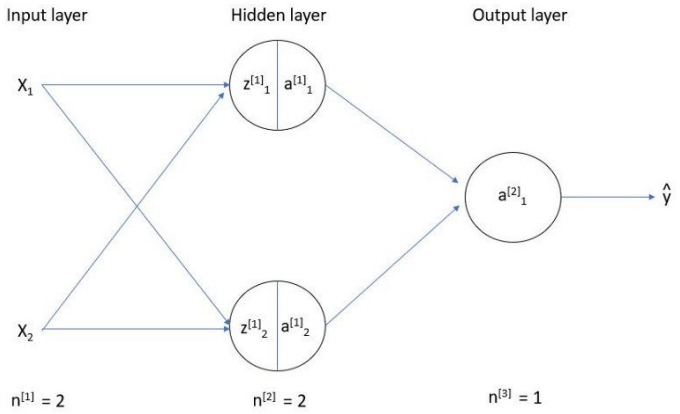
SOFE 4620U

Machine Learning & Data Mining

Mini Project 3

Mackenzie Zahn	100559676
Abdirisak Ibrahim	100582257

Design Details



To solve the given problem, we have implemented a two layer feed-forward neural network. There is an input layer, hidden layer, and an output layer. The network is trained and run for a defined number of epochs. The results are shown as an array of certainty values (from 0-1).

Our Input layer has 45 nodes, each representing one of the bits in the bitmap for a digit. The hidden layer has 5 nodes as outlined in the project requirements. The output layer has 10 nodes, each one represents the digits 0-9, and the value of the output layer represents how confident the network is about the input being that respective digit.

We considered two approaches for the actual implementation of the neural network: objects, and matrices. We ultimately decided on matrices because it is generally lighter and less memory intensive to scale up than objects. The downside of this approach is that we needed to perform many operations on matrices, however we found a java class online that contained functions for all the matrix operations we would need. This allowed us to just focus on the logic for the neural network(np.java, https://gist.github.com/Jeraldy/7d4262db0536d27906b1e397662512bc).

Bitmap Inputs
An array of bitmaps of length 45 representing the 10 digits

```
//bitmap inputs
double[][] X = {
                    {0,0,1,0,0,0,1,1,0,0,1,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0},
                    {0,1,1,1,0,1,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,0,0,1,1,1,1,1},
                    {0,1,1,1,0,1,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,1,0,0,0,0,0,1,0,0,0,0,1,1,0,0,0,1,0,1,1,1,1,0},
                    {0,0,0,1,0,0,0,1,1,0,0,0,1,1,0,0,1,0,1,0,0,1,0,1,0,1,0,0,1,0,1,1,1,1,1,1,0,0,0,1,0,0,0,0,1,0},
                    {1,1,1,1,1,1,1,0,0,0,0,1,1,0,0,0,0,1,1,1,1,1,0,1,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,0,0,0,1,0,1,1,1,1,0},
                    {0,1,1,1,0,1,0,0,0,0,1,1,0,0,0,0,1,1,0,0,0,0,1,0,0,0,0,1,1,1,1,1,0,1,0,0,0,1,1,0,0,0,1,0,1,1,1,1,0},
                    {1,1,1,1,1,1,0,0,0,0,1,0,0,0,1,0,0,0,0,1,0,0,0,1,0,0,0,0,1,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0},
                    {0,1,1,1,0,1,0,0,0,1,1,1,0,0,0,1,1,1,0,0,0,1,0,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,0,1,1,1,1,0},
                    {0,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,0,1,1,1,1,1,0,0,0,0,1,0,0,0,0,1,1,0,0,0,1,0,1,1,1,1,0},
                    {0,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,0,1,1,1,1,0},
                };
```

10 arrays representing the expected outputs. (1 representing a 100% match)

```
//Expected outputs
double[][] Y = {
                    {1,0,0,0,0,0,0,0,0,0},
                    {0,1,0,0,0,0,0,0,0,0},
                    {0,0,1,0,0,0,0,0,0,0},
                    {0,0,0,1,0,0,0,0,0,0},
                    {0,0,0,0,1,0,0,0,0,0},
                    {0,0,0,0,0,1,0,0,0,0},
                    {0,0,0,0,0,0,1,0,0,0},
                    {0,0,0,0,0,0,0,1,0,0},
                    {0,0,0,0,0,0,0,0,1,0},
                    {0,0,0,0,0,0,0,0,0,1}
                };
```

