Příloha B: Komentované zdrojové kódy

Obsah

KALIBRACE

Program pro kalibraci měřicího systému pro čtyřmikrofonovou metodu měření v akustické impedanční trubici.

Autor: Michal Kašpárek

```
% DAQ: NI USB-6341 (Device ID: 'Dev1')
% version 1.0 - new MIC connection layout

% pouziti: Amplitudovou kalibracni funkci nasobit namerene hodnoty
%         -> A(MIC) * kA(f)
% Fazovou kalibracni funkci / konstantu odecist od namerenych hodnot
%         -> Ph(MIC)- kPh(f)
```

```
clc;
close all;
clear all;


FsReq =  40000;         % max 125k for 4 channels
duration = 0.81;        % sec
Aout = 0.2;             % V

timeOffset    = 0.3;      % sec
timeToAnalyze = 0.5;      % sec

f_min = 40;    % [Hz]
f_max = 1100;

n_measures = 120;
% freqVector = round(logspace(log10(f_min),log10(f_max),n_measures));
freqVector = f_min:f_max;

% [i][MIC no][char]   (freq, Amp, Phase)
freqChar4MIC = zeros(length(freqVector), 4, 3);
freqChar4MICLocalMax = zeros(length(freqVector), 4);
```

initialisation of DAQ

```
daq.getDevices();

s = daq.createSession('ni');

ch1 = addAnalogInputChannel(s,'Dev1',4,'Voltage');      % MIC1
ch2 = addAnalogInputChannel(s,'Dev1',5,'Voltage');      % MIC2
ch3 = addAnalogInputChannel(s,'Dev1',6,'Voltage');      % MIC3
ch4 = addAnalogInputChannel(s,'Dev1',7,'Voltage');      % MIC4

ch1.TerminalConfig = 'SingleEnded';
ch2.TerminalConfig = 'SingleEnded';
ch3.TerminalConfig = 'SingleEnded';
ch4.TerminalConfig = 'SingleEnded';
```

```
ch1.Range = [-0.2,0.2];
ch2.Range = [-0.2,0.2];
ch3.Range = [-0.2,0.2];
ch4.Range = [-0.2,0.2];

cho = addAnalogOutputChannel(s,'Dev1','ao0','Voltage');

s.Rate = FsReq;
FsReal = s.Rate;

for count=1:length(freqVector),
```

Output data definition

```
    F = freqVector(count);
    dataOut = Aout * sin(linspace(0,(duration*2*pi*F), duration*s.Rate));
```

Measurement

```
    queueOutputData(s, dataOut');
    [dataMeasured,timeStamps,~]= startForeground(s);
    dataM = dataMeasured;                       % used channels 0, 2, 4, 6
```

Calculations

dataM crop - zahodit prvnich "timeOffset" ms a pouzit cele nasobky periody F => NP period

```
    NP = floor(F * timeToAnalyze);  % pocet period k analyze behem "timeToAnalyze"
    dataCrop = dataM((floor(timeOffset*FsReal):floor(timeOffset*FsReal+(NP*FsReal/F))),:);

    W = [cos(2*pi*F*timeStamps(1:(length(dataCrop(:,1)))))
sin(2*pi*F*timeStamps(1:(length(dataCrop(:,1)))))]; % tmp vector

    phasorMconst = zeros(2,4);   % phasorMconst(:,1) = Amp MIC1, Phase MIC1
    phasorMcomplex = zeros(1,4);

    for im = 1:4,
        C = (W\dataCrop(:,im));
        amp=sqrt(C(1)^2 + C(2)^2);
        phi=atan2(C(1),C(2));

        phasorMconst(:, im) = [amp, phi];
        phasorMcomplex(im) = amp * exp(1i*phi);
    end;

    freqChar4MIC (count,:, 1) = F;                  % freq
    freqChar4MIC (count,:, 2) =  phasorMconst(1,:); % Amp   [V]
    freqChar4MIC (count,:, 3) =  phasorMconst(2,:); % Phase [rad]

    freqChar4MICLocalMax(count, :) = max(dataCrop);

end;
```

Normalization

```
normalized4MICFreq = freqChar4MIC (:,1, 1);
normalizedChar4MICAmp = 1./freqChar4MIC (:,:, 2);
normalizedChar4MICPhase = freqChar4MIC (:,:, 3);

% prepocet fazoveho zpozdeni ve vztahu ke kanalu MIC1
normalizedChar4MICPhase = normalizedChar4MICPhase - normalizedChar4MICPhase(:,1)*[1 1 1 1];
normalizedChar4MICAmp = normalizedChar4MICAmp./(normalizedChar4MICAmp(:,1)*[1 1 1 1]);

%normalizace rozsahu faze do (-pi, pi)
for ii=1:length(normalizedChar4MICPhase),
    for jj = 1:4,
        while (normalizedChar4MICPhase(ii, jj) >= pi || normalizedChar4MICPhase(ii, jj) < -pi)
            if normalizedChar4MICPhase(ii, jj) >= pi,
                normalizedChar4MICPhase(ii, jj) = normalizedChar4MICPhase(ii, jj) - 2*pi;
            end;
            if normalizedChar4MICPhase(ii, jj) < -pi,
                normalizedChar4MICPhase(ii, jj) = normalizedChar4MICPhase(ii, jj) + 2*pi;
            end;
        end;
    end;
end;
```

Visualisation (optional)

figure('position',[100 100 800 500]); plot(freqChar4MIC(:,1,1), freqChar4MIC(:,:,2)); legend('MIC1','MIC2', 'MIC3', 'MIC4'); title(strcat(['Amplitudová charakteristika 4 MIC pro F = ', num2str(f_min), '-', num2str(f_max), 'Hz'])); xlabel('F [Hz]'); ylabel('Napětí [V]'); grid on;

figure('position',[100 100 800 500]); plot(freqChar4MIC(:,1,1), freqChar4MIC(:,:,3)); legend('MIC1','MIC2', 'MIC3', 'MIC4'); title(strcat(['Fázová charakteristika 4 MIC pro F = ', num2str(f_min), '-', num2str(f_max), 'Hz'])); xlabel('F [Hz]'); ylabel('Fáze [rad]'); grid on;

```
% figure('position',[100 100 800 500]);
% plot(normalized4MICFreq, normalizedChar4MICAmp); legend('MIC1','MIC2', 'MIC3', 'MIC4');
% title(strcat(['Amplitudové kalibrační charakteristiky 4 MIC pro F =  ', num2str(f_min), '-',
num2str(f_max), 'Hz']));
% xlabel('F [Hz]'); ylabel('Relativní napětí'); grid on;
%
% figure('position',[100 100 800 500]);
% plot(normalized4MICFreq, normalizedChar4MICPhase); legend('MIC1','MIC2', 'MIC3', 'MIC4');
% title(strcat(['Fázové kalibrační charakteristiky 4 MIC pro F =  ', num2str(f_min), '-',
num2str(f_max), 'Hz']));
% xlabel('F [Hz]'); ylabel('relativní fáze [rad]'); grid on;
```

Saving (optional)

Ukládané proměnné: normalized4MICFreq; normalizedChar4MICAmp; normalizedChar4MICPhase;

```
% FileName = strcat('.\Kalibrace\CalibrationTrubice4_', datestr(now,'yyyymmdd_HHMMSS'));
% save(FileName, 'normalized4MICFreq', 'normalizedChar4MICAmp', 'normalizedChar4MICPhase');
```

TRUBICE4_SWEEP

Program pro měření akustických veličin čtyřmikrofonovou metodou v akustické impedanční trubici.

