian Matrix is much more expensive than using just the derivative, as is done with Gradient descent.
Next to this, this report will cover the possibility of SVD in PCA, and the link between EVD and SVD (see abbreviations below).

2 Preliminaries

The notation is similar to the notation of the study guide of IB3702 Mathematics for Machine Learning.
$\frac{\partial^2 f}{\partial x \partial y}$ denotes the second partial derivative of function ($f(x, y)$ with regard to y en then to x.

Notation of Hessian matrix with second partial derivatives:

$$\mathbf{H}_f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Abbreviations

1. EVD Eigenvalue decomposition
2. PCA Principle Component Analysis
3. SVD Singular Value Decomposition

After reading the report, please solve the following problems:
Question 1: Compute the Hessian Matrix for the function $f(x) = x^3 + y^3 + 2xy$ in the point $x = 1, y = 2$.

Answer 1: The Hessian matrix becomes $H = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} = \begin{bmatrix} 6x & 2 \\ 2 & 6y \end{bmatrix}$ The values for $x = 1, y = 2$ are $H = \begin{bmatrix} 12 & 2 \\ 2 & 12 \end{bmatrix}$

Question 2:
Answer 2:

3 Methods

3.1 Gradient descent with one variable

The purpose of using gradient descent is to find a local or global minimum of a differentiable function by iteratively adjusting the parameters in the direction of the steepest descent. This a mouthful.
It can be clarified by the following metaphor: When you want to find the path from the top to the foot of the mountain in the mist. You walk a few meters down the path with the steepest descent. Then you determine the next path with the steepest descent[1]. Appendix A.1 contains a more detailed version of this metaphor.
Assume that we have a continuous function f defined on $\mathbb{R}$ (fig 1).This function is differentiable with derivative $f'(x)$ and has a starting point x_0. Then we get x_1 by subtracting $f'(x_0) \cdot \alpha$ from x_0, where α is called the learning rate. This is equal to walking the path

[1] https://en.wikipedia.org/wiki/Gradient_descent

in the steepest descent by α meters in the metaphore. The value of α will be chosen before starting the procedure. Usual values for α are 0.01 or 0.05.
We iterate this, so that we get a **sequence** which is recursively defined as:

$$x_{k+1} = x_k - f'(x_k) \cdot \alpha$$

This sequence will converge to the minimum of f (fig 1). The pitchfalls here are, that the procedure may end in a local minimum, while f has a stronger minimum elsewhere. Or with a less than optimal choice for the learning rate, the sequence could even diverge.

3.2 Newton's method with one variable

Newton's method finds the zeroes of a function f, in cases where it is more challenging or expensive, to find the minimum directly by solving an equation. It works by iterating through the following steps:

1. Start at a random point on the curve.
2. Determine the tangent line at that point (using the derivative).
3. Determine the point where the tangent line hits the X-axis
4. From the point found in previous step, continue with step, 2 until the value does not change significantly anymore.

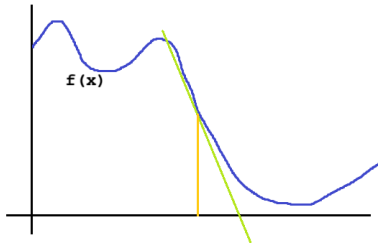

Figure 1: A continuous function f defined on $\mathbb{R}$

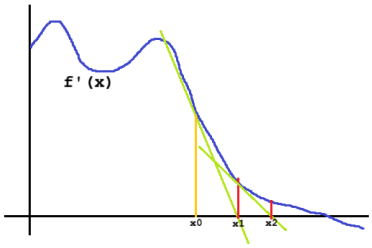

Figure 2: Newton's method

To state it more mathematically: starting from x_0 find the value x_1 where the tangent line intersects with the x-axis. By iterating this procedure we get an sequence $(x_k)_{k=0,1,\ldots}$. From the geometrical aspect of the procedure, we can give a formula between x_{k+1} and x_k. The idea is that the sequence (x_k) converges to the value of x where $f(x) = 0$.
Below formula takes into account that we want to find the *minimum of a function f*, so we are finding a *zero of f'*. So after substituting f' for f and f'' for f':

$$x_{k+1} = x_k - (f'(x_k)/f''(x_k))$$

An example is depicted in (fig 2).
In order to know if $f'(x)$ points to a minimum of f, we need to look at the second derivative $f''(x)$:
$f''(x) > 0 \Rightarrow$ f has a minimum at x
$f''(x) < 0 \Rightarrow$ f has a maximum at x
$f''(x) = 0 \Rightarrow$ inconclusive, perhaps an inflection point