Constant values

10 inputs, 20000 epochs, 5 nodes, learning rate of 0.01.

```
int nSamples = 10;
int nInputs = 45;
int nodes = 5;
int nOutputs = 10;
int epocs = 20000;
double learnRate = 0.01;
```

Assign and apply weights for the first and second layers as 2D arrays (matrices) populated by random numbers between 0 and 1.0

```
//layer 1 weights (ninputs columns, nodes rows)
//mth row is mth input, nth column is nth node
double[][] W1 = np.random(nInputs, nodes);
double[][] b1 = new double[nSamples][nodes];

//layer 2 (output layer) weights
double[][] W2 = np.random(nodes, nOutputs);
double[][] b2 = new double[nSamples][nOutputs];
```

```
//apply weights layer 1
double[][] sumL1 = np.add(np.dot(X,W1), b1);

//apply sigmoid layer 1
double[][] Z1 = np.sigmoid(sumL1);
```

```
//apply weights layer 2
double[][] sumL2 = np.add(np.dot(Z1,W2),b2);
double[][] Z2 = np.sigmoid(sumL2);
```

Back Propagation

Calculate the errors & the weight deltas

```
//Layer 2
double[][] deltaE2 = np.subtract(Z2, Y);

double[][] dW2 = np.divide(np.dot(np.T(Z1), deltaE2), nSamples);

double[][] db2 = np.divide(deltaE2, nSamples);
```

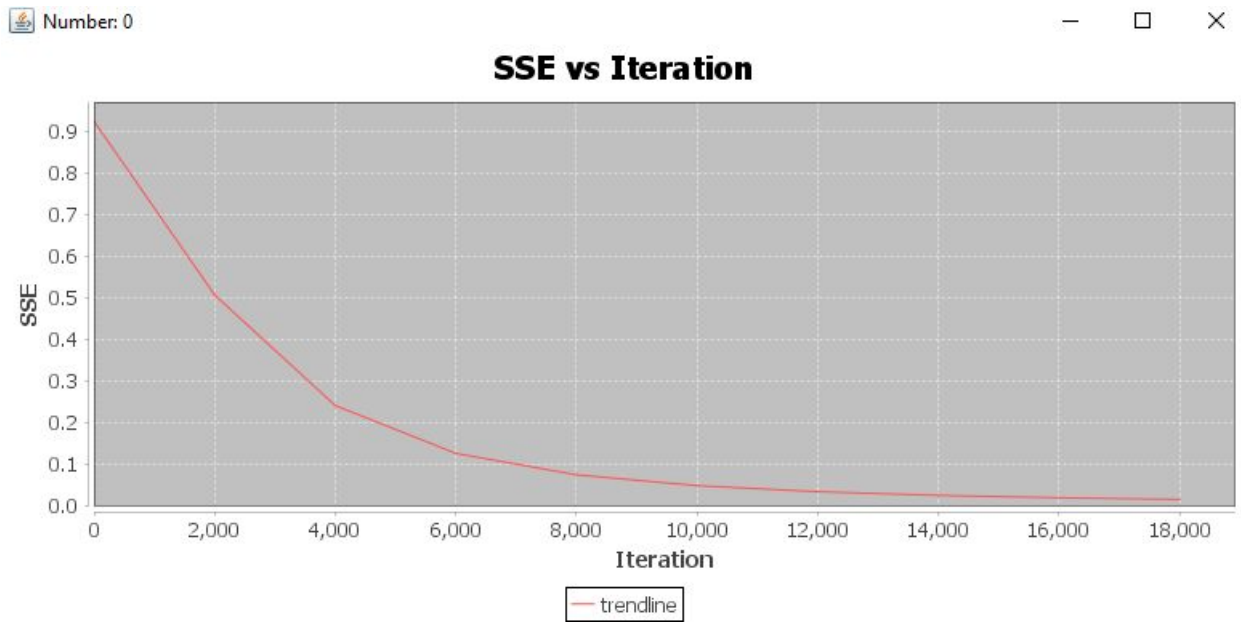
```
//layer 1
double[][] deltaE1 = np.multiply(np.dot(deltaE2, np.T(W2)), np.subtract(1.0,np.power(Z1, 2)));

double[][] dW1 = np.divide(np.dot(np.T(X),deltaE1),nSamples);

double[][] db1 = np.divide(deltaE1, nSamples);
```

