**Product Inference Engine with Center Average Defuzzification, and**

**Output MF Center tuning using Serial and Parallel Steepest Descent**

%% Input and Output Membership Functions

clc

clear

x = [0:1:100];

cold = zeros(1,length(x));

cool = zeros(1,length(x));

warm = zeros(1,length(x));

slow = zeros(1,length(x));

medium = zeros(1,length(x));

fast = zeros(1,length(x));

for i=1:length(x)

% X input MF's

cold(i) = max([0 min([1 (30-x(i))/(30-0)])]);

cool(i) = max([0 min([(x(i)-0)/(20-0) (70-x(i))/(70-20)])]);

warm(i) = max([0 min([(x(i)-40)/(70-40) 1])]);

% Z output MF's

slow(i) = max([0 min([(x(i)-15)/(20-15) (25-x(i))/(25-20)])]);

medium(i) = max([0 min([(x(i)-55)/(60-55) (65-x(i))/(65-60)])]);

fast(i) = max([0 min([(x(i)-75)/(80-75) (85-x(i))/(85-80)])]);

end

subplot(2,1,1);

hold on

plot(x,cold);

plot(x,cool);

plot(x,warm);

xlabel('Universe X - Temperature (F)');

ylabel('MF Grade');

title('Input X Membership Functions');

legend('Cold', 'Cool', 'Warm');

subplot(2,1,2);

hold on

plot(x,slow);

plot(x,medium);

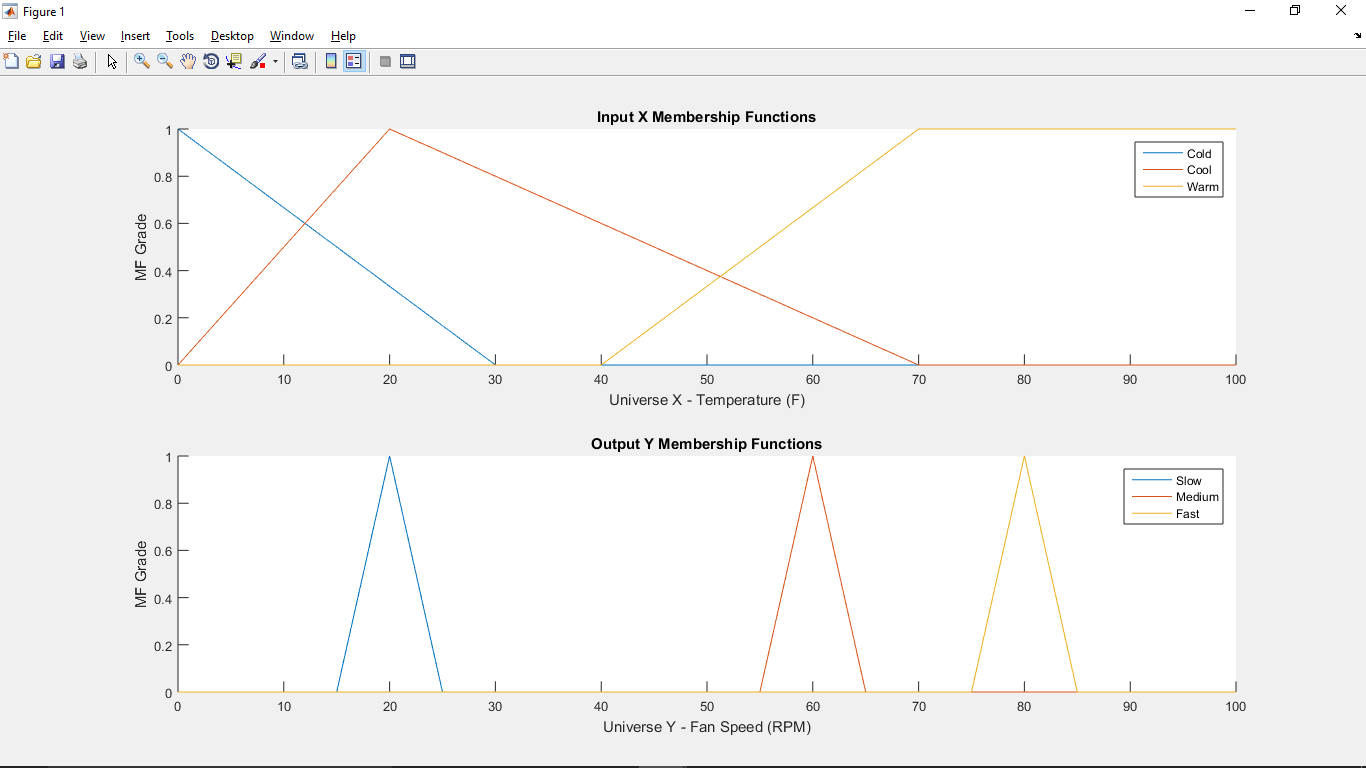
plot(x,fast);

