Content

I Calibration and regression equations

Calibration

```
function [fitresult, Eps_IndexD, Eps_IndexS] = CalibrationDP()

%% Clear and Load
clear; clc; close all;

load CalibrationDP_CalData.mat % Calibration data

%% Table split in columns
Points = CalData(:,1);      % Point number where the measurement was made
Alpha = CalData(:,4);       % Angle alpha, kept 0
Epsilon = CalData(:,5);     % Angle epsilon
pb = CalData(:,6);          % Absolute atmospheric Pressure
Temp = CalData(:,7);        % Temperature
p1 = CalData(:,10) + pb;    % Absolute pressure p1
p2 = CalData(:,11) + pb;    % Absolute pressure p2
p3 = CalData(:,12) + pb;    % Absolute pressure p3
p4 = CalData(:,13) + pb;    % Absolute pressure p4
pB = CalData(:,14) + pb;    % Absolute pressure pB
pP = CalData(:,15) + pb;    % Absolute pressure pP
pL = CalData(:,16) + pb;    % Absolute pressure pL
prefC = CalData(:,17) + pb; % Absolute total pressure of a reference probe
prefS = CalData(:,18) + pb; % Absolute static pressure of a reference probe
prefD = prefC - prefS;      % Absolute dynamic pressure of a reference probe

%% Pressure unification
P = [p1 p2 p3 p4]; % matrix of pressures p1 - p4

%% Pressure coefficients
k = zeros(size(P,1),size(P,2),size(P,2));
% Matrix of pressure coefficients
for i = 1:4
    A = P(:,i);   % Helping matrix to pick out the specific vector
    for j = 1:4
        for l = 1:size(P,1)
            if j ~= i
                B = P(:,j); % the same as A
                k(l,j,i) = (A(l) - pB(l))./(B(l) - pB(l)); % 3D matrix of this calculation
            else
                k(l,j,i) = 0;
            end
        end
    end
end

%% Coefficients of Kx1
K11 = k(:,1,1);
```

```
K21 = k(:,1,2);
K31 = k(:,1,3);
K41 = k(:,1,4);

% Plot
figure()
plot(K21, Epsilon, '+r')
hold on
plot(K31, Epsilon, 'ob')
plot(K41, Epsilon, 'xg')
hold off
xlabel('Coefficient k_{\epsilon}')
ylabel('Angle \epsilon')
legend('K21', 'K31', 'K41')

%% Coefficients of Kx2
K12 = k(:,2,1);
K22 = k(:,2,2);
K32 = k(:,2,3);
K42 = k(:,2,4);

% Plot
figure()
plot(K12, Epsilon, '+r')
hold on
plot(K32, Epsilon, 'ob')
plot(K42, Epsilon, 'xg')
hold off
xlabel('Coefficient k_{\epsilon}')
ylabel('Angle \epsilon')
title('Progress of calibration coefficient k_{\epsilon32}')
legend('K12', 'K32', 'K42')

%% Coefficients of Kx3
K13 = k(:,3,1);
K23 = k(:,3,2);
K33 = k(:,3,3);
K43 = k(:,3,4);

% Plot
figure()
plot(K13, Epsilon, '+r')
hold on
plot(K23, Epsilon, 'ob')
plot(K43, Epsilon, 'xg')
hold off
xlabel('Coefficient k_{\epsilon}')
ylabel('Angle \epsilon')
legend('K13', 'K23', 'K43')
```

```
%% Coefficients of Kx4
K14 = k(:,4,1);
K24 = k(:,4,2);
K34 = k(:,4,3);
K44 = k(:,4,4);

% Plot
figure()
plot(K14, Epsilon, '+r')
hold on
plot(K24, Epsilon, 'ob')
plot(K34, Epsilon, 'xg')
hold off
xlabel('Coefficient k_{\epsilon}')
ylabel('Angle \epsilon')
legend('K14', 'K24', 'K34')

%% Static and dynamic pressure coefficients with a help of reference probe
kD = zeros(size(P,1), size(P,2));
kS = zeros(size(P,1), size(P,2));
    for i = 1:4
        for l = 1:size(P,1)
            kD(l,i) = (P(l,i) - pB(l))/prefD(l); % Static pressure coefficient
            kS(l,i) = (P(l,i) - prefS(l))/prefD(l); % Dynamic pressure coefficient
        end
    end

%% Plots of static and dynamic pressure coefficients kDx and kSx

%kDi
figure()
plot(Epsilon,kD(:,1), '+r')
hold on
plot(Epsilon,kD(:,2), 'ob')
plot(Epsilon,kD(:,3), 'xg')
plot(Epsilon,kD(:,4), 'vk')
hold off
ylabel('Coefficient k_{Di} [-]')
xlabel('Angle \epsilon [°]')
legend('kS1', 'kS2', 'kS3', 'kS4')
title('Calibration coefficients for dynamic pressure')

% kSi
figure()
plot(Epsilon,kS(:,1), '+r')
hold on
plot(Epsilon,kS(:,2), 'ob')
plot(Epsilon,kS(:,3), 'xg')
plot(Epsilon,kS(:,4), 'vk')
hold off
```

```
ylabel('Coefficient k_{Di} [-]')
xlabel('Angle \epsilon [°]')
legend('kD1', 'kD2', 'kD3', 'kD4')
title('Calibration coefficients for static pressure')

%% Point where kD2 and kD3 intersect
kD2 = kD(:,2);
kD3 = kD(:,3);

InterD = kD2 - kD3;
InterD = abs(InterD);
Min = 10;
IndexD = zeros(1);
for i = 1:size(kD2,1)
   if InterD(i) < Min
      Min = InterD(i);
      IndexD = i;
   end
end

Eps_IndexD = Epsilon(IndexD); % limiting Epsilon value for dynamic pressure coefficient

kS2 = kS(:,2);
kS3 = kS(:,3);

InterS = kS2 - kS3;
InterS = abs(InterS);
Min = 10;
IndexS = zeros(1);
for i = 1:size(kS2,1)
   if InterS(i) < Min
      Min = InterS(i);
      IndexS = i;
   end
end
Eps_IndexS = Epsilon(IndexS); % limiting Epsilon value for static pressure coefficient