Apply the weight deltas then repeat the loop

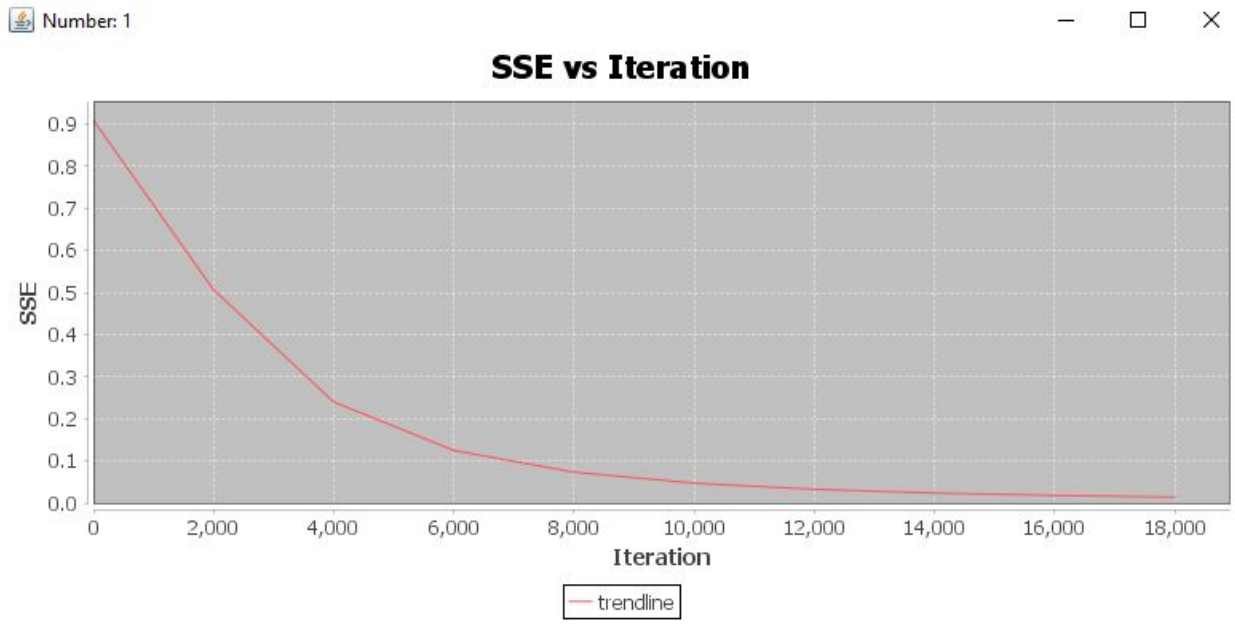
```
W1 = np.subtract(W1, np.multiply(learnRate, dW1));
b1 = np.subtract(b1, np.multiply(learnRate, db1));

W2 = np.subtract(W2, np.multiply(learnRate, dW2));
b2 = np.subtract(b2, np.multiply(learnRate, db2));
```

