1. Erzeuge Container, in dem alle Parameter gespeichert werden:

rc = rietveld.base.RVContainer

rc =

RVContainer with properties:

phaseCnt: 1

dataX: 0

dataY: 0

fitFunction: []

1. Setze Anzahl der Phasen

rc.setPhaseCnt(1)

1. Laden der Messdaten (die vorher schon aufbereitet werden müssen) und zuweisen an den Container

load(fullfile(rvpath, 'data', 'CurrentProject', 'LaB6\_14tth\_Daniel.mat'));

EDSpectrum = S.EDSpectrum;%([30:71]);%,33:61,65:93,97:125]);

dataY = zeros(1, numberOfSpecs);

for i = 1:numberOfSpecs

dataX = EDSpectrum{1,i}(:,1);

dataY(1:length(EDSpectrum{1,i}(:,1)),i) = EDSpectrum{1,i}(:,2);

end

% Set the X and Y data in the rietveld container.

rc.setDataX(dataX);

rc.setDataY(dataY);

1. Erzeugen der Rietveld Funktion

Create the Rietveld function. The fit function is assembled using multiple "section functions", where each one has a individual purpose. If for example a correction should not be executed, the "Interface" of the respective function should be loaded instead. This way the function has no influence on the calculations. If the respective function needs a subfunction to work properly, the respective sub function needs to be defined.

% FitFunction

% Define the subfunctions to calculate the spectrum.

% Calculation of the energy positions. Different function are available. 1) EnergyPos and 2) EnergyPosCalib. Function 2) allows to refine calibration parameters in case the dead time correction is not working properly. If changed, tmpTau also needs to be changed.

tmpEPos = rietveld.func.spec.EnergyPos();

% Define the function used to fit the background.

tmpBkg = rietveld.func.spec.bkg.Polynomial();

% Define the modul to consider Dummy Peaks.

tmpDummy = rietveld.func.spec.dummy.DummyPeaks();

% Define the model to analyze each peak separately (similar to the mathematica data evaluation)

tmpSinglePeakAnalysis = rietveld.func.spec.dummy.DummyInterface();

% Define the corrections that are being executed.

% Absorption correction. Air absorption and material absorption is considered.

tmpACAir = rietveld.func.spec.corr.AttenuationCoeffAir();

tmpACMat = rietveld.func.spec.corr.AttenuationCoeffMat();

% Wiggler spectrum correction.

tmpWiggler = rietveld.func.spec.corr.WigglerInterface();

% Define the modul that calculates the fluorescence lines.

tmpFluor = rietveld.func.spec.fluor.FluorescenceKL();

tmpFluor.setSubFunction('AttenuationCoeffAir', tmpACAir);

% Define the functions that are used/needed to calculate all peak related factors.

% Define how the intensity is calculated. One can chose between the Rietveld method, where the intensities are calculated using the atomic scattering factors or the Le Bail method, where the intensities are treated as an additional fit parameter.

tmpFHKL = rietveld.func.spec.diffpeaks.fhkl.LeBail();

tmpIntensity = rietveld.func.spec.diffpeaks.Intensity();

tmpIntensity.setSubFunction('AttenuationCoeffAir', tmpACAir);

tmpIntensity.setSubFunction('Wiggler', tmpWiggler);

tmpIntensity.setSubFunction('FHKL', tmpFHKL);

% Define the functions used to analyze the residual stresses.

% Define the functions used to calculate the information depth tau.

tmpTau = rietveld.func.spec.diffpeaks.Tau();

% Function used to describe the residual stress depth distribution.

tmpStrain = rietveld.func.spec.diffpeaks.strain.ModSigmatauFit();

tmpStrain.setSubFunction('Tau', tmpTau);

% Define the profile function used to describe the peak shape.

tmpPeaks = rietveld.func.spec.diffpeaks.TCHPV();

tmpPeaks.setSubFunction('AttenuationCoeffMat', tmpACMat);

tmpPeaks.setSubFunction('EnergyPos', tmpEPos);

tmpPeaks.setSubFunction('Intensity', tmpIntensity);

tmpPeaks.setSubFunction('Strain', tmpStrain);

% tmpPeaks.setSubFunction('StressMod', tmpStressMod);

% tmpPeaks.setSubFunction('Tau', tmpTau);

% Define the modul that calculates the fluorescence lines.

tmpFluor = rietveld.func.spec.fluor.FluorescenceKL();

tmpFluor.setSubFunction('AttenuationCoeffAir', tmpACAir);

% Define the functions that are used/needed to calculate all peak related factors.

% Define how the intensity is calculated. One can chose between the Rietveld method, where the intensities are calculated using the atomic scattering factors or the Le Bail method, where the intensities are treated as an additional fit parameter.

tmpFHKL = rietveld.func.spec.diffpeaks.fhkl.LeBail();

tmpIntensity = rietveld.func.spec.diffpeaks.Intensity();

tmpIntensity.setSubFunction('AttenuationCoeffAir', tmpACAir);

tmpIntensity.setSubFunction('Wiggler', tmpWiggler);

tmpIntensity.setSubFunction('FHKL', tmpFHKL);

% Define the functions used to analyze the residual stresses.

% Define the functions used to calculate the information depth tau.

tmpTau = rietveld.func.spec.diffpeaks.Tau();

% Function used to describe the residual stress depth distribution.

tmpStrain = rietveld.func.spec.diffpeaks.strain.ModSigmatauFit();

tmpStrain.setSubFunction('Tau', tmpTau);

% Define the profile function used to describe the peak shape.

tmpPeaks = rietveld.func.spec.diffpeaks.TCHPV();

tmpPeaks.setSubFunction('AttenuationCoeffMat', tmpACMat);

tmpPeaks.setSubFunction('EnergyPos', tmpEPos);

tmpPeaks.setSubFunction('Intensity', tmpIntensity);

tmpPeaks.setSubFunction('Strain', tmpStrain);

% tmpPeaks.setSubFunction('StressMod', tmpStressMod);

% tmpPeaks.setSubFunction('Tau', tmpTau);

% Assembling of the "Spectrum function"

tmpSpec = rietveld.func.spec.Spectrum();

tmpSpec.setSubFunction('Background', tmpBkg);

tmpSpec.setSubFunction('DummyPeaks', tmpDummy);

tmpSpec.setSubFunction('SinglePeakAnalysis', tmpSinglePeakAnalysis);

tmpSpec.setSubFunction('Fluorescence', tmpFluor);

tmpSpec.setSubFunction('DiffPeaks', tmpPeaks);

% tmpSpec.setSubFunction('Wiggler', tmpWiggler);

% tmpSpec.setSubFunction('EnergyCalibCorr', tmpEnergyCalibCorr);

% tmpSpec.setSubFunction('Escape', tmpEscape);

% Define the function used to correct for dead time induced diffraction line shifts.

tmpChannelToEnergy = rietveld.func.spec.corr.ChannelToEnergy;

% tmpChannelToEnergy = rietveld.func.spec.corr.ChannelToEnergy('30.Januar 2017');

% tmpWigglerSpectrum = rietveld.func.spec.corr.Wiggler();

% Define the Rietveld fit function.

tmpFitFunc = rietveld.func.RVFitFunc();

tmpFitFunc.setSubFunction('Spectrum', tmpSpec);

tmpFitFunc.setSubFunction('ChannelToEnergy', tmpChannelToEnergy);

% tmpFitFunc.setSubFunction('Wiggler', tmpWigglerSpectrum);

% Define the fit function in the rietveld container.

rc.setFitFunction(tmpFitFunc);

* In tmpFitFunc wird festgelegt, welche Parameter im rc-Container definiert sind, also welche Parameter dementsprechend auch vorgegeben werden müssen (über die einzelnen SubFunctions)

% Define the "fitter" that executes the fit.

fitter = fitting.DefaultFitter;

% Define whether the fit errors should be calculated or not.

fitter.setFitOptions('ComputeFitErrors', false);