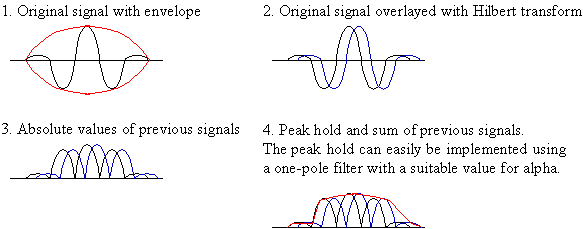
Development Notes: Envelope and the Damping-Constant method for the Acoustical-Resonance-Analysis

***The Envelope:***

The need for an envelope calculation method arises from the calculation of the damping constant where the envelope is an integral part of the calculation, so we decide to implement an envelope function corresponding to the standard methods usually taken in the digital signal processing. The calculation routine is based on the so called Analytic-Function which insists of a complex function where the core signal is the real part and the Hilbert-Transformation of the signal is the imaginary part. The Hilbert-Transformation of a signal has no negative imaginary parts and can be seen as a 90 degree phase shift of the original signal. The resulting analytic/causal signal can be used to build the envelope as shown in the picture above.



The calculation of the Hilbert Transformation can either be done in the frequency domain or in the time domain. The frequency domain method tries to phase shift the signals complex frequencies while the time domain approach is to extract suitable FIR filter parameters for the Hilbert-Transformation and put the signal through a FIR filter. The FIR filter coefficients can be easily generated corresponding equations found in the literature. Because we haveour very well outperformed FFT implementation with the Radix and Winograd optimization, we decide to use the calculation into the frequency domain by solving the complex equation **H(X(jw))=-j\*sign(w)\*X(jw)** and transform the result back into the time domain. The Hilbert-Transformation can be found into the real part of the back transformation.

We implemented two methods for the envelope extraction where both calculate the analytic function first. The absolute value of the complex analytic function will be taken for the first function called **ENVELOPE\_CALCULATION\_METHOD\_ABSOLUTE\_VALUE** and a second method applies an FIR-filter with variable filter constant after the calculation of the absolute value. This calculation method is called **ENVELOPE\_CALCULATION\_METHOD\_FIR\_LOW\_PASS**.The methods is available in the RMathLib DLL via the COM Interface **RMLTimeDomain::HilbertEnvelope.**As follows the function signature from the DLL Specification. More details about the implementation can be found in the source code in the appendix.

**enum ML\_HILBERT\_ENVELOPE\_CALCULATION\_TYPE**

**{**

**ML\_HILBERT\_ENEVLOPE\_CALCULATION\_TYPE\_ABS,**

**ML\_HILBERT\_ENEVLOPE\_CALCULATION\_TYPE\_FIR**

**}**

**HilbertEnvelope**

**(**

**[in] VARIANT inputSignal,**

**[in] ML\_HILBERT\_ENVELOPE\_CALCULATION\_TYPE method,**

**[in] double firLowPassAlpha,**

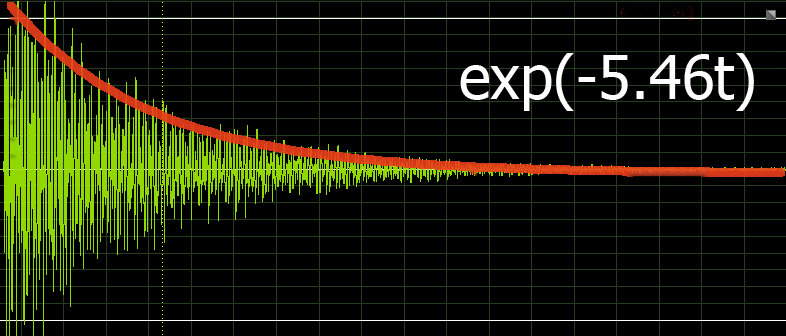
**[out, retval] VARIANT\* envelope**

**);**

The FIR low passfilter in this function is implemented as **y[i]:=y[i-1]+alpha\*(x[i]-y[i-1])** and has the need for a suitable filter coefficient. The default value for the filter alpha is set to 0.54 corresponding the Hamming value.An empirical evaluated good value for the filter coefficient is **τ/N**with **τ**in **[30…60]** and **N** as the signal length.

