VECTOR

Acoustic Analysis Using FFT Function

2022-02-16
Support Note SN-IMC-1-097

Author(s) Sokratov, Stanislav
Restrictions Public Document

Table of Contents

1 Description

This script shows how a simple NVH (**N**oise **V**ibration **H**arshness) analysis can be done using FFT (**F**ast **F**ourier **T**ransform) function provided by the function dll. The main purpose of this example is to illustrate how real physical values for the amplitude and frequency of the FFT can be interpreted. For this, amplitude spectrum and overall level for a signal representing sound pressure will be considered in detail. Please refer to the **FFT-Analysis with CANape** -Support Note (see KnowledgeBase article KB0011731) on which this support note is based on.

2 Disclaimer

3 Requirements

This example is based on the script from the Knowledge Base article KB0011731 in which the `BLOCK_FFT_MODE_PEAK_AMPLITUDE` and `BLOCK_FFT_MODE_RMS_AMPLITUDE` flags for the `BlockFFT` function are relevant.

For the `Funktion_DoTheFFT_online` function, create global variables: `fftArray`, `FFT_normalized`, `FFT_dB` and `Overall`.

For the `Funktion_DoTheFFT_offline` function, create global variables: `fftArray_blockFFT`, `counter`.

4 Source code

4.1 Online use case

```
/*****************************************************************************
this function calculates the FFT over the previous BUFFER_SIZE number of
data points.

Within this function six parameters are defined (with R and I denote the
real and imaginary part of the FFT):

BUFFER_SIZE = defines the number of data points that are taken into account
by the function BlockFFT. If the number of datapoints previous to the
current analyzed data point is less than BUFFER_SIZE then the missing
```

```
parameter values for BlockFFT are omitted.

    BLOCK_FFT_FLAG_NORMALIZE = flag to normalize the result of BLOCK_FFT
    BLOCK_FFT_MODE_PEAK_AMPLITUDE = ( R^2 + I^2 )^0.5
    BLOCK_FFT_MODE_RMS_AMPLITUDE = ( ( R^2 +I^2 )^0.5 ) / ( 2 )^0.5
    BLOCK_FFT_MODE_AUTO_SPECTRUM = R^2 + I^2
    BLOCK_FFT_MODE_POWER_SPECTRUM = ( R^2 + I^2 ) / 2
*****************************************************************************
********/

function Funktion_DoTheFFT_online (var signal, var fftArray[])
{
/********************** DEFINE PARAMETERS *******************************/
  #define BUFFER_SIZE   2048
  #define BLOCK_FFT_FLAG_NORMALIZE               0x0001
// Flag, can/must be combined with a mode
  #define BLOCK_FFT_MODE_PEAK_AMPLITUDE          0x0002
  #define BLOCK_FFT_MODE_RMS_AMPLITUDE           0x0004
  #define BLOCK_FFT_MODE_AUTO_SPECTRUM           0x0008
  #define BLOCK_FFT_MODE_POWER_SPECTRUM          0x0010

/************************ LOCAL VARIABLES ***********************************/
  double buffer[BUFFER_SIZE];
  double n;
  double dim;
  double checkVal;
  double dt;
  double t_start;
  double t_end;
  long initial = 1;
  long idx;
  long raster;
  long i = -1;
  long isPowerOfTwo;
  double EvalMode;
  double overall_lin;
  double p_ref=0.00002;
  double corr_factor;

/************************ MAIN CODE ************************************/

  // ----------initialize variables----------------
  switch (initial)
  {
    case 1:  // validate fftArray settings
      initial = 2;

      // check if dimension of fftArray is less than 4096
      dim = xdimension(fftArray);
      if(dim >= 4096)
      {
        Write("The number of columns of fftArray is too large
(currently %d). The calculation cannot be computed and is stopped. Reduce
the number of columns to a value less than 4096. NOTE: The number of
columns has to be a power of two. ", dim);
        cancel;
