Bradley Morgan Id:1731302

Computational Methods – Worksheet 1

Task 1 - Find the Determinants of A & B

Determinants are very useful when solving linear systems, particularly finding the determinant of the $A - \lambda I$ matrix allows one to derive the quadratic equation used to find the principal components of a data stream (eigenvalues & eigenvectors). Such components are vital in traversing higher dimensional space, for example in the use of dimension reduction algorithms(principal component analysis (PCA)).

Task 1 – Part A

$$A = \begin{matrix} 2 & 3 \\ 1 & 1 \end{matrix}$$

To calculate the Determinant of a 2x2 matrix one must cross multiply each element and then subtract the two values generated by such multiplication.

$$Det(A) = (2x1) - (3x1) = -1$$

MATLAB validation:

```
A =

     2     3
     1     1

>> det(A)

ans =

    -1
```

Please Continue to the next page

Task 1 – Part B

$$B = \begin{matrix} 4 & 7 & 5 \\ 2 & 4 & 3 \\ 1 & 3 & 6 \end{matrix} = \begin{matrix} + & - & + \\ - & + & - \\ + & - & + \end{matrix}$$

In order find the determinant for this larger matrix it is not possible to solely perform the simple operations shown in part A. Instead matrix B must be deconstructed into 2x2 matrices termed minors, the determinant of 3 minors should be sufficient enough to calculate the overall determinant of the major B matrix. In order to select 3 minors a column from the matrix can be targeted, the minors emerge via the following rule:

Step 1: Select a Column:

$$B = \begin{matrix} \boxed{4} & 7 & 5 \\ \boxed{2} & 4 & 3 \\ \boxed{1} & 3 & 6 \end{matrix} = \begin{matrix} + & - & + \\ - & + & - \\ + & - & + \end{matrix}$$

Step 2: Isolate the minor relative to the first element in the selected column:

$$B = \begin{matrix} \boxed{4} & \boxed{7} & \boxed{5} \\ \boxed{2} & \boxed{4} & \boxed{3} \\ \boxed{1} & \boxed{3} & \boxed{6} \end{matrix} = \begin{matrix} + & - & + \\ - & + & - \\ + & - & + \end{matrix}$$

The blue bars indicate the Rows and columns to be ignored, whilst the red bar reveals the minor respective to 4.

Step 3: Calculate the determinant of the minor using the $\det(x) = ad - bc$ equation.

Step 4: Repeat the above steps for each element in the selected column. The Horizontal blue bar will descend as one moves down the rows of elements, any elements that are not encapsulated within a blue bar will be used to formulate a minor respective to its element. This process can be shown as follows:

$$1st\ Det(B) = \begin{bmatrix} 4 & 3 \\ 3 & 6 \end{bmatrix} = +4\big((4\ x\ 6) - (3\ x\ 3)\big) = 60$$

$$2nd\ Det(B) = \begin{bmatrix} 7 & 5 \\ 3 & 6 \end{bmatrix} = -2\big((7\ x\ 6) - (5\ x\ 3)\big) = -54$$

$$3rd\ Det(B) = \begin{bmatrix} 7 & 5 \\ 4 & 3 \end{bmatrix} = +1\big((7\ x\ 3) - (4\ x\ 5)\big) = 1$$

$$Sum\ all\ anwsers\colon\ Det(B) = 60 - 54 + 1 = 7$$

Each determinant of each minor is multiplied by their element; however the sign of the element must follow the specified pattern of occurrence as shown by the matrix on the right side of matrix B. The results can be validated via MATLAB:

```
A =

     4     7     5
     2     4     3
     1     3     6

>> det(A)

ans =

    7.0000
```

Please Continue to the next page

Bradley Morgan Id:1731302

Task 2 – Calculating Eigenvalues and eigenvectors

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

Define the $\lambda * I$ Matrix:

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \lambda I = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}$$

Define the $A - \lambda I$ matrix:

$$Lambda\ Matrix = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} = \begin{bmatrix} 2-\lambda & 1 \\ 1 & 2-\lambda \end{bmatrix}$$

Find the determinant of Lambda Matrix:

$$(2 - \lambda)(2 - \lambda) - (1\ x\ 1)$$

Expand out the brackets:

$$4 - 2\lambda - 2\lambda + \lambda^2 - 1 = 0$$

Simplify the above equation:

$$\lambda^2 - 4\lambda + 3 = 0$$

The above quadratic equation can be used to solve for λ which will be the Eigenvalues.