***The Damaping-Constant:***

The availability of an envelope function offersthe extraction of the damping-constant with the help of an exponential regression. After the envelope is extracted, we begin with the peak point and build a point-cloud from the peak to the end of the signal which will be used to lay over an corresponding exponential function **y(t)=A\*exp(-b\*t)**where A is the proposed amplitude and b is the so proposed called damping-constant. The extraction of the exponential regression parameters can be outperformed while fall back to the linear regression with a logarithmization of the result. The filter coefficients for the envelope calculation will be trimmed to an empirical evaluated value, and will be set a priori inside the damping-constant function depending on the length of the signal.In this way an autonomous operation can be accomplished. The results of the damping-constantextraction are in real physical SI-Units (**[b]=1/s**) and independent from the sample rate. The Method is fully implemented and tested concerning memory leaks and exceptional input values. Following an example of an one second silica crucibleimpulse response signal, where the method found an damping constant from 5.46. As in the picture can be seen the proposed envelope matches very well the envelope of the impulseresponse signal.



The function is located at **RMLTimeDomain::DampingConstant** in the RMathLib DLL. As follows the COM function signature.

**HRESULT DampingConstant**

**(**

**[in] VARIANT inputSignal,**

**[in] long sampleRate,**

**[out, retval] double\* dampingConstant**

**);**

Because the function works fully autonomous while searching the peak and find a suitable filter constant,no further information apart from the signal and the sample rate is needed to find the damping-constant. The exponential regression results aside from the damping-constant the amplitude of the exponential function and the position of the peak, which will not pass to the user and only calculated for internal use.

***Special explanatory notes for the usage of the damping-constant extraction algorithmus:***

We propose the usage of the damping-constant algorithm only with a **dropped band-pass** filter. An example as shown in the picture should illustrate this: An as OK declared component has a given impulse response fingerprint with a corresponding long decay signal. As we can see in the picture above the signal second order dominant own frequency is the only order which has a significant difference between the OK and NOK component. But in the NOK signal, the fourth order component has a significant level and influences the damping-constant more than the information carrying second order own frequency. An adequate dimensioned band pass filter should avoid this problem.