Autor: Michal Kašpárek

```
% DAQ: NI USB-6341 (Device ID: 'Dev1')
% version 1.0 - new MIC connection layout

clc;
close all;
clear all;

FsReq =  40000;         % max 125k for 4 channels
duration = 0.81;        % sec
Aout = 0.05;            % V

timeOffset    = 0.3;    % sec
timeToAnalyze = 0.5;    % sec


f_min = 130;
f_max = 800;
n_measures = 200;
freqVector = round(logspace(log10(f_min),log10(f_max),n_measures));

% Tloušťka měřeného vzorku [m]
d = 0.002;

% Vzdalenosti mikrofonu MIC1-MIC4 od cela vzorku: x1, x2, x3, x4
x1 = -0.6875;
x2 = -0.4575;
x3 = 0.460;
x4 = 0.690;

disp(strcat(['Sweep measurement with ', num2str(n_measures), ' measurements in range ',
num2str(f_min), ' - ', num2str(f_max), ' Hz']));
```

Initialisation of DAQ

```
% daq.getDevices();
s = daq.createSession('ni');

ch1 = addAnalogInputChannel(s,'Dev1',4,'Voltage');      % MIC1
ch2 = addAnalogInputChannel(s,'Dev1',5,'Voltage');      % MIC2
ch3 = addAnalogInputChannel(s,'Dev1',6,'Voltage');      % MIC3
ch4 = addAnalogInputChannel(s,'Dev1',7,'Voltage');      % MIC4

ch1.TerminalConfig = 'SingleEnded';
ch2.TerminalConfig = 'SingleEnded';
ch3.TerminalConfig = 'SingleEnded';
ch4.TerminalConfig = 'SingleEnded';

ch1.Range = [-0.2,0.2];
```

```
ch2.Range = [-0.2,0.2];
ch3.Range = [-0.2,0.2];
ch4.Range = [-0.2,0.2];

cho = addAnalogOutputChannel(s,'Dev1','ao0','Voltage');

s.Rate = FsReq;     % FsIdeal
FsReal = s.Rate;
```

Loading calibration data

```
fileName = 'CalibrationTrubice4_VUTS';
load(fileName);

% Data loaded:
% normalized4MICFreq;
% normalizedChar4MICAmp;
% normalizedChar4MICPhase;
```

RAW data structure initialisation

```
pocetKanalu = 4;
delkaMereni = round(duration*s.Rate);
RawSweep4MIC = zeros(n_measures, pocetKanalu, delkaMereni);
```

Measurement cycle

```
for count=1:length(freqVector),
% Output data definition
    F = freqVector(count);
       dataOut = Aout * sin(linspace(0,(duration*2*pi*F), duration*s.Rate));

% Measurement
    queueOutputData(s, dataOut');
    [dataMeasured,timeStamps,~]= startForeground(s);
    dataM = dataMeasured;
% Calculations
    NP = floor(F * timeToAnalyze);  % pocet period k analyze behem "timeToAnalyze"
    dataCrop = dataM((floor(timeOffset*FsReal):floor(timeOffset*FsReal+(NP*FsReal/F))),:);

    W = [cos(2*pi*F*timeStamps(1:(length(dataCrop(:,1)))))
sin(2*pi*F*timeStamps(1:(length(dataCrop(:,1)))))];

    phasorMconst = zeros(2,4);   % phasorM(:,1) = Amp Mic1, PhaseMic1 (rad)
    phasorMcomplex = zeros(1,4);

    % Find calibration coefficient
    normalizedCharIndF = find(normalized4MICFreq == F);
    if isempty(normalizedCharIndF),
        disp('ERROR: frequency not found in calibration file');
        break;
    end
```

```
    for im = 1:4,
        C = (W\dataCrop(:,im));
        amp=sqrt(C(1)^2 + C(2)^2);
        phi=atan2(C(1),C(2));

% Aplication of callibration coef.
        amp = amp * normalizedChar4MICAmp(normalizedCharIndF, im);
        phi = phi - normalizedChar4MICPhase(normalizedCharIndF, im);

        phasorMconst(:, im) = [amp, phi];
        phasorMcomplex(im) = amp * exp(1i*phi);
    end;

    soundSpeed = 343.2*sqrt((291)/293); % m/s at cca 18°
  k = 2*pi*F/soundSpeed;          % wave number

% COEFFICIENTS A, B, C, D,

  A =  1i*(phasorMcomplex(1)*exp(1i*k*x2) - phasorMcomplex(2)*exp(1i*k*x1)) / ...
       (2*sin(k*(x1 - x2)));
  B =  1i*(phasorMcomplex(2)*exp(-1i*k*x1) - phasorMcomplex(1)*exp(-1i*k*x2)) / ...
       (2*sin(k*(x1 - x2)));
  C =  1i*(phasorMcomplex(3)*exp(1i*k*x4) - phasorMcomplex(4)*exp(1i*k*x3)) / ...
       (2*sin(k*(x3 - x4)));
  D =  1i*(phasorMcomplex(4)*exp(-1i*k*x3) - phasorMcomplex(3)*exp(-1i*k*x4)) / ...
       (2*sin(k*(x3 - x4)));

  t = 22;             % air temperature in °C
  T = (t + 273.15);     % air temperature in K
  Z = 300;            % Altitude/Elevation above sea level (m) - Lbc 374 m.n.m.
  pAir = 101325 * (1.0 - Z*0.0000225577)^5.2559;  % Pressure dependent on altitude
  RAir = 287.058;                  % specific gas constant for dry air 287.058 J/(kg·K)
  rhoA = pAir/((RAir * T));              % Air Density [kg/m^3]

  Px0 = A + B;
  Vx0 = (A - B) / (rhoA * soundSpeed);



  Pxd = C*exp(-1i*k*d) + D*exp(1i*k*d);
  Vxd = (C*exp(-1i*k*d) - D*exp(1i*k*d)) / (rhoA * soundSpeed);

%   T = matrix 2x2 [T11 T12; T21 T22]
  T = (1/(Px0*Vxd + Pxd*Vx0)) .* [(Pxd*Vxd + Px0*Vx0)  (Px0^2-Pxd^2); (Vx0^2-Vxd^2) (Pxd*Vxd + Px0*Vx0)];

  Ta = (2*exp(1i*k*d)) / (T(1,1) + (T(1,2)/(rhoA * soundSpeed)) + (rhoA * soundSpeed * T(2,1) + T(2,2)));
  Ra = (T(1,1) + (T(1,2)/(rhoA * soundSpeed)) - (rhoA * soundSpeed * T(2,1)) - T(2,2)) / ...
     (T(1,1) + (T(1,2)/(rhoA * soundSpeed)) + (rhoA * soundSpeed * T(2,1)) + T(2,2));
```

```
    dataR(count, 1) = F;
    dataR(count, 2) = abs(Ta);
    dataR(count, 3) = abs(Ra);

    RawSweep4MIC(count, :, :) = dataMeasured';

end;
```

Saving (optional)

Uložení všech získaných časových průběhů - hrubá data k další analýze

```
% FileName = strcat('RawSweep4MIC_', datestr(now,'yyyymmdd_HHMMSS'), '.mat');
% save(FileName, 'RawSweep4MIC');
```

Visualisation (optional)

```
% figure;
% subplot(2,1,1);
% plot(dataR(:, 1), dataR(:, 2));
% grid on; title ('Ra absolutni hodnota transmise'); xlabel('F [Hz]');
% subplot(2,1,2);
% plot(dataR(:, 1), dataR(:, 3));
% grid on; title ('Ra absolutni hodnota odrazu'); xlabel('F [Hz]');
%
% figure
% plot(dataR(:, 1), sqrt((dataR(:, 2)+dataR(:, 3))));
% % plot(dataR(:, 1), 20*log10(dataR(:, 2)));
% grid on; title ('soucet Tr+Re'); xlabel('F [Hz]');
%
% figure('position',[100 100 800 500]);
% plotIndexes = floor(FsReal*timeOffset):floor(FsReal*timeOffset+length(dataCrop)-1);
% plot(timeStamps(plotIndexes), dataCrop(:, :));  legend('MIC1','MIC2', 'MIC3', 'MIC4');
% title(strcat(['Naměřený průběh napětí na kanálech připojených a odpojených mikrofonů pro  F
=  ', num2str(F), ' Hz']));
% xlabel('Čas [s])'); ylabel('Napětí [V]'); grid on; xlim([timeOffset
(timeOffset+timeToAnalyze)]);
```

TRUBICE4_SWEEP_VoltageMFC

Program pro měření akustických a elektrických veličin čtyřmikrofonovou metodou v akustické impedanční trubici.