Results

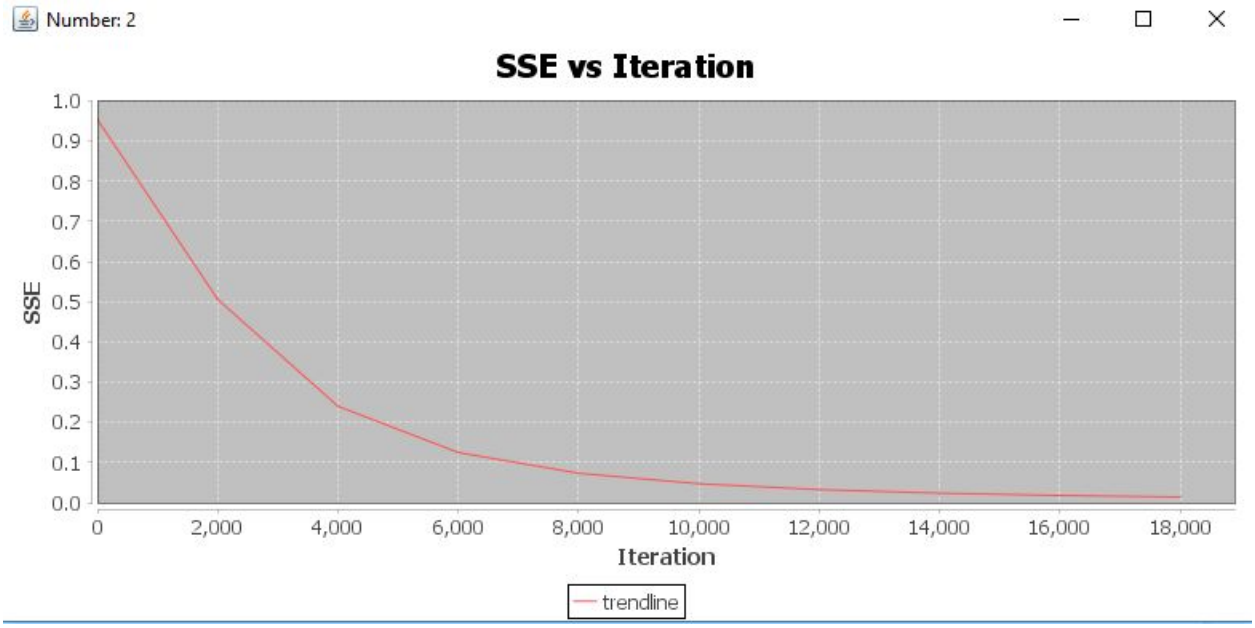
[0.9006024853294066, 0.012046970948388015, 0.012043686775581573, 0.012043758550926504, 0.012056474278545429, 0.012051186202137637, 0.012011701119102127, 0.012049114739987443, 0.012048267809411081, 0.012051249841754236,]



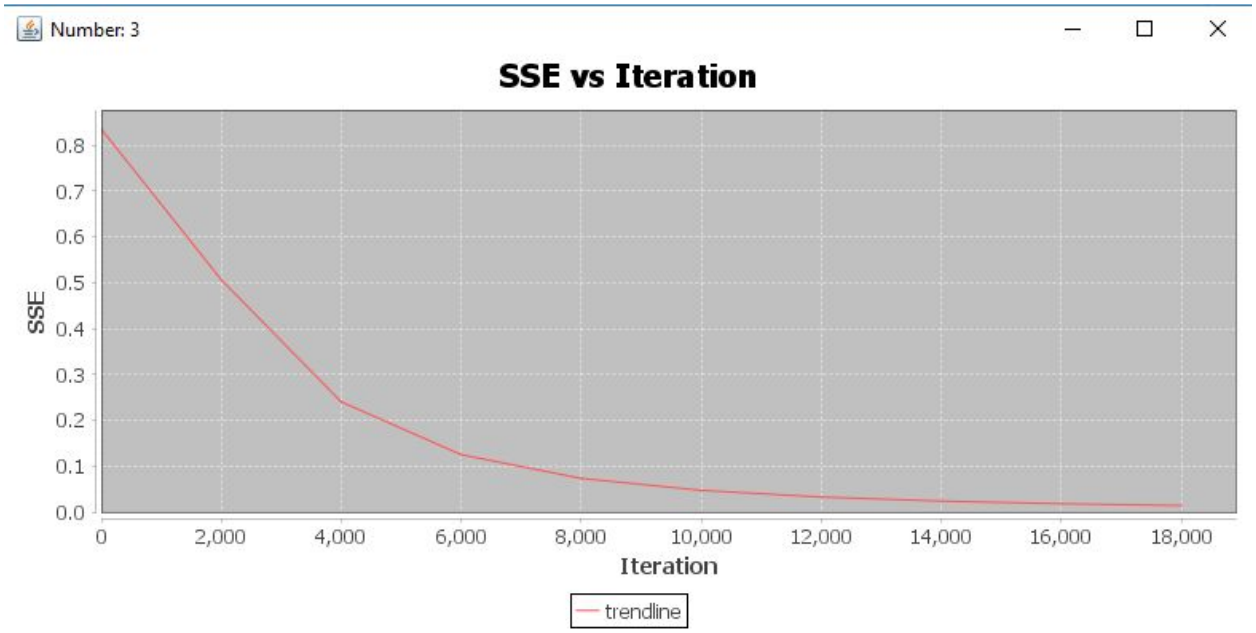
[0.011920910696172847, 0.9005051037966858, 0.011959480485310946, 0.011963997653696993, 0.011964526488619738, 0.011963435143629029, 0.011932606225997532, 0.011964921759101294, 0.011965058642546267, 0.011966422341596838,]