Solving the Quadratic equation:

$$\lambda = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$\lambda^2 - 4\lambda + 3 = 0$$

$$a = 1 \quad b = -4 \quad c = 3$$

With Addition:

$$\lambda = \frac{4 + \sqrt{16 - 12}}{2} = \frac{4 + 2}{2} = 3$$

With Subtraction:

$$\lambda = \frac{4 - \sqrt{16 - 12}}{2} = \frac{4 - 2}{2} = 1$$

$$Eigenvalue: \lambda_1 = 1$$

$$Eigenvalue: \lambda_2 = 3$$

Finding the Eigenvectors

The eigenvector respective its eigenvalue by substituting λ_1 into the Lambda matrix:

$$B\ matrix = \begin{bmatrix} 2-\lambda & 1 \\ 1 & 2-\lambda \end{bmatrix} = \begin{bmatrix} 2-1 & 1 \\ 1 & 2-1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

The next step is to Solve $B\bar{x} = \bar{0}$ via row reduction operations, this will allow me to ascertain the first eigenvector:

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Augmented Matrix for row reduction:

$$\left|\begin{matrix} 1 & 1 \\ 1 & 1 \end{matrix}\right| \begin{matrix} 0 \\ 0 \end{matrix} \quad -1(R1) + R2$$

(-1 x 1) + 1 = 0 // (-1 x 1) + 1 = 0 //

$$\left|\begin{matrix} 1 & 1 \\ 0 & 0 \end{matrix}\right| \begin{matrix} 0 \\ 0 \end{matrix} = \ x_1 + \ x_2 = 0$$

$$x_1 = \ -1$$

$$x_2 = 1$$

$$Eigenvalue = 1 \quad Eigenvector = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

Please Continue to the next page

The eigenvector respective its eigenvalue by substituting λ_2 into the Lambda matrix:

$$B\ matrix = \begin{bmatrix} 2-\lambda & 1 \\ 1 & 2-\lambda \end{bmatrix} = \begin{bmatrix} 2-3 & 1 \\ 1 & 2-3 \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}$$

As before the $B\bar{x} = \bar{0}$ equation solved via row reduction operations, this will allow me to ascertain the second eigenvector:

$$\begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Augmented Matrix for row reduction:

$$\left|\begin{matrix} -1 & 1 \\ 1 & -1 \end{matrix}\right| \begin{matrix} 0 \\ 0 \end{matrix} \quad 1(R2) + R1$$

(1 x 1) + -1 = 0 // (1 x -1) + 1 = 0 //

$$\left|\begin{matrix} 0 & 0 \\ 1 & -1 \end{matrix}\right| \begin{matrix} 0 \\ 0 \end{matrix} = \ x_1 - x_2 = 0$$

$$x_1 = 1$$

$$x_2 = 1$$

$$Eigenvalue = 3 \quad Eigenvector = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

<u>Validation of results via MATLAB</u>:

As shown on the right the result produced by the compute eig() function produces the same results as the handwritten solution. V represents the Eigen vectors, they tend towards -1 or 1 due to MATLAB floating point computation procedures. Finally it is clear the eigen values of D match with the eigenvalues produced using the hand written method.

```
A =

     2     1
     1     2

>> eig(A)

ans =

     1
     3

>> [V, D] = eig(A)

V =

   -0.7071    0.7071
    0.7071    0.7071


D =

     1     0
     0     3
```

Bradley Morgan Id:1731302

Task 3 – Dimension reduction

In short, the task requires that a ten-dimensional data set be projected into a reduced two-dimensional data space. Dimension reduction processes are highly useful when dealing with extremely large data sets for three main reasons. Firstly, when a multidimensional data set is sampled from the external distribution, there are often many correlations between different dimensions of the data. However, due to the size of these data sets it is very hard to isolate such correlations when it is masked by so much background variance. Not only this, but some correlations may be more important than others depending on the research aims. Therefore, it would be useful to be able to identify how much variance each dimension contributes to the total variance and thus would allow one to discard unwanted dimensions or dimensions that contribute less to the total variance, but with minimal data loss. Secondly, if we can find patterns within large data sets more efficiently through dimension reduction, then this would allow for the prediction of future behaviour for such patterns, which increases ones analytical power. Thirdly, it allows the visualisation of a data set to be more easily attainable. Higher dimensional data could be reduced to four-dimensional data or below which is comprehendible to the human brain.