3.3 Functions with two or more variables and their derivatives

A function with one variable describes the result of a calculation with respect to just 1 variable. In practice the result of a function is dependent on more that one variable. For example the price of a house is not only dependent on the floor area of the house, but for example the distance to the nearest shops count, the number of crimes in the area per year etc. This means the function has multiple variables. This is what we see in machine learning most of the time too. An example of a graph of the multi value function can be seen in figure 5. This depicts the function $f(x, y) = 85 - \frac{1}{90}x^2(x-6)y^2(y-6)$. As you can see the function is defined by two variables: $f(x, y)$.
Finding the slope in a point using a function with one variable is easily done by determining the derivative. For multi value functions this is done in only 1 direction at a time. This means the derivatie has te determined with regard to 1 variable the rest of the variables is considered as a constant. [**?**] describes this very well.
Mathematically:

$$\frac{\partial f}{\partial x} = f_x(x, y) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x, y) - f(x, y)}{\Delta x}$$

$$\frac{\partial f}{\partial y} = f_y(x, y) = \lim_{\Delta y \to 0} \frac{f(x, y + \Delta y) - f(x, y)}{\Delta y}$$

3.4 Gradient descent with two or more variables

Considering the metaphore in section 3.1 the x is the number of steps to walk to the west and y is the number of meters to walk to the north, instead of just walking straight ahead. With a function $f(x, y)$ of more variables, we can determine the gradient:

$$\nabla f(x, y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)$$

The method is the same, but where we took the derivative for one variable, we will now take the gradient, and the recursive definition of our sequence (x_k, y_k) becomes:

$$(x_{k+1}, y_{k+1}) = (x_k, y_k) - \nabla f(x, y) \cdot \alpha$$

3.5 Hessian matrix

The second derivative of a function tells something about the curvature of a function in a certain point. A function is convex at a point when the second derivative is positive, and concave if it is negative. When the value of the second derivative is zero, the point may be an inflection point, rather than a minimum or maximum of the function.
A Hessian matrix is a squared matrix containing all second derivatives of a multi valued function. Say we have function $f(x, y)$ of 2 variables. Then we get a square matrix having te following format:

$$H_f(x,y) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

Generally a Hessian Matrix has the form: $\mathbf{H}_f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$

The Hessian matrix is a symmetric matrix most of the times. (Because there is Clairaut's Theorem that states, that $\frac{\partial^2 f}{\partial x \partial y} = \frac{\partial^2 f}{\partial y \partial x}$ under specific conditions which ar usually true when we have smooth functions.)

With Hessian matrix local minima, maxima and stationary points of a function can be found. It helps to identify saddle points, local minima and local maxima. A saddle point (figure 4) is a stationary point that is neither a local maximum nor a local minimum. From this point the function looks like a minimum in some directions and a maximum in other directions. The type of extreme can be found by calculating the eigenvalues of the Hessian matrix (section 3.6).
Hessian matrices can be become quite large. The matrix for a function with n different variables the matrix will have the size of $n \times n$ (in modern machine learning models the value of n can become significantly large). The size of the matrix to be calculated and stored in memory will have a size of order n^2. For such situations approximations of the Hessian Matrix are being used. An often used algorithm that uses an approximation is BFGS.

3.6 Newton's method with two or more variables

The main principles of Newton's method with one variable apply to Newton's method with two or more variables. Say we have function $f(x, y)$ (containing 2 variables).

