|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **ASSIGNMENT COVERSHEET** | | | | UTS LOGO | | | |
| **UTS: ENGINEERING & INFORMATION TECHNOLOGY** | | | | | | | |
|  | **NAME OF STUDENT(s) (PRINT CLEARLY)**  Joel Cappelli  *SURNAME FIRST NAME* | | | | | | **STUDENT ID(s).**  12137384 |
| **STUDENT EMAIL**  joel.cappell@gmail.com | | | | **STUDENT CONTACT NUMBER**  0422 384 087 | | | |
| **NAME OF TUTOR** | | | **TUTORIAL GROUP** | | | **DUE DATE**  22 April 2016 | |
| **ASSESSMENT ITEM NUMBER/ TITLE**  Hung Nguyen  49275 NEURAL NETWORKS AND FUZZY LOGIC Assignment 1 | | | | | | | |
| 🗆 I confirm that I have read, understood and followed the guidelines for assignment submission and presentation on page 2 of this cover sheet.  🗆 I confirm that I have read, understood and followed the advice in my Subject Outline about assessment requirements.  🗆 I understand that if this assignment is submitted after the due date it may incur a penalty for lateness unless I have previously had an extension of time approved and have attached the written confirmation of this extension.  **Declaration of Originality**: The work contained in this assignment, other than that specifically attributed to another source, is that of the author(s) and has not been previously submitted for assessment. I understand that, should this declaration be found to be false, disciplinary action could be taken and penalties imposed in accordance with University policy and rules. In the statement below, I have indicated the extent to which I have collaborated with others, whom I have named.  **Statement of Collaboration**:  22 April 2016  **Signature of Student(s) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**Date**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | | | | | | | |
| - -- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - | | | | | | | |
| **ASSIGNMENT RECEIPT** | | | | To be completed by the student if a receipt is required | | | |
| **SUBJECT NAME/NUMBER** | | **NAME OF TUTOR**  49275 NEURAL NETWORKS AND FUZZY LOGIC | | | | | |
| **SIGNATURE OF TUTOR** | | | | | **RECEIVED DATE**  22 April 2016 | | |

# **49275 NEURAL NETWORKS AND FUZZY LOGIC**

## Assignment 1

### Joel Cappelli

### 12137384 21 April 2016

## Qu 1.1

Discrete Perceptron Training

Initial weights:

Final weights after 60 steps:

The following formulae were used to compute cycle and pattern error.

There are 6 training patterns in the question. For every 1 cycle, there are 6 updates to the weights vector. Weights were updated using the discrete perceptron learning rule with a constant learning rate of 0.05 and augmented inputs of -1.

Figure 1 and Figure 2 show the cycle and pattern error curves over 10 cycles (60 steps) of training. classifies the entire training set perfectly. In fact after 3 cycles, .

Classification with :



Figure : Qu1.1 Cycle error curve



Figure : Qu1.1 Pattern error curve

## Qu 1.2

Continuous Perceptron Training

The following formulae were used to compute cycle and pattern error.

There are 6 training patterns in the question. For every 1 cycle, there are 6 updates to the weights vector. Weights were updated using the delta training rule with a constant learning rate of 0.5 and augmented inputs of 1.

Initial weights:

Weights after 1 cycle and 50 cycles:

Desired output:

Classification with :

Cycle Error with :

Classification with :

Cycle Error with :

Figure 3 and Figure 4 how the cycle and pattern error curves over 50 cycles (300 steps) of training. The cycle error is reducing over each cycle as the weights vector is converging to a minimum in the weights space. The classification with is closer to the desired output than . The discrete perceptron training allows for perfect classification because of the use of the logic threshold unit which has saturated +1/-1 outputs and the desired outputs are +1/-1 while the bipolar logistic function asymptotically approaches +/-1.



Figure : Qu1.2 Cycle error curve



Figure : Qu1.2 Pattern error curve

## Qu 2.1

Given the following in the question

Where is given as the relation between credit limits and average account balance and is relation between average account balance and profits.

Relation can be found between credit limits and profits using compositional operations.

