Table of Contents


```
function [yL, yH, tS, tau, SSEmod, string] =
 Project_M4Algorithm_002_08(timeCol, tempCol)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ENGR 132 FINAL PROJECT
%
% Function Call
% [yL, yH, tS, tau, SSEmod, string] =
 Project_M4Algorithm_002_08(timeCol, tempCol)
%
% Input Arguments
% 1. timeCol: the time value columns of the data set
% 2. tempCol: the temperature value
%
% Output Arguments
% 1. yL: initial low temperature/asymptoting lowest temperature
% 2. yH: initial high temperature/asymptoting highest temperature
% 3. tS: the threshold t-value of the change in temperature
% 4. tau: represents the time it takes for the dependent
%    variable to achieve a value of yTau = yL + 0.632(yH ? yL).
% 5. SSEmod: the SSE mod of the data set given
% 6. string: cooling or heating
%
% Assignment Information
%   Assignment:        M4 algorithm
%   Author:             Tomoki Koike, koike@purdue.edu
%                       Yi Zhou, zhou823@purdue.edu
%                       Ian Pitman, ipitman@purdue.edu
%                       Eu Jin Lee, lee2219@purdue.edu
%   Team ID:            002-08
%   Contributor:   no contributor
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

STEP ONE - Calculating yL (F) and yH (F)

```
% This is for the bottom sth percent of the data

percent1 = 0.001; % the percent of data that we are going to focus on
avg1 = 1; % mean (F)
md1 = 100; % median (F)
```

```
% Find the total number of indices in the temperature column
numY = numel(tempCol);

% The while loop below calculates the mean and the median and compares
%them until the two values become approximately the same value. For
%each iteration the range of data that we focus on is changed/expanded
%by the percent (counter) of the loop. I
while  round(avg1,1) ~= round(md1,1)  % round to first decimal (F)
    % the range of temperatures (F)we are observing
    %(depends on percent)
    range1 = tempCol(1:round(numY*percent1));
    % take the average of the temperatures (F) in the range
    avg1 = mean(range1);
    % take the median of the temperatures (F) in the range
    md1 = median(range1);
    % increase the percent to expand the range
    percent1 = percent1 + 0.001;
end

% obtain normal distribution parameters - mu and sigma
pd1 = fitdist(tempCol(1:round(numY*(percent1-0.001))),'normal');
% obtain the confidence interval
ci1 = paramci(pd1);

% now we take the mu as the y1
y1 = mean(ci1(:,1));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This is for the top sth percent of the data

% Setting variables likewise to the previous operation
percent2 = 0.999; % this time it starts from 100% and decreases
                  %the percentage
avg2 = 1;   % the mean (F)
md2 = 100;  % the median (F)

% This time is the top sth percent of the data
while round(avg2,1)~= round(md2,1)  % round to first decimal (F)
    % range of temperatures (F) we are observing
    %(depends on the percent)
    range2 = tempCol(round(numY*percent2):numY);
    % take the average of the temperatures (F) in the range
    avg2 = mean(range2);
    % take the median of the temperatures (F) in the range
    md2 = median(range2);
    %
    percent2 = percent2 - 0.001;
end

% normal distribution parameters - mu and sigma
pd2 = fitdist(tempCol((round(numY*(percent2+0.001):numY))),'normal');
% the confidence interval (F)
ci2 = paramci(pd2);
```

```
% Since in the bell shaped probability density function the median is
%always located closer to the mode we say that the y2 is median (F)
y2 = mean(ci2(:,1));

% The next operation switches the value so that yL and yH are assigned
%proper values from y1 and y2 according to whether the data is a
%cooling or heating process  (F)
if y1 > y2   % cooling
    yH = y1;
    yL = y2;
    string = "cooling";
else         % heating
    yH = y2;
    yL = y1;
    string = "heating";
end
```

```
Not enough input arguments.

Error in Project_M4Algorithm_002_08 (line 42)
numY = numel(tempCol);
```

STEP TWO - Finding the tS value

```
% For this step we use the yL and yH values that we have figured out
%in the previous step

% Finding the indices for the values in the data set that are equal
%to yL. Then finding the highest index value. (F)
yLowRange = find(tempCol >= (yL - 0.5) & tempCol <= (yL + 0.5));
yHighRange = find(tempCol >= (yH - 0.5) & tempCol <= (yH + 0.5));

if y1 > y2   % cooling
    yLowLimit = min(yLowRange);
    yHighLimit = max(yHighRange);
    % the range of temperature values to scrutinize
    tempRange = tempCol(yHighLimit : yLowLimit);
    % preallocating a vector which will contain the slope values
    slope = zeros(1, yHighLimit - yLowLimit + 1);
    for n = 1:(yLowLimit - yHighLimit)
        % finding the time values that correspond with the
        %temperature range (s)
        timeValues = timeCol(yHighLimit : yLowLimit);
        % calculating the slopes (F/s)
        slope(n) = (tempRange(n+1) - tempRange(n)) / ...
            (timeValues(n+1) - timeValues(n));
    end
    % slope must be negative
    validSlopes = slope(slope < 0);
    % y = m*x + b1
    avgSlope = mean(validSlopes);  % m
    % y = m*x + b2
 intercept2 = tempCol(yHighLimit) - avgSlope*timeCol(yHighLimit);
```