%% Selection of kD2 and kD3 coefficients, Selection of kS2 and kS3 coefficients
kD2_Old = kD2;
kD2 = kD2_Old(1:IndexD);

kD3_Old = kD3;
kD3 = kD3_Old((IndexD + 1):size(P,1));

kS2_Old = kS2;
kS2 = kS2_Old(1:IndexS);

kS3_Old = kS3;
kS3 = kS3_Old((IndexS + 1):size(P,1));
```

```
EpsilonD2 = Epsilon(1:IndexD);
EpsilonD3 = Epsilon((IndexD + 1):size(P,1));

EpsilonS2 = Epsilon(1:IndexS);
EpsilonS3 = Epsilon((IndexS + 1):size(P,1));

%% Graph for the two main curves

%kDi
figure()
plot(EpsilonD2,kD2, 'ob')
hold on
plot(EpsilonD3,kD3, 'xg')
hold off
ylim([-0.2, 1.4])
ylabel('Coefficient k_{Di} [-]')
xlabel('Angle \epsilon [°]')
legend('kD2', 'kD3')
title('Calibration coefficients for dynamic pressure')

%kSi
figure()
plot(EpsilonS2,kS2, 'ob')
hold on
plot(EpsilonS3,kS3, 'xg')
hold off
ylim([-0.2, 1.4])
ylabel('Coefficient k_{Si} [-]')
xlabel('Angle \epsilon [°]')
legend('kS2', 'kS3')
title('Calibration coefficients for static pressure')

%% Graphs with regression equations
[fitresult, gof] = FittedCurves(K32, Epsilon, K12, K41, K42, EpsilonD2, kD2, EpsilonD3,
kD3, EpsilonS2, kS2, EpsilonS3, kS3);
end
```

Regression equations

```
function [fitresult, gof] = FittedCurves(K32, Epsilon, K12, K41, K42, EpsilonD2, kD2,
EpsilonD3, kD3, EpsilonS2, kS2, EpsilonS3, kS3)
%CREATEFITS(K32,EPSILON,K12,K41,K42,EPSILOND2,KD2,EPSILOND3,KD3,EPSILONS2,KS2,EPSILONS3,KS3)
%  Create fits.
%
%  Data for 'K32' fit:
%      X Input : K32
%      Y Output: Epsilon
%  Data for 'K12' fit:
%      X Input : K12
```

```
%      Y Output: Epsilon
%  Data for 'K41' fit:
%      X Input : K41
%      Y Output: Epsilon
%  Data for 'K42' fit:
%      X Input : K42
%      Y Output: Epsilon
%  Data for 'kD2' fit:
%      X Input : EpsilonD2
%      Y Output: kD2
%  Data for 'kD3' fit:
%      X Input : EpsilonD3
%      Y Output: kD3
%  Data for 'kS2' fit:
%      X Input : EpsilonS2
%      Y Output: kS2
%  Data for 'kS3' fit:
%      X Input : EpsilonS3
%      Y Output: kS3
%  Output:
%      fitresult : a cell-array of fit objects representing the fits.
%      gof : structure array with goodness-of fit info.
%
%  See also FIT, CFIT, SFIT.

%  Auto-generated by MATLAB on 06-Mar-2020 19:57:24

%% Initialization.

% Initialize arrays to store fits and goodness-of-fit.
fitresult = cell( 8, 1 );
gof = struct( 'sse', cell( 8, 1 ), ...
    'rsquare', [], 'dfe', [], 'adjrsquare', [], 'rmse', [] );

%% Fit: 'K32'.
[xData, yData] = prepareCurveData( K32, Epsilon );

% Set up fittype and options.
ft = fittype( 'poly2' );

% Fit model to data.
[fitresult{1}, gof(1)] = fit( xData, yData, ft );

% Plot fit with data.
figure( 'Name', 'K32' );
h = plot( fitresult{1}, xData, yData, 'predobs' );
legend( h, 'Epsilon vs. K32', 'K32', 'Lower bounds (K32)', 'Upper bounds (K32)', 'Location', 'NorthEast' );
[Message] = Equation(fitresult{1})
text(0.5,36,Message);
```

```
% Label axes
xlabel K32
ylabel Epsilon
grid on

%% Fit: 'K12'.
[xData, yData] = prepareCurveData( K12, Epsilon );

% Set up fittype and options.
ft = fittype( 'poly3' );

% Fit model to data.
[fitresult{2}, gof(2)] = fit( xData, yData, ft );

% Plot fit with data.
figure( 'Name', 'K12' );
h = plot( fitresult{2}, xData, yData, 'predobs' );
legend( h, 'Epsilon vs. K12', 'K12', 'Lower bounds (K12)', 'Upper bounds (K12)', 'Location',
'NorthEast' );
[Message] = Equation(fitresult{2})
text(0.3,39,Message);
% Label axes
xlabel K12
ylabel Epsilon
grid on

%% Fit: 'K41'.
[xData, yData] = prepareCurveData( K41, Epsilon );

% Set up fittype and options.
ft = fittype( 'poly8' );

% Fit model to data.
[fitresult{3}, gof(3)] = fit( xData, yData, ft );

% Plot fit with data.
figure( 'Name', 'K41' );
h = plot( fitresult{3}, xData, yData, 'predobs' );
legend( h, 'Epsilon vs. K41', 'K41', 'Lower bounds (K41)', 'Upper bounds (K41)', 'Location',
'NorthEast' );
[Message] = Equation(fitresult{3})
text(-1.2,45,Message);
% Label axes
xlabel K41
ylabel Epsilon
grid on

%% Fit: 'K42'.
[xData, yData] = prepareCurveData( K42, Epsilon );
```

```
% Set up fittype and options.
ft = fittype( 'poly3' );

% Fit model to data.
[fitresult{4}, gof(4)] = fit( xData, yData, ft );

% Plot fit with data.
figure( 'Name', 'K42' );
h = plot( fitresult{4}, xData, yData, 'predobs' );
legend( h, 'Epsilon vs. K42', 'K42', 'Lower bounds (K42)', 'Upper bounds (K42)', 'Location',
'NorthEast' );
[Message] = Equation(fitresult{4})
text(0.2,34,Message);
% Label axes
xlabel K42
ylabel Epsilon
grid on

%% Fit: 'kD2'.