Je zde využito 8 vstupních kanálů. Kanály AI 0, AI 1 měří průběh napětí na elektrodách MFC aktuátorů. Kanál AI 2 představuje vstup laserového vibrometru. AI 3 je uzemněný vstupní kanál, který slouží k jednoznačnému oddělení akustických – mikrofonních vstupů tak, aby nemohlo dojít k jevu charge injection (viz dokumentace NI). Získaná data kanálu AI 3 jsou zahozena, neslouží k žádným výpočtům ani nejsou ukládána. Kanály AI 4 – AI 7 představují jednotlivé měřicí mikrofony, resp. výstup z mikrofonního předzesilovače.

Autor: Michal Kašpárek

```
% DAQ: NI USB-6341 (Device ID: 'Dev1')
% version 1.0 - new MIC connection layout

% Definice vstupů DAQ (8 kanálů):
% 0 - MFC1 [V] Differential (-1, 1)
% 1 - MFC2 [V] Differential (-1, 1)
% 2 - Vibrometr [V] Differential (-5, 5)
% 3 - GND SingleEnded (-0.2, 0.2) - uzemnění, oddělení MIC vstupů
% 4 - MIC1 [V] SingleEnded (-0.2, 0.2)
% 5 - MIC2 [V] SingleEnded (-0.2, 0.2)
% 6 - MIC3 [V] SingleEnded (-0.2, 0.2)
% 7 - MIC4 [V] SingleEnded (-0.2, 0.2)

% Galvanické oddělení vstupů 0,1,2 (MFC1, MFC2, Vibro) přes Dewetron
% DEWE settings:
% DEWEsetAin(0,2.5,10,0,0,1);
% DEWEsetAin(1,2.5,10,0,0,1);
% DEWEsetAin(2,5,10,0,0,1);
```

```
clc;
close all;
clear all;

FsReq =  40000;       % max 125k for 4 channels
duration = 0.81;                    % sec
Aout = 0.3;                      % V

timeOffset    = 0.3;      % sec
timeToAnalyze = 0.5;      % sec

f_min = 50;
f_max = 1000;
n_measures = 120;
freqVector = round(logspace(log10(f_min),log10(f_max),n_measures));

% Tloušťka měřeného vzorku [m]
    d = 0.002;

% VZDALENOSTI MIKROFONU MIC1+MIC4 od cela vzorku: x1, x2, x3, x4
    x1 = -0.6875;
```

```
    x2 = -0.4575;
    x3 = 0.460;
    x4 = 0.690;

disp(strcat(['Sweep measurement with ', num2str(n_measures), ' measurements in range ',
num2str(f_min), ' - ', num2str(f_max), ' Hz']));
```

Initialisation of DAQ

```
daq.getDevices();

s = daq.createSession('ni');

ch0 = addAnalogInputChannel(s,'Dev1',0,'Voltage');     % V MFC1
ch0.TerminalConfig = 'Differential';
ch0.Range = [-1,1];
ch1 = addAnalogInputChannel(s,'Dev1',1,'Voltage');     % V MFC2
ch1.TerminalConfig = 'Differential';
ch1.Range = [-1,1];

ch2 = addAnalogInputChannel(s,'Dev1',2,'Voltage');     % laser Vibrometer
ch2.TerminalConfig = 'Differential';
ch2.Range = [-5,5];

ch3 = addAnalogInputChannel(s,'Dev1',3,'Voltage');     % GND
ch3.TerminalConfig = 'SingleEnded';
ch3.Range = [-0.2,0.2];

ch4 = addAnalogInputChannel(s,'Dev1',4,'Voltage');      % MIC1
ch5 = addAnalogInputChannel(s,'Dev1',5,'Voltage');      % MIC2
ch6 = addAnalogInputChannel(s,'Dev1',6,'Voltage');      % MIC3
ch7 = addAnalogInputChannel(s,'Dev1',7,'Voltage');      % MIC4

ch4.TerminalConfig = 'SingleEnded';
ch5.TerminalConfig = 'SingleEnded';
ch6.TerminalConfig = 'SingleEnded';
ch7.TerminalConfig = 'SingleEnded';

ch4.Range = [-0.2,0.2];
ch5.Range = [-0.2,0.2];
ch6.Range = [-0.2,0.2];
ch7.Range = [-0.2,0.2];

cho = addAnalogOutputChannel(s,'Dev1','ao0','Voltage');

s.Rate = FsReq;
FsReal = s.Rate;
```

Loading calibration data

```
fileName = 'CalibrationTrubice4_VUTS';
load(fileName);

% Data loaded:
```

```
% normalized4MICFreq;
% normalizedChar4MICAmp;
% normalizedChar4MICPhase;
```

RAW data structure initialisation

```
pocetKanalu = 8;
delkaMereni = round(duration*s.Rate);
RawSweep2Voltage1Vibro4MIC = zeros(n_measures, pocetKanalu, delkaMereni);
```

Measurement cycle

```
for count=1:length(freqVector),
% Output data definition
    F = freqVector(count);
       dataOut = Aout * sin(linspace(0,(duration*2*pi*F), duration*s.Rate));

% Measurement
    queueOutputData(s, dataOut');
    [dataMeasured,timeStamps,~]= startForeground(s);
    dataM = dataMeasured(:,5:8);

% Calculations
    NP = floor(F * timeToAnalyze);           % pocet period k analyze behem 500ms
    dataCrop = dataM((floor(timeOffset*FsReal):floor(timeOffset*FsReal+(NP*FsReal/F))),:);

    W = [cos(2*pi*F*timeStamps(1:(length(dataCrop(:,1)))))
sin(2*pi*F*timeStamps(1:(length(dataCrop(:,1)))))]; % tmp vector

    phasorMconst = zeros(2,4);   % phasorM(:,1) = Amp Mic1, PhaseMic1 (rad)
    phasorMcomplex = zeros(1,4);

% Find calibration coefficient
    normalizedCharIndF = find(normalized4MICFreq == F);
    if isempty(normalizedCharIndF),
        disp('ERROR: frequency not found in calibration file');
        break;
    end
    mldivide
    for im = 1:4,
        C = (W\dataCrop(:,im));
        amp=sqrt(C(1)^2 + C(2)^2);
        phi=atan2(C(1),C(2));

% Aplication of callibration coef.
        amp = amp * normalizedChar4MICAmp(normalizedCharIndF, im);
        phi = phi - normalizedChar4MICPhase(normalizedCharIndF, im);

        phasorMconst(:, im) = [amp, phi];
        phasorMcomplex(im) = amp * exp(1i*phi);
    end;

    soundSpeed = 343.2*sqrt((291)/293); % m/s at cca 18°
    k = 2*pi*F/soundSpeed;              % wave number
```

```
% COEFFICIENTS A, B, C, D,

    A =  1i*(phasorMcomplex(1)*exp(1i*k*x2) - phasorMcomplex(2)*exp(1i*k*x1)) / ...
            (2*sin(k*(x1 - x2)));
    B =  1i*(phasorMcomplex(2)*exp(-1i*k*x1) - phasorMcomplex(1)*exp(-1i*k*x2)) / ...
            (2*sin(k*(x1 - x2)));
    C =  1i*(phasorMcomplex(3)*exp(1i*k*x4) - phasorMcomplex(4)*exp(1i*k*x3)) / ...
            (2*sin(k*(x3 - x4)));
    D =  1i*(phasorMcomplex(4)*exp(-1i*k*x3) - phasorMcomplex(3)*exp(-1i*k*x4)) / ...
            (2*sin(k*(x3 - x4)));

    t = 22;                 % air temperature in °C
    T = (t + 273.15);       % air temperature in K
    Z = 300;                % Altitude/Elevation above sea level (m) - Lbc 374 m.n.m.
    pAir = 101325 * (1.0 - Z*0.0000225577)^5.2559;  % Pressure dependent on altitude
    RAir = 287.058;                                 % specific gas constant for dry air
287.058 J/(kg·K)
    rhoA = pAir/((RAir * T));                       % Air Density [kg/m^3]

    Px0 = A + B;
    Vx0 = (A - B) / (rhoA * soundSpeed);


    Pxd = C*exp(-1i*k*d) + D*exp(1i*k*d);
    Vxd = (C*exp(-1i*k*d) - D*exp(1i*k*d)) / (rhoA * soundSpeed);

%    T = matrix 2x2 [T11 T12; T21 T22]
    T = (1/(Px0*Vxd + Pxd*Vx0)) .* [(Pxd*Vxd + Px0*Vx0)  (Px0^2-Pxd^2); (Vx0^2-Vxd^2) (Pxd*Vxd
+ Px0*Vx0)];

    Ta = (2*exp(1i*k*d)) / (T(1,1) + (T(1,2)/(rhoA * soundSpeed)) + (rhoA * soundSpeed *
T(2,1) + T(2,2)));
    Ra = (T(1,1) + (T(1,2)/(rhoA * soundSpeed)) - (rhoA * soundSpeed * T(2,1)) - T(2,2)) / ...
         (T(1,1) + (T(1,2)/(rhoA * soundSpeed)) + (rhoA * soundSpeed * T(2,1)) + T(2,2));

    dataR(count, 1) = F;
    dataR(count, 2) = abs(Ta);
    dataR(count, 3) = abs(Ra);

    RawSweep2Voltage1Vibro4MIC(count, :, :) = dataMeasured';
end;
```