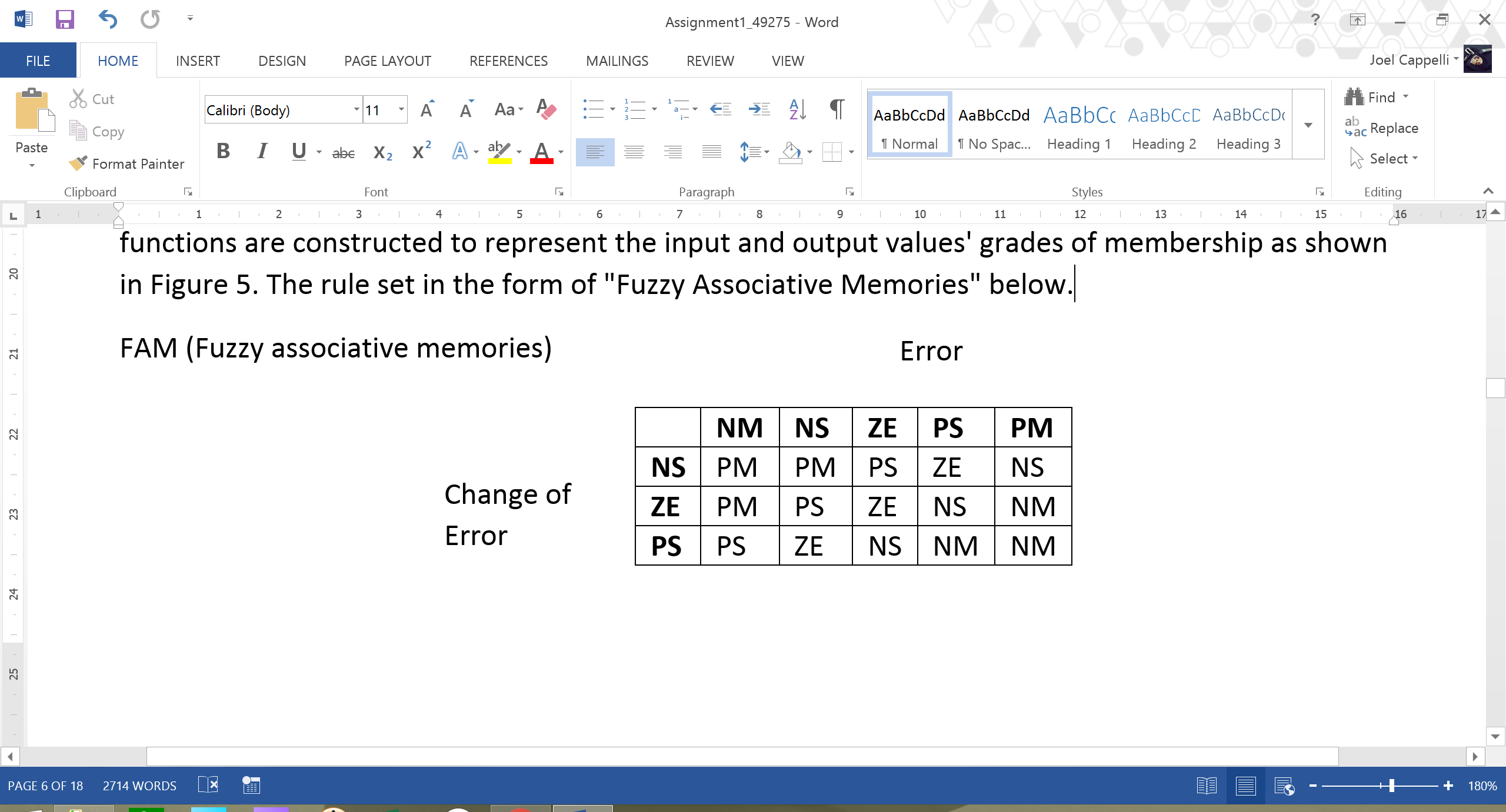
Qu 2.1.1 R Fuzzy Relation - Max-min compositional

Qu 2.1.2 R Fuzzy Relation - Sum-product compositional (using matrix multiplication for sum-product)

If the sum-product compositional operations are used to recover variables, the variables should be normalised

## Qu 2.2

To represent the error input to the controller, a set of linguistic variables is chosen to represent 5 degrees of error, 3 degrees of change of error, and 5 degrees of armature voltage. Membership functions are constructed to represent the input and output values' grades of membership as shown in Figure 5. The rule set in the form of "Fuzzy Associative Memories" below.



Error input = 3.25, Error input = -0.2

With controller constant gains GE =1, GCE = 1, GU = 1



Figure : Qu2.2 Membership functions of laser beam alignment system

In fuzzy notation;

Using the FAM above given in question and inference for each associated method;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | U(MOM [sum-prod])) | U(COA [Max-min]) |
| Rule 1 |  |  |  |  |
| Rule 2 |  |  |  |  |
| Rule 3 |  |  |  |  |
| Rule 4 |  |  |  |  |

According to the MOM (Mean of maximum) method, the output is:

MOM Method (sum-prod) - Defuzzified Output for Error = 3.25, deltaError = -0.2

Output voltage = -6.1395V

According to the COA (Centre of area) method, the output can be found using Max-min inference as:

Defuzzification Piecewise Functions

Function 1 between -10 and -7.66667 is u(U) = 0.666667

Area 1 = 1.55556

Moment 1 = -13.7407

Function 2 between -7.66667 and -6.66667 is u(U) = -0.25\*u + -1.25

Area 2 = 0.541667

Moment 2 = -3.90278

Function 3 between -6.66667 and -2.66667 is u(U) = 0.416667

Area 3 = 1.66667

Moment 3 = -7.77778

Function 4 between -2.66667 and -2.33333 is u(U) = -0.25\*u + -0.25

Area 4 = 0.125

Moment 4 = -0.313272

Function 5 between -2.33333 and 3.33333 is u(U) = 0.333333

Area 5 = 1.88889

Moment 5 = 0.944444

Function 6 between 3.33333 and 5 is u(U) = -0.2\*u + 1

Area 6 = 0.277778

Moment 6 = 1.08025

COA Method (max-min) - Defuzzified Output for Error = 3.25, deltaError = -0.2

Total Area = 6.0556

Total Moment = -23.7099

Output voltage = -3.9154V





Figure : Qu2.2 Defuzzification of controller output

## Source code

clear all;

close all;

clc;

%% Neural Networks Assignment 1

% Joel Cappelli

% 12137384

%% Qu1.1

x1 = [0.8;0.5;0.0;0.1];

x2 = [0.2;0.1;1.3;0.9];

x3 = [0.9;0.7;0.3;0.3];

x4 = [0.2;0.7;0.8;0.2];

x5 = [1.0;0.8;0.5;0.7];

x6 = [0.0;0.2;0.3;0.6];

setX = [x1 x2 x3 x4 x5 x6];

augInput = -1;

x = [setX; augInput\*ones(1,size(setX,2))];

trainingSize = size(x,2);

d1 = 1;

d2 = -1;

d3 = 1;

d4 = -1;

d5 = 1;

d6 = -1;

d = [d1 d2 d3 d4 d5 d6];

eta = 0.05;

wts = [0.0976;0.8632;-0.3296;0.3111;-0.2162];

sse = Inf;

tol = 1e-2;

maxCycles = 10;

patternErrors = zeros(maxCycles,trainingSize);

patternErrorsOverSteps = zeros(maxCycles\*trainingSize,1);

wtsArray = zeros(size(wts,1),maxCycles\*trainingSize + 1);

cycle = 1;

wtsArray(:,1) = wts;

while(cycle <= maxCycles)

for i = 1:trainingSize

v = wts'\*x(:,i);

z = TLU(v);

error = d(i)-z;

r = error;

patternErrors(cycle,i) = 0.5\*error\*error;

patternErrorsOverSteps((cycle-1)\*trainingSize+i) = 0.5\*error\*error;

wts = wts + eta\*x(:,i)\*r;

wtsArray(:,(cycle-1)\*trainingSize+i+1) = wts;

end

cycle = cycle +1;

end

cycleError = sum(patternErrors,2);

figure;

plot(cycleError);

title('Qu1.1 - Cycle Error');

grid on;

xlabel('Cycle');

figure;

plot(patternErrorsOverSteps);

title('Qu1.1 - Pattern Error Over Steps');

grid on;

xlabel('Step');

fprintf('\nQu1.1 Discrete Perceptron Training\n');

disp('Initial weights:');

disp(wtsArray(:,1));

fprintf('Final weights after %d steps:\n',maxCycles\*trainingSize);

disp(wtsArray(:,end));

figure;

for i = 1:trainingSize

subplot(trainingSize,1,i);plot(patternErrors(:,i));

grid on;

title(strcat('Pattern ',num2str(i)));

end

xlabel('Cycle');

%% Qu1.2

fprintf('Qu1.2 Continuous Perceptron Training\n');

%re-init

setX = [x1 x2 x3 x4 x5 x6];

augInput = 1;

x = [setX; augInput\*ones(1,size(setX,2))];

trainingSize = size(x,2);

eta = 0.5;

wts = [0.0976;0.8632;-0.3296;0.3111;-0.2162];

maxCycles = 50;

patternErrors = zeros(maxCycles,trainingSize);

patternErrorsOverSteps = zeros(maxCycles\*trainingSize,1);

wtsArray = zeros(size(wts,1),maxCycles\*trainingSize + 1);

cycle = 1;

wtsArray(:,1) = wts;

while(cycle <= maxCycles)

for i = 1:trainingSize

v = wts'\*x(:,i);

z = bipolarLog(v);

error = d(i)-z;

r = error\*bipolarLog(z,'deriv');

patternErrors(cycle,i) = 0.5\*error\*error;

patternErrorsOverSteps((cycle-1)\*trainingSize+i) = 0.5\*error\*error;

wts = wts + eta\*x(:,i)\*r;

wtsArray(:,(cycle-1)\*trainingSize+i+1) = wts;

end

cycle = cycle +1;

end

cycleError = sum(patternErrors,2);

fprintf('Desired output:\n')

disp(d);

disp('Initial weights:');

disp(wtsArray(:,1));

fprintf('Weights after %d steps (%d cycle):\n',6,1);