[xData, yData] = prepareCurveData( EpsilonD2, kD2 );

% Set up fittype and options.
ft = fittype( 'poly4' );

% Fit model to data.
[fitresult{5}, gof(5)] = fit( xData, yData, ft );

% Plot fit with data.
figure( 'Name', 'kD2' );
h = plot( fitresult{5}, xData, yData, 'predobs' );
legend( h, 'kD2 vs. EpsilonD2', 'kD2', 'Lower bounds (kD2)', 'Upper bounds (kD2)',
'Location', 'NorthEast' );
[Message] = Equation(fitresult{5})
text(-8,1.24,Message);
% Label axes
xlabel EpsilonD2
ylabel kD2
grid on

%% Fit: 'kD3'.
[xData, yData] = prepareCurveData( EpsilonD3, kD3 );

% Set up fittype and options.
ft = fittype( 'poly4' );

% Fit model to data.
[fitresult{6}, gof(6)] = fit( xData, yData, ft );

% Plot fit with data.
figure( 'Name', 'kD3' );
```

```
h = plot( fitresult{6}, xData, yData, 'predobs' );
legend( h, 'kD3 vs. EpsilonD3', 'kD3', 'Lower bounds (kD3)', 'Upper bounds (kD3)', 'Location', 'NorthEast' );
[Message] = Equation(fitresult{6})
text(30,1.18,Message);
% Label axes
xlabel EpsilonD3
ylabel kD3
grid on

%% Fit: 'kS2'.
[xData, yData] = prepareCurveData( EpsilonS2, kS2 );

% Set up fittype and options.
ft = fittype( 'poly4' );

% Fit model to data.
[fitresult{7}, gof(7)] = fit( xData, yData, ft );

% Plot fit with data.
figure( 'Name', 'kS2' );
h = plot( fitresult{7}, xData, yData, 'predobs' );
legend( h, 'kS2 vs. EpsilonS2', 'kS2', 'Lower bounds (kS2)', 'Upper bounds (kS2)', 'Location', 'NorthEast' );
[Message] = Equation(fitresult{7})
text(-6,0.975,Message);
% Label axes
xlabel EpsilonS2
ylabel kS2
grid on

%% Fit: 'kS3'.
[xData, yData] = prepareCurveData( EpsilonS3, kS3 );

% Set up fittype and options.
ft = fittype( 'poly4' );

% Fit model to data.
[fitresult{8}, gof(8)] = fit( xData, yData, ft );

% Plot fit with data.
figure( 'Name', 'kS3' );
h = plot( fitresult{8}, xData, yData, 'predobs' );
legend( h, 'kS3 vs. EpsilonS3', 'kS3', 'Lower bounds (kS3)', 'Upper bounds (kS3)', 'Location', 'NorthEast' );
[Message] = Equation(fitresult{8})
text(20,0.95,Message);
% Label axes
xlabel EpsilonS3
ylabel kS3
```

```
grid on
```

Equations script

```
function [Message] = Equation(X)
% this function writes out the regression equations into the graphs
Xcoeffs = coeffvalues(X); % coefficients of the function
n = size(Xcoeffs,2);
    switch (n-1)
        case 0
            Message = sprintf('y = (%f)',Xcoeffs(1));
        case 1
            Message = sprintf('y = (%f) x + (%f)',Xcoeffs(1),Xcoeffs(2));
        case 2
            Message = sprintf('y = (%f) x^2 + (%f) x + (%f)',Xcoeffs(1),Xcoeffs(2),Xcoeffs(3));
        case 3
            Message = sprintf('y = (%f) x^3 +(%f) x^2 + (%f) x +
(%f)',Xcoeffs(1),Xcoeffs(2),Xcoeffs(3),Xcoeffs(4));
        case 4
            Message = sprintf('y =(%f) x^4 +(%f) x^3 +(%f) x^2 + (%f) x +
(%f)',Xcoeffs(1),Xcoeffs(2),Xcoeffs(3),Xcoeffs(4),Xcoeffs(5));
        case 5
            Message = sprintf('y = (%f) x^5 +(%f) x^4 +(%f) x^3 +(%f) x^2 + (%f) x +
(%f)',Xcoeffs(1),Xcoeffs(2),Xcoeffs(3),Xcoeffs(4),Xcoeffs(5),Xcoeffs(6));
        case 6
            Message = sprintf('y = (%f) x^6 + (%f) x^5 +(%f) x^4 +(%f) x^3 +(%f) x^2 + (%f) x
+ (%f)',Xcoeffs(1),Xcoeffs(2),Xcoeffs(3),Xcoeffs(4),Xcoeffs(5),Xcoeffs(6),Xcoeffs(7));
        case 7
            Message = sprintf('y = (%f) x^7 + (%f) x^6 + (%f) x^5 +(%f) x^4 +(%f) x^3 +(%f)
x^2 + (%f) x +
(%f)',Xcoeffs(1),Xcoeffs(2),Xcoeffs(3),Xcoeffs(4),Xcoeffs(5),Xcoeffs(6),Xcoeffs(7),Xcoeffs
(8));
        case 8
            Message = sprintf('y = (%f) x^8 + (%f) x^7 + (%f) x^6 + (%f) x^5 +(%f) x^4 +(%f)
x^3 +(%f) x^2 + (%f) x +
(%f)',Xcoeffs(1),Xcoeffs(2),Xcoeffs(3),Xcoeffs(4),Xcoeffs(5),Xcoeffs(6),Xcoeffs(7),Xcoeffs
(8),Xcoeffs(9));
    end
end
```

II Calculation

```
%% Clear
clear;clc; close all;

%% Choose and Load Data Before the Blade, comment out when not used
% load DataProcessingDP_BeforeTheBlade2018_Full.mat % Loads complete data from the
front blade

% QoS = (1-W);      % Quality of Steam = Dryness from a measurement
% OO = 0;           % Cuts off arbitrary amount of rows
```

```
% q = (size(QoS,1)-OO);     % Vector rendering number of rows
% figure()          % Diagram of steam quality along the front blade
% plot(QoS(1:q),PP(1:q),'-or')
% xlabel('Quality of steam [-]')
% ylabel('Distance from the bladeroot [mm]')
% title('Steam quality along the front side of the blade')

%% Choose and Load Data Behind the Blade, comment out when not used
load DataProcessingDP_BehindTheBlade2018_Full.mat % loads complete data from the back blade

QoS = (1-W);                  % Quality of Steam = Dryness from a measurement
OO = 0;                       % Cuts off arbitrary amount of rows