Saving (optional)

Uložení všech získaných časových průběhů - hrubá data k další analýze

```
% FileName = strcat('RawSweep2Voltage1Vibro4MIC', datestr(now,'yyyymmdd_HHMMSS'), '.mat');
% save(FileName, 'RawSweep2Voltage1Vibro4MIC');
```

Visualisation (optional)

```
% figure('position',[100 100 800 500]);
% subplot(2,1,1);
```

```
% plot(dataR(:, 1), dataR(:, 2));
% grid on; title ('Ta absolutni hodnota transmise'); xlabel('F [Hz]');
% % ylim([0 1]);
% subplot(2,1,2);
% plot(dataR(:, 1), dataR(:, 3));
% grid on; title ('Ra absolutni hodnota odrazu'); xlabel('F [Hz]');
% % ylim([0 1]);

% figure('position',[100 100 800 500]);
% plot(dataR(:, 1), sqrt((dataR(:, 2)+dataR(:, 3))));
% % plot(dataR(:, 1), 20*log10(dataR(:, 2)));
% grid on; title ('soucet Tr+Re'); xlabel('F [Hz]');

% figure('position',[100 100 800 500]);
% plotIndexes = floor(FsReal*timeOffset):floor(FsReal*timeOffset+length(dataCrop)-1);
% plot(timeStamps(plotIndexes), dataCrop(:, :));  legend('MIC1','MIC2', 'MIC3', 'MIC4');
% title(strcat(['Naměřený průběh napětí na kanálech připojených a odpojených mikrofonů pro  F
=  ', num2str(F), ' Hz']));
% xlabel('Čas [s])'); ylabel('Napětí [V]'); grid on; xlim([timeOffset
(timeOffset+timeToAnalyze)]);

% figure('position',[100 100 800 500], 'name', 'Vizualizace kopírování vstupu s vysokou Z a
řešení kompenzací');
% grid on; title ('Problém multiplexoru NI 6341 s injektáží náboje');
% xlabel('Čas [s])'); ylabel('Napětí [V]');
% plot(timeStamps(:), dataMeasured(:, 1:3));
% legend('1: MIC1','2: Nepripojeny vstup', '3: MIC2- odpojeny mikrofon');
% xlim([0.16 0.17]);

% figure('position',[100 100 800 500], 'name', 'Verifikace "čistoty" kanálu s odpojeným
mikrofonem - nové zapojení');
% grid on; title ('Verifikace "čistoty" kanálu s odpojeným mikrofonem - nové zapojení');
% xlabel('Čas [s])'); ylabel('Napětí [V]');
% plot(timeStamps(:), dataMeasured(:, 5:6));
% legend('1: MIC1', '2: MIC2 - odpojeny mikrofon');
% xlim([0.16 0.17]);
```

Analýza a vizualizace RAW dat - měřeni Sweep VUTS - prázdná trubice

Program pro analýzu a vizualizaci výsledků měření prázdné trubice bez vloženého vzorku při verifikaci funkčnosti systému impedanční trubice v bezodrazné komoře VUTS.

Autor: Michal Kašpárek

```
% Kalibracni soubor 40-1100 Hz
% Datový soubor 01 -> 50-600 Hz 200p


clc;
clear all;
close all;

FsReq =  40000;
FsReal = 40000;
duration = 0.81;          % sec

timeOffset    = 0.3;      % sec
timeToAnalyze = 0.5;      % sec

f_min = 50;
f_max = 600;
n_measures = 200;
freqVector = round(logspace(log10(f_min),log10(f_max),n_measures));

% Tloušťka měřeného vzorku [m]
    d = 0.001;
% VZDALENOSTI MIKROFONU MIC1-MIC4 od cela vzorku: x1, x2, x3, x4
    x1 = -0.6875;
    x2 = -0.4575;
    x3 = 0.460;
    x4 = 0.690;
```

Loading

```
    names = { '01RawSweep_VUTS_prazdny_201709204MIC_50-600Hz_200points_A-0.2.mat' };
% Calibration data
    fileName = 'CalibrationTrubice4_VUTS.mat';
    load(fileName);

% Data loaded:
    % normalized4MICFreq;
    % normalizedChar4MICAmp;
    % normalizedChar4MICPhase;
```

Calculations

```
% freqChar(noMeasR-26, F-200, ch-4, A/Ph-2)
freqChar = zeros(length(names),length(freqVector), 4, 2);
```

```
TRall = zeros(length(names),length(freqVector), 2); % (measR, F, F/TR)
RRall = zeros(length(names),length(freqVector), 2); % (measR, F, F/RR)

% 50-600 Hz 200p
for noMeasR = 1:length(names),          % Jednotliva mereni
    load(char(names(noMeasR,:)));       % RawSweep4MIC

    for countF=1:length(freqVector),    % Frekvence v jdntl mereni
        F = freqVector(countF);
        NP = floor(F * timeToAnalyze);  % pocet period k analyze

        cropLength = floor(NP*FsReal/F);
        sampleRange = (floor(timeOffset*FsReal):floor(timeOffset*FsReal+cropLength-1));
        dataCrop = reshape(RawSweep4MIC(countF, :, sampleRange), [4 cropLength]);
        dataCrop = dataCrop';

        phasorMconst = zeros(2,4);   % phasorM(:,1) = Amp Mic1, PhaseMic1 (rad)

% Find calibration coefficient
        normalizedCharIndF = find(normalized4MICFreq == F);
        if isempty(normalizedCharIndF),
            disp('ERROR: frequency not found in calibration file');
            break;
        end
% Calculating phasor
        timeStamps = (linspace(0, timeToAnalyze, length(dataCrop(:,1))))';
        W = [cos(2*pi*F*timeStamps(1:(length(dataCrop(:,1)))))
sin(2*pi*F*timeStamps(1:(length(dataCrop(:,1)))))]; % tmp vector
        C = (W\dataCrop(:,:));
        amp=sqrt(C(1,:).^2 + C(2,:).^2);
        phi=atan2(C(1,:),C(2,:));

        for im = 1:4,
            amp(im) = amp(im) * normalizedChar4MICAmp(normalizedCharIndF, im);
            phi(im) = phi(im) - normalizedChar4MICPhase(normalizedCharIndF, im);

            phasorMconst(:, im) = [amp(im), phi(im)];
            phasorMcomplex(im) = amp(im) * exp(1i*phi(im));
        end;

% Calculating TR, RR
    soundSpeed = 343.2*sqrt((291)/293); % m/s at cca 18°
  k = 2*pi*F/soundSpeed;          % rad/m, wave number -> complex??

    A =  1i*(phasorMcomplex(1)*exp(1i*k*x2) - phasorMcomplex(2)*exp(1i*k*x1)) / ...
        (2*sin(k*(x1 - x2)));
    B =  1i*(phasorMcomplex(2)*exp(-1i*k*x1) - phasorMcomplex(1)*exp(-1i*k*x2)) / ...
        (2*sin(k*(x1 - x2)));
    C =  1i*(phasorMcomplex(3)*exp(1i*k*x4) - phasorMcomplex(4)*exp(1i*k*x3)) / ...
        (2*sin(k*(x3 - x4)));
    D =  1i*(phasorMcomplex(4)*exp(-1i*k*x3) - phasorMcomplex(3)*exp(-1i*k*x4)) / ...
        (2*sin(k*(x3 - x4)));
```