Illustration 1: Sample OK spectrum Illustration 2: Sample NOK spectrum

***Appendix:***

**Header Function Description Envelope-Method:**

**/\*\***

**\* (c) RTE Akustik + Prüftechnik GmbH**

**\***

**\* @file envelope.h**

**\* @class Envelope**

**\* @version 1.0**

**\* @date 2008**

**\* @author Christoph Lauer**

**\* @brief This function implements a basic envelope calculation which uses the hilbert transformation.**

**\* @see http://www.numerix-dsp.com/envelope.html**

**\* @todo finished and tested so far.**

**\*/**

**//------------------------------------------------------------------------------------------------**

**#ifndef RTE\_MATH\_ENVELOPE**

**#define RTE\_MATH\_ENVELOPE**

**//------------------------------------------------------------------------------------------------**

**namespace rte**

**{**

**namespace math**

**{**

**//------------------------------------------------------------------------------------------------**

**/\*\***

**\* This class implements an envelope calculation with the hilbert transformation. More details about the**

**\* implementation can be found into the functions. The Hilbert transform works best with AC coupled,**

**\* band-limited signals. To perform an inverste Hilbert Transformation can be computed with the aid of**

**\* the Hilbert Transformation and post negation. Beacause our implemenation is based on the FFT we can**

**\* say thast the performance of the Hilbert Transformation is blindingly fast. THe speed should be a little**

**\* bit more than the 2\*FFT time --> approximately 2\*N\*log(N)+rest. Because we implemented this algorithms by**

**\* hand this collection of the hilbert and the envelope calculation benifits heavyly from the copmpiler**

**\* optimizations so we advise for perfomance purposes to compile this code with the compiler optimization**

**\* "-O3", because we are not sure to make the best perfomant implementation here.**

**\*/**

**class Envelope**

**{**

**public:**

**//-------------------------------------------------------------------------------------------------**

**/// this enummeration specifies both calculation methods for spectrum envelope**

**enum ENVELOPE\_CALCULATION\_METHOD**

**{**

**/// For this method the signal will be decomutated and the max taken from the hilbert and signal**

**ENVELOPE\_CALCULATION\_METHOD\_ABSOLUTE\_VALUE,**

**/// For this method the absolute value from both will be shifted trough a FIR filter**

**ENVELOPE\_CALCULATION\_METHOD\_FIR\_LOW\_PASS**

**};**

**//-------------------------------------------------------------------------------------------------**

**/\*\***

**\* This function calculates the envelope of a given time domain input signal with two**

**\* different methos derived from the hilbert transfromation. Both methods build first the**

**\* DC absolute value of the so called analytiv function from the given input signal. The calcualtion**

**\* of the analytic function has a need for the hilbert transformation phase shifter which is**

**\* also implemented into this file. After the analytic function is calcualted two methods**

**\* are implemented for the envelope calculation:**

**\* 1.) The second method calcualtes simple the absolute value of the analytic signal sqrt(h^1+x^2)**

**\* 2.) The first method build the max of them and allpies an simple FIR low pass.**

**\* The calculation method can be switched with the "method" flag.**

**\***

**\* @param inputSignal This is a pointer to the input signal.**

**\* @param length This value specifies the length of the input vector.**

**\* @param method This enummeration flag specifies the mwthod used for the calculation**

**\* of the nevelope.**

**\* @param firLowPassAlpha For the FIR filtered method this value specifies the filter constant**

**\* used to smooth the signal. A lower value results a higher low pass.**

**\* @return The return value of the calculation is the signal envelope. Note that**

**\* the result vector of this function will be allocated into this function**

**\* so no pre allocation is needed.**

**\*/**

**staticdouble\* calculateEnvelope(constdouble\* inputSignal, constint length, const ENVELOPE\_CALCULATION\_METHOD method, double firLowPassAlpha = 0.54);**