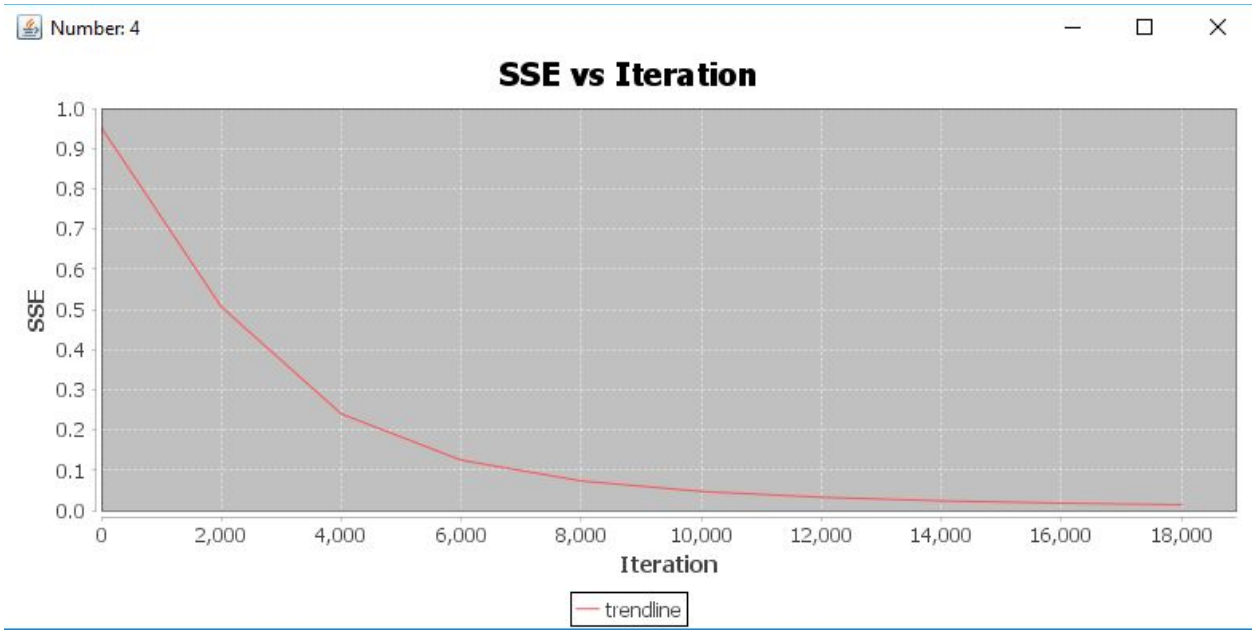
[0.011925381792716766, 0.011967565304743943, 0.9005122040095773, 0.011968493684385606, 0.01196970859050178, 0.011968181528746078, 0.011937075660227683, 0.011969592240692018, 0.011969600805196216, 0.011971114553209832,]