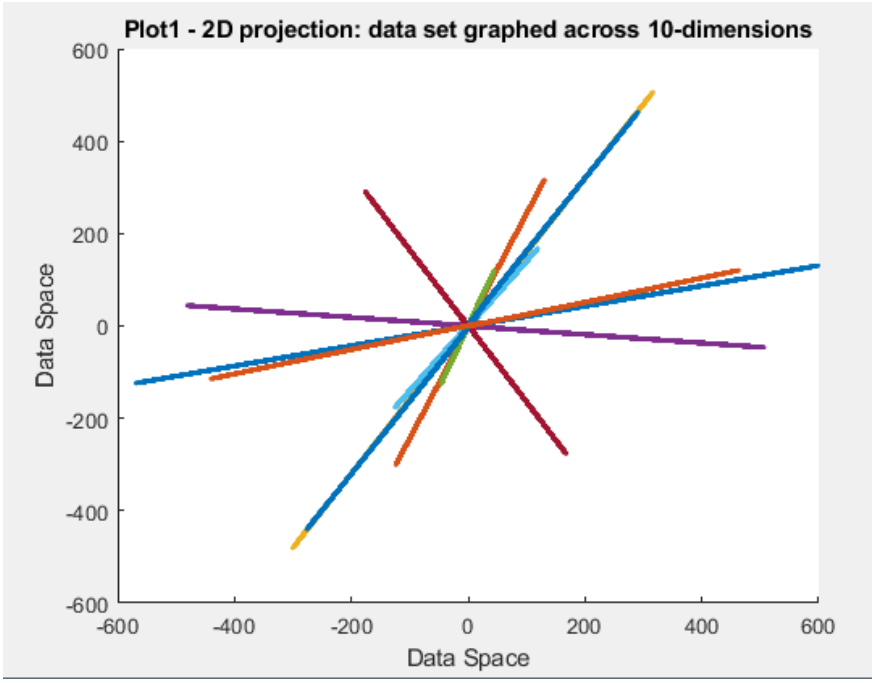
Visualisation of the data across 10 dimensions:

Editor - C:\Users\Novus\Documents\MASTERS UNI DOCS\Lin

Data.m | Data2.m | Oja_neuron.m | Worksh

```
1
2 -     X = data;
3
4       %Plots all the dimensions overlayed
5
6 -     for i = 1:9
7 -         hold on
8 -          plot(X(:, i), X(:, i +1), '.')
9
10 -    end
11
12 -    hold off
13
14
15 -    Cov = cov(X);
```

The following code on the left uses simple for loop with the hold on/off function to overlay all of the ten-dimensions, each comprised of 20,000 data points into a single 2D data space.



Principal Component Analysis algorithm:

Code image 2: Covariance matrix – eigen function

```
17 -    Cov = cov(X);
18
19      %Calculate Eigenvalues of C
20 -    [W, Lamda] = eig(Cov);
21
22 -     u = diag(Lamda)/trace(Lamda) * 100;
23 -    figure; bar(u)
24 -    xlabel('Dimensions')
25 -    ylabel('Percentage of Explained Variance')
26
```

The first step (line 17) is to calculate the covariance of data across all 10-dimensions. This will allow one to establish how all of the data points for each dimension vary in relation to the other dimensions.

The second step (line 20) is to calculate the magnitude by which each dimension contributes to the total variance of a data set. This can be done by calculating the eigenvalues and eigenvectors of the covariance matrix, in Matlab, by using the built in eig() function. The matrix W has been assigned to hold the eigenvectors, whilst lamda has been assigned to hold eigenvalues.

Code image 3: Percentage Variance

```
>> u

u =

   -0.0000
   -0.0000
   -0.0000
   -0.0000
   -0.0000
    0.0000
    0.0000
    0.0000
    0.0038
   99.9962

fx >>
```

Thirdly, the variable U is used to extract the diagonal elements of lamda (eigenvalues) and then divide by the trace (sum of all diagonal values) of lamda x 100, which gives the dimensional variance contribution as a percentage.