q = (size(QoS,1)-OO);         % Vector rendering number of rows
figure()                      % Diagram of steam quality along the back blade
plot(QoS(1:q),PP(1:q),'-or')
xlabel('Quality of steam [-]')
ylabel('Distance from the bladeroot [mm]')
title('Steam quality along the back side of the blade')

%% Table split in columns
Points = CalcData(:,3);       % Point number where the measurement was made
Dist = CalcData(:,4);         % Distance [mm]
Alpha = CalcData(:,5);         % Zeroed Angle alpha, where dp = pL - pP = 0
pb = CalcData(:,13);          % Absolute atmospheric Pressure
p1 = pb - CalcData(:,6) ;      % Absolute pressure p1
p2 = pb - CalcData(:,7);       % Absolute pressure p2
p3 = pb - CalcData(:,8);       % Absolute pressure p3
p4 = pb - CalcData(:,9);       % Absolute pressure p4
pB = pb - CalcData(:,10);       % Absolute pressure pB
pL = pb - CalcData(:,11);       % Absolute pressure pL
pP = pb - CalcData(:,12);       % Absolute pressure pP
dryness = 1 - W;              % Dryness

%% Pressure unification
P = [p1 p2 p3 p4]; % matrix of pressures p1 - p4

%% Pressure coefficients
k = zeros(size(P,1),size(P,2),size(P,2));

% Matrix of pressure coefficients
for i = 1:4
    A = P(:,i);   % Helping matrix to pick out the specific vector
    for j = 1:4
        for l = 1:size(P,1)
            if j ~= i
                B = P(:,j); % the same as A
                k(l,j,i) = (A(l) - pB(l))./(B(l) - pB(l)); % 3D matrix
            else
                k(l,j,i) = 0;
```

```
      end
    end
  end
end

%% Selected Coefficients
K41 = k(:,1,4);
K12 = k(:,2,1);
K32 = k(:,2,3); % Further only this coefficient is used
K42 = k(:,2,4);

% Diagram of pressure coefficient K32 over the blade length
figure()
plot(PP(1:q),K32(1:q))
xlabel('Length of blade [mm]')
ylabel('Coefficient k_{\epsilon32} [°]')
title('Progress of coefficient k_{\epsilon32} over blade length')

%% Epsilon Determination Regression Equations Coefficients
% calling function CalibrationDP(): acquiring of calibration equations and
% limit values for selection of appropriate calibration equation based on calculated Epsilon
[fitresults, Eps_IndexD, Eps_IndexS] = CalibrationDP();

% calculation of angle epsilon based on k32 calibration equation and coefficient K32
Eps32 = feval(fitresults{1}, K32);

figure()
plot(PP(1:q), Eps32(1:q),'*b')
title('Angle \epsilon calculated from the regression equation')
xlabel('Blade length [mm]')
ylabel('Angle \epsilon [°]')
legend('Eps32')

%% Dynamic Coefficient from Epsilon
% calibration equations for dynamic pressure coefficient, yielding dynamic pressure from
% epsilon values
kD2 = feval(fitresults{5}, Eps32);
kD3 = feval(fitresults{6}, Eps32);

kD = zeros(size(Eps32,1),1);
% selects appropriate kD values based on epsilon value in comparison with
% the limit value
for i = 1:size(Eps32,1)
  if Eps32(i) <= Eps_IndexD
    kD(i) = kD2(i);
  else
    kD(i) = kD3(i);
  end
end
```

```
figure()
plot(PP(1:q),kD(1:q), 'xr')
title('Coefficient of a dynamic pressure')
xlabel('Blade length [mm]')
ylabel('Coefficient values [-]')
legend('kD')

%% Static Coefficient from Epsilon
% calibration equations for static pressure coefficient, yielding dynamic pressure from
% epsilon values
kS2 = feval(fitresults{7}, Eps32);
kS3 = feval(fitresults{8}, Eps32);

kS = zeros(size(Eps32,1),1);
% selects appropriate kS values based on epsilon value in comparison with
% the limit value
for i = 1:size(Eps32,1)
    if Eps32(i) <= Eps_IndexS
        kS(i) = kS2(i);
    else
        kS(i) = kS3(i);
    end
end

figure()
plot(PP(1:q),kS(1:q), 'xr')
title('Coefficient of a static pressure')
xlabel('Blade length [mm]')
ylabel('Coefficient values [-]')
legend('kS')

%% Calculation of Static and Dynamic Pressures
pD = zeros(size(P,1)-1,1); % allocation of vector size
pS = zeros(size(P,1)-1,1); % allocation of vector size
% calculation of static and dynamic pressure from kD (kD32) and kS (kS32)
for i = 1 : size(P,1)
    pD(i) = (p3(i) - pB(i))/kD(i); % Calculated from kD32
    pS(i) = (p3(i) - (kS(i)/kD(i))*(p3(i) - pB(i))); % Calculated from kD32 and kS32
end

pC = pS + pD; % Calculation of total pressure
% diagram of static, dynamic and total pressure along the blade length
figure()
plot(PP(1:q),pD(1:q), 'xr')
hold on
plot(PP(1:q),pS(1:q), 'ob')
plot(PP(1:q),pC(1:q),'vg')
hold off
title('Calculated pressures - dynamic, static and total')
xlabel('Blade length [mm]')
```

```
ylabel('Pressure values [Pa]')
legend('pD', 'pS', 'pC')

%% Speed calculation
psteam('open');     % opens steam tables IAPWS - IF97
vol = zeros(size(pS,1),1); % allocation for volume vector - f(x,p)
SoS = zeros(size(pS,1),1); % allocation for Speed of Sound vector  - f(x,p)