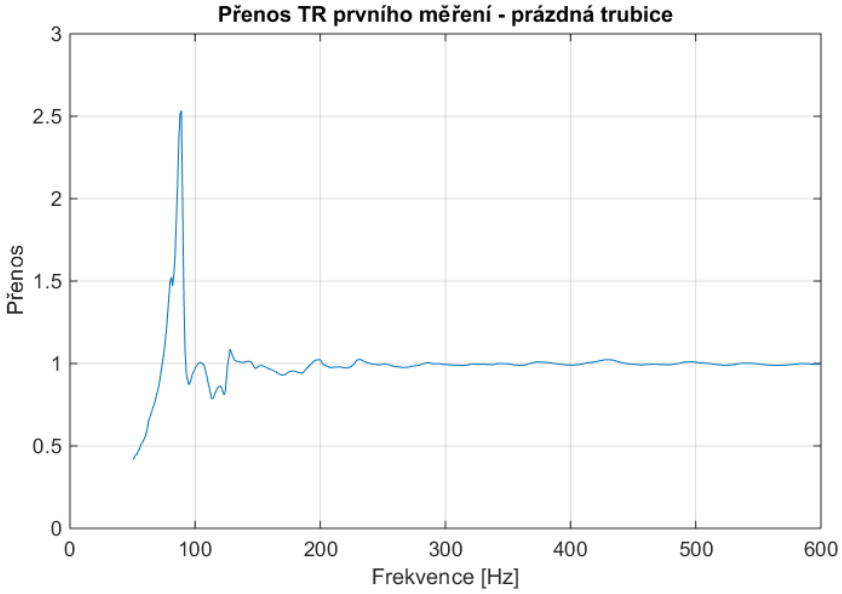
```
    t = 22;             % air temperature in °C
    T = (t + 273.15);   % air temperature in K
    Z = 300;            % Altitude/Elevation above sea level (m) - Lbc 374 m.n.m.
    pAir = 101325 * (1.0 - Z*0.0000225577)^5.2559;  % Pressure dependent on altitude
    RAir = 287.058;                 % specific gas constant for dry air 287.058 J/(kg·K)
    rhoA = pAir/((RAir * T));       % Air Density [kg/m^3]

    Px0 = A + B;
    Vx0 = (A - B) / (rhoA * soundSpeed);



    Pxd = C*exp(-1i*k*d) + D*exp(1i*k*d);
    Vxd = (C*exp(-1i*k*d) - D*exp(1i*k*d)) / (rhoA * soundSpeed);

  %   T = matrix 2x2 [T11 T12; T21 T22]
    T = (1/(Px0*Vxd + Pxd*Vx0)) .* [(Pxd*Vxd + Px0*Vx0)  (Px0^2-Pxd^2); (Vx0^2-Vxd^2) (Pxd*Vxd + Px0*Vx0)];

    Ta = (2*exp(1i*k*d)) / (T(1,1) + (T(1,2)/(rhoA * soundSpeed)) + (rhoA * soundSpeed * T(2,1) + T(2,2)));
    Ra = (T(1,1) + (T(1,2)/(rhoA * soundSpeed)) - (rhoA * soundSpeed * T(2,1)) - T(2,2)) / ...
        (T(1,1) + (T(1,2)/(rhoA * soundSpeed)) + (rhoA * soundSpeed * T(2,1)) + T(2,2));


  freqChar(noMeasR, countF, :, :) = phasorMconst';

  TRall(noMeasR, countF, 1) = F;
  TRall(noMeasR, countF, 2) = abs(Ta);
  RRall(noMeasR, countF, 1) = F;
  RRall(noMeasR, countF, 2) = abs(Ra);



  end;

  clear RawSweep4MIC;
end;
```

VISUALISATION

TR prenos prvniho mereni prazdne trubice

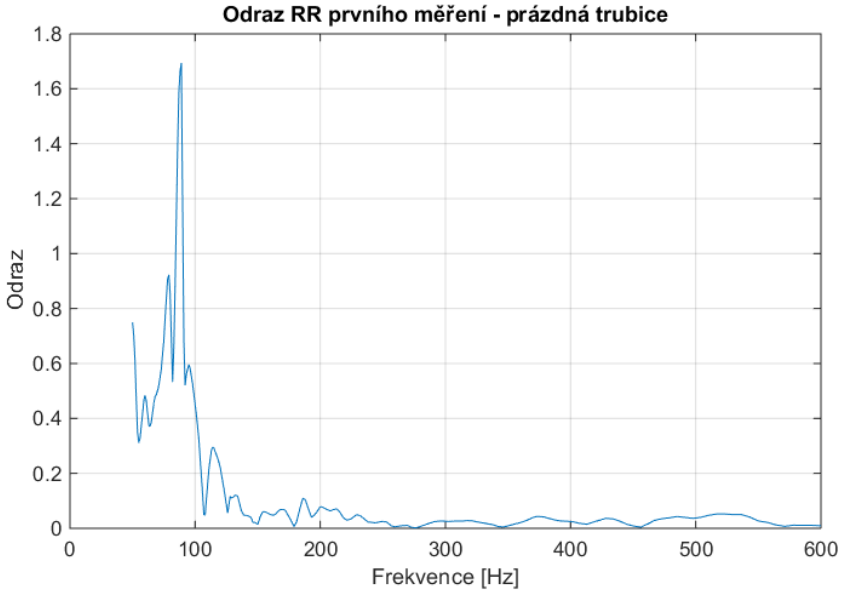
```
figure('position',[100 100 800 500], 'name', 'Přenos TR prvního měření - prázdná trubice');
plot(TRall(1, :, 1), TRall(1, :, 2));
% legend('Jen polykarbonnát','kompozit el.volně long', 'kompozit el. zkrat. long','kompozit
el.volně short','kompozit el. zkrat short');
title('Přenos TR prvního měření - prázdná trubice');
```

```
xlabel('Frekvence [Hz]'); ylabel('Přenos'); grid on;
% ylim([0 1]); xlim([50 600]);
```



RR odraz prvniho mereni prazdne trubice

```
figure('position',[100 100 800 500], 'name', 'Odraz RR prvního měření - prázdná trubice');
plot(RRall(1, :, 1), RRall(1, :, 2));
% legend('Jen polykarbonnát','kompozit el.volně long', 'kompozit el. zkrat. long','kompozit
el.volně short','kompozit el. zkrat short');
title('Odraz RR prvního měření - prázdná trubice');
xlabel('Frekvence [Hz]'); ylabel('Odraz'); grid on;
% ylim([0 1]); xlim([50 600]);
```



Analyza a vizualizace RAW dat - mereni Sweep VUTS

Program pro analýzu a vizualizaci akustických veličin zjištěných čtyřmikrofonovou metodou v akustické impedanční trubici při měření v bezodrazné komoře VUTS. Algoritmus lze jednoduchou modifikací využít i pro jiné množiny vstupních dat, zde je vše připraveno a optimalizováno (z hlediska nároků na operační paměť RAM) pro efektivní načtení a zpracování měření z bezodrazné komory VUTS.

Sekce VISUALISATION obsahuje vybrané výsledky akustických veličin i s vlastními grafy. Zde lze jednoduchou obměnou parametrů u jednotlivých sekcí zobrazovat různé veličiny z různých měření, celkových možných kombinací je příliš vysoké množství na to, aby bylo vhodné je všechny pojmout do jednoho dokumentu. A to zejména s přihlédnutím k faktu, že základní principy a průběhy se často opakují.

