# Lukas Fu Homework 3

# Chaotic Time Series - Matlab Code

Lukas Fu

## Matlab Code

### Main

```matlab
clear;clc;clf;

% initialize variables
xTrain = load('training-set.csv');
xTest = load('test-set-5.csv');
nInput = size(xTrain,1);
nReservoir = 500;
kRidge = 0.01;
[wIn,wReservoir] = InitializeNetwork(nInput,nReservoir);

predictTimeStep = 500;

wOut = TrainReservoir(wIn,wReservoir,kRidge,xTrain);
predictionOutput = Predict(wIn,wReservoir,xTest,wOut,predictTimeStep);
hold on
grid on
plot3(predictionOutput(1,:),predictionOutput(2,:),predictionOutput(3,:), ...
    'b',LineWidth=0.01)
plot3(xTest(1,:),xTest(2,:),xTest(3,:),'r')
v = [-2 3 2];
[caz,cel] = view(v);
legend('Test Data','Prediction')
%% csv write
writematrix(predictionOutput(2,:),'prediction.csv')
```

### Initialize Network

```matlab
function [wIn,wReservoir] = InitializeNetwork(nInput,nReservoir)
variance1 = sqrt(0.002);
variance2 = 2/nReservoir;
wIn = normrnd(0,variance1,nReservoir,nInput);
wReservoir = normrnd(0,variance2,nReservoir,nReservoir);
end
```

### Train Reservoir

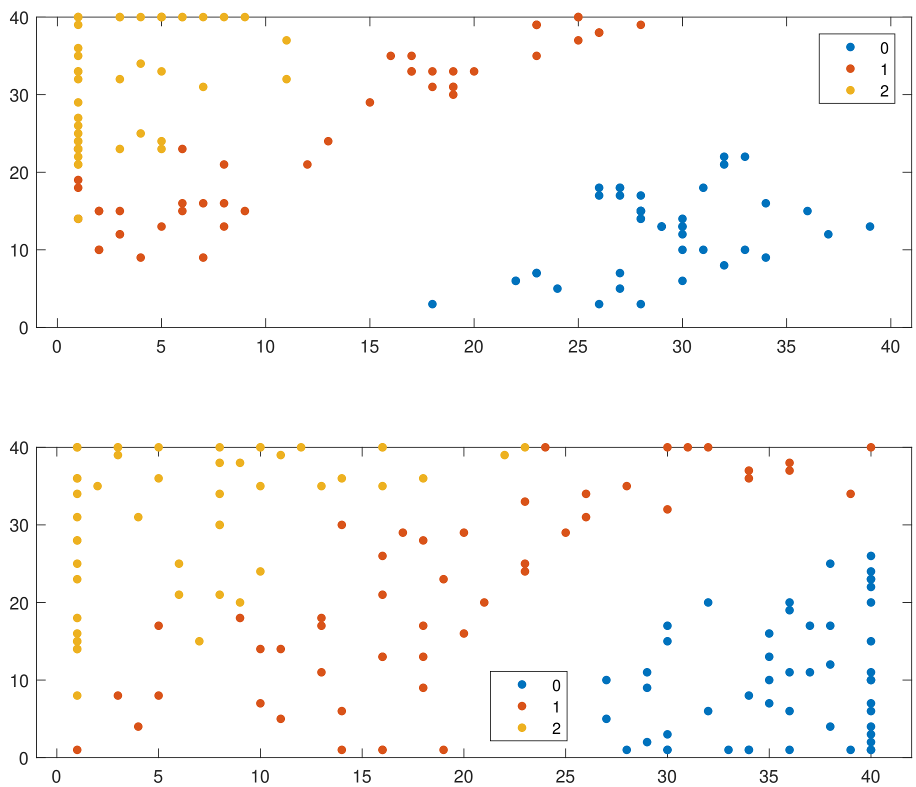
```matlab
function wOut = TrainReservoir(wIn,wReservoir,kRidge,xTrain)
    nReservoir = size(wReservoir,1);
    trainT = size(xTrain,2);
    R = zeros(nReservoir,trainT);
    rNext = zeros(nReservoir,1);

```

# Self-Organizing Maps

Lukas Fu

The figure shows the difference between the output using the randomly initialized weights compared to the output after 10 epochs of training. While the clustering is not very clear, the regions are somewhat clearly defined. It does seem like the program is struggling with separating 1 and 2, seeing as they occasionally intrude on each others areas. The empty boundary between 0 and 1 is still present in the lower figure, although to a lesser extent. I find it likely that both of these could be solved by iterating more, since the direction of movement shown between the epoch 0 and 10 tend toward filling out the output space and more clearly differentiating between the classes.




Figur 1: The upper figure shows the output using the randomly chosen weights for the 0:th epoch. The lower figure shows the result of iterating the learning rule for 10 epochs.

# Self-Organizing Maps - Matlab Code

Lukas Fu

## Matlab Code

### Main

```
clear; clc; clf;

nEpoch = 10;
eta = 0.1;
sigma = 10;
irisData = load('iris-data.csv');
irisLabels = load('iris-labels.csv');
maxIrisData = max(irisData);
stdIrisData = irisData/maxIrisData;

nPattern = size(irisData,1);
outputDimensions = 40; % dimensions as in matrix dimensions dim x dim

weights = zeros(40,40,4);
for k = 1:size(weights,1)
    for l = 1:size(weights,2)
        for m = 1:size(weights,3)
            weights(k,l,m) = rand;
        end
    end
end

for epoch = 1:nEpoch

    for trial = 1:nPattern
        iPattern = 1 + fix(rand*nPattern);
        pattern(1,1,:) = irisData(iPattern,:);
        [i0,j0] = FindWinningNeuron(pattern,weights);
        i0 = i0 + normrnd(0,0.02);
        j0 = j0 + normrnd(0,0.02);
        r0 = [i0 j0];

        for i = 1:outputDimensions
            for j = 1:outputDimensions
                r = [i j];
                distanceToR0 = vecnorm(r-r0);

                if distanceToR0 < 3 * sigma
                    h = exp(-distanceToR0^2 / (2*sigma^2));
                    dw = eta * h * (pattern - weights(i,j,:));
                    weights(i,j,:) = weights(i,j,:) + dw;
```