**//------------------------------------------------------------------------------------------------**

**private:**

**/\*\***

**\* THis method applies a standart FIR low pass Filter corresponding the formular**

**\* y(n)=( alpha\*x(n) + (1-alpha) \* y(n-1)), where is in the intervall 0..1 and has a typical**

**\* value of 0.97. The low pass filtered signal will be pre allocated into thsi fucntion**

**\* so no pre allocation of the output signal is needed. The legnth of the output signal**

**\* is the same as the length of the input signal.**

**\***

**\* @param input This value points to the input signal.**

**\* @param length Ths is simpel the length if the input signal.**

**\* @param alpha This value prepresents the FIR filter coefficcient.**

**\* @return The result of the calculation is the low pass filtered vector**

**\* of the input signal.**

**\*/**

**staticdouble\* applyFirLowPass(constdouble\* input, constint length, constdouble alpha = 0.54);**

**//------------------------------------------------------------------------------------------------**

**/\*\***

**\* This function calculates the hilbert transformation for the given input sample vector.**

**\* We decide to use the FFT hilbert transformation where the given input signal will be first**

**\* transformed in the complex frequency domain. In the frequency domain we shift the signal**

**\* in the frequency doamin by 90 degrees while applying the hilbert transformation formular:**

**\* H(X(jw)) = -j \* sign (w) \* X(jw). The corresponding result will be back transformed into**

**\* the time domain and given back as result. Note that we implement a FFT version of the Hilbert**

**\* transformation so we need an input vector which has a power of two.**

**\***

**\* @param timeDomain The time domain input Signal.**

**\* @param length The length of the input signal. The length should be have a length**

**\* from a power of two.**

**\* @return hilbertDomain The hilbert domain output signal. Note that the output vector**

**\* will be allocated into the fucntion and must not explicitely**

**\* preallocated. THe length of the vector is the same as the**

**\* length of the input signal.**

**\***

**\*/**

**staticdouble\* fastHilbertTransform(constdouble\* timeDomain, constint length);**

**//------------------------------------------------------------------------------------------------**

**}; // class Envelope**

**//------------------------------------------------------------------------------------------------**

**} // namespace math**

**} // namespace rte**

**//------------------------------------------------------------------------------------------------**

**#endif// RTE\_MATH\_ENVELOPE**

**Header Function Description Damping-Constant-Method:**

**/\*\***

**\* (c) RTE Akustik + Prüftechnik GmbH**

**\***

**\* @file damping\_constant.h**

**\* @class DampingConstant**

**\* @version 1.0**

**\* @date 2008**

**\* @author Christoph Lauer**

**\* @brief Try to propose the Damping Constant of an given input inpulse signal**

**\* @see http://de.wikipedia.org/wiki/Dämpfungskonstante**

**\* @see http://en.wikipedia.org/wiki/Regression\_analysis**

**\* @todo finished and tested so far**

**\*/**

**//------------------------------------------------------------------------------------------------**

**#ifndef RTE\_MATH\_DAMPING\_CONSTANT**

**#define RTE\_MATH\_DAMPING\_CONSTANT**

**//------------------------------------------------------------------------------------------------**

**namespace rte**

**{**

**namespace math**

**{**

**//------------------------------------------------------------------------------------------------**

**/\*\***

**\* This class collects functions to find a suitable exponential expression for an exponential function**

**\* which matches best the envelope of a given impulse resonse from a resonance analysis. The class**

**\* implements only two functions, the main function wehich steers everythig, and the exponential**

**\* regression function which try to find a suitable exponential function. This function makes heavily**

**\* use of the envelope function class-collection whish was defined outside of this function file.**

**\* The main function is implemented to find the damping constant autonomously and try to match the**

**\* best values. The higher the damping constant the faster signal convergates to the zero line =>**

**\* Higher Damping. The fucntion try to match the best physical units and the result shound return a**

**\* value with unit 1/s which can be guarated by the sampleing rate. Another core function of this**

**\* class-collection is the exponential regression whis here implemented with a fallback to a linear**

**\* regression where the solution can found in every statistical book. THe exponential regression**

**\* returns the amplitude and damping factor of the function f(t)=A\*exp(-bt), where b is thedamping**

**\* factor and the amplitude A can be forgotten.**

**\*/**

**class DampingConstant**

**{**

**//------------------------------------------------------------------------------------------------**

**public:**

**//------------------------------------------------------------------------------------------------**

**/\*\***

**\* This function does the following steps to extract the damping constatnt of the given signal.**

**\* 1.) It trys to find the envelope of the given signal with the envelope function.**

**\* 2.) Second step is to find the peak value and some significant values to build a suifficient**

**\* point cloud form the following exponential regression.**

**\* 3.) In the third step this function try to find via an exponential regression the exponetial**

**\* curve of the given data cloud, which is given back by this function.**

**\* This function steers all the other functions like the envelope function of the exponential**

**\* regression function and try to do all the things autonomously to find the correct values for the**

**\* peak and all the other values for the damping constant extraction. The sample rate is needed to**

**\* make the result independent from any A/D wanler values. The output has the unit 1/s which will**

**\* outperform from the sample rate. This function is the interface function for the outside of this**

**\* class.**

**\***

**\* @param signal A pointer to the input signal.**

**\* @param length The length of the given input signal.**

**\* @param sampleRate The sample rate of the given input signal.**

**\* @return This fucntion returns the found damping constant in 1/s.**

**\*/**

**staticdouble proposeDampingConstant(constdouble\* signal, constint length, constint sampleRate);**

**//------------------------------------------------------------------------------------------------**

**private:**

**//------------------------------------------------------------------------------------------------**

**/\*\***

**\* This function calculates the exponential regression for the given point-loud and try to find the**

**\* best exponential function which matches the point cloud. The resulting exponential function can**

**\* be described with the formular: f(t) = amplitude \* exp ( - dampingConstant \* t). The exponential**

**\* regression can be break down to a linear regression and that's the way it is implemented into this**

**\* function. We first transformate the values from the exponential space into the linear domain,**

**\* solve the simple linear regression values for the slope and the intersection and transformate**

**\* the result back into the exponential domain. Note that our impelmentation works only for positive**

**\* numbers. For the envelope calculation this case is not relevant, because the all the numbers from**

**\* the envelope are positive.**

**\***

**\* @param pointsX A pointer to the given point cloud X-values.**

**\* @param pointsY A pointer to the given point cloud Y-values. Note taht htis function has a**

**\* side effect which modifyes the y points vector.**

**\* @param length The number of points in the point cloud.**

**\* @param amplitude The resulting amplitude of the exponential function.**

**\* @param dampingConstant The resulting damping constant of the exponential function.**

**\*/**

**staticvoid exponentialRegression(constdouble\* pointsX, double\* pointsY, constint length, double& amplitude, double& dampingConstant);**

**//------------------------------------------------------------------------------------------------**

**}; // class DampingConstant**

**//------------------------------------------------------------------------------------------------**

**} // namespace math**

**} // namespace rte**

**//------------------------------------------------------------------------------------------------**

**#endif// RTE\_MATH\_DAMPING\_CONSTANT**