      }

      // check if dimension of fftArray is a power of two
      isPowerOfTwo = 1;
      checkVal = dim;
      while((checkVal != 1) && (isPowerOfTwo == 1))
```

```
      {
        if(checkVal%2 == 0)
        {
          checkVal = checkVal / 2;
        }else{
          isPowerOfTwo = 0;
        }
      }
      if(isPowerOfTwo == 0)
      {
        Write("The number of columns of fftArray is not a power of two
(currently %d). The calculation cannot be computed and is stopped. Adjust
the number of columns.", dim);
        cancel;
      }

      // initialize global array fftArray, all elements equal to zero
      for(idx = 0; idx < dim; idx++)
      {
        fftArray[idx] = -1;
      }
      break;

    case 2:   // initialize time raster to calculate the FFT. NOTE:
increasing BUFFER_SIZE automatically increases the calculation time for the
FFT. Hence, the time raster to repeat the FFT calculation is increased.
Otherwise data loss may occur.
      initial = 0;
      dt = diffTime(signal) * 1000;
      raster = ceil(1 / dt);
      i = 1;
      break;
  }

  if (i == raster)
  {
    i = 1;
    t_start = GetClockHR();

    // copy the last BUFFER_SIZE number of data points to the array buffer
to be FFT analyzed.
    for (n = 0; n < BUFFER_SIZE; n++)
    {
      buffer[n] = signal[-n];
    }
    //set evaluation mode
    EvalMode= BLOCK_FFT_MODE_PEAK_AMPLITUDE;

    // define the factor to correct the FFT result depending on selected
mode: PEAK or RMS
    // The FFT has to be corrected by sqrt(2) in case
EvalMode=BLOCK_FFT_MODE_PEAK_AMPLITUDE because
    // the FFT Spectrum result (sometimes called the linear spectrum or rms
spectrum) is derived from the FFT auto-spectrum,
    // with the spectrum being scaled to represent the rms level at each
frequency as per definition => FFT(RMS)=FFT(PEAK)/sqrt(2)
    // Note that FFT transformation will divide the power between the
positive and negative sides,
    // so as only one side of the FFT result is taken into consideration,
the FFT result has to be multiplied by 2

    if (EvalMode==BLOCK_FFT_MODE_PEAK_AMPLITUDE)
```

```
    {
      corr_factor=2/sqrt(2);
    }
    else if(EvalMode==BLOCK_FFT_MODE_RMS_AMPLITUDE)
    {
      corr_factor=2;
    }


    //calculate FFT
    BlockFFT(buffer, BUFFER_SIZE, EvalMode, fftArray);

    // calculate overall and frequency level
    overall_lin=0;
    Overall=0;
    if (EvalMode==BLOCK_FFT_MODE_PEAK_AMPLITUDE ||
EvalMode==BLOCK_FFT_MODE_RMS_AMPLITUDE)
    {
    for(n = 0; n<dim;n++)
    {
    // calculate overall and freq. level
    FFT_normalized[n]=fftArray[n]*corr_factor/ dim; // normalize FFTs by
the number of sample points

    FFT_dB[n]=20*log(FFT_normalized[n]/p_ref); //calculate level in dB that
is relevant for sound pressure

    overall_lin=overall_lin+pow(FFT_normalized[n],2); //overall level (the
sum of amplitudes at each frequency) for each blocklength
    }
    }
    overall_lin=sqrt(overall_lin); // calculate overall level[lin]
    Overall=20*log(overall_lin/p_ref); // calculate overall level in dB
that is relevant for sound pressure

    Write("Lin overall is %2.3f", overall_lin); // overall (linear value)
for each blocklength
    Write("Overall level is %2.1f dB", Overall); // overall (dB value) for
each blocklength



    t_end = GetClockHR();


    // check if raster is sufficient to calculate the FFT. If not adjust
raster setting.
    if ( ((t_end - t_start) / dt) > raster )
    {
      raster = ceil( (t_end - t_start + 1) / dt);
    }

    return t_end - t_start;

  }

  i++;

  return;
}
/********************************** END **********************************
```