disp(wtsArray(:,7));

fprintf('Classification with w%d:\n',7);

classW7 = zeros(size(d));

for i = 1:trainingSize

classW7(i) = bipolarLog(wtsArray(:,7)'\*x(:,i));

end

disp(classW7);

fprintf('Cycle Error with w%d:\n',7);

disp(cycleError(1));

fprintf('Weights after %d steps (%d cycles):\n',300,50);

disp(wtsArray(:,301));

fprintf('Classification with w%d:\n',301);

classW301 = zeros(size(d));

for i = 1:trainingSize

classW301(i) = bipolarLog(wtsArray(:,301)'\*x(:,i));

end

disp(classW301);

fprintf('Cycle Error with w%d:\n',301);

disp(cycleError(50));

figure;

plot(cycleError);

title('Qu1.2 - Cycle Error');

grid on;

xlabel('Cycle');

figure;

plot(patternErrorsOverSteps);

title('Qu1.2 - Pattern Error Over Steps');

grid on;

xlabel('Step');

figure;

for i = 1:trainingSize

subplot(trainingSize,1,i);plot(patternErrors(:,i));

grid on;

title(strcat('Pattern ',num2str(i)));

end

xlabel('Cycle');

%% Qu 2.1

Credit\_limits = [500, 1000, 1500, 2000];

Average\_account\_balance = [20, 50, 100, 500, 1000, 1200];

Profits = [-50, 0, 50, 100, 500];

[mcl,ncl]=size(Credit\_limits);

[maab,naab]=size(Average\_account\_balance);

[mp,np]=size(Profits);

A = [

0.8 1.0 0.6 0.2 0.0 0.0;...

0.2 0.3 0.5 0.8 0.1 0.0;...

0.1 0.2 0.4 0.9 0.6 0.1;...

0.1 0.1 0.4 0.8 0.8 0.3

];

E = [

0.8 0.9 0.7 0.1 0.0;...

0.7 1.0 0.8 0.2 0.0;...

0.5 0.9 0.9 0.5 0.1;...

0.2 0.5 0.7 0.8 0.9;...

0.1 0.4 0.6 0.9 0.8;...

0.0 0.3 0.5 0.8 0.7...

];

% R = A o E = (Credit\_limits X Average\_account\_balance) o (Average\_account\_balance X Profits)

% columns of Q with rows of P

% using max-min composition

R\_maxmincomp = zeros(ncl,np);

for i = 1:ncl

for j = 1:np

R\_maxmincomp(i,j) = max(min(A(i,:),E(:,j)'));

end

end

disp('Qu 2.1.1 R Fuzzy Relation - Max-min compositional');

R\_maxmincomp

R\_sumprodcomp = zeros(ncl,np);

for i = 1:ncl

for j = 1:np

R\_sumprodcomp(i,j) = dot(A(i,:),E(:,j)');

end

end

disp('Qu 2.1.2 R Fuzzy Relation - Sum-product compositional');

max\_R = max(max(R\_sumprodcomp));

R\_sumprodcomp = R\_sumprodcomp/max\_R

%% Qu 2.2

fprintf('\nQu2.2 Fuzzy logic Controller\n\n');

error = 3.25;

deltaError = -0.2;

minErrorRange = -5;

maxErrorRange = 5;

maxError = 1;

minDeltaErrorRange = -2;

maxDeltaErrorRange = 2;

maxDeltaError = 1;

minOutputRange = -10;

maxOutputRange = 10;

maxOutput = 1;

setError = {'NM','NS','ZE','PS','PM'};

errorTextXpos = [-4 -0.8 0 0.8 4];%(minErrorRange - [-4 -0.8 0 0.8 4])./(minErrorRange-maxErrorRange);

setDeltaError = {'NS','ZE','PS'};

deltaErrorTextXpos = [-0.6 0 0.6];%(minDeltaErrorRange - [-0.6 0 0.6])./(minDeltaErrorRange-maxDeltaErrorRange);

degError = size(setError,2);

degDeltaError = size(setDeltaError,2);

if(degError > degDeltaError)

setOutput = setError;

else

setOutput = setDeltaError;

end

contOutputTextXpos = [-9 -5 0 5 9];%(minOutputRange -[-8 -4 0 4 8])./(minOutputRange-maxOutputRange);

degOutput = size(setOutput,2);

FAM\_error\_DeltaError = {'PM','PM','PS','ZE','NS';...

'PM','PS','ZE','NS','NM';...