Then here we have it's Hessian matrix:

$$H_f(x,y) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

Now at a given point (x,y) we'll calculate it's eigenvalues $\lambda_1, \lambda_2, \ldots$
(A 2 by 2 matrix would have at most two eigenvalues)

If the gradient has value (0,0) at point (x,y) then:
- If all the eigenvalues of H_f at (x,y) are positive, it's a minimum
- If all the eigenvalues of H_f at (x,y) are negative, it's a maximum
- If one of the eigenvalues of H_f at (x,y) are zero, or we have mixed signs, it's inconclusive or we may have a saddle point

In Newton's method generalized to more than one variables, the formula for the next point is:

$$(x_{k+1}, y_{k+1}) = (x_k, y_k) - (H_f^{-1}(x_k, y_k) \cdot \nabla f(x_k, y_k))$$

3.7 SVD in PCA

The aim of this section is not to provide an explanation about Principle Component Analysis in general. The aim is to zoom in to the possibility, at a certain point, to apply SVD in PCA, instead of the more widely used definition of PCA based on EVD. The explanation will be made clearer by a diagram of the process, that visualizes how the different steps and alternatives relate.

We start with an mxn matrix A, feature by observation, so there are m observations and n features. We will mean-center it and call it $A_{mean-centered}$ or Amc.

In the most common definition we will now create the covariance matrix, which is

$$A_{Cov} = \frac{Amc^T \cdot Amc}{m-1}$$

Then we perform eigendecomposition (EVD) on this covariance matrix. The outcome is a set of eigenvalues which we shall call λ_3.

> On a side note, at this point we could perform SVD on the covariance matrix, instead of EVD. This would produce values σ. But because we are doing this on a covariance matrix, these values σ will take the same values as the values λ_3 found using EVD. (By the way in section 3.8 this fact is clarified.) That is because of the covariance matrix being positive semidefinite, and symmetrical. But this is not the SVD opportunity that will be discussed here mainly.

Alternatively, we could perform SVD on the original data instead of EVD on the covariance matrix. But in this case we take the mean-centered data, so matrix Amc.

Because of how SVD is done by definition, we would calculate:

$$Amc \cdot Amc^T$$
$$Amc^T \cdot Amc$$

On each of these we would take EVD. (Please note that $Amc^T \cdot Amc$ is almost the covariance matrix but without scaling by $\frac{1}{m-1}$).

The EVD produces eigenvalues λ_1 and λ_2 respectively which are of the same value. We have already noticed the difference in scaling between this step, and EVD from the covariance matrix.

So we have:

$$\sigma = \sqrt{\lambda_2}$$
$$\frac{\sigma^2}{m-1} = \lambda_3$$

This was the point where SVD can be used for PCA, from here PCA proceeds as usual. Below fig 3 there is a diagram that shows the steps and how they relate:

We have also created a small python program that does the following:

- Define testmatrix A
- Calculate PCA values
- Show that EVD on the covariance matrix and SVD on the covariance matrix produces the same values
- Show that SVD on the mean-centered data produces the same values apart from scaling

Link to this python program: `https://github.com/evert823/ejk_work_ib3702/blob/main/pca_evd_svd.py`

3.8 Connection between EVD and SVD

This paragraph is describing some of the connections between Eigen Value Decomposition (EVD) and Singualar Value Decomposition (SVD). The theory of these are part of the course and will not be introduced.
EVD rewrites a matrix A into separate components using the formula $A = Q\Lambda Q^{-1}$. Where Λ is a diagonal matrix with the eigen values and Q contains the eigen vectors.
SVD is used for dimensionality reduction of matrices. By using the formula $A = U\Sigma V^T$ where Σ contains the eigen values squared and is sorted from the highest value to the smallest. To calculate U one uses:

$$AA^T = \left(U\Sigma V^T\right)\left(U\Sigma V^T\right)^T = U\Sigma V^T V\Sigma^T U^T = U\Sigma^2 U^T$$