Autor: Michal Kašpárek

```
% 28 souboru dat z 28 mereni, z toho pouzito 26 + kalibrace:

% Kalibracni soubor 40-1100 Hz
%  01 -> 50-600 Hz 200p
%  02-20, 22-28 -> 130-800 Hz 200p
%  21 kratsi - JEN 60 BODU - VYNECHAT
%  0.81 sec -> 32400 points


clc;
clear all;
close all;

FsReq =  40000;
FsReal = 40000;
duration = 0.81;          % sec

timeOffset    = 0.3;      % sec
timeToAnalyze = 0.5;      % sec

f_min = 130;
f_max = 800;
n_measures = 200;
freqVector = round(logspace(log10(f_min),log10(f_max),n_measures));

% Tloušťka měřeného vzorku [m]
    d = 0.001;
% VZDALENOSTI MIKROFONU MIC1-MIC4 od cela vzorku: x1, x2, x3, x4
    x1 = -0.6875;
    x2 = -0.4575;
    x3 = 0.460;
    x4 = 0.690;
```

Load init - file names

Vzhledem k velkému množství dat jsou soubory načítány a zpracovávány postupně v sekci Calculations

```
    names = { '02RawSweep_VUTS_JenPolykarbonat_201709204MIC_130-800Hz_200points_A-0.2.mat';...
    '03RawSweep_VUTS_MFC-openCircuitNaDlouho_201709204MIC_130-800Hz_200points_A-0.2.mat';...
    '04RawSweep_VUTS_MFC-shortCircuitNaDLouho_201709204MIC_130-800Hz_200points_A-0.2.mat';...
    '05RawSweep_VUTS_MFC-shortCircuitNaKratko_201709204MIC_130-800Hz_200points_A-0.2.mat';...
    '06RawSweep_VUTS_MFC-openCircuitNaKratko_201709204MIC_130-800Hz_200points_A-0.2.mat';...
    '07RawSweep_VUTS_MFC-MFCZvlastRozpojene_201709204MIC_130-800Hz_200points_A-0.2.mat';...
    '08RawSweep_VUTS_MFC-MFCZvlastZkratovane_201709204MIC_130-800Hz_200points_A-0.2.mat';...
    '09RawSweep_VUTS_MFC-MFCZvlast300Ohm_201709204MIC_130-800Hz_200points_A-0.2.mat';...
    '10RawSweep_VUTS_MFC-antiparalelne300ohm_201709204MIC_130-800Hz_200points_A-0.2.mat';...
    '11RawSweep_VUTS_NIC1_201709204MIC_130-800Hz_200points_A-0.2.mat';...
    '12RawSweep_VUTS_NIC2_201709204MIC_130-800Hz_200points_A-0.1.mat';...
    '13RawSweep_VUTS_NIC3_201709204MIC_130-800Hz_200points_A-0.05.mat';...
    '14RawSweep_VUTS_NIC4_201709204MIC_130-800Hz_200points_A-0.05.mat';...
    '15RawSweep_VUTS_NIC5_201709204MIC_130-800Hz_200points_A-0.05.mat';...
    '16RawSweep_VUTS_NIC6_201709204MIC_130-800Hz_200points_A-0.05.mat';...
    '17RawSweep_VUTS_NIC7_201709204MIC_130-800Hz_200points_A-0.05.mat';...
    '18RawSweep_VUTS_NIC8_201709204MIC_130-800Hz_200points_A-0.05.mat';...
    '19RawSweep_VUTS_NIC9_201709204MIC_130-800Hz_200points_A-0.05.mat';...
    '20RawSweep_VUTS_NICH1_201709204MIC_130-800Hz_200points_A-0.05.mat';...
    '22RawSweep_VUTS_NICH3_201709204MIC_130-800Hz_200points_A-0.05.mat';...
    '23RawSweep_VUTS_NICH4_201709204MIC_130-800Hz_200points_A-0.05.mat';...
    '24RawSweep_VUTS_NICH5_201709204MIC_130-800Hz_200points_A-0.05.mat';...
    '25RawSweep_VUTS_NICH6_201709204MIC_130-800Hz_200points_A-0.05.mat';...
    '26RawSweep_VUTS_NICH7_201709204MIC_130-800Hz_200points_A-0.05.mat';...
    '27RawSweep_VUTS_NICH8_201709204MIC_130-800Hz_200points_A-0.05.mat';...
    '28RawSweep_VUTS_NICH9_201709204MIC_130-800Hz_200points_A-0.05.mat';...
    };
```

Loading calibration data

```
fileName = 'CalibrationTrubice4_VUTS.mat';
load(fileName);

% Data loaded:
    % normalized4MICFreq;
    % normalizedChar4MICAmp;
    % normalizedChar4MICPhase;
```

Calculations

```
% freqChar(noMeasR-26, F-200, ch-4, A/Ph-2)
freqChar = zeros(length(names),length(freqVector), 4, 2);

TRall = zeros(length(names),length(freqVector), 2); % (measR, F, F/TR)
RRall = zeros(length(names),length(freqVector), 2); % (measR, F, F/RR)


% 130-800 Hz 200p
```

```
for noMeasR = 1:length(names),          % Jednotliva mereni
    load(char(names(noMeasR,:)));       % RawSweep4MIC

    for countF=1:length(freqVector),    % Frekvence v jdntl mereni
        F = freqVector(countF);
        NP = floor(F * timeToAnalyze);  % pocet period k analyze

        cropLength = floor(NP*FsReal/F);
        sampleRange = (floor(timeOffset*FsReal):floor(timeOffset*FsReal+cropLength-1));
        dataCrop = reshape(RawSweep4MIC(countF, :, sampleRange), [4 cropLength]);
        dataCrop = dataCrop';

        phasorMconst = zeros(2,4);   % phasorM(:,1) = Amp Mic1, PhaseMic1 (rad)

% Find calibration coefficient
        normalizedCharIndF = find(normalized4MICFreq == F);
        if isempty(normalizedCharIndF),
            disp('ERROR: frequency not found in calibration file');
            break;
        end

% Calculating phasor
        timeStamps = (linspace(0, timeToAnalyze, length(dataCrop(:,1))))';
        W = [cos(2*pi*F*timeStamps(1:(length(dataCrop(:,1)))))
sin(2*pi*F*timeStamps(1:(length(dataCrop(:,1)))))]; % tmp vector
        C = (W\dataCrop(:,:));
        amp=sqrt(C(1,:).^2 + C(2,:).^2);
        phi=atan2(C(1,:),C(2,:));

        for im = 1:4,
            amp(im) = amp(im) * normalizedChar4MICAmp(normalizedCharIndF, im);
            phi(im) = phi(im) - normalizedChar4MICPhase(normalizedCharIndF, im);

            phasorMconst(:, im) = [amp(im), phi(im)];
            phasorMcomplex(im) = amp(im) * exp(1i*phi(im));
        end;

% Calculating TR, RR
    soundSpeed = 343.2*sqrt((291)/293); % m/s at cca 18°
    k = 2*pi*F/soundSpeed;              % rad/m, wave number -> complex??

        A =  1i*(phasorMcomplex(1)*exp(1i*k*x2) - phasorMcomplex(2)*exp(1i*k*x1)) / ...
               (2*sin(k*(x1 - x2)));
        B =  1i*(phasorMcomplex(2)*exp(-1i*k*x1) - phasorMcomplex(1)*exp(-1i*k*x2)) / ...
               (2*sin(k*(x1 - x2)));
        C =  1i*(phasorMcomplex(3)*exp(1i*k*x4) - phasorMcomplex(4)*exp(1i*k*x3)) / ...
               (2*sin(k*(x3 - x4)));
        D =  1i*(phasorMcomplex(4)*exp(-1i*k*x3) - phasorMcomplex(3)*exp(-1i*k*x4)) / ...
               (2*sin(k*(x3 - x4)));

        t = 22;                 % air temperature in °C
        T = (t + 273.15);       % air temperature in K
        Z = 300;                % Altitude/Elevation above sea level (m) - Lbc 374 m.n.m.
        pAir = 101325 * (1.0 - Z*0.0000225577)^5.2559;  % Pressure dependent on altitude
        RAir = 287.058;                                 % specific gas constant for dry air
287.058 J/(kg·K)
```

```
        rhoA = pAir/((RAir * T));                       % Air Density [kg/m^3]

        Px0 = A + B;
        Vx0 = (A - B) / (rhoA * soundSpeed);


        Pxd = C*exp(-1i*k*d) + D*exp(1i*k*d);
        Vxd = (C*exp(-1i*k*d) - D*exp(1i*k*d)) / (rhoA * soundSpeed);

    %    T = matrix 2x2 [T11 T12; T21 T22]
        T = (1/(Px0*Vxd + Pxd*Vx0)) .* [(Pxd*Vxd + Px0*Vx0)  (Px0^2-Pxd^2); (Vx0^2-Vxd^2)
(Pxd*Vxd + Px0*Vx0)];

        Ta = (2*exp(1i*k*d)) / (T(1,1) + (T(1,2)/(rhoA * soundSpeed)) + (rhoA * soundSpeed *
T(2,1) + T(2,2)));
        Ra = (T(1,1) + (T(1,2)/(rhoA * soundSpeed)) - (rhoA * soundSpeed * T(2,1)) - T(2,2)) /
...
             (T(1,1) + (T(1,2)/(rhoA * soundSpeed)) + (rhoA * soundSpeed * T(2,1)) + T(2,2));


        freqChar(noMeasR, countF, :, :) = phasorMconst';

        TRall(noMeasR, countF, 1) = F;
        TRall(noMeasR, countF, 2) = abs(Ta);
        RRall(noMeasR, countF, 1) = F;
        RRall(noMeasR, countF, 2) = abs(Ra);
    end;
    clear RawSweep4MIC;
end;
```