kappa = zeros(size(pS,1),1); % allocation for heat capacity ratio
% determining of volume, SoS and kappa based on static pressure and dryness
for i = 1:size(pS,1)
    vol(i) = psteam('v_xp',dryness(i),pS(i)*10^-6);
    if dryness(i) >= 1
        SoS(i) = psteam('w_gsat_p',pS(i)*10^-6);
    elseif dryness(i) < 1
        SoS(i) = psteam('w_xp',dryness(i),pS(i)*10^-6);
    end
    kappa(i) = psteam('ka_xp',dryness(i),pS(i)*10^-6);
end
psteam('close'); % closes the steam tables

ro = 1./vol;   % medium density

% allocation of speed vectors
c = zeros(size(pS,1),1); % speed vector
cu = zeros(size(c,1),1); % tangential component of speed vector
cz = zeros(size(c,1),1); % axial component of speed vector
cr = zeros(size(c,1),1); % radial component of speed vector
deltac = zeros(size(c,1),1); %

for i = 1:size(pS,1)
    c(i) = sqrt(2*kappa(i)/(kappa(i)-1)*pS(i)/ro(i)*(((pC(i)/pS(i)).^((kappa(i)-1)/kappa(i)))-1));
    cu(i) = c(i)*cos(deg2rad(Eps32(i)))*sin(deg2rad(Alpha(i) - 90));
    cz(i) = c(i)*cos(deg2rad(Eps32(i)))*cos(deg2rad(Alpha(i) - 90
    cr(i) = c(i)*sin(deg2rad(Eps32(i)));
    deltac(i) = rad2deg(atan(cr(i)/cz(i)));
end

%% Mach Number
Ma = c ./ SoS; % Mach number

ErrMa = zeros(size(Ma,1),1);
% Measurement Error based on Mach number
for i = 1 : size(Ma,1)
    ErrMa(i) = 100*abs(1 - sqrt(2/(kappa(i)*(Ma(i).^2))*((1 + (kappa(i) -
1)/2*(Ma(i).^2))^(kappa(i)/(kappa(i)-1)) - 1)));
end

figure()
plot(Ma,ErrMa,'xr')
title('Measurement error due to fluid compressibility')
```

```
xlabel('Mach number [-]')
ylabel('Measurement error [%]')
```

III Uncertainty

```
%% Uncertainty of regression equations
ErrNS = 0.0015 * 103421;   % NetScanner error [Pa], Type B
ErrRM = 0.00035 * 207000;   % Rosemount error [Pa], Type B

% pb
stdpb = std(pb)./sqrt(size(pb,1)); % Type A error
ErrRM_pb = 2*sqrt(stdpb.^2+ErrRM.^2); % Combined and factored by 2 -> CL = 95 %
% pB
stdpB = std(CalcData(:,10))./sqrt(size(CalcData(:,10),1)); % Type A error
ErrNS_pB = 2*sqrt(stdpB.^2+ErrNS.^2); % Combined and factored by 2 -> CL = 95 %
% p2
stdp2 = std(CalcData(:,7))./sqrt(size(CalcData(:,7),1)); % Type A error
ErrNS_p2 = 2*sqrt(stdp2.^2+ErrNS.^2); % Combined and factored by 2 -> CL = 95 %
% p3
stdp3 = std(CalcData(:,8))./sqrt(size(CalcData(:,8),1)); % Type A
ErrNS_p3 = 2*sqrt(stdp3.^2+ErrNS.^2); % Combined and factored by 2 -> CL = 95 %


% pB margins
pB_p = (pb + ErrRM_pb) - (CalcData(:,10) - ErrNS_pB); % Upper boundary
pB_m = (pb - ErrRM_pb) - (CalcData(:,10) + ErrNS_pB); % Lower boundary
% p2 margins
p2_p = (pb + ErrRM_pb) - (CalcData(:,7) - ErrNS_p2); % Upper boundary
p2_m = (pb - ErrRM_pb) - (CalcData(:,7) + ErrNS_p2); % Lower boundary
% p3 margins
p3_p = (pb + ErrRM_pb) - (CalcData(:,8) - ErrNS_p3); % Upper boundary
p3_m = (pb - ErrRM_pb) - (CalcData(:,8) + ErrNS_p3); % Lower boundary
% K32
K32_p = (p3_p - pB_p)./(p2_p - pB_p); % Upper boundary
K32_m = (p3_m - pB_m)./(p2_m - pB_m); % Lower boundary

% Eps32
% Calculated epsilon values through calibration equation and upper/lower boundary
% of coefficient K32
Eps_p = feval(fitresults{1}, K32_p);
Eps_m = feval(fitresults{1}, K32_m);

%% kD2 and kD3
% Calculated dynamic pressure coefficients through calibration equations and upper/lower boundary
% of epsilon
kD2_p = feval(fitresults{5}, Eps_p);
kD2_m = feval(fitresults{5}, Eps_m);
kD3_p = feval(fitresults{6}, Eps_p);
kD3_m = feval(fitresults{6}, Eps_m);
```

```
kD_p = zeros(size(Eps32,1),1);
kD_m = zeros(size(Eps32,1),1);

% Selection of calculated pressure coefficients based on epsilon value and
% limited epsilon value
for i = 1:size(Eps32,1)
   if Eps_p(i) <= Eps_IndexD
      kD_p(i) = kD2_p(i);
   else
      kD_p(i) = kD3_p(i);
   end

   if Eps_m(i) <= Eps_IndexD
      kD_m(i) = kD2_m(i);
   else
      kD_m(i) = kD3_m(i);
   end
end

%% kS2 and kS3
% Calculated static pressure coefficients through calibration equations and upper/lower
boundary
% of epsilon
kS2_p = feval(fitresults{7}, Eps_p);
kS2_m = feval(fitresults{7}, Eps_m);
kS3_p = feval(fitresults{8}, Eps_p);
kS3_m = feval(fitresults{8}, Eps_m);

kS_p = zeros(size(Eps32,1),1);
kS_m = zeros(size(Eps32,1),1);

% Selection of calculated pressure coefficients based on epsilon value and
% limited epsilon value
for i = 1:size(Eps32,1)
   if Eps_p(i) <= Eps_IndexD
      kS_p(i) = kS2_p(i);
   else
      kS_p(i) = kS3_p(i);
   end

   if Eps_m(i) <= Eps_IndexD
      kS_m(i) = kS2_m(i);
   else
      kS_m(i) = kS3_m(i);
   end
end

%% Static and Dynamic pressure - Measurement uncertainty
pD_p = zeros(size(P,1),1);
```

```
pD_m = zeros(size(P,1),1);

pS_p = zeros(size(P,1),1);
pS_m = zeros(size(P,1),1);