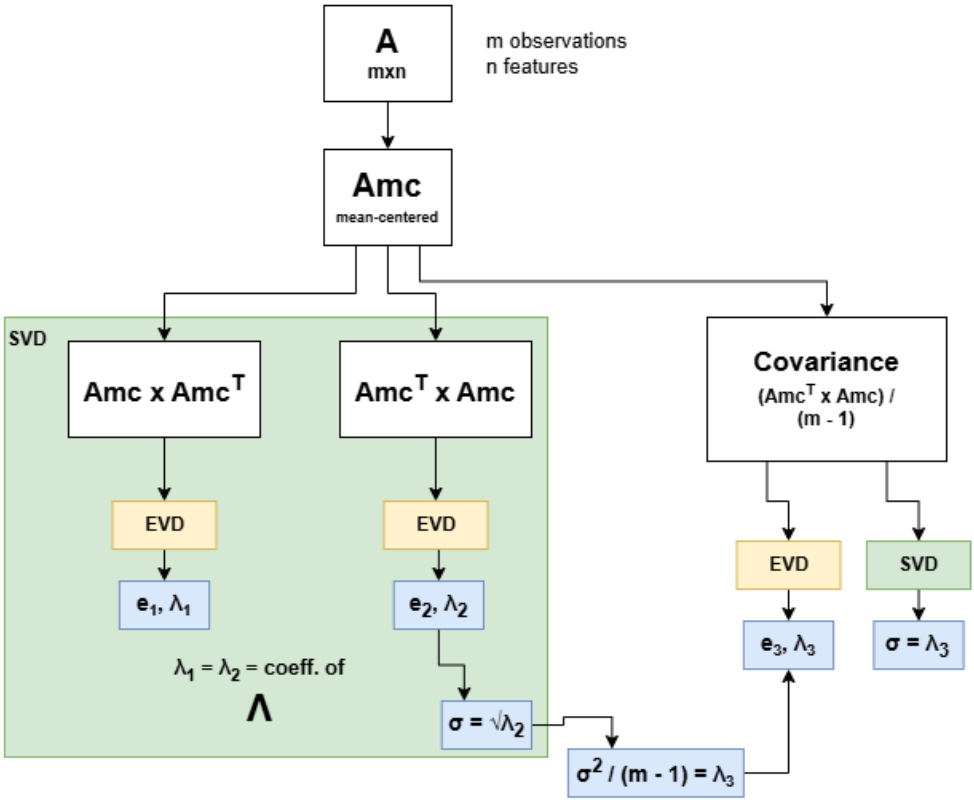

Figure 3: PCA overview and steps

Because $A.A^T$ is symmetrical the eigen decomposition becomes: $A.A^T = Q\Lambda Q^T$ when substituting Λ for Σ^2 and because Σ. When calculating values voor Σ one has to take te square root of the values of Λ. This can easily be done because Λ is a diagonal matrix. A similar computation is for matrix V

$$A^T A = \left(U\Sigma V^T\right)^T \left(U\Sigma V^T\right) = V\Sigma^T U^T U\Sigma V^T = V\Sigma^2 V^T$$

If a matrix S is square, symmetric, and positive semi-definite, meaning $A.A^T = A^t.A$ then both U and V from SVD are the same (see previous two equations). The SVD can then be written as

$$S = U\Sigma^2 U^T \text{ or } S = V\Sigma^2 V^T$$

this is the same. And because U is orthogonal (definition of SVD), we the transpose is equal to the inverse. Meaning we have the following formula:

$$S = U\Sigma' U^{-1}$$

This is the eigen value decomposition, where $\Sigma' = \Sigma^2$.

4 Numerical Examples

We will look at this function (fig 5):

$$f(x,y) = 85 - \frac{1}{90}x^2(x-6)y^2(y-6)$$

Visually we see a possible minimum near point $(x,y) = (4,4)$. We'll take $(x_0, y_0) = (1,1)$ and $\alpha = 0.01$ and from there, carry out gradient descent as actual example.
For gradient descent, we know that we need the formula for the gradient:

$$\nabla f(x,y) = \left(-\frac{1}{90}(3x^2 - 12x)y^2(y-6), -\frac{1}{90}x^2(x-6)(3y^2 - 12y)\right)$$

When we fill in x=1 and y=1 we calculate a gradient of (-0.5, -0.5). We subtract this, multiplied by the learning rate, from (0, 0) and go to the next iteration. Here we continued with a Python script that produced the output shown in appendix B.1. We see (x,y) approach (4,4) and we see the gradient approach (0,0). It takes 249 iterations. We also tried learning rate 0.05. Then we saw the same behaviour, but only 56 iterations (a tolerance of 1e-4 was used for comparing floats).

