Question 4

Given:

- 1-D lattice Kohonen
- 2-D inputs
- 100 neurons
- Random initial weights
    - [.25, .75]
- Varied learning rate
    - .5 initially, linearly decreased over training period
- Training period is 100 iterations
- Varied radius
    - R = 5 initially, linearly decreased to 1 over training phase

Find:

- Plot of SOM with connecting lines
    - t = 0, 50, 100, 1000, 5000

Net Construction:

We will be using a 1-D Kohonen Lattice, which is a form of self-organizing map. Since we are plotting a 1-D lattice, our graph will take the form of a line. We will be training by linear topology using R as a radius. So, if node J is to be trained, we will also train nodes in the range of J - R to J + R. Our 2-D input data will require us to have 2 input units X and Y.

With a cap of 100 neurons, we can create 50 cluster units. This implementation will have a link between every input and output. With 50 cluster units, our weight matrix will be of size 2 x 50. We will initialize this matrix with random values in the range [.25, .75]. X values will be mapped to the first rows, and Y values will be mapped to the second row.

Input data has been provided as a 100 x 2 matrix with X in column 1 and Y in column 2. Training will be done using the Kohonen SOM architecture. Since we have 100 unique inputs, we will define one iteration as one vector fed into the training algorithm. The learning rate alpha will be decreased over the course of each training period, and the radius R will be decreased over the entire training phase of 5000 iterations.

Our training phase will consist of 5000 iterations with plots shown at 10, 50, 100, 1000, and 5000 iterations. This setup will show how the map organizes through 1 run of the entire set from 0 to 100. It will also show how the map organizes through subsequent runs of the set from iterations 100 to 5000. Finally, this setup will have alpha decreasing much quicker than R, keeping in line with the standard Kohonen procedure.

Algorithm:

First, we will begin by initializing the data into the training set.

```
x = SOM_Data;
```

Next, we will allocate the weights with random values and bind the results to [.25, .75].

```
w = rand(2,50);
w = w * .5 + .25;
```

Following, we will allocate a 1 x 100 matrix with alpha.

```
alpha_step = (.5-.01)/99;
alpha = .5:-alpha_step:.01;
```

Next, we wil allocate a 1 x 5000 matrix with the radius.

```
R_step = (5 - 1)/4999;
Radius = 5:-R_step:1;
```

Next, we will pre allocate a matrix to hold the values for elucidian distance and initialize counters for R and alpha.

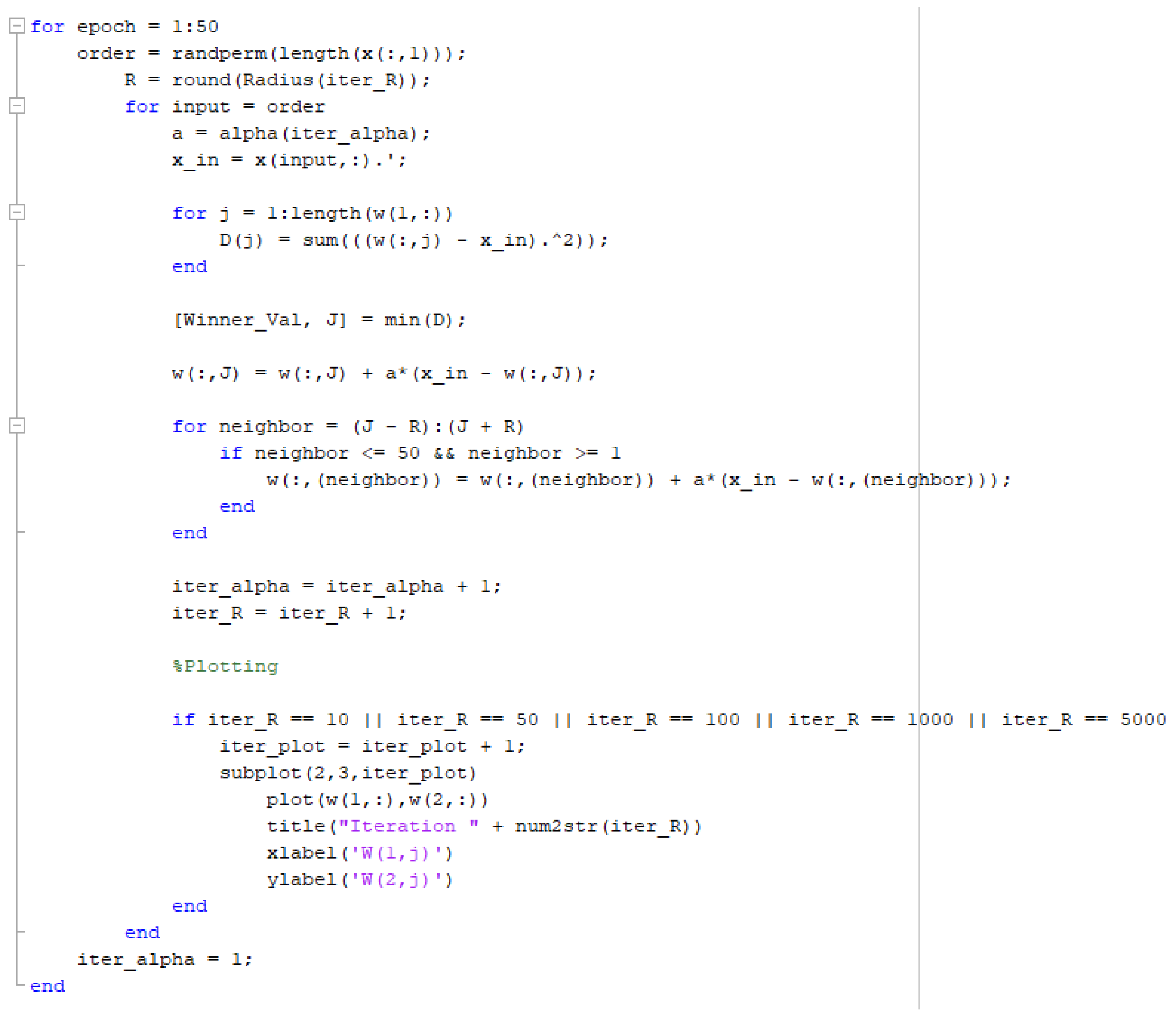
```
D = zeros(1,length(w(1,:)));
iter_alpha = 1;
iter_R = 1;
```