4.2 Offline use case

4.2.1 Ecexute_FFT script

```
double n;
double dim;

 ClearWriteWindow();

 CalculateSignal(Calculated.Funktion_DoTheFFT_offline); // Execute
Funktion_DoTheFFT_offline

 dim = xdimension(fftArray_blockFFT);
 for(n=0;n<dim;n++)
  {
  fftArray[n]=sqrt(fftArray[n]/counter); // RMS averaging: square root of
the arithmetic mean : The counter represents the overall number of FFT
blocks
  fftArray[n]=fftArray[n]*2/dim;
  }
```

4.2.2 Funktion_DoTheFFT_Offline function

```
function Funktion_DoTheFFT_offline (var signal, var fftArray_blockFFT[])
{
/********************** DEFINE PARAMETERS ******************************/
 #define BUFFER_SIZE 2048 //it's recommended to define the same buffer size
like the dimension of fftArray
 #define BLOCK_FFT_FLAG_NORMALIZE 0x0001 // Flag, can/must be combined with
a mode
 #define BLOCK_FFT_MODE_PEAK_AMPLITUDE 0x0002
 #define BLOCK_FFT_MODE_RMS_AMPLITUDE 0x0004
 #define BLOCK_FFT_MODE_AUTO_SPECTRUM 0x0008
 #define BLOCK_FFT_MODE_POWER_SPECTRUM 0x0010
/************************ LOCAL VARIABLES ******************************/
 double buffer[BUFFER_SIZE];
 double n;
 double size;
 double t_end;
 double dim;
 double checkVal;
 long initial = 1;
 long idx;
 long isPowerOfTwo;
 double overall_lin; //linear overall level
 long sample;
 long blocklength;

 //! After 'steps' input samples, the FFT is executed
 int inkr = 2048;
/************************ MAIN CODE ************************************/

 // ----------initialize variables---------------
 if (initial == 1)
  {
  initial = 0;
  counter=0; //The counter represents the overall number of FFT blocks
  sample=0;
  // time of last measured datapoint
  size = sizeof(signal) - 1;

  //t_end = time(signal[BUFFER_SIZE-1]);
```

```
  blocklength=BUFFER_SIZE;
  //Write("t %f2.2",t_end);

  // check if dimension of fftArry is less than 4096
  dim = xdimension(fftArray_blockFFT);

   if(dim >= 4096)
    {
    Write("The number of columns of fftArray is too large (currently %d).
    The calculation cannot be computed and is stopped. Reduce the number of
    columns to a value less than 4096. NOTE: The number of columns has to
be a
    power of two. ", dim);
    cancel;
    }
  // check if dimension of fftArray is a power of two
  isPowerOfTwo = 1;
  checkVal = dim;
   while((checkVal != 1) && (isPowerOfTwo == 1))
    {
     if(checkVal%2 == 0)
      {
      checkVal = checkVal / 2;
      }else{
      isPowerOfTwo = 0;
      }
    }
   if(isPowerOfTwo == 0)
    {
    Write("The number of columns of fftArray is not a power of two
    (currently %d). The calculation cannot be computed and is stopped.
Adjust
    the number of columns.", dim);
    cancel;
    }

   // initialize global array fftArray, all elementens equal to zero
    for(idx = 0; idx < dim; idx++)
     {
     fftArray_blockFFT[idx] = 0;
     fftArray[idx] = 0;
     }
  }

 sample++;
 if(sample%inkr==0) //execute BlockFFT() only if number of samples is equal
to inkr
   {
    counter++;
    Write("sample %d",sample);
    Write("counter %d",counter);
    t_end = time(signal.mbuffer[sample-(inkr*counter)-1]); //the BlockFFT
will be computed after inkr-samples
    //t_end = time(FILE1.XCPsim.channel1.mbuffer[inkr-1]);
    Write("t end %f2.2",t_end);
    //blocklength=inkr;

    // copy the last BUFFER_SIZE number of datapoints to the array buffer
to be FFT analyzed.
     for (n = 0; n < BUFFER_SIZE; n++)
      {
      buffer[n] = signal[-n];
```

```
      }

    //calculate FFT
    BlockFFT(buffer, BUFFER_SIZE,  BLOCK_FFT_MODE_RMS_AMPLITUDE,
fftArray_blockFFT);

    overall_lin=0; //reset overall_lin
     for(n=0;n<dim;n++)
      {
      fftArray[n]=fftArray[n]+pow(fftArray_blockFFT[n],2); // first part of
RMS avaraging : calculate the sum of squares

      fftArray_blockFFT[n]=fftArray_blockFFT[n]*2/dim; // normalization of
the FFT result to represent phys. correct values
      // normalize FFTs by the number of sample points --> dim
      // Note that FFT transformation will divide the power between the
positive and negative sides,
      // so if you only look at one side of the FFT (that's the case here),
the FFT result has to be multiplied by 2

      overall_lin=overall_lin+pow(fftArray_blockFFT[n],2); //overall level
(the sum of amplitudes at each frequency) for each FFT block
      }

     overall_lin=sqrt(overall_lin);

     Write("overall lin %f2.3",overall_lin);
   }
 return;
}
```