'PS','ZE','NS','NM','NM'};

cellfind = @(string)(@(cell\_contents)(strcmp(string,cell\_contents)));

logical\_col = cellfun(cellfind('NS'),setError);

logical\_row = cellfun(cellfind('NS'),setDeltaError);

findOutput = cellfun(cellfind(FAM\_error\_DeltaError(logical\_row,logical\_col)),setOutput);

test = setOutput(findOutput);

MAX = 1;

LINEAR = 2;

MIN = 3;

errorFuzzySet1\_Func = [MAX LINEAR MIN];

errorFuzzySet1\_output = [1 1 0 0];

errorFuzzySet1\_inputBreakPts = [minErrorRange -4 -0.8 maxErrorRange];

errorFuzzySet2\_Func = [LINEAR LINEAR MIN];

errorFuzzySet2\_output = [0 1 0 0];

errorFuzzySet2\_inputBreakPts = [minErrorRange -0.8 -0.2 maxErrorRange];

errorFuzzySet3\_Func = [MIN LINEAR LINEAR MIN];

errorFuzzySet3\_output = [0 0 1 0 0];

errorFuzzySet3\_inputBreakPts = [minErrorRange -0.8 0 0.8 maxErrorRange];

errorFuzzySet4\_Func = [MIN LINEAR LINEAR];

errorFuzzySet4\_output = [0 0 1 0];

errorFuzzySet4\_inputBreakPts = [minErrorRange 0.2 0.8 maxErrorRange];

errorFuzzySet5\_Func = [MIN LINEAR MAX];

errorFuzzySet5\_output = [0 0 1 1];

errorFuzzySet5\_inputBreakPts = [minErrorRange 0.8 4 maxErrorRange];

errorFuzzySet\_Funcs = {errorFuzzySet1\_Func;errorFuzzySet2\_Func;errorFuzzySet3\_Func;errorFuzzySet4\_Func;errorFuzzySet5\_Func};

errorFuzzySet\_output = {errorFuzzySet1\_output;errorFuzzySet2\_output;errorFuzzySet3\_output;errorFuzzySet4\_output;errorFuzzySet5\_output};

errorFuzzySet\_inputBreakPts = {errorFuzzySet1\_inputBreakPts;errorFuzzySet2\_inputBreakPts;errorFuzzySet3\_inputBreakPts;errorFuzzySet4\_inputBreakPts;errorFuzzySet5\_inputBreakPts};

numErrorFuzzySets = size(errorFuzzySet\_Funcs,1);

deltaErrorFuzzySet1\_Func = [MAX LINEAR MIN];

deltaErrorFuzzySet1\_output = [1 1 0 0];

deltaErrorFuzzySet1\_inputBreakPts = [minDeltaErrorRange -0.6 0 maxDeltaErrorRange];

deltaErrorFuzzySet2\_Func = [MIN LINEAR LINEAR MIN];

deltaErrorFuzzySet2\_output = [0 0 1 0 0];

deltaErrorFuzzySet2\_inputBreakPts = [minDeltaErrorRange -0.6 0 0.6 maxDeltaErrorRange];

deltaErrorFuzzySet3\_Func = [MIN LINEAR MAX];

deltaErrorFuzzySet3\_output = [0 0 1 1];

deltaErrorFuzzySet3\_inputBreakPts = [minDeltaErrorRange 0 0.6 maxDeltaErrorRange];

deltaErrorFuzzySet\_Funcs = {deltaErrorFuzzySet1\_Func;deltaErrorFuzzySet2\_Func;deltaErrorFuzzySet3\_Func};

deltaErrorFuzzySet\_output = {deltaErrorFuzzySet1\_output;deltaErrorFuzzySet2\_output;deltaErrorFuzzySet3\_output};

deltaErrorFuzzySet\_inputBreakPts = {deltaErrorFuzzySet1\_inputBreakPts;deltaErrorFuzzySet2\_inputBreakPts;deltaErrorFuzzySet3\_inputBreakPts};

numdeltaErrorFuzzySets = size(deltaErrorFuzzySet\_Funcs,1);

contOutputFuzzySet1\_Func = [MAX LINEAR MIN];

contOutputFuzzySet1\_output = [1 1 0 0];

contOutputFuzzySet1\_inputBreakPts = [minOutputRange -9 -5 maxOutputRange];

contOutputFuzzySet2\_Func = [LINEAR LINEAR MIN];

contOutputFuzzySet2\_output = [0 1 0 0];

contOutputFuzzySet2\_inputBreakPts = [minOutputRange -5 -1 maxOutputRange];

contOutputFuzzySet3\_Func = [MIN LINEAR LINEAR MIN];

contOutputFuzzySet3\_output = [0 0 1 0 0];

contOutputFuzzySet3\_inputBreakPts = [minOutputRange -5 0 5 maxOutputRange];

contOutputFuzzySet4\_Func = [MIN LINEAR LINEAR];

contOutputFuzzySet4\_output = [0 0 1 0];

contOutputFuzzySet4\_inputBreakPts = [minOutputRange 1 5 maxOutputRange];

contOutputFuzzySet5\_Func = [MIN LINEAR MAX];