As you can see in Code Image 3 the eigenvalue corresponding to the 10th dimension contributes 99.99% towards the total variance, whilst the eigenvalue corresponding to dimension 9 seems to be contributing 0.03% towards the total variance, whilst all other dimensions do not contribute to the total variance of the data set.

Please Continue to the next page

Code image 4: Dimension reduction.

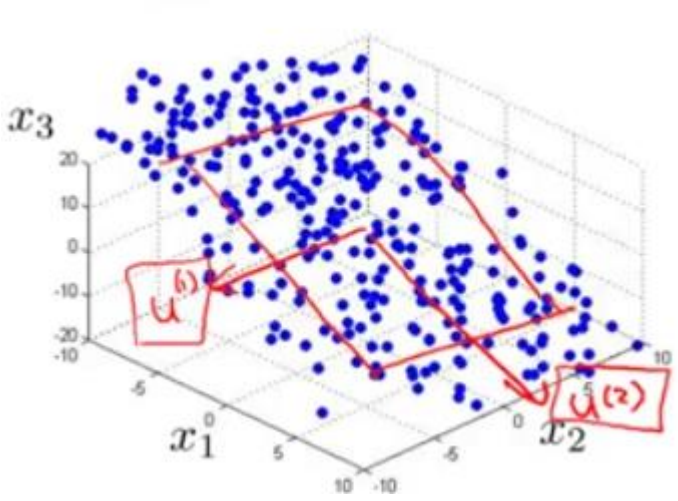
```
27      %Reordering W and Lamda
28
29 -    W = W(:, end: -1:1);
30 -    Lamda = Lamda(:, end: -1: 1);
31
32      %Projecting Data onto sub-dimension(2D)
33
34 -    W_Reduce = W(:, 1:2);
35
36 -    sub_Dim = data * W_Reduce;
37
38
39 -    figure; plot(sub_Dim(:, 1), sub_Dim(:, 2), '.k');
40
```

The eig() function does not explicitly order and match any of the eigenvector outputs to their corresponding eigenvalue. Therefore, this must be done manually via the commands shown on line 29-30.

Dimension Reduction.

It is clear from above analysis that all dimensions other than 9 and 10 are redundant and we can afford to discard these dimensions without increasing data loss at all. Therefore, the goal now is to project the most descriptive data (dimension with the highest variance) into a two-dimensional space. The first eigenvector corresponds to a set of coordinates that lay in the direction of the highest variance of the data in Euclidian Space. The second eigenvector points towards the greatest variance of the data points that are orthogonal to the first eigenvector. These two eigenvectors represent the directions of the new two-dimensional axis of the sub-dimensional plane that the data will be projected towards. This can be shown by diagram 1. where U_{1-2} represents the eigenvectors and the two dimensional projection plane that they form Andrew Ng (2017).

Diagram 1 – Conceptual illustration of eigen-vectors.