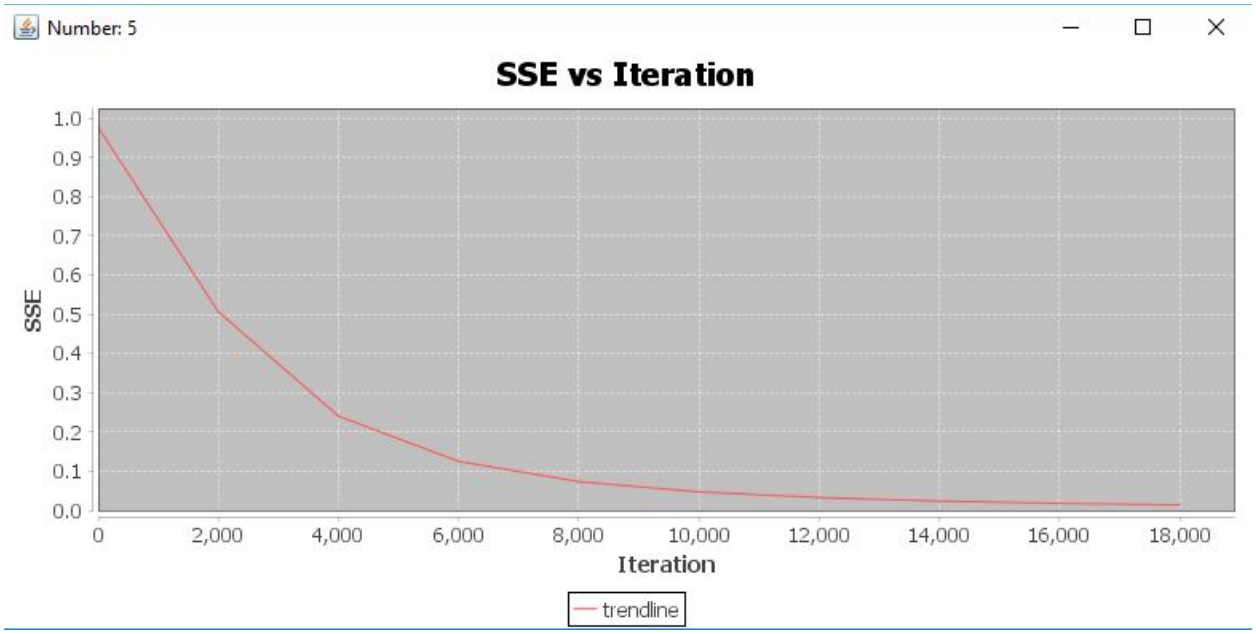
[0.011915973686079854, 0.011958148697358832, 0.01195368091721484, 0.9004985162322119, 0.011960250080049799, 0.01195835930867858, 0.011928793233463388, 0.011960943631138378, 0.011960488150142045, 0.011962227628389267,]