contOutputFuzzySet5\_output = [0 0 1 1];

contOutputFuzzySet5\_inputBreakPts = [minOutputRange 5 9 maxOutputRange];

contOutputFuzzySet\_Funcs = {contOutputFuzzySet1\_Func;contOutputFuzzySet2\_Func;contOutputFuzzySet3\_Func;contOutputFuzzySet4\_Func;contOutputFuzzySet5\_Func};

contOutputFuzzySet\_output = {contOutputFuzzySet1\_output;contOutputFuzzySet2\_output;contOutputFuzzySet3\_output;contOutputFuzzySet4\_output;contOutputFuzzySet5\_output};

contOutputFuzzySet\_inputBreakPts = {contOutputFuzzySet1\_inputBreakPts;contOutputFuzzySet2\_inputBreakPts;contOutputFuzzySet3\_inputBreakPts;contOutputFuzzySet4\_inputBreakPts;contOutputFuzzySet5\_inputBreakPts};

numContOutputFuzzySets = size(contOutputFuzzySet\_Funcs,1);

figure;

for i = 1:numErrorFuzzySets

str = setError{i};

subplot(3,1,1); plot(errorFuzzySet\_inputBreakPts{i},errorFuzzySet\_output{i},'b');

subplot(3,1,1); text(errorTextXpos(i),1,str);

hold on;

end

subplot(3,1,1);plot([error error],[0 1],'--r','linewidth',2);

grid on;

ylabel('Error');

for i = 1:numdeltaErrorFuzzySets

str = setDeltaError{i};

subplot(3,1,2); plot(deltaErrorFuzzySet\_inputBreakPts{i},deltaErrorFuzzySet\_output{i},'b');

subplot(3,1,2);text(deltaErrorTextXpos(i),1,str);

hold on;

end

subplot(3,1,2);plot([deltaError deltaError],[0 1],'--k','linewidth',2);

grid on;

ylabel('Delta Error');

for i = 1:numContOutputFuzzySets

str = setOutput{i};

subplot(3,1,3); plot(contOutputFuzzySet\_inputBreakPts{i},contOutputFuzzySet\_output{i},'b');

subplot(3,1,3);text(contOutputTextXpos(i),1,str);

hold on;

end

grid on;

ylabel('Controller Output');

inputs = minErrorRange:0.01:maxErrorRange;

output = zeros(size(inputs));

numinputs = size(inputs,2);

fuzzySet\_Error\_PM = polyval(linearEqu([0.8 0],[4 1]),error);

fuzzySet\_Error\_PS = polyval(linearEqu([0.8 1],[5 0]),error);

fuzzySet\_deltaError\_NS = polyval(linearEqu([-0.6 1],[0 0]),deltaError);

fuzzySet\_deltaError\_ZE = polyval(linearEqu([-0.6 0],[0 1]),deltaError);

logical\_col = cellfun(cellfind('PS'),setError);

logical\_row = cellfun(cellfind('NS'),setDeltaError);

findOutput = cellfun(cellfind(FAM\_error\_DeltaError(logical\_row,logical\_col)),setOutput);

Rule1\_output = setOutput(findOutput);

logical\_col = cellfun(cellfind('PS'),setError);

logical\_row = cellfun(cellfind('ZE'),setDeltaError);

findOutput = cellfun(cellfind(FAM\_error\_DeltaError(logical\_row,logical\_col)),setOutput);

Rule2\_output = setOutput(findOutput);

logical\_col = cellfun(cellfind('PM'),setError);

logical\_row = cellfun(cellfind('NS'),setDeltaError);

findOutput = cellfun(cellfind(FAM\_error\_DeltaError(logical\_row,logical\_col)),setOutput);

Rule3\_output = setOutput(findOutput);

logical\_col = cellfun(cellfind('PM'),setError);

logical\_row = cellfun(cellfind('ZE'),setDeltaError);

findOutput = cellfun(cellfind(FAM\_error\_DeltaError(logical\_row,logical\_col)),setOutput);

Rule4\_output = setOutput(findOutput);

Rule\_output = {Rule1\_output{:},Rule2\_output{:},Rule3\_output{:},Rule4\_output{:}};

UMOM\_Rule1\_E\_PS\_deltaE\_NS = fuzzySet\_Error\_PS\*fuzzySet\_deltaError\_NS;

UMOM\_Rule2\_E\_PS\_deltaE\_ZE = fuzzySet\_Error\_PS\*fuzzySet\_deltaError\_ZE;

UMOM\_Rule3\_E\_PM\_deltaE\_NS = fuzzySet\_Error\_PM\*fuzzySet\_deltaError\_NS;

UMOM\_Rule4\_E\_PM\_deltaE\_ZE = fuzzySet\_Error\_PM\*fuzzySet\_deltaError\_ZE;