We will then plot the intial weights.

```
iter_plot = 1;
subplot(2,3,iter_plot)
    plot(w(1,:),w(2,:))
        title('Initial')
        xlabel('W(1,j)')
        ylabel('W(2,j)')
```

Next, we will begin the training phase of the net. The algorithm will start by deciding a random order to update the weights. Then, it will follow this order. The algorithm will calculate the elucidean distance between all the inputs and their respective cluster units. A winner will be decided on the basis of the node with the smallest distance.

Weights will be updated in accordance with the Kohonen net architecture. Then, all nodes within range of the radius will be updated the same way. Finally, the alpha and R values will be updated. At iteration counts 10, 50, 100, 1000, and 5000 a plot will be drawn of the cluster units.

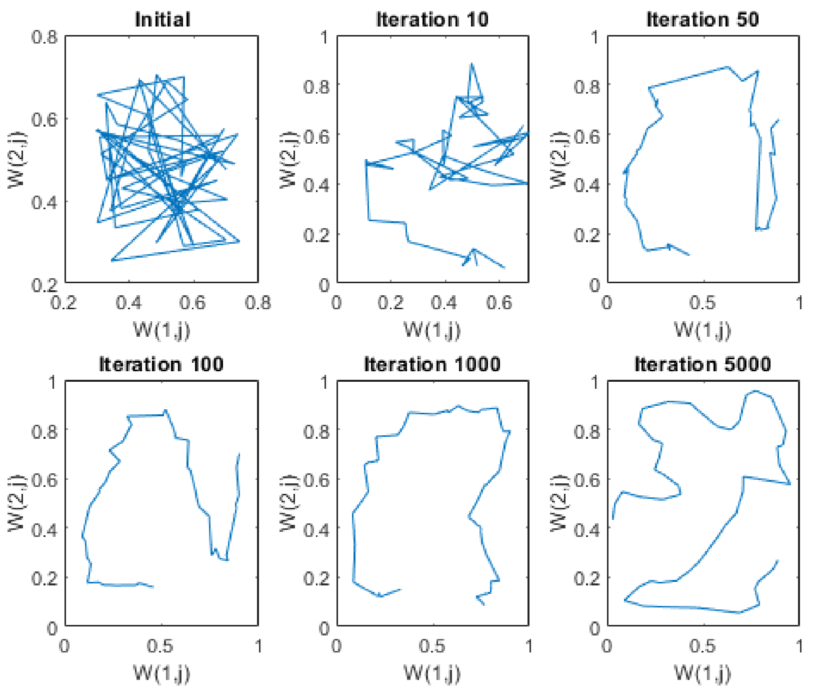


```
for epoch = 1:50
    order = randperm(length(x(:,1)));
        R = round(Radius(iter_R));
        for input = order
            a = alpha(iter_alpha);
            x_in = x(input,:).';

            for j = 1:length(w(1,:))
                D(j) = sum(((w(:,j) - x_in).^2));
            end

            [Winner_Val, J] = min(D);

            w(:,J) = w(:,J) + a*(x_in - w(:,J));

            for neighbor = (J - R):(J + R)
                if neighbor <= 50 && neighbor >= 1
                    w(:,(neighbor)) = w(:,(neighbor)) + a*(x_in - w(:,(neighbor)));
                end
            end

            iter_alpha = iter_alpha + 1;
            iter_R = iter_R + 1;

            %Plotting

            if iter_R == 10 || iter_R == 50 || iter_R == 100 || iter_R == 1000 || iter_R == 5000
                iter_plot = iter_plot + 1;
                subplot(2,3,iter_plot)
                    plot(w(1,:),w(2,:))
                    title("Iteration " + num2str(iter_R))
                    xlabel('W(1,j)')
                    ylabel('W(2,j)')
            end
        end
    iter_alpha = 1;
end
```