```
    tS = (yH - intercept2) / avgSlope;
else         % heating
    yLowLimit = max(yLowRange);
    yHighLimit = min(yHighRange);
    % the range of temperature values to scrutinize
    tempRange = tempCol(yLowLimit : yHighLimit);
    % preallocating a vector which will contain the slope values
    slope = zeros(1, yHighLimit - yLowLimit + 1);
    for n = 1:(yHighLimit - yLowLimit)
        % finding the time values that correspond with the
        %temperature range
        timeValues = timeCol(yLowLimit : yHighLimit);
        % calculating the slopes
        slope(n) = (tempRange(n+1) - tempRange(n)) / ...
            (timeValues(n+1) - timeValues(n));
    end
    % slope must be positive
    validSlopes = slope(slope > 0);
    % y = m*x + b1
    avgSlope = mean(validSlopes);  % m
    % y-intercept b1 will be
    intercept1 = tempCol(yLowLimit) - avgSlope*timeCol(yLowLimit);
    % First equation is y = avgSlope*x + intercept1
    tS = (yL - intercept1) / avgSlope;
end
```

STEP THREE - Compute Tau

M4 Method

```
counter = 1;    % counter for the for while loop
index_tS = 1;   % initialize the predicted index of tS in the timeCol

% The loop below is a linear search for the tS value in the timeCol.
%The loop is used to round the tS to from 0.1 to 0.01 to 0.001 and on
%and on until there is not a value in the timeCol that is equivalent
%to the rounded tS value.
while isempty(index_tS) ~= 1
    holdIndex_tS = index_tS;
    index_tS = find(timeCol == round(tS, counter));
    counter = counter + 1;
end

% this sets the index of the tS value in the timeCol
index_tS = holdIndex_tS;

% We establish the timeCol and tempCol with the range from the index
 of
%the index of tS (which we have just found above) to the last index of
%timeCol/tempCol.
limitTimeCol = timeCol(index_tS:length(timeCol));
limitTempCol = tempCol(index_tS:length(tempCol));
```

```
% Set the model equation of the thermocouple
if string == "cooling"
    modelEqn = @(b,t)(yL + (yH - yL)*(exp((-t/b(1)) + b(2)/b(1))));
else
    modelEqn = @(b,t)(yL + (yH - yL)*(1 - exp((-t/b(1)) + b(2)/
b(1))));
end

% Set the array for the variables t and y
t = limitTimeCol;
y = limitTempCol;
% give the coefficients initial values
b0 = [2;2];

% Models the equation with the estimated coefficients
modl = fitnlm(t,y,modelEqn,b0);
% Establish the array of the confidence interval for the tau and tS
%value (2-by-2 matrix; the first row is the tau, the second row is the
%tS, and each row is consisted of the lower limit (col 1) and upper
%limit (col 2)).
tau_and_tS = coefCI(modl);
% Take the average of the confidence interval for each tau and tS (s)
% THESE AVERAGES WILL BE THE RESULTS OF tau AND tS
% *because the tS we have modelled here is more accurate than the
%previous one we will overwrite the tS value. (the first tS parameter
%still played a pivotal role in limiting the interval of the tempCol
%and timeCol to the range for the model equation)
tau = mean(tau_and_tS(1,:));  %(s)
tS = mean(tau_and_tS(2,:));   %(s)
```

STEP FOUR - Compute the SSEmod

```
% This step is particular to M2
% Now that we have all values yL, yH, tS, and tau, we can plub this
%this into the y(t) equation. And then plug in the time column values
%to get all the calculated y-values based on our code. And using these
%y-values and the y-values from the original data we can calculate the
%SSEmod.

syms y(t)
if y1 > y2    % cooling
    yExp = yL + (yH - yL)*(exp(-(t - tS)/tau));
    y(t) = piecewise(t<=tS, yH, tS<t, yExp);
    yVals = subs(y(t), t, timeCol);
    SSE = sum((tempCol - yVals).^2);
    SSEmod = SSE / length(tempCol);
else          % heating
    yExp = yL + (yH - yL)*(1 - exp(-(t - tS)/tau));
    y(t) = piecewise(t<=tS, yL, tS<t, yExp);
    yVals = subs(y(t), t, timeCol);
    SSE = sum((tempCol - yVals).^2);
    SSEmod = SSE / length(tempCol);
```

```
end

SSEmod = vpa(SSEmod);

%fprintf("\nThe SSEmod for algorithm 1 is -> %f\n\n", SSEmod);

end
```

Published with MATLAB® R2018a