UMOM\_vals = [UMOM\_Rule1\_E\_PS\_deltaE\_NS,UMOM\_Rule2\_E\_PS\_deltaE\_ZE,UMOM\_Rule3\_E\_PM\_deltaE\_NS,UMOM\_Rule4\_E\_PM\_deltaE\_ZE];

UCOA\_Rule1\_E\_PS\_deltaE\_NS = min(fuzzySet\_Error\_PS,fuzzySet\_deltaError\_NS);

UCOA\_Rule2\_E\_PS\_deltaE\_ZE = min(fuzzySet\_Error\_PS,fuzzySet\_deltaError\_ZE);

UCOA\_Rule3\_E\_PM\_deltaE\_NS = min(fuzzySet\_Error\_PM,fuzzySet\_deltaError\_NS);

UCOA\_Rule4\_E\_PM\_deltaE\_ZE = min(fuzzySet\_Error\_PM,fuzzySet\_deltaError\_ZE);

UCOA\_vals = [UCOA\_Rule1\_E\_PS\_deltaE\_NS,UCOA\_Rule2\_E\_PS\_deltaE\_ZE,UCOA\_Rule3\_E\_PM\_deltaE\_NS,UCOA\_Rule4\_E\_PM\_deltaE\_ZE];

disp('Fuzzification UMOM - sum-prod');

for i = 1:size(UMOM\_vals,2)

fprintf('Rule %d: %g/%s \n',i, UMOM\_vals(i),Rule\_output{i});

end

fprintf('\nMOM Method (sum-prod) - Defuzzified Output for Error = %g, deltaError = %g\n',error,deltaError);

outputVoltage = dot(UMOM\_vals,[0 -5 -5 -9])/sum(UMOM\_vals);

fprintf('Output voltage = %gV\n',outputVoltage);

fprintf('\n');

disp('Fuzzification UCOA - max-min');

for i = 1:size(UMOM\_vals,2)

fprintf('Rule %d: %g/%s \n',i, UCOA\_vals(i),Rule\_output{i});

end

UCOA\_Rule1\_inputBreakPts = [-5 linearPosx(linearEqu([-5 0],[0 1]),UCOA\_vals(1)) linearPosx(linearEqu([5 0],[0 1]),UCOA\_vals(1)) 5];

UCOA\_Rule1\_output = [0 UCOA\_vals(1) UCOA\_vals(1) 0];

figure;

for i = 1:numContOutputFuzzySets

str = setOutput{i};

subplot(2,2,1);plot(contOutputFuzzySet\_inputBreakPts{i},contOutputFuzzySet\_output{i},'b');

subplot(2,2,1);text(contOutputTextXpos(i),1,str);

hold on;

end

subplot(2,2,1);plot(UCOA\_Rule1\_inputBreakPts,UCOA\_Rule1\_output,'b','linewidth',3);

grid on;

title('MF1');

UCOA\_Rule2\_inputBreakPts = [-10 linearPosx(linearEqu([-10 0],[-5 1]),UCOA\_vals(2)) linearPosx(linearEqu([-5 1],[-1 0]),UCOA\_vals(2)) -1];

UCOA\_Rule2\_output = [0 UCOA\_vals(2) UCOA\_vals(2) 0];

for i = 1:numContOutputFuzzySets

str = setOutput{i};

subplot(2,2,2);plot(contOutputFuzzySet\_inputBreakPts{i},contOutputFuzzySet\_output{i},'b');

subplot(2,2,2);text(contOutputTextXpos(i),1,str);

hold on;

end

subplot(2,2,2);plot(UCOA\_Rule2\_inputBreakPts,UCOA\_Rule2\_output,'b','linewidth',3);

grid on;

title('MF2');

UCOA\_Rule3\_inputBreakPts = [-10 linearPosx(linearEqu([-10 0],[-5 1]),UCOA\_vals(3)) linearPosx(linearEqu([-5 1],[-1 0]),UCOA\_vals(3)) -1];

UCOA\_Rule3\_output = [0 UCOA\_vals(3) UCOA\_vals(3) 0];

for i = 1:numContOutputFuzzySets

str = setOutput{i};

subplot(2,2,3);plot(contOutputFuzzySet\_inputBreakPts{i},contOutputFuzzySet\_output{i},'b');

subplot(2,2,3);text(contOutputTextXpos(i),1,str);

hold on;

end

subplot(2,2,3);plot(UCOA\_Rule3\_inputBreakPts,UCOA\_Rule3\_output,'b','linewidth',3);

grid on;

title('MF3');

UCOA\_Rule4\_inputBreakPts = [-10 linearPosx(linearEqu([-9 1],[-5 0]),UCOA\_vals(4)) -5];