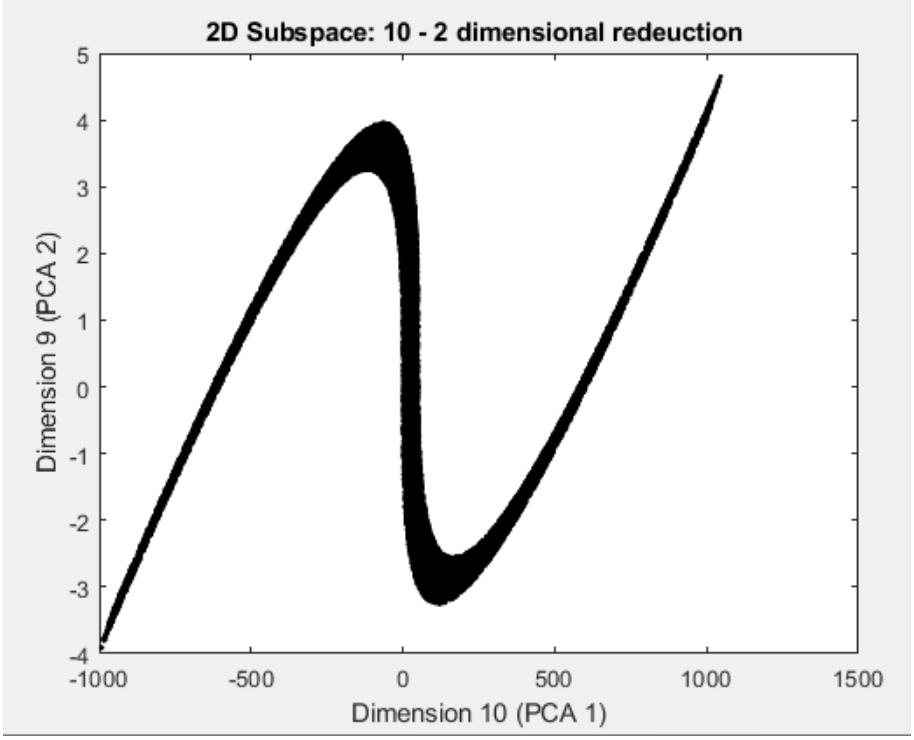


The W_Reduce variable (line 34) is used to hold the extracted target dimension eigenvectors from the larger eigen vector matrix of W to form the reduced W matrix(10x2).

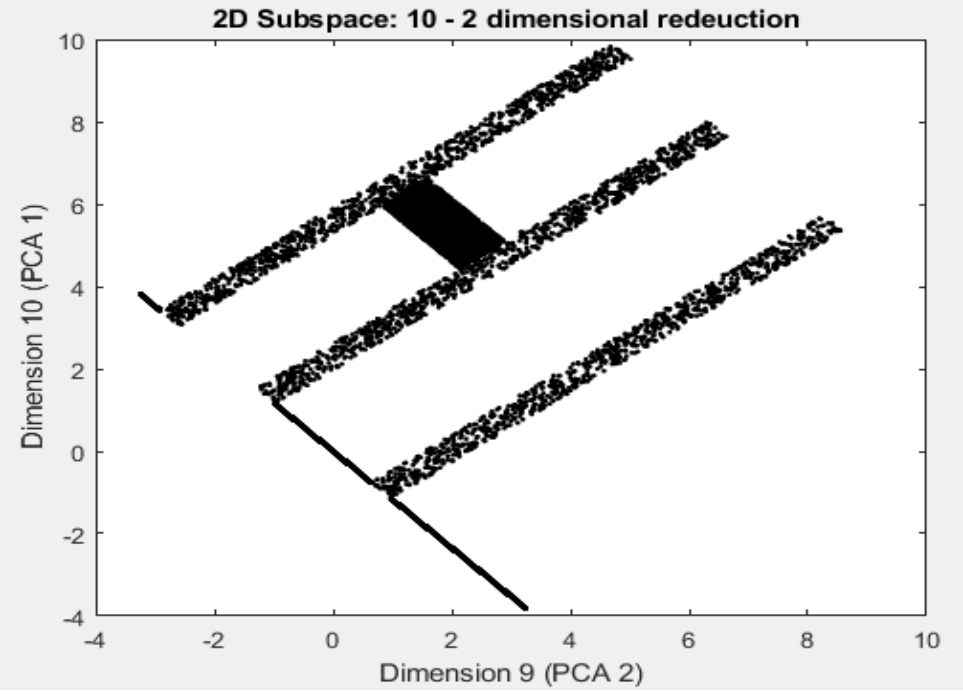
Line 36 enables the projection of the data-set to be projected onto the new sub-dimensional plane via a simple matrix multiplication command for the original data and the reduced eigen vector matrix W.

Finally, both the principal components (10 – 9 dimensions) can be plotted to allow for the visualisation of their newly mapped data shape across 2D sub-space. The resultant plot image can be seen below on the next page.

Plot 2: 2D sub-dimensional projection of dataset – 'data'



Plot 3: 2D sub-dimensional projection of dataset – 'data2'



Plot 3 shows the result of dimension reduction on the 'data2' dataset. The code used to implement the reduction process is in essence the same as the above code that describes the process respective to the 'data' dataset.

With data2, again a 20000 x 10 matrix , it was required that this dataset be reduced from ten to two dimensions. Analysis revealed that both dimensions ten and nine where the highest dimensional variance contributors with a percentage of 77.8% (Dimension 10) and a 22.1% contribution from dimension 9. This is shown via code image 5.

