Exercises for imc FAMOS II – Digital Course

- Block 2 -

Doc. Rev.: 1.2- 28.08.2025

Exercise A

Exercise objective:

In exercise B from block 1, you "manually" created a time channel **Time_Of_Interest** by graphically reading time values from a curve window. Now create **Time_Of_Interest** automatically based on the plateau positions from the channel **displacement1**.

Here you will learn new basic functions such as derivative or smoothing, how to determine the measurement duration of a channel and how to isolate interesting areas from it.

Task formulation:

To perform the exercise, proceed as follows:

- Calculate the derivative of **displacement1** and smooth the result with a time window of 20 seconds
- Use **SearchLevel()** to determine a channel with temporal start points and a channel with temporal end points of the plateaus.
- Compare the start and end point channels in the data editor and correct their start and/or end values with the sequence if necessary.
- Create **Time_Of_Interest** by mathematical operations of the start and end point channels.

If you have done exercise B or C from block 1, replace your old definition for **Time_Of_Interest** with your new algorithm there.

Result:

The calculated elements of the **Time_Of_Interest** channel are as follows:

```
Time_Of_Interest[1] = 142,452
Time_Of_Interest[2] = 408,489
Time_Of_Interest[3] = 577,293
Time_Of_Interest[4] = 703,7
Time_Of_Interest[5] = 828,76
Time_Of_Interest[6] = 988,578
Time_Of_Interest[7] = 1437,16
```

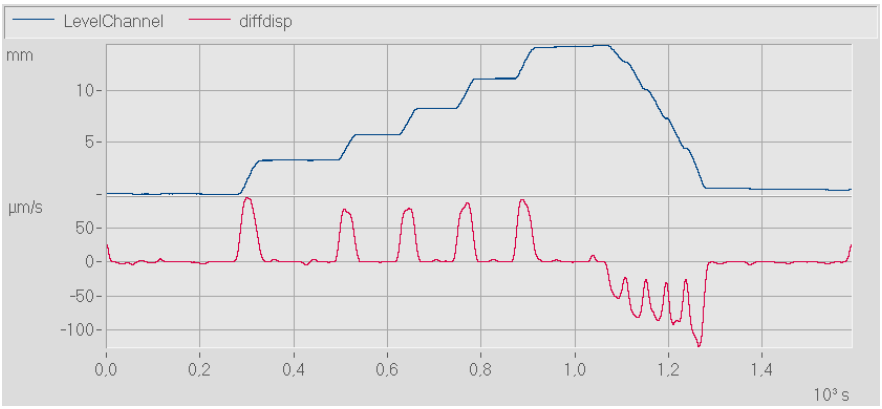
Exercise steps:

- Load the sequence from block 1 exercise C. If necessary, copy it from the exercise description if you have not saved it and use it to create a new sequence.
- Load the **LoadTestSample.dat** data set and group again **load1..4** in a **Load** group, **displacement1..3** in a **Displacement** group and **Force1..2** in a **Force** group.
- First, determine the derivative of **Displacement:displacement1** as an auxiliary data set **DiffDisp**. Since the derivative of a plateau is zero, this auxiliary dataset contains values around 0 as long as the plateaus persist and values significantly different from 0 for the ramp runs.

```
LevelChannel = Displacement:displacement1
diffdisp = Diff(LevelChannel)
```


- Show the auxiliary data set in a curve window together with the original data set. Since the derivative is very noisy, smooth it with a time window of 20 seconds:

```
diffdisp = Smo(diffdisp, 20)
```



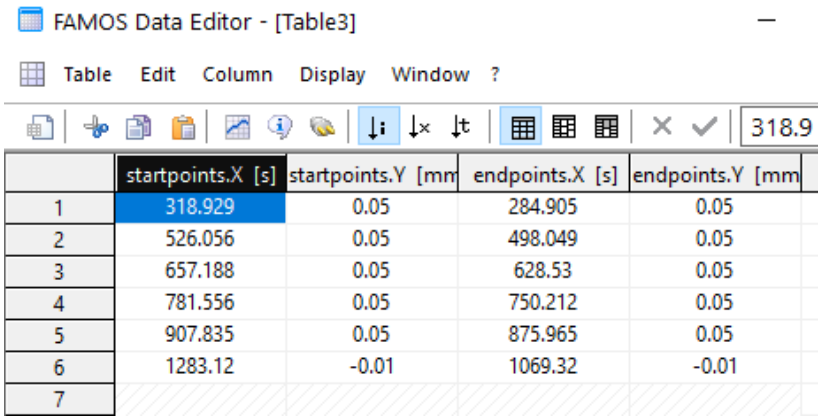
- During the ramp runs, all derivatives are higher than 50 µm/s or lower than -10 µm/s. A plateau always lies between two ramp runs, so use this fact to determine the start and end points of the plateaus. You obtain the start points by determining all intersection points of the descending derivatives with 50 µm/s as well as the ascending derivatives with -10 µm/s:

```
startpoints = SearchLevel(diffdisp, 3, -0.01, 0.05, 0, 0, 0, 1)
```