xlabel('Universe Y - Fan Speed (RPM)');

ylabel('MF Grade');

title('Output Y Membership Functions');

legend('Slow', 'Medium', 'Fast')



%% Normal Product Inference Engine (PIE) with CA Defuzzification

% Current Z fuzzy set center points for defuzzification

fast\_c = 80;

medium\_c = 60;

slow\_c = 20;

% Crisp inputs are used as fuzzy singleton facts

xfacts = [0 10 30 40 50 70];

% Optimal output for given inputs (training data)

y\_optimal = [150 128.5 100 100 72.7 50];

% "Fuzzify" the inputs at the given measurement values

% this output known as "firing strength" or "rule weight"

xfacts\_cold\_MFgrade = zeros(1,length(xfacts));

xfacts\_cool\_MFgrade = zeros(1,length(xfacts));

xfacts\_warm\_MFgrade = zeros(1,length(xfacts));

for i=1:length(xfacts)

xfacts\_cold\_MFgrade(i) = max([0 min([1 (30-xfacts(i))/(30-0)])]);

xfacts\_cool\_MFgrade(i) = max([0 min([(xfacts(i)-0)/(20-0) (70-xfacts(i))/(70-20)])]);

xfacts\_warm\_MFgrade(i) = max([0 min([(xfacts(i)-40)/(70-40) 1])]);

end

% Rule 1, if temp is cold then blower is fast

wR1 = xfacts\_cold\_MFgrade; % only 1 input per rule

wR1\_center = wR1 \* fast\_c; % instead, using center point of output MF

% Rule 2, if temp is cool then blower is medium

wR2 = xfacts\_cool\_MFgrade;

wR2\_center = wR2 \* medium\_c;

% Rule 3, if temp is warm then blower is slow

wR3 = xfacts\_warm\_MFgrade;

wR3\_center = wR3 \* slow\_c;

sum\_wR\_center = wR1\_center + wR2\_center + wR3\_center;

sum\_wR = wR1 + wR2 + wR3; % in this case = 1

y\_ca = sum\_wR\_center ./ sum\_wR; % "Defuzzify" ouput using center average

figure()