Code image 5: Percentage Variance

```
u =

   -0.0000
   -0.0000
   -0.0000
   -0.0000
    0.0000
    0.0000
    0.0000
    0.0000
   22.1349
   77.8651

>> Data2
```

Task 4 – Oja's Neuron

Oja's rule represents an extension to the Hebbian learning algorithm that was applied to eigenvector filtration. It was shown that Hebb's law allows a simple single artificial neuron to learn and extract an unknown distributions eigen direction of covariance, which in turn holds correlational patterns of the data input stream. This is achieved by the weight of neuron being represented as a vector that corresponds to a location within dataspace. Over the course of several training cycles the weight vector will converge towards the same direction as the first eigenvector, which corresponds highest eigenvalue of a covariance matrix. This occurs due to the fact that more frequently occurring inputs will generate higher influence during learning. Thus, more frequently occurring input vectors will bias the weight vector towards themselves, which will result in greater signal values and thus greater weight updates in these particular inputs directions, which lay in the region of highest covariance. Oja's rule modifies the Hebbian processes by implementing a weight decay proportional to the signal and activation of the neuron, this counteracts the issues of unbounded growth of weight vector magnitudes found in the original Hebbian algorithm. To implement Oja's rule programmatically the following code was written via MATLAB:

Code image 6 – Input stream

```
x = ojax(:, 1);
y = ojax(:, 2);

input = [x' ; y'];
```

Code image 6 shows how each of the columns of the original data-set 'ojax' has been converted into two separate input vector streams both x and y.

```
%Initilisation of parameters

learnrate = 0.5;
weights = [0.2 ; 0.6];
plot(0.2, 0.6, 'o'); %plots starting point of weights
```

Next The learning rate is set to a small value from the real numbers, the learning rate aids in controlling the rate of convergence towards the eigenvector position. Finally weights are initialised in vector form, the location of the starting weights is plotted into the data space.

Please Continue to the next page

Code image 7: Oja – learning loop

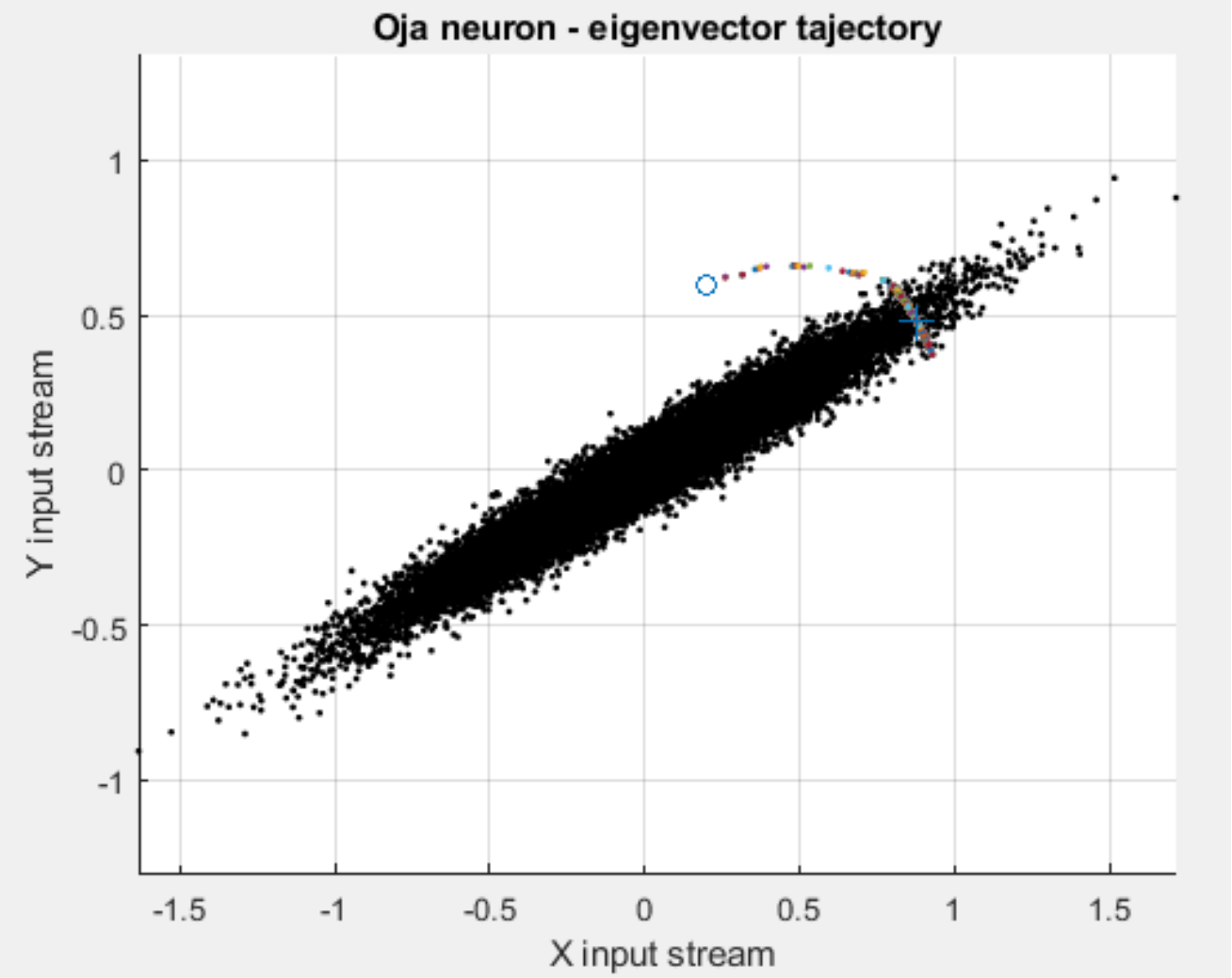
```
24 -  for epoch = 1:6
25 -      for i = 1:20000
26            %neuron summation
27 -          sum = input(:, i)' * weights;
28            %Equation to update weights
29 -          weights = weights + learnrate*sum*(input(:, i) - sum * weights);
30 -          plot(weights(1), weights(2), '.');
31 -      end
32 -  end
33
34 -   plot(weights(1), weights(2), '+', 'markersize', 10);
```