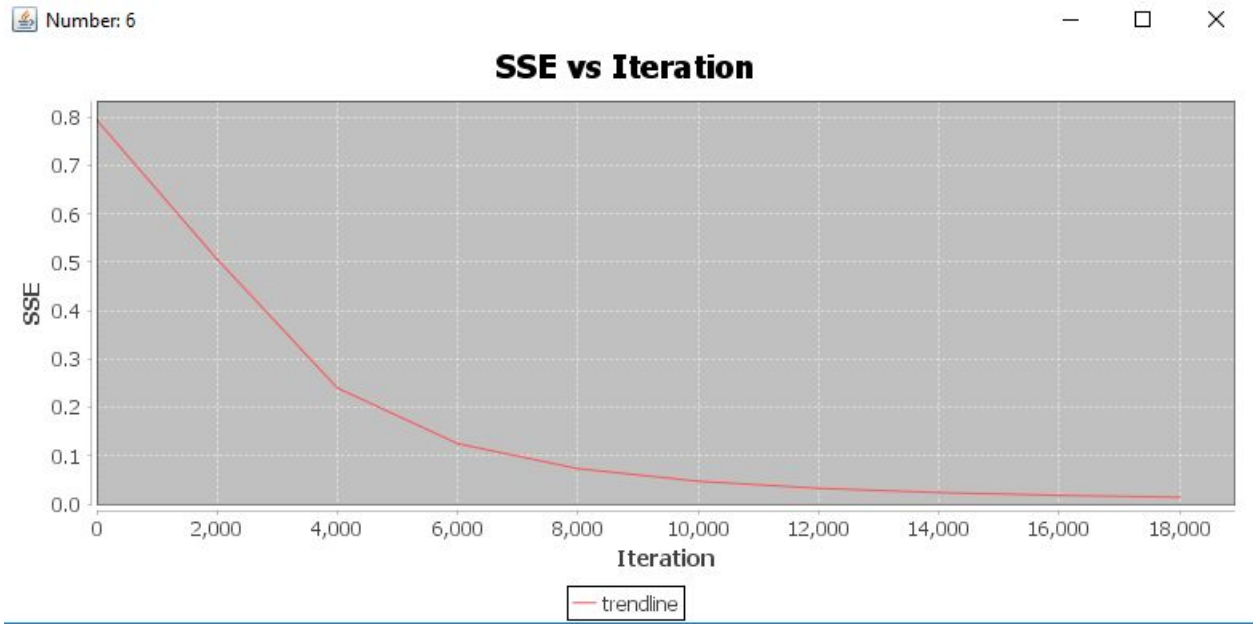
[0.011912258668296516, 0.01195431011166149, 0.011950379008638377, 0.011954964396308864, 0.9004904480013473, 0.01195493796817214, 0.011923579418814565, 0.011956114064969027, 0.011956255684508279, 0.011958211758878973,]



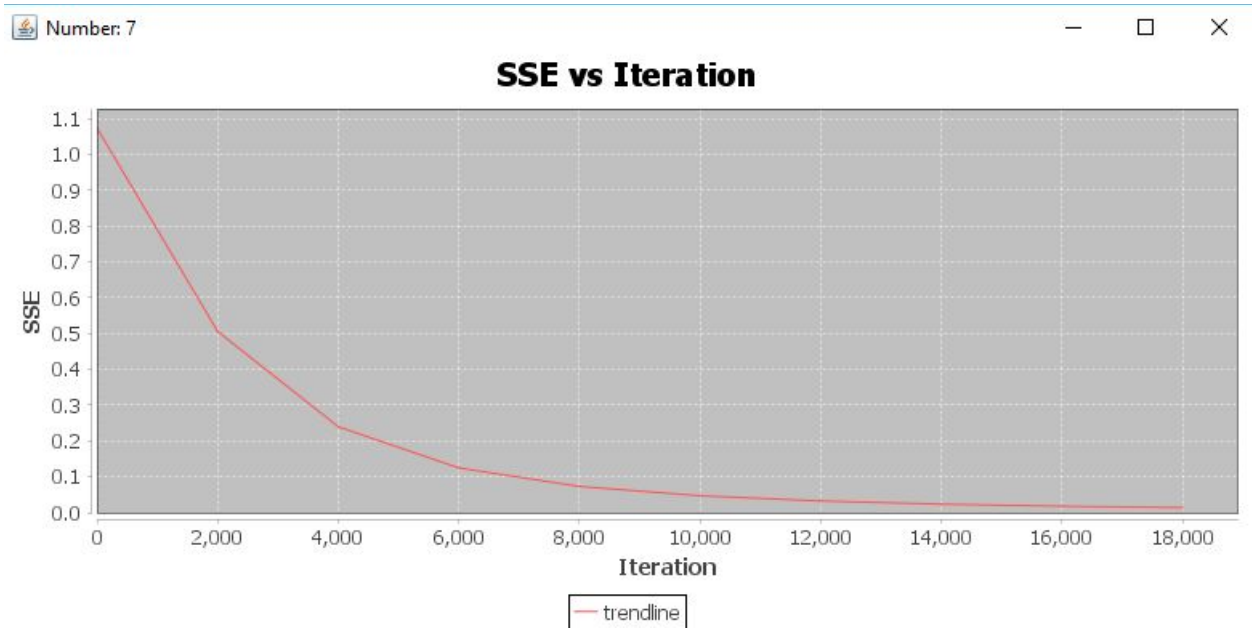
[0.011914007865211717, 0.011956624853202697, 0.011953197884171778, 0.011956840553446342, 0.011959295180787463, 0.9005003156275316, 0.01192541229319077, 0.01195830024287787, 0.011958261029264951, 0.011959982032600416,]