hold on

plot(xfacts,y\_ca) % rule surface

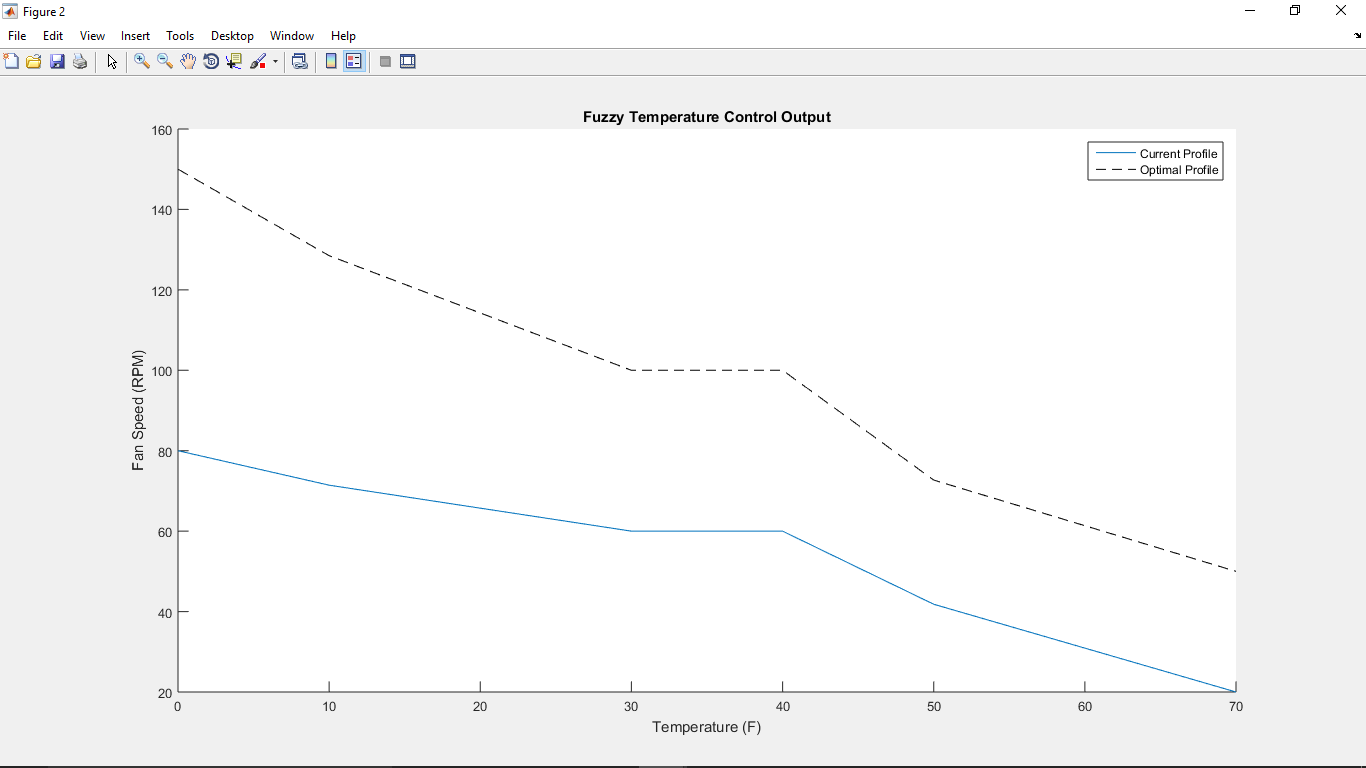
plot(xfacts,y\_optimal, '--k')

xlabel('Temperature (F)');

ylabel('Fan Speed (RPM)');

title('Fuzzy Temperature Control Output');

legend('Current Profile', 'Optimal Profile')



%% Parallel Steepest Descent (SD)

% Estimate better y output membership function center points using SD

xfacts = [0 10 30 40 50 70]; % input training data

y\_optimal = [150 128.5 100 100 72.7 50]; % output training data

thetahat = [80; 60; 20]; % initial estimates

% Because I'm using a static step size, I chose an identical cost\_delta to

% avoid overstepping. That's what happens here if you choose something like

% cost\_delta = 1d-10

step\_size = 1d-3; % denoted as eta (n) or lamda in the notes

cost\_delta = 1d-3; % loop exit criteria

cost\_value = 0; % e(j)

prev\_cost\_value = 1d5; % e(j-1)

y\_ca\_defuzz = zeros(1,length(xfacts)); % defuzzified outputs using Center Average method

cost\_gradient = zeros(3,length(xfacts));

save\_cost\_parallel = [];

% thetahat = zeros(3,length(xfacts));

% Loop while cost (error) is still changing by more than a chosen delta

% The assumption is that the cost difference decreases each iteration

% Iterations are called 'j' in notes, but we don't track them b/c we're

% not indexing anything based on them

while (abs(cost\_value - prev\_cost\_value) > cost\_delta)

prev\_cost\_value = cost\_value; % must update prev\_cost\_value to converge!

save\_cost\_parallel = [save\_cost\_parallel prev\_cost\_value];

wR = zeros(3,length(xfacts)); % wR is a matrix of rule weights, rows are each Rule, columns are each input xfact

for i=1:length(xfacts) % num data I/O, parallel SD uses all data at once

xfact\_cold\_MFgrade = max([0 min([1 (30-xfacts(i))/(30-0)])]);

xfact\_cool\_MFgrade = max([0 min([(xfacts(i)-0)/(20-0) (70-xfacts(i))/(70-20)])]);

xfact\_warm\_MFgrade = max([0 min([(xfacts(i)-40)/(70-40) 1])]);

wR(:,i) = [xfact\_cold\_MFgrade xfact\_cool\_MFgrade xfact\_warm\_MFgrade]';