Now we will try to find the minimum using Newton's method. For this we need the second order derivatives:

$$\frac{\partial^2 f}{\partial x^2} = -\frac{1}{90}(6x - 12)y^2(y-6)$$

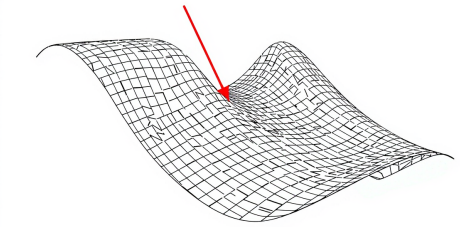

Figure 4: A saddle point

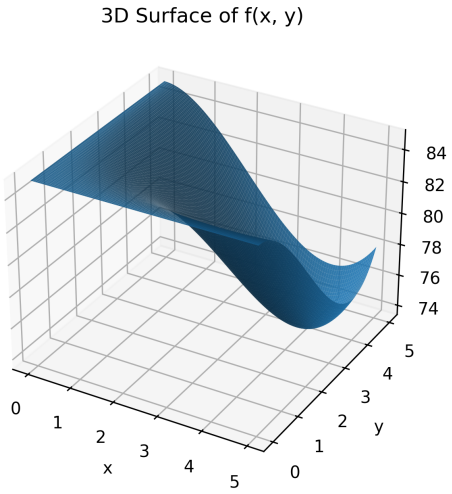

Figure 5: 3D surface of $f(x, y)$

$$\frac{\partial^2 f}{\partial x \partial y} = -\frac{1}{90}(3x^2 - 12x)(3y^2 - 12y)$$

$$\frac{\partial^2 f}{\partial y \partial x} = -\frac{1}{90}(3x^2 - 12x)(3y^2 - 12y)$$

$$\frac{\partial^2 f}{\partial y^2} = -\frac{1}{90}x^2(x - 6)(6y - 12)$$

We implemented the steps above in a python script and saw the output from appendix B.2. This converges to point (0,0). The eigenvalues of the Hessian are almost zero. So Newton's method brings us from (1,1) to a stationary point (0,0) which is not a minimum. This happened from many other points. When we choose a starting point real close to (4,4) only then it will converge to our expected minimum as can be seen in appendix B.3.
The point reached here is indeed (4,4). The eigenvalues of the Hessian are positive, and indeed this is in line with that we already know, that (4,4) is a minimum. From (3.2,3.2) it took 5 iterations. When we try Gradient Descent from this point (3.2,3.2) we will see the number of iterations for that:

```
Gradient Descent from (3.2,3.2) with learning rate 0.01 takes 109 iterations to reach (4,4).
Gradient Descent from (3.2,3.2) with learning rate 0.05 takes 27 iterations to reach (4,4).
```

The python code can be found on
`https://github.com/evert823/ejk_work_ib3702/blob/main/dogradientdescent.py`
`https://github.com/evert823/ejk_work_ib3702/blob/main/donewton.py`

5 Collaboration

Text...

6 Reflection

6.1 Student a

Text...

6.2 Student b

Text...

A Metaphores

A.1 Gradient descent

The basic intuition behind gradient descent can be illustrated by a hypothetical scenario. People are stuck in the mountains and are trying to get down (i.e., trying to find the global minimum). There is heavy fog such that visibility is extremely low. Therefore, the path down the mountain is not visible, so they must use local information to find the minimum. They can use the method of gradient descent, which involves looking at the steepness of the hill at their current position, then proceeding in the direction with the steepest descent (i.e., downhill). If they were trying to find the top of the mountain (i.e., the maximum), then they would proceed in the direction of steepest ascent (i.e., uphill). Using this method, they would eventually find their way down the mountain or possibly get stuck in some hole (i.e., local minimum or saddle point), like a mountain lake. However, assume also that the steepness of the hill is not immediately obvious with simple observation, but rather it requires a sophisticated instrument to measure, which the people happen to have at that moment. It takes quite some time to measure the steepness of the hill with the instrument. Thus, they should minimize their use of the instrument if they want to get down the mountain before sunset. The difficulty then is choosing the frequency at which they should measure the steepness of the hill so as not to go off track.