% uncertainty boundaries of static and dynamic pressures
for i = 1 : size(P,1)
    pD_p(i) = (p3_p(i) - pB_p(i))/kD_p(i); % pD higher boundary
    pD_m(i) = (p3_m(i) - pB_m(i))/kD_m(i); % pD lower boundary
    pS_p(i) = (p3_p(i) - kS_p(i)*pD_p(i)); % pS higher boundary
    pS_m(i) = (p3_m(i) - kS_m(i)*pD_m(i)); % pS lower boundary
end
% Total pressure
pC_p = pS_p + pD_p; % higher bound of total pressure
pC_m = pS_m + pD_m; % lower bound of total pressure

%% Speed calculation uncertainty
psteam('open');                  % opens steam tables
vol_p = zeros(size(pS,1),1); % volume upper boundary - f(x,p)
vol_m = zeros(size(pS,1),1); % volume lower boundary - f(x,p)
SoS_p = zeros(size(pS,1),1); % Speed of Sound upper boundary - f(x,p)
SoS_m = zeros(size(pS,1),1); % Speed of Sound lower boundary - f(x,p)
kappa_p = zeros(size(pS,1),1); % Heat capacity ratio upper boundary
kappa_m = zeros(size(pS,1),1); % Heat capacity ratio lower boundary

for i = 1:size(pS,1)
    vol_p(i) = psteam('v_xp',dryness(i),(pS_p(i))*10^-6);
    vol_m(i) = psteam('v_xp',dryness(i),(pS_m(i))*10^-6);
    if dryness(i) >= 1
        SoS_p(i) = psteam('w_gsat_p',dryness(i),(pS_p(i))*10^-6);
        SoS_m(i) = psteam('w_gsat_p',dryness(i),(pS_m(i))*10^-6);
    elseif dryness(i) < 1
        SoS_p(i) = psteam('w_xp',dryness(i),(pS_p(i))*10^-6);
        SoS_m(i) = psteam('w_xp',dryness(i),(pS_m(i))*10^-6);
    end
    kappa_p(i) = psteam('ka_xp',dryness(i),(pS_p(i))*10^-6);
    kappa_m(i) = psteam('ka_xp',dryness(i),(pS_m(i))*10^-6);
end
psteam('close');

ro_p = zeros(size(vol_p,1),1);
ro_m = zeros(size(vol_m,1),1);

for i = 1:size(vol,1)
    ro_p(i) = 1/vol_p(i);   % medium density upper boundary
    ro_m(i) = 1/vol_m(i);   % medium density lower boundary
end

%% upper and lower boundary of speed vector and its components
c_p = zeros(size(pS,1),1);
c_m = zeros(size(pS,1),1);
```

```
cu_p = zeros(size(c_p,1),1);
cu_m = zeros(size(c_p,1),1);
cz_p = zeros(size(c_p,1),1);
cz_m = zeros(size(c_p,1),1);
cr_p = zeros(size(c_p,1),1);
cr_m = zeros(size(c_p,1),1);
deltac_p = zeros(size(c_p,1),1);
deltac_m = zeros(size(c_p,1),1);

for i = 1:size(pS,1)
   c_p(i) = sqrt(2*kappa_p(i)/(kappa_p(i)-
1)*pS_p(i)/ro_p(i)*((pC_p(i)/pS_p(i)).^((kappa_p(i)-1)/kappa_p(i))-1));
   c_m(i) = sqrt(2*kappa_m(i)/(kappa_m(i)-
1)*pS_m(i)/ro_m(i)*((pC_m(i)/pS_m(i)).^((kappa_m(i)-1)/kappa_m(i))-1));

   cu_p(i) = c_p(i)*cos(deg2rad(Eps_p(i)))*sin(deg2rad(Alpha(i) - 90));
   cu_m(i) = c_m(i)*cos(deg2rad(Eps_m(i)))*sin(deg2rad(Alpha(i) - 90));
   cz_p(i) = c_p(i)*cos(deg2rad(Eps_p(i)))*cos(deg2rad(Alpha(i) - 90));
   cz_m(i) = c_m(i)*cos(deg2rad(Eps_m(i)))*cos(deg2rad(Alpha(i) - 90));
   cr_p(i) = c_p(i)*sin(deg2rad(Eps_p(i)));
   cr_m(i) = c_m(i)*sin(deg2rad(Eps_m(i)));

   deltac_p(i) = rad2deg(atan(cr_p(i)/cz_p(i)));
   deltac_m(i) = rad2deg(atan(cr_m(i)/cz_m(i)));
end

%% Highest uncertainties
% Determining of highest uncertainty of dynamic pressure
if max(abs(pD_p - pD)) > max(abs(pD_m - pD))
   [deltapD,I] = max(abs(pD_p - pD));
else
   [deltapD,I] = max(abs(pD_m - pD));
end
[MeanCounted] = MeanIntervalCounter(Points, I, pD)
deltapD
perc_pD = max(deltapD/MeanCounted)*100 % relative uncertainty in %

% Determining of highest uncertainty of static pressure
if max(abs(pS_p - pS)) > max(abs(pS_m - pS))
   [deltapS,I] = max(abs(pD_p - pD));
else
   [deltapS,I] = max(abs(pD_p - pD));
end
[MeanCounted] = MeanIntervalCounter(Points, I, pS)
deltapS
perc_pS = max(deltapS./MeanCounted)*100 % relative uncertainty in %

% Determining of highest uncertainty of steam velocity c
if max(abs(c_p - c)) > max(abs(c_m - c))
   [delta_c,I] = max(abs(c_p - c));
```

```
else
    [delta_c,I] = max(abs(c_m - c));
end
[MeanCounted] = MeanIntervalCounter(Points, I, c)
delta_c
perc_c = max(delta_c./MeanCounted)*100 % relative uncertainty in %

% Determining of highest uncertainty of steam velocity component cu
if max(abs(cu_p - cu)) > max(abs(cu_m - cu))
    [delta_cu,I] = max(abs(cu_p - cu));
else
    [delta_cu,I] = max(abs(cu_m - cu));
end
[MeanCounted] = MeanIntervalCounter(Points, I, cu)
delta_cu
perc_cu = max(delta_cu./MeanCounted)*100 % relative uncertainty in %

% Determining of highest uncertainty of steam velocity component cz
if max(abs(cz_p - cz)) > max(abs(cz_m - cz))
    [delta_cz,I] = max(abs(cz_p - cz));
else
    [delta_cz,I] = max(abs(cz_m - cz));
end
[MeanCounted] = MeanIntervalCounter(Points, I, cz)
delta_cz
perc_cz = max(delta_cz./MeanCounted)*100 % relative uncertainty in %

% Determining of highest uncertainty of steam velocity cr
if max(abs(cr_p - cr)) > max(abs(cr_m - cr))
    [delta_cr,I] = max(abs(cr_p - cr));
else
    [delta_cr,I] = max(abs(cr_m - cr));