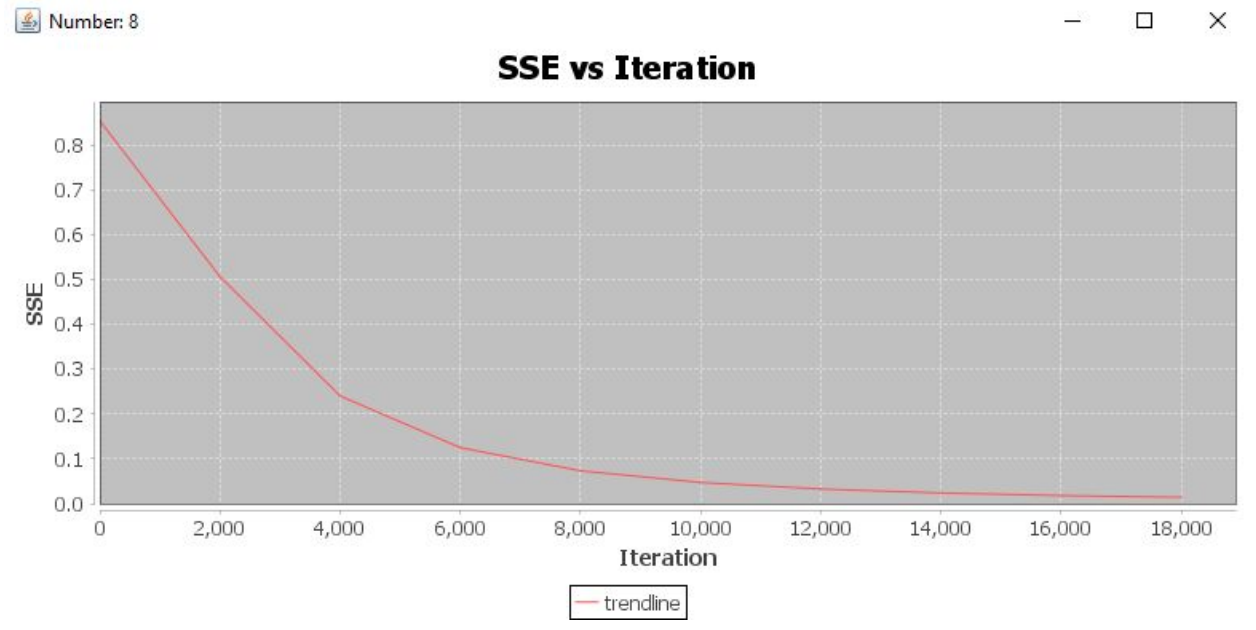
[0.011968878156666507, 0.012013076300634397, 0.012021549770832233, 0.012031934239838473, 0.012029614962112826, 0.012027034464441406, 0.9005820487918709, 0.012034414277566081, 0.012030566228765869, 0.012022323495134173,]



[0.011911987266028472, 0.011954020806963657, 0.011949917365740121, 0.011954635080399241, 0.011955869321177584, 0.01195453429903055, 0.011923266020258236, 0.9004943116863555, 0.011955882695558761, 0.011957935140369103,]



[0.011912415510177354, 0.011954424885793067, 0.011950401642420063, 0.01195517484348139, 0.011956351123803383, 0.011955076444078696, 0.011923778943721394, 0.011956296571985381, 0.9004972909942777, 0.011958433257455294,]



[0.011912059380073156, 0.011953975476863594, 0.011949781241338902, 0.011954621634773349, 0.011955697847025694, 0.011954536836318917, 0.011923263306880292, 0.011955705994456127, 0.011955911074837327, 0.9004925077240994,]