In this analogy, the people represent the algorithm, and the path taken down the mountain represents the sequence of parameter settings that the algorithm will explore. The steepness of the hill represents the slope of the function at that point. The instrument used to measure steepness is differentiation. The direction they choose to travel in aligns with the gradient of the function at that point. The amount of time they travel before taking another measurement is the step size.

Copied from [**?**].

B Output of Python scripts for Hessian Matrices

B.1 Gradient descent

```
0. x 1 y 1 --> (-0.500000000000000,-0.500000000000000)
1. x 1.00500000000000 y 1.00500000000000 --> (-0.506184974895937,-0.506184974895937)
2. x 1.01006184974896 y 1.01006184974896 --> (-0.512483652519824,-0.512483652519824)
3. x 1.01518668627416 y 1.01518668627416 --> (-0.518898669704649,-0.518898669704649)
...
246. x 3.98976321048153 y 3.98976321048153 --> (-0.0435643360994635,-0.0435643360994638)
247. x 3.99019885384252 y 3.99019885384252 --> (-0.0417150068300141,-0.0417150068300138)
248. x 3.99061600391082 y 3.99061600391082 --> (-0.0399437948089941,-0.0399437948089938)
249. x 3.99101544185891 y 3.99101544185891 --> (-0.0382474329314844,-0.0382474329314846)
```

B.2 Newton's method, first iteration

```
(1.0000,1.0000) - (0.4054,0.4054) = (0.5946,0.5946)
(0.5946,0.5946) - (0.2190,0.2190) = (0.3756,0.3756)
(0.3756,0.3756) - (0.1327,0.1327) = (0.2429,0.2429)
(0.2429,0.2429) - (0.0839,0.0839) = (0.1589,0.1589)
(0.1589,0.1589) - (0.0542,0.0542) = (0.1047,0.1047)
(0.1047,0.1047) - (0.0354,0.0354) = (0.0693,0.0693)
(0.0693,0.0693) - (0.0233,0.0233) = (0.0460,0.0460)
(0.0460,0.0460) - (0.0154,0.0154) = (0.0305,0.0305)
(0.0305,0.0305) - (0.0102,0.0102) = (0.0203,0.0203)
(0.0203,0.0203) - (0.0068,0.0068) = (0.0135,0.0135)
(0.0135,0.0135) - (0.0045,0.0045) = (0.0090,0.0090)
(0.0090,0.0090) - (0.0030,0.0030) = (0.0060,0.0060)
(0.0060,0.0060) - (0.0020,0.0020) = (0.0040,0.0040)
(0.0040,0.0040) - (0.0013,0.0013) = (0.0027,0.0027)
(0.0027,0.0027) - (0.0009,0.0009) = (0.0018,0.0018)
(0.0018,0.0018) - (0.0006,0.0006) = (0.0012,0.0012)
(0.0012,0.0012) - (0.0004,0.0004) = (0.0008,0.0008)
(0.0008,0.0008) - (0.0003,0.0003) = (0.0005,0.0005)
(0.0005,0.0005) - (0.0002,0.0002) = (0.0004,0.0004)
(0.0004,0.0004) - (0.0001,0.0001) = (0.0002,0.0002)
(0.0002,0.0002) - (0.0001,0.0001) = (0.0002,0.0002)

Hessian from last iteration: [[-4.37584933e-08 -8.75203986e-08]
 [-8.75203986e-08 -4.37584933e-08]]
Eigenvalues of Hessian: [-1.31278892e-07  4.37619053e-08]
```

B.3 Newton's method, second iteration

```
(3.2000,3.2000) - (-1.4933,-1.4933) = (4.6933,4.6933)
(4.6933,4.6933) - (0.7598,0.7598) = (3.9336,3.9336)
(3.9336,3.9336) - (-0.0677,-0.0677) = (4.0013,4.0013)
(4.0013,4.0013) - (0.0013,0.0013) = (4.0000,4.0000)
(4.0000,4.0000) - (0.0000,0.0000) = (4.0000,4.0000)

Hessian from last iteration: [[ 4.26666750e+00 -2.45347276e-13]
 [-2.45347276e-13  4.26666750e+00]]
Eigenvalues of Hessian: [4.2666675 4.2666675]
```