A nested for loop is utilised to allow the neuron to be repetitively trained across 6 epochs for the full 20000 data points for the x and y input streams. Line 27 indicates the equation that produces the signal of the neuron by multiplying the sum of the current x and y inputs transposed, multiplied by the neurons weights. Furthermore line 29 programmatically represents the weight update equation defined in Oja's algorithm. As said in the opening paragraph both the sum equation and weight update equation will cause the biasing of the weights in the direction of the frequently occurring inputs that correspond to the region in, coordinate space, of highest variance. As the program proceeds through each iteration one would expect weight vector biasing to increase, thus each weight vector coordinate is plotted to generate a trajectory path.

At the end of the final epoch cycle the neuron should be tending towards convergence with eigen-vector of the distribution, the final coordinate of the weights is then plotted. When graphed the convergence trajectory can be seen on the next page:

Please Continue to the next page

Plot4: Eigen-vector trajectory of Oja neuron



The blue circle marker indicates the starting position of the weights of the neuron in data space, whilst the blue cross indicates the final weight vector position of the neuron. The multicoloured sequence of points shows the trajectory of the neurons course to convergence, with the trajectory reaching an equilibrium of sorts upon the final vector(notice how the trajectory travels back and forth across the height of the distribution). When accessing the final vector values of the weights it is shown to be as follows:

Code image 8: Final Oja neuron weights

```
>> weights

weights =

    0.8748
    0.4848
```

Code Image 9: Validation 1

```
>> A = cov(ojax)

A =

    0.1527    0.0864
    0.0864    0.0522

>> [W Lamda] = eig(A)

W =

    0.4988   -0.8667
   -0.8667   -0.4988
```

The final code images(9-10) shows the resultant eigen vectors of the datasets covariance matrix when calculated manually using the eig() function. Its is clear to see the that the weight vector position is extremely close to the manually calculated eigen vector on the right. This right(2nd column) Eigen vector corresponds to the highest eigen-value as shown when the values Lambda is accessed(code image 10). Finally it should be noted that the scaling differences between the weight and the manually calculated eigen-vector is arbitrary as both will provide the same directional line through the highest covariance of the distribution. Therefore one can conclude that the Oja neuron can indeed converge towards the highest eigen-vector to eigen-value pair when presented with an unknown distribution during unsupervised learning.

Code image 10: Validation 2

```
>> Lamda

Lamda =

    0.0025         0
         0    0.2024
```

Reference:

Andrew Ng: Lecture 14.4 — Dimensionality Reduction | Principal Component Analysis Algorithm, 2017.Youtube, Link: https://www.youtube.com/watch?v=rng04VJxUt4

Thank-you for reading :D!!