% Calculate cost update, notes page 34

% the cost function actually uses the "Defuzzified" ouput

y\_ca\_defuzz(i) = sum(wR(:,i) .\* thetahat) ./ sum(wR(:,i));

cost\_value = .5 \* ((y\_ca\_defuzz(i) - y\_optimal(i))^2);

% Calculate gradiant, notes page 37

% this is partial derivative of cost function

cost\_gradient(:,i) = cost\_value \* wR(:,i) / sum(wR(:,i));

% Calculate new y output MF center

% thetanew = theta -+ n\*g

% Must hard code which direction to go, positive/negative

% if you choose wrong, the algorithm won't converge

thetahat = thetahat + step\_size \* cost\_gradient(:,i);

end

end

thetahat % the new estimated output centers

y\_parallel = y\_ca\_defuzz

%% Serial Steepest Descent (SD)

% Estimate better y output membership function center points using SD

xfacts = [0 10 30 40 50 70]; % input training data

y\_optimal = [150 128.5 100 100 72.7 50]; % output training data

thetahat = [80; 60; 20]; % initial estimates

step\_size = 1d-2; % denoted as eta (n) or lamda in the notes

cost\_delta = 1d-10; % loop exit criteria

cost\_value = zeros(1,length(xfacts)); % e(j)

prev\_cost\_value = 1d5; % e(j-1)

wR = []; % wR is a matrix of rule weights, rows are each Rule, columns are each input xfact

y\_ca\_defuzz = zeros(1,length(xfacts)); % defuzzified outputs using Center Average method

cost\_gradient = zeros(3,length(xfacts));

% thetahat = zeros(3,length(xfacts));

save\_costxy = []

for i=1:length(xfacts) % num data I/O, this is serial SD training loop

xfact\_cold\_MFgrade = max([0 min([1 (30-xfacts(i))/(30-0)])]);

xfact\_cool\_MFgrade = max([0 min([(xfacts(i)-0)/(20-0) (70-xfacts(i))/(70-20)])]);

xfact\_warm\_MFgrade = max([0 min([(xfacts(i)-40)/(70-40) 1])]);

wR = [wR [xfact\_cold\_MFgrade xfact\_cool\_MFgrade xfact\_warm\_MFgrade]'];

% Loop while cost (error) is still changing by more than a chosen delta

% The assumption is that the cost difference decreases each iteration

% Iterations are called 'j' in notes, but we don't track them b/c we're

% not indexing anything based on them

cost\_value(i) = 0;

prev\_cost\_value = 1d5;

save\_costx = [];

j = 0;

while (abs(cost\_value(i) - prev\_cost\_value) > cost\_delta && j < 1d3)

prev\_cost\_value = cost\_value(i); % must update prev\_cost\_value to converge!

% Calculate cost update, notes page 34

% the cost function actually uses the "Defuzzified" ouput

y\_ca\_defuzz(i) = sum(wR(:,i) .\* thetahat) ./ sum(wR(:,i));

cost\_value(i) = .5 \* ((y\_ca\_defuzz(i) - y\_optimal(i))^2);

% Calculate gradiant, notes page 37

% this is partial derivative of cost function

cost\_gradient(:,i) = cost\_value(i) \* wR(:,i) / sum(wR(:,i));

% Calculate new y output MF center

% thetanew = theta -+ n\*g

% Must hard code which direction to go, positive/negative

% if you choose wrong, the algorithm won't converge

thetahat = thetahat + step\_size \* cost\_gradient(:,i);

j = j + 1;

save\_costx = [save\_costx prev\_cost\_value];

end

save\_costxy = [save\_costxy; save\_costx];

end

thetahat % the new estimated output centers

y\_serial = y\_ca\_defuzz

%% Data Plots

figure()

hold on

scatter(xfacts,y\_optimal, 'k')

plot(xfacts,y\_serial, 'r')