5 How to use

5.1 Online use case

In this example, channel 1 from CANape XCPDemo example is used as input signal for FFT calculation. Let's assume that this synthetic signal represents sound pressure. The period and the amplitude are set to 1 s (or frequency 1 Hz) and 1 Pa, respectively (Figure 1).

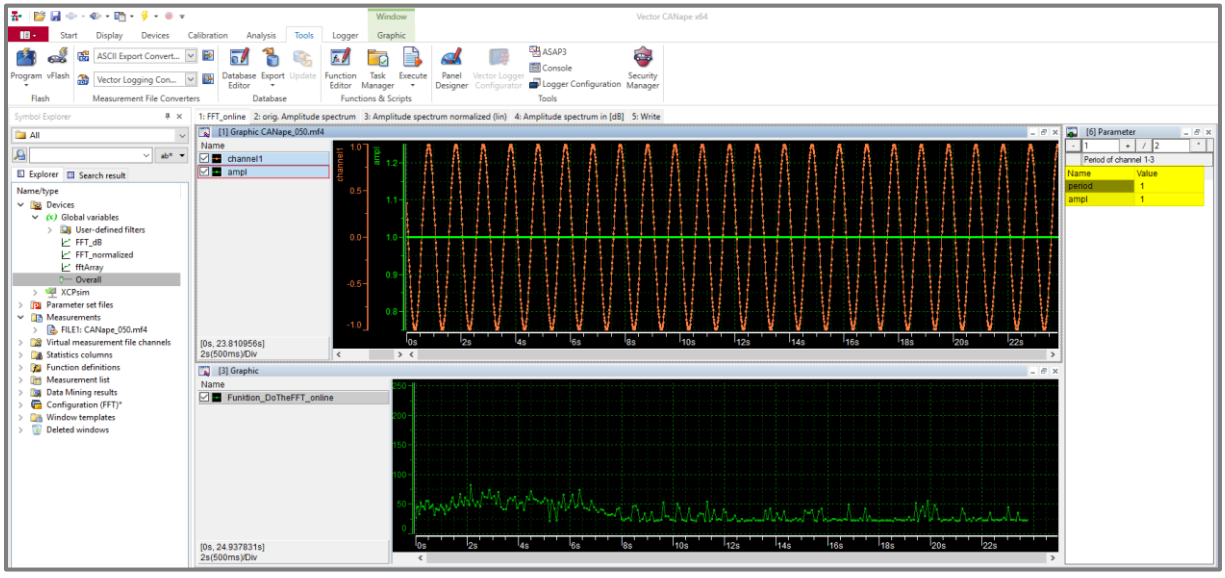

Figure 1 – Setting period and amplitude for channel 1 signal

On the **orig. Amplitude spectrum** page, the original FFT result is shown (Figure 2).