UCOA\_Rule4\_output = [UCOA\_vals(4) UCOA\_vals(4) 0];

for i = 1:numContOutputFuzzySets

str = setOutput{i};

subplot(2,2,4);plot(contOutputFuzzySet\_inputBreakPts{i},contOutputFuzzySet\_output{i},'b');

subplot(2,2,4);text(contOutputTextXpos(i),1,str);

hold on;

end

subplot(2,2,4);plot(UCOA\_Rule4\_inputBreakPts,UCOA\_Rule4\_output,'b','linewidth',3);

grid on;

title('MF4');

figure;

for i = 1:numContOutputFuzzySets

str = setOutput{i};

plot(contOutputFuzzySet\_inputBreakPts{i},contOutputFuzzySet\_output{i},'b');

text(contOutputTextXpos(i),1,str);

hold on;

end

plot(UCOA\_Rule1\_inputBreakPts,UCOA\_Rule1\_output,'b','linewidth',3);

for i = 1:numContOutputFuzzySets

plot(contOutputFuzzySet\_inputBreakPts{i},contOutputFuzzySet\_output{i},'b');

hold on;

end

plot(UCOA\_Rule2\_inputBreakPts,UCOA\_Rule2\_output,'b','linewidth',3);

for i = 1:numContOutputFuzzySets

plot(contOutputFuzzySet\_inputBreakPts{i},contOutputFuzzySet\_output{i},'b');

hold on;

end

plot(UCOA\_Rule3\_inputBreakPts,UCOA\_Rule3\_output,'b','linewidth',3);

for i = 1:numContOutputFuzzySets

plot(contOutputFuzzySet\_inputBreakPts{i},contOutputFuzzySet\_output{i},'b');

hold on;

end

plot(UCOA\_Rule4\_inputBreakPts,UCOA\_Rule4\_output,'b','linewidth',3);

grid on;

defuzz\_output = [UCOA\_vals(4) UCOA\_vals(4) UCOA\_vals(2) UCOA\_vals(2) UCOA\_vals(1) UCOA\_vals(1) 0];

defuzz\_breakPts = [-10 linearPosx(linearEqu([-9 1],[-5 0]),UCOA\_vals(4)) linearPosx(linearEqu([-9 1],[-5 0]),UCOA\_vals(2)) ...

linearPosx(linearEqu([-5 1],[-1 0]),UCOA\_vals(2)) linearPosx(linearEqu([-5 1],[-1 0]),UCOA\_vals(1)) linearPosx(linearEqu([0 1],[5 0]),UCOA\_vals(1)) 5];

plot(defuzz\_breakPts,defuzz\_output,'--r','linewidth',3);

grid on;

title('Defuzzification');

func1 = [0 UCOA\_vals(4)];

func2 = linearEqu([-9 1],[-5 0]);

func3 = [0 UCOA\_vals(2)];

func4 = linearEqu([-5 1],[-1 0]);

func5 = [0 UCOA\_vals(1)];

func6 = linearEqu([0 1],[5 0]);

defuzz\_Funcs = [func1 func2 func3 func4 func5 func6];

areas = zeros(1,size(defuzz\_breakPts,2)-1);

moments = zeros(1,size(defuzz\_breakPts,2)-1);

fprintf('\n');

disp('Defuzzification Piecewise Functions');

for i = 1:(size(defuzz\_breakPts,2)-1)

if(defuzz\_Funcs(2\*i-1) ~= 0)

fprintf('Function %d between %g and %g is u(U) = %g\*u + %g \n',i, defuzz\_breakPts(i),defuzz\_breakPts(i+1),defuzz\_Funcs(2\*i-1),defuzz\_Funcs(2\*i));

else

fprintf('Function %d between %g and %g is u(U) = %g \n',i, defuzz\_breakPts(i),defuzz\_breakPts(i+1),defuzz\_Funcs(2\*i));

end

areas(i) = integral(@(x)(defuzz\_Funcs(2\*i-1)\*x + defuzz\_Funcs(2\*i)),defuzz\_breakPts(i),defuzz\_breakPts(i+1));

moments(i) = integral(@(x)(defuzz\_Funcs(2\*i-1)\*(x.\*x) + defuzz\_Funcs(2\*i)\*x),defuzz\_breakPts(i),defuzz\_breakPts(i+1));

fprintf('Area %d = %g\n',i,areas(i));

fprintf('Moment %d = %g\n',i,moments(i));

end

fprintf('\nCOA Method (max-min) - Defuzzified Output for Error = %g, deltaError = %g\n',error,deltaError);

fprintf('Total Area = %g\n',sum(areas));

fprintf('Total Moment = %g\n',sum(moments));

outputVoltage = sum(moments)/sum(areas);

fprintf('Output voltage = %gV\n',outputVoltage);

text(outputVoltage,0.6,strcat('COA = ',num2str(outputVoltage)));

plot([outputVoltage outputVoltage],[0 1],'--g','linewidth',2);