end
[MeanCounted] = MeanIntervalCounter(Points, I, cr)
delta_cr
perc_cr = max(delta_cr./MeanCounted)*100 % relative uncertainty in %

% Determining of highest uncertainty of steam velocity angle delta
if max(abs(deltac_p - deltac)) > max(abs(deltac_m - deltac))
    [delta_dc,I] = max(abs(deltac_p - deltac));
else
    [delta_dc,I] = max(abs(deltac_m - deltac));
end
[MeanCounted] = MeanIntervalCounter(Points, I, deltac)
delta_dc
perc_dc = max(abs(delta_dc./MeanCounted))*100 % relative uncertainty in %

%% Graphs

% Speed of sound along the blade
```

```
figure()
plot(SoS(1:q),PP(1:q), '--xb')
title('Speed of sound along the blade')
xlabel('Speed of sound [m/s]')
ylabel('Distance from the blade root [mm]')

% Steam velocity c and its boundaries (uncertainty)
figure()
plot(c(1:q),PP(1:q), '-xr')
hold on
plot(c_p(1:q),PP(1:q), '--k')
plot(c_m(1:q),PP(1:q), '--k')
hold off
legend('Steam velocity c','Lower boundary','Upper boundary')
title('Steam velocity along the blade')
xlabel('Steam velocity [m/s]')
ylabel('Distance from the blade root [mm]')
```

MeanIntervalCounter

```
function [MeanCounted] = MeanIntervalCounter(Points, I, Variab)
% this function calculates the mean values for the set of values measured at
% the same point where the highest uncertainty is located.
   Vel = size(Points,1);  % number of all measurements
   OneInt = Vel/size(unique(Points),1); % number of all measurements at one point
   Breaker = I/OneInt; % in which point is the highest uncertainty located
   Fl = floor(Breaker); % boundaries of the interval with the highest uncertainty
   Ce = ceil(Breaker); % boundaries of the interval with the lowest uncertainty
   % selects rows of the interval where the highest uncertainty is located
   if Fl ~= Ce && Fl ~= 0
      lim1 = OneInt*Fl;
      lim2 = OneInt*Ce;
   elseif Fl ~= Ce && Fl == 0
         lim1 = 1;
         lim2 = OneInt*(Ce);
   elseif Fl == Ce && (Fl == 0 || Fl == 1)
         lim1 = 1;
         lim2 = OneInt;
   else
         lim1 = OneInt*(Fl-1);
         lim2 = OneInt*Ce;
   end
   % mean value of an arbitrary variable within the determined interval
   MeanCounted = mean(Variab(lim1:lim2));
end
```

IV Statistics

```
%% Clear and close
```

```
clear;clc; close all;

%% Data loading and random selection from Turbine Raw Data
load Statistics_InFront.mat  % loads data from the front blade
% load Statistics_Behind.mat % loads data from the back blade

Data = zeros((size(unique(DataAll(:,3)),1)),1);
l = 1;

Number = floor(size(unique(DataAll(:,3)),1)*rand) % randomly selects number

if Number < 1
   Number = 23;
end

% Selects only the data of a specific measurement point given by variable 'Number'
for i = 1 : size(DataAll,1)
   if DataAll(i,3) == Number
      Data(l,1:size(DataAll,2)) = DataAll(i,:);
      l = l + 1;
   end
end

%% Column Selection for further examination
close all;
TestedVar = Data(:,7); % Tested variable is pressure p2

%% Trend and Periodicity
t = size(Data,1)/100; % Number of Samples for 1 point, frequency = 100 samples/s
T = linspace(1,t,size(Data,1));
T = T';
% T = T(60:80); % for Trend only
% TestedVar = TestedVar(60:80); % for Trend only

[p,s] = polyfit(T,TestedVar,1); % polynomial fitresult
[fitresult, gof] = fit( T, TestedVar, 'poly1' ); % cfit fitresult for prediction interval

[yfit,dy] = polyconf(p,T,s, 'predopt', 'curve'); % confidence band
predInt = predint(fitresult,T,0.95,'observation','off'); % prediction interval

figure()
plot(T,TestedVar, 'bx')
hold on
plot(T, yfit,'color','r')
plot(T, yfit-dy, ':b') % confidence band 95 %
plot(T, yfit+dy, ':b') % confidence band 95 %
plot(T, predInt, '--g')
hold off
xlabel('Time [s]')
ylabel('Pressure p2 [Pa]')
```

```
legend('Variable points','Trend','Confidence band lower','Confidence band upper', 'Prediction bounds')

%% Normality tests
h = kstest(TestedVar);
hh = adtest(TestedVar);
hhh = ttest(TestedVar);

%% Histogram and basic statistics
 % histogram
Fit = fitdist(TestedVar,'Normal');
Stred = Fit.mu; % Mean
sig = Fit.sigma; % standard deviation

figure()
histfit(TestedVar)
xlabel('Pressure p_{2} [Pa]')
ylabel('Number of appearances [-]')
title('Histogram of pressure p_{2}')
hold on
line([Stred, Stred], ylim, 'Color', 'r', 'LineWidth', 2);
line([Stred + sig, Stred + sig], ylim, 'Color', 'r', 'LineWidth', 1.5);
line([Stred - sig, Stred - sig], ylim, 'Color', 'r', 'LineWidth', 1.5);
line([Stred + 2*sig, Stred + 2*sig], ylim, 'Color', 'r','LineStyle', '-.', 'LineWidth', 1);
line([Stred - 2*sig, Stred - 2*sig], ylim, 'Color', 'r', 'LineStyle', '-.', 'LineWidth', 1);
hold off
yl = ylim; % Get limits of y axis so we can find a nice height for the text labels.