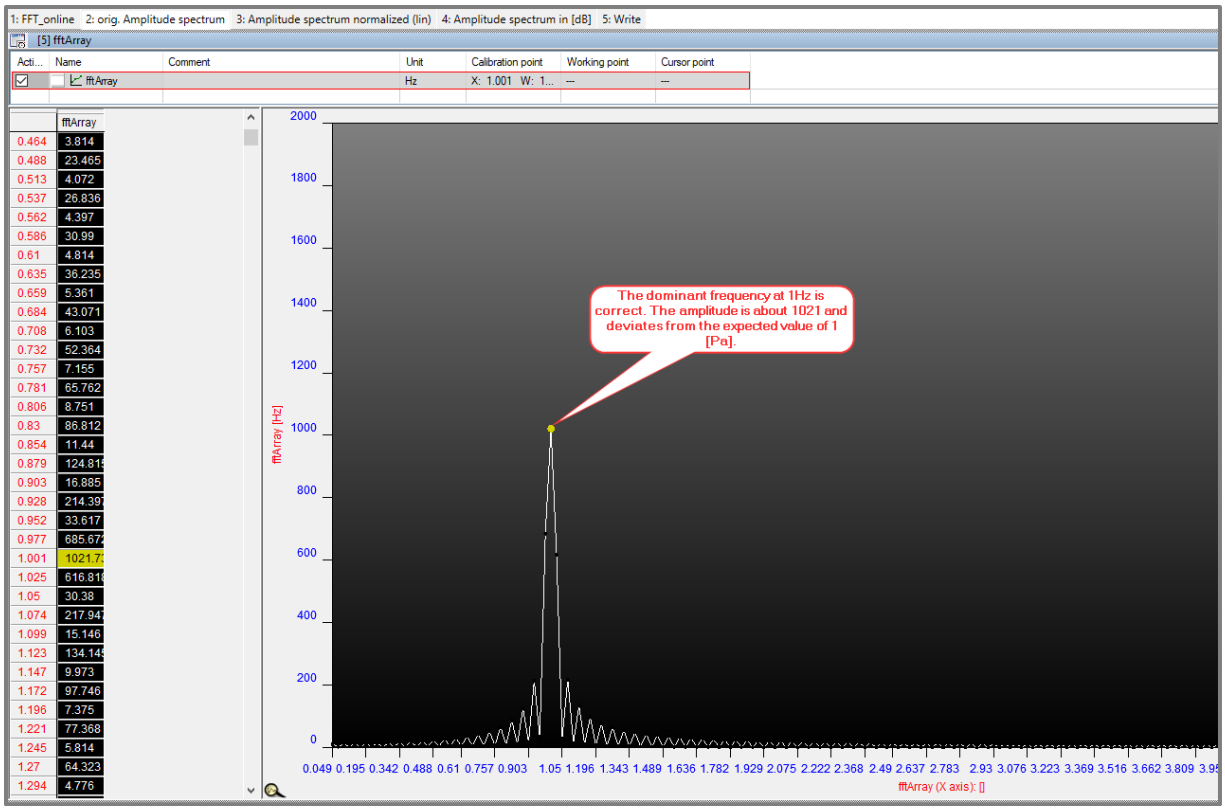

Figure 2 – Original FFT result

The peak is at 1 Hz just as expected for a sine wave with a period of 1 s. However, the amplitude is expected to be 1 [Pa] instead of 1021 (Figure 2). In order to obtain physically correct amplitude values, follow these steps: First, the original FFT result has to be normalized by the block length (number of samples in a single FFT block) that is 2048 in this example.