- Similarly, obtain the end points of the plateaus by determining the intersection points of all ascending derivatives at 50 µm/s and the descending derivatives at -10 µm/s:

```
endpoints = SearchLevel(diffdisp, 4, -0.01, 0.05, 0, 0, 0, 1)
```

- View the results of **startpoints** and **endpoints** in the data editor. The desired values are located in the respective X-tracks.



	startpoints.X [s]	startpoints.Y [mm	endpoints.X [s]	endpoints.Y [mm
1	318.929	0.05	284.905	0.05
2	526.056	0.05	498.049	0.05
3	657.188	0.05	628.53	0.05
4	781.556	0.05	750.212	0.05
5	907.835	0.05	875.965	0.05
6	1283.12	-0.01	1069.32	-0.01
7				

*Tip: If you do not see the X-components of the variables, then activate their view in the data editor via the menu **Display > Options >** Tab **Various > Both components.***

- For the start values, the value for the first ramp is still missing. Extract the X-track of the XY-data set for the start values into a new variable and add a zero at the beginning:

```
;StartpointsX = Join(0, startpoints.X); Standard until FAMOS 7.5
StartpointsX = [0, startpoints.X]; alternatively since FAMOS 2021
```


- For the endpoints, the value for the last plateau is still missing. This end point corresponds to the time of the last sample, in other words the duration of the data set. Extract the X-track of the XY data set for the end values into a new variable and append the total duration of the data set:

```
EndpointsX = [Endpoints.X, XOff?(LevelChannel) +
             XDel?(LevelChannel)*(Leng?(LevelChannel)-1)]
```

- The desired data set Time_Of_Interest results from the midpoints between the start and end points.

```
Time_Of_Interest = (endpointsX - startpointsX)/2 + startpointsX
```

  Replace the old definition of the time channel and the following line for the definition of the unit from exercise B/C block 2 with the new definition.

Exercise B

Exercise objective:

A formula is to be fitted to existing data by means of polynomial regression. In this example, a resistance curve is to be fitted as a function of the temperature above 0°C using a 2nd degree polynomial. A comparison with the literature values is to verify that the measured resistance is a Pt100.

Result:

By second order polynomial regression with the function **Poly()** for the range from 0 to 800 °C the following coefficients are obtained:

```
R0 = 99.43794037919
a = 0.3916019174779
b = -5.82847254123e-05
```

According to the literature, the 1st coefficient should be 100 Ω and the following two should be 3.9083-10^{-1} and - 5.775-10^{-5}, which is sufficiently satisfied. It is therefore obvious that the measured values correspond to a 100 Ω measuring resistor made of platinum, also known as Pt100.

Exercise steps:

- Load the file pt100_Meas.dat from the sample data and cut out a range above 0°C.

```
part = Cut(pt100_Meas, 0,800)
```

- Apply a second-order polynomial regression to the cut-out section and extract the coefficients to seperate variables.

```
coeff = Poly(part,2,2)
R0 = coeff[1]
a = coeff[2]/R0
b = coeff[3]/R0
```

- Repeat the steps and observe the behavior of the cut out part using a limit above 852°C.
  Note: The cut part is longer than the actual data set, which causes the difference to be filled with zeros and distorts the calculation of the polynomial. Note that this can cause trouble when using sequences.

Exercise C

Exercise objective:

Instead of a polynomial regression, in this exercise the fitting of any given function to measured data is to be performed. In this example, the parameters of a damped oscillation are to be determined.

Task formulation:

The formula for a damped oscillation includes as variable parameters the amplitude, the frequency and the damping constant τ.