message = sprintf('%.1f  ', Stred);
text(Stred, 0.95 * yl(2), message, 'Color', 'r', 'HorizontalAlignment', 'right', 'FontWeight', 'bold');
message = sprintf('  %.1f', Stred+sig);
text(Stred+sig, 0.9 * yl(2), message, 'Color', 'r', 'FontSize', 15);
message = sprintf('%.1f  ', Stred-sig);
text(Stred-sig, 0.9 * yl(2), message, 'Color', 'r', 'HorizontalAlignment', 'right', 'FontSize', 15);

message = sprintf('  %.1f', Stred+2*sig);
text(Stred+2*sig, 0.88 * yl(2), message, 'Color', 'r', 'FontSize', 14);
message = sprintf('%.1f  ', Stred-2*sig);
text(Stred-2*sig, 0.88 * yl(2), message, 'Color', 'r', 'HorizontalAlignment', 'right', 'FontSize', 14);
hold on

% Basic statistics
MEDI = median(TestedVar)      % Median
Skew = skewness(TestedVar)    % Skewness
Kurt = kurtosis(TestedVar)    % Kurtosis
SmerOdch = std(TestedVar)     % STD
Rozpyl = var(TestedVar)       % Variance
Prum = mean(TestedVar)        % Mean
Modus = mode(TestedVar)       % Mode
```

```
Diff_abs = abs(Prum - MEDI);        % Absolute mean-median difference [Pa]
Diff_rel = abs(Prum - MEDI)/Prum;    % Relative mean-median difference [-]
line([MEDI, MEDI], ylim, 'Color', 'k', 'LineWidth', 2);
message = sprintf('%.1f  ', MEDI);
text(MEDI + 0.5*sig, 0.95 * yl(2), message, 'Color', 'k', 'HorizontalAlignment', 'right',
'FontWeight', 'bold');

hold off
if Diff_rel > 0.1
   disp('Nonnormality')
else
   disp('Normality')
end
Quantiles = quantile(TestedVar, [0.05 .25 0.5 0.75 0.95]); % 5th, 25th quantile, 50th, 75th,
and 95th

%% Ridge diagram
Sorted = sort(TestedVar);
P = zeros(size(Sorted,1),1);
for i = 1:size(Sorted)
   P(i) = (i - 1/3)/(size(Sorted,1) + 1/3);
   if P(i) <= 0.5
      P(i) = 100*P(i);
   else
       P(i) = 100 - 100*P(i);
   end
end

figure()
plot(Sorted,P, '+r')
xlabel('Sorted pressure p2 [Pa]') % změna, napsat o jaký tlak jde
ylabel('Modified Order Probability y [-]')
title('Ridge diagram')

%% Box plot, PP plot, QQ plot
figure()
boxplot(TestedVar, 'Notch', 'on')
ylabel('Pressure p_{2} [Pa]')
title('Boxplot')

figure()
probplot(TestedVar)
xlabel('Sample Data')
ylabel('Probability')
title('Comparison of Sampla Data and Normal Distributions')

figure()
qqplot(TestedVar)
xlabel('Normal Distribution Quantiles')
```

```
ylabel('Sample Data Quantiles')
title('QQ Plot of Sample Data vs. Standard Normal')
```

V Probe zeroing

```
%% Clear and Load
clear, clc
load SelDataSm_ProbeZeroing.mat; % calling the data
SelData = SelDataSm;

%% Table split in columns
Points = SelData(:,1);      % Points measured
SubPoints = SelData(:,2);
Alpha = SelData(:,3);       % Angle Alpha, looking for 'pL - pP = 0' position
Epsilon = SelData(:,4);     % Angle Epsilon, rather constant
pB = SelData(:,5);          % relative pB pressure
pP = SelData(:,6);          % relative pP pressure
pL = SelData(:,7);          % relative pL pressure

%% Dependency of progress of relative pressure on angle Alpha
figure()
plot(Alpha, pP, '-b')
hold on
grid on
plot(Alpha, pL, '-r')
plot(Alpha, pB, '-g')
hold off
title('Progress of relative pressures at angle \alpha')
xlabel('Angle \alpha  [°]')
ylabel('Relative pressure [Pa]')
legend('pressure pP','pressure pL','pressure pB')

%% Dependency of delta p (pL-pP) on angle Alpha
figure()
dp = pL - pP;
plot(Alpha, dp, '-r')
grid on
title('Progress of pressure difference on angle \alpha')
xlabel('Angle \alpha [°]')
ylabel('\Deltap = p_{L} - p_{P} [Pa]')

%% Probe sensitivity
dAlpha = diff(Alpha);
ddp = diff(dp);
der = dAlpha ./ ddp;
Mean_der = mean(der);

Array_der = zeros((size(Alpha,1)-1),1);
for i = 1:(size(Alpha,1))
```

```
    Array_der(i) = Mean_der;
end

figure()
plot(Alpha(1:(size(Alpha,1)-1)), der)
grid on
hold on
plot(Alpha(1:size(Alpha,1)),Array_der, '--r', 'LineWidth', 3.5)
hold off
title('Probe sensitivity')
xlabel('Angle \alpha [°]')
ylabel('Derivative d\alpha/d\Delta{p} [°/Pa]')

%% Regression equation of pL - pP progress based on angle Alpha
poly = polyfit(dp,Alpha,3);
X = 1.1*min(dp):0.1:1.1*max(dp);
Y = polyval(poly,X);

figure()
plot(dp, Alpha,'xb')
hold on
plot(X, Y, '-r', 'LineWidth', 2)
hold off
s = sprintf('y = (%.9f) x^3 +(%.9f) x^2 + (%.4f) x + (%.4f)',poly(1),poly(2),poly(3),poly(4));
title('Progress of pressure difference \Deltap = pP - pL')
xlabel('Angle \Deltap = p_{L} - p_{P} [°]')
ylabel('Angle \alpha [°]')

%% Determination of probe rotation towards the interval
dp2 = pL - pB;

figure()
plot(dp2, Alpha)
title('Graph used for probe rotation determination to move towards the desired interval')
xlabel('Angle \deltap = p_{L} - p_{B} [°]')
ylabel('Angle \alpha [°]')
```