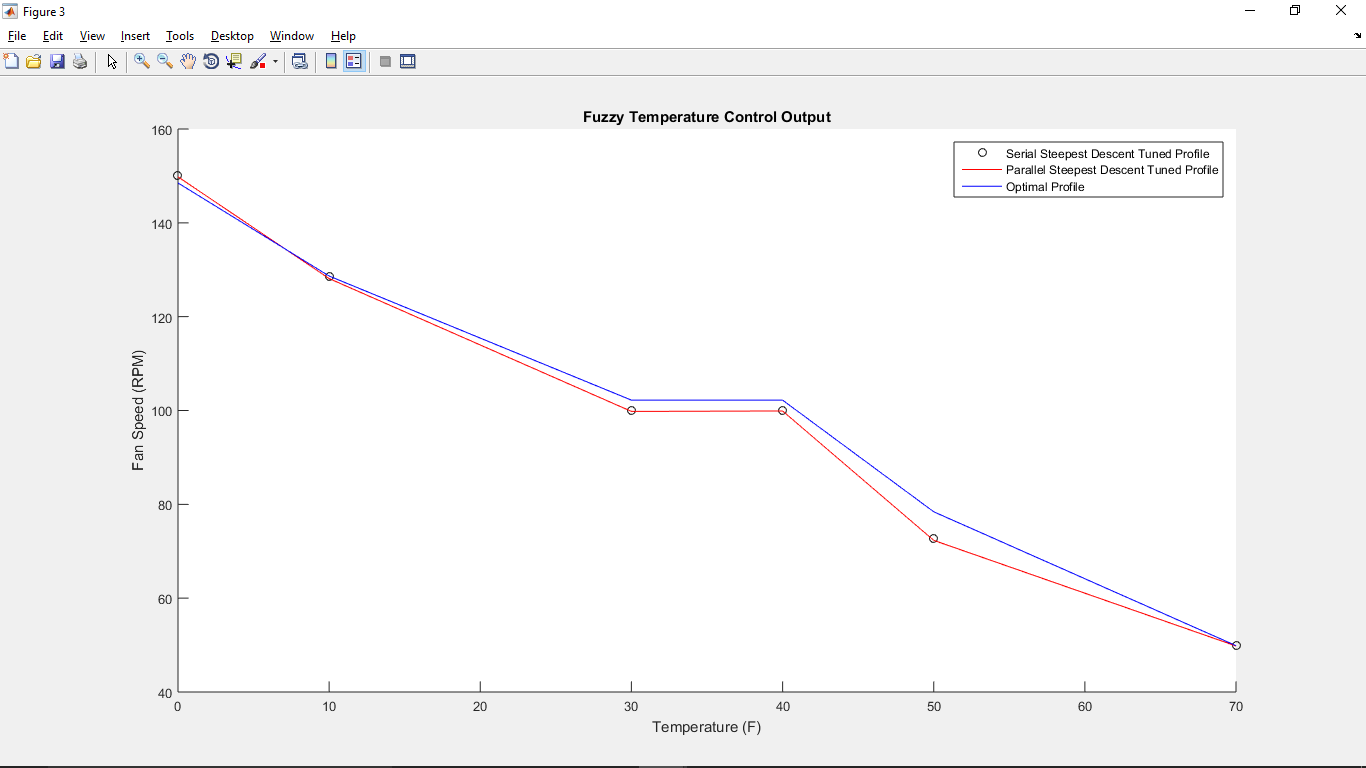
plot(xfacts,y\_parallel, 'b')

xlabel('Temperature (F)');

ylabel('Fan Speed (RPM)');

title('Fuzzy Temperature Control Output');

legend('Optimal Profile', 'Serial Steepest Descent Tuned Profile', 'Parallel Steepest Descent Tuned Profile')



Black circles are **optimal** output, red line is **serial Steepest Descent** tuned output, blue line is **parallel Steepest Descent** tuned output.

**Parallel –** seems better than the serial method at training the overall output MF centers for all MF's, but it doesn't seem to fit each data point as well to the optimal, seen in the plot of the actual output.

output MF centers –

**Serial** **–** does a good job of estimating output MF centers for each individual input, but it's not necessarily good at training your overall output MF centers for all MF's.

output MF centers –

figure()

axis([ 0 200 0 2500])

hold on

for i=1:length(xfacts)

plot(save\_costxy(i,:))

end

xlabel('Iteration');

ylabel('Cost');

title('Serial SD Cost VS Iteration');

legend('xfact 1', 'xfact 2', 'xfact 3')

figure()

hold on

plot(save\_cost\_parallel)

xlabel('Iteration');

ylabel('Cost');

title('Parallel SD Cost VS Iteration');

legend('all xfacts')