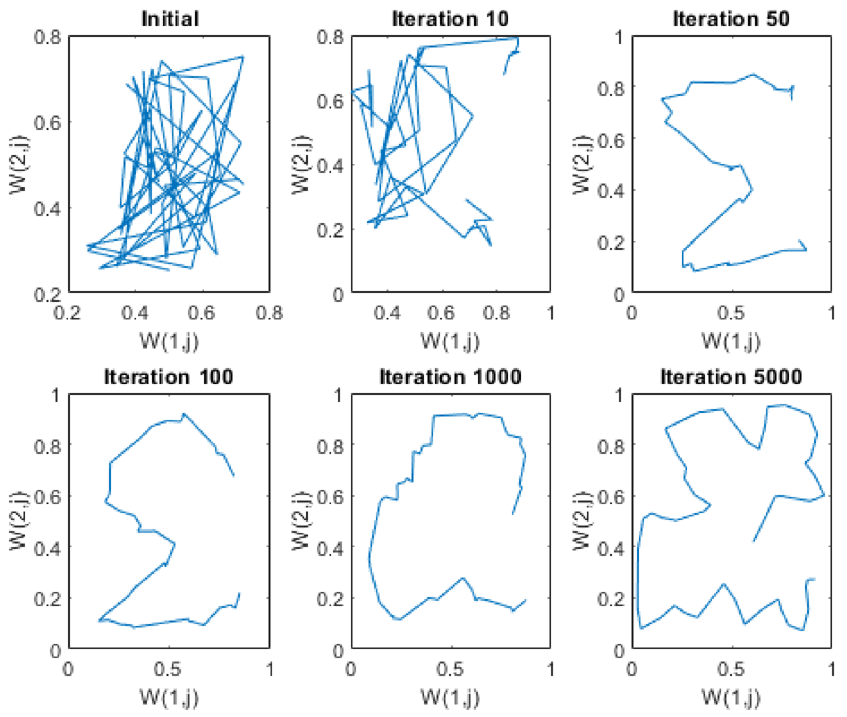
SOM Plots:

Plots for 3 separate runs of the algorithm are shown.

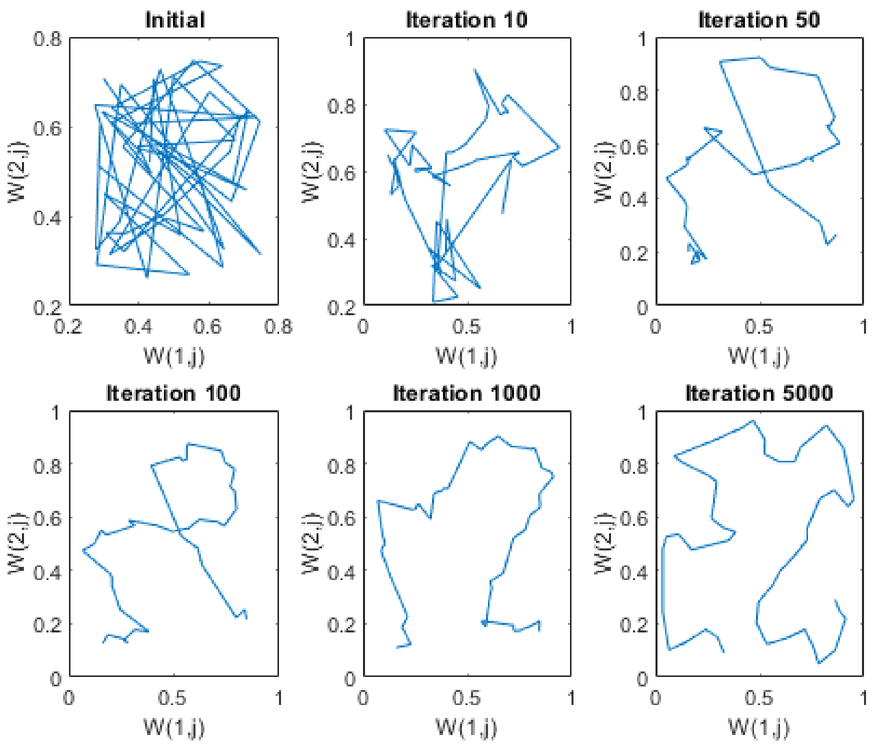
Run 1:



Run 2:



Run 3:



Analysis:

We reach a fitted curve around 1000 iterations. Going up to 5000 iterations seems to overfit the SOM, as the path looks less efficient at that point. Furthermore, we seem to have varying outputs based on the run. The only properties that could change between the net between runs are the initial weights as they are selected using a random function. I am not too sure how much this discrepancy would affect the final result, as the general curve at 5000 iterations still seems to be very similar between the runs.