VISUALISATION

```
indexUvod1 = (1:5);
indexUvod2 = (6:9);
indexSoft1 = (10:12);
indexSoft2 = [12, 13, 14, 15, 18, 16, 17];
```

TR = Acoustic Transmisibility of selected measurements

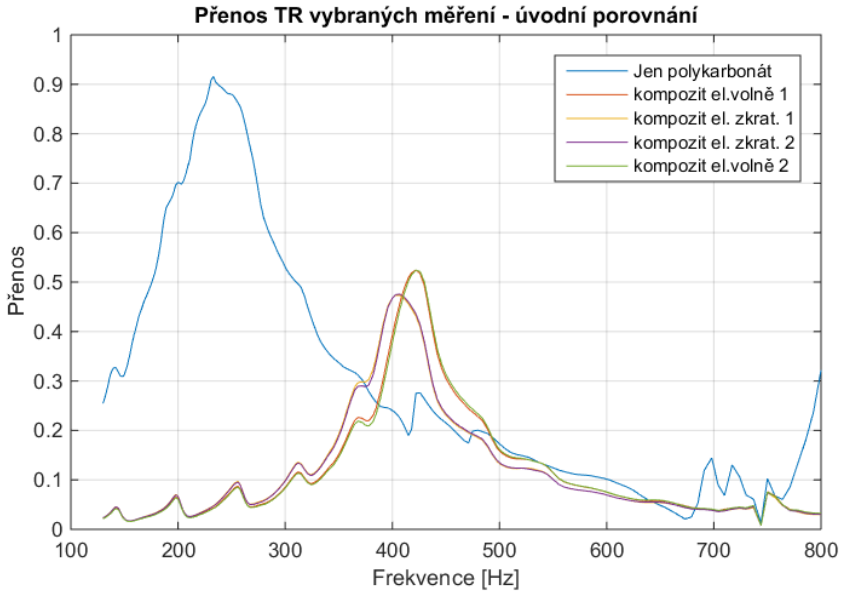
Struktura použitých polí:

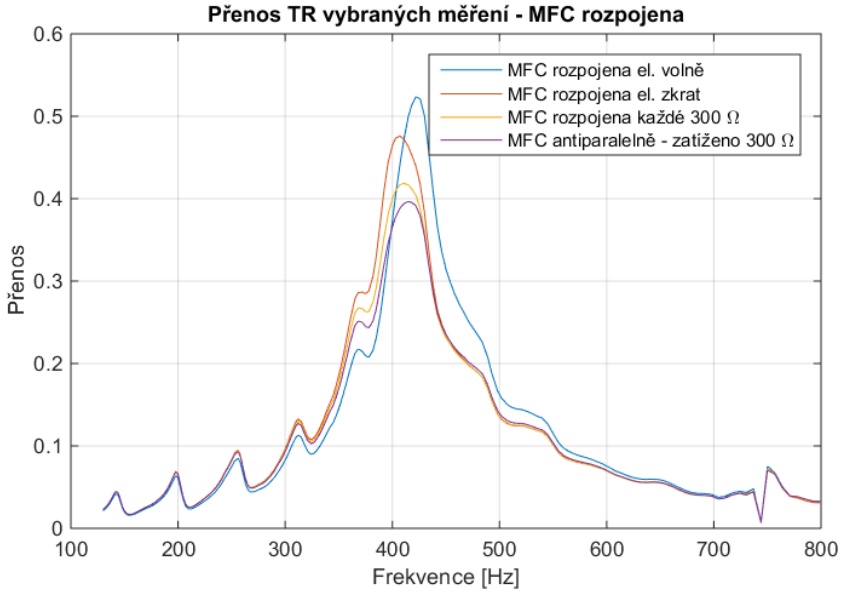
freqChar[noMeasR-7, F-120, ch-7, A/Ph-2]

TRall = zeros(length(names),length(freqVector), 2); % [measR, F, F/TR]

```
figure('position',[100 100 800 500], 'name', 'Přenos TR vybraných měření - úvodní porovnání');
% plot(TRall(1, :, 1), TRall(indexUvod1, :, 2), 'LineWidth', 1.5);
plot(TRall(1, :, 1), TRall(indexUvod1, :, 2));
legend('Jen polykarbonát','kompozit el.volně 1', 'kompozit el. zkrat. 1','kompozit el. zkrat.
2', 'kompozit el.volně 2');
title('Přenos TR vybraných měření - úvodní porovnání');
xlabel('Frekvence [Hz]'); ylabel('Přenos'); grid on;
```

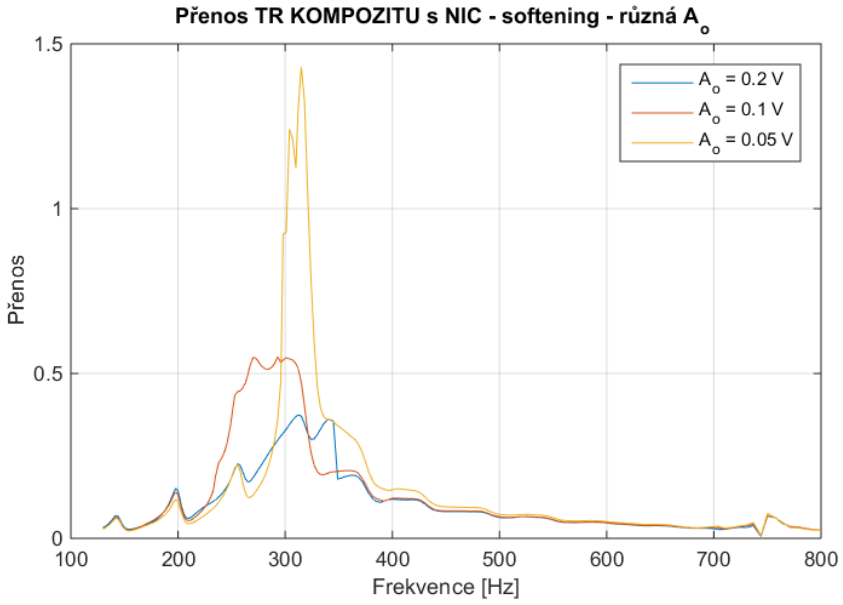
```
% ylim([0 1]); xlim([130 800]);


figure('position',[100 100 800 500], 'name', 'Přenos TR vybraných měření - MFC rozpojena');
plot(TRall(1, :, 1), TRall(indexUvod2, :, 2));
legend('MFC rozpojena el. volně','MFC rozpojena el. zkrat', 'MFC rozpojena každé 300
\Omega','MFC antiparalelně - zatíženo 300 \Omega');
title('Přenos TR vybraných měření - MFC rozpojena');
xlabel('Frekvence [Hz]'); ylabel('Přenos'); grid on;
% ylim([0 1]); xlim([130 800]);
```

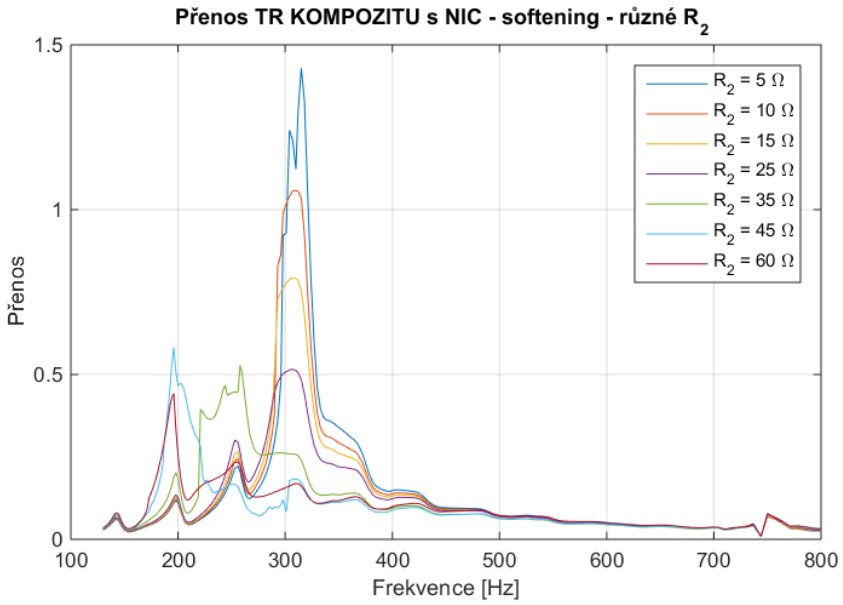




Composite sample - NIC Softening

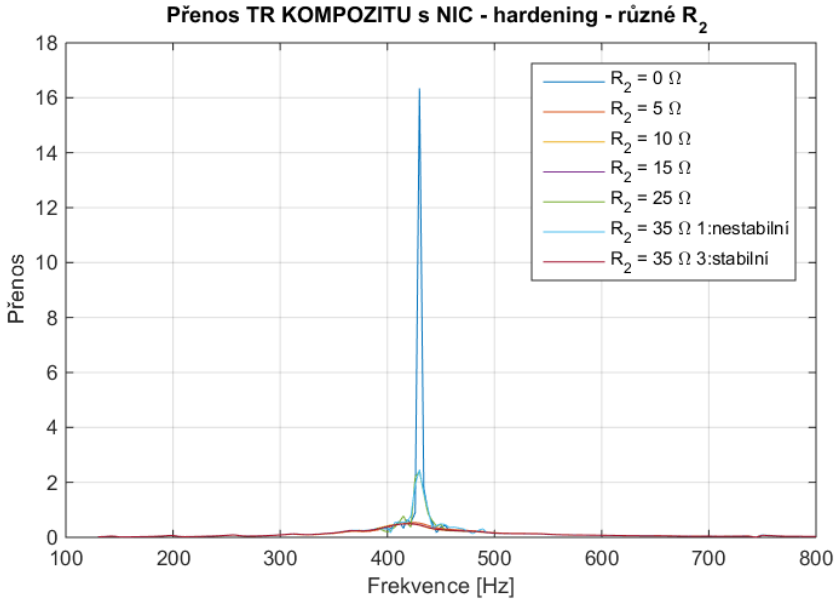
```
figure('position',[100 100 800 500], 'name', 'Přenos TR KOMPOZITU s NIC - softening - různá
A_o ');
plot(TRall(1, :, 1), TRall(indexSoft1, :, 2));
legend('A_o = 0.2 V', 'A_o = 0.1 V', 'A_o = 0.05 V');
title('Přenos TR KOMPOZITU s NIC - softening - různá A_o ');
xlabel('Frekvence [Hz]'); ylabel('Přenos'); grid on;
% ylim([0 1]); xlim([130 800]);

figure('position',[100 100 800 500], 'name', 'Přenos TR KOMPOZITU s NIC - softening - různé
R_2 ');
plot(TRall(1, :, 1), TRall(indexSoft2, :, 2));
legend('R_2 = 5 \Omega', 'R_2 = 10 \Omega', 'R_2 = 15 \Omega', 'R_2 = 25 \Omega', 'R_2 = 35
\Omega', 'R_2 = 45 \Omega', 'R_2 = 60 \Omega');
title('Přenos TR KOMPOZITU s NIC - softening - různé R_2 ');
xlabel('Frekvence [Hz]'); ylabel('Přenos'); grid on;
% ylim([0 1]); xlim([130 800]);
```

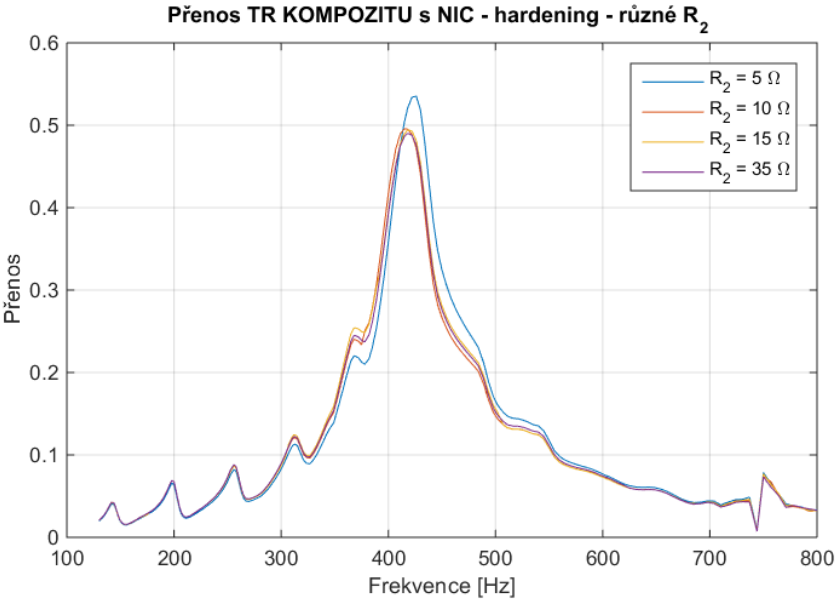




Composite sample - NIC Hardening

```
indexHard = [19, 20, 21, 22, 23, 24, 26];

figure('position',[100 100 800 500], 'name', 'Přenos TR KOMPOZITU s NIC - hardening - různé
R_2 ');
plot(TRall(1, :, 1), TRall(indexHard, :, 2));
legend('R_2 = 0 \Omega', 'R_2 = 5 \Omega', 'R_2 = 10 \Omega', 'R_2 = 15 \Omega', 'R_2 = 25
\Omega', 'R_2 = 35 \Omega 1:nestabilní', 'R_2 = 35 \Omega 3:stabilní');
title('Přenos TR KOMPOZITU s NIC - hardening - různé R_2 ');
xlabel('Frekvence [Hz]'); ylabel('Přenos'); grid on;
% ylim([0 1]); xlim([130 800]);


indexHard2 = [20, 21, 22, 26];

figure('position',[100 100 800 500], 'name', 'Přenos TR KOMPOZITU s NIC - hardening - různé
R_2 ');
plot(TRall(1, :, 1), TRall(indexHard2, :, 2));
legend('R_2 = 5 \Omega', 'R_2 = 10 \Omega', 'R_2 = 15 \Omega', 'R_2 = 35 \Omega');
title('Přenos TR KOMPOZITU s NIC - hardening - různé R_2 ');
xlabel('Frekvence [Hz]'); ylabel('Přenos'); grid on;
% ylim([0 1]); xlim([130 800]);
```


Přenos TR KOMPOZITU s NIC - hardening - různé R2
Přenos
Frekvence [Hz]
R2 = 0 Ω
R2 = 5 Ω
R2 = 10 Ω
R2 = 15 Ω
R2 = 25 Ω
R2 = 35 Ω 1:nestabilní
R2 = 35 Ω 3:stabilní


Přenos TR KOMPOZITU s NIC - hardening - různé R2
Přenos
Frekvence [Hz]
R2 = 5 Ω
R2 = 10 Ω
R2 = 15 Ω
R2 = 35 Ω