$$u(t) = A \sin(2 \pi f t)\, e^{-\frac{t}{\tau}}$$

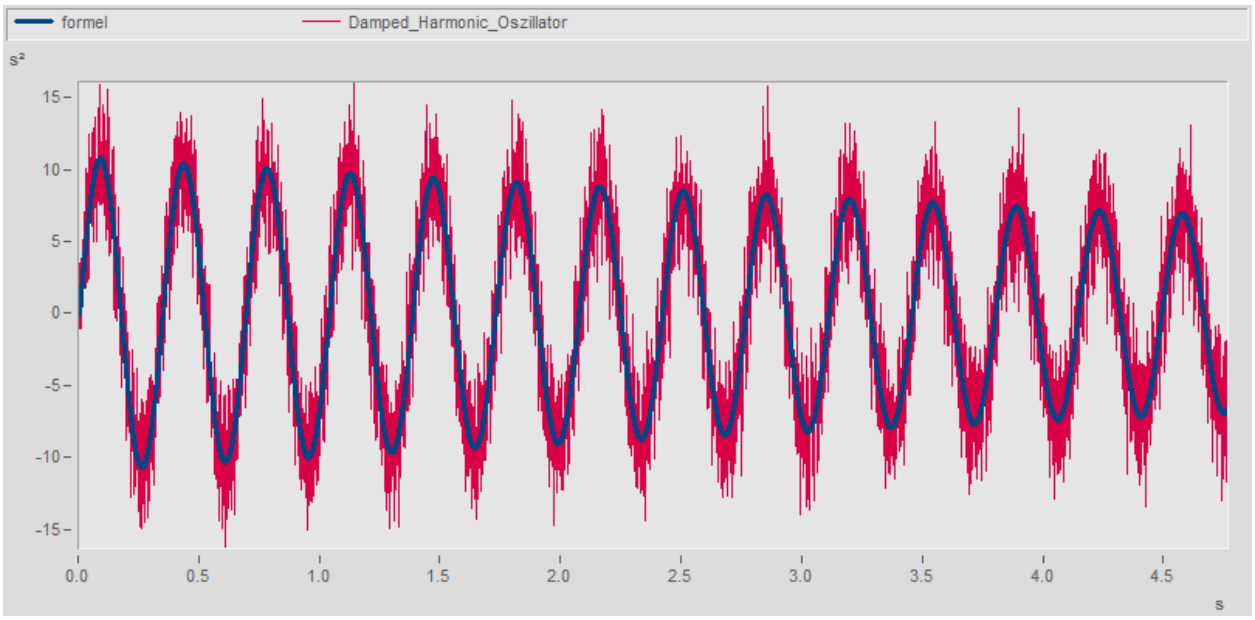
The dataset **Damped_Harmonic_Oszillator.dat** is the result of the measurement on such a damped oscillation.

- Use the function **ApproNonLin()** to determine a set of values for A, f, and τ that describes the measured values as accurately as possible.
- Using the determined parameters as well as the formula for the damped oscillation, calculate a data set with the same length of the measured data set and compare the two data sets in a curve window.

Result:

The results of the calculation in the curve window (right, display: last value as a number) and the calculated formula against the measurement for the first few seconds (below).

Amplitude	=	10.9356
Frequency	=	2.8900
tau	=	10.3516



Exercise steps:

- For the application of the function **ApproNonLin()** the sought coefficients for an expression are adjusted in such a way that it approaches zero as closely as possible. Therefore formulas of the type **y = f(x)** must be entered as parameters of the function **ApproNonLin()** in the notation **f(x)-y**. For the formula of the damped oscillation this means:

```
"A1*sin(pi2*A2*x)*exp(-1*x/A3)-y"
```

- Use this expression to generate the fit to the original dataset.:

```
Result = ApproNonLin(Damped_Harmonic_Oszillator ,
         "A1*sin(pi2*A2*x)*exp(-1*x/A3)-y", 1200, 1, "", empty, 1)
```

Note: The function tries to adjust the quadratic expression for XY value pairs from the original data set so that it approaches zero as closely as possible.

- Extract amplitude, frequency and tau each into separate variables:

```
Amplitude = Result[1]
Frequency = Result[2]
tau = Result[3]
```

- Using the coefficients as well as the formula given above for the damped harmonic oscillation, generate a data set of the length of the original data set in order to be able to compare them with each other:

```
xvalues = Ramp(XOff?(Damped_Harmonic_Oszillator),
          XDel?(Damped_Harmonic_Oszillator),
          Leng?(Damped_Harmonic_Oszillator ))
formula = Amplitude*Sin(pi2*xvalues*Frequency)*Exp(-1*xvalues/tau)
```

Then display the optimized and the original data set in a curve window.

Additional task:

- Determine the maximum frequency of the spectrum for the original data set:

```
; calculation via Spec()
spectrum = Spec(Red2(Damped_Harmonic_Oszillator ))
delta_F = XDel?(spectrum)
frequency_by_spectrum = Posi(spectrum.B, Max(spectrum.B))
```

- Compare the calculated maximum with the calculated frequency from the nonlinear optimization.
  Note: You will notice that the fitting by means of the function **ApproNonLin()** provides a more suitable value than the maximum frequency of the spectrum, because even with the high number of points the frequency resolution of the Fourier analysis is only 0.25 Hz. Thus it can be quite meaningful to determine a frequency by means of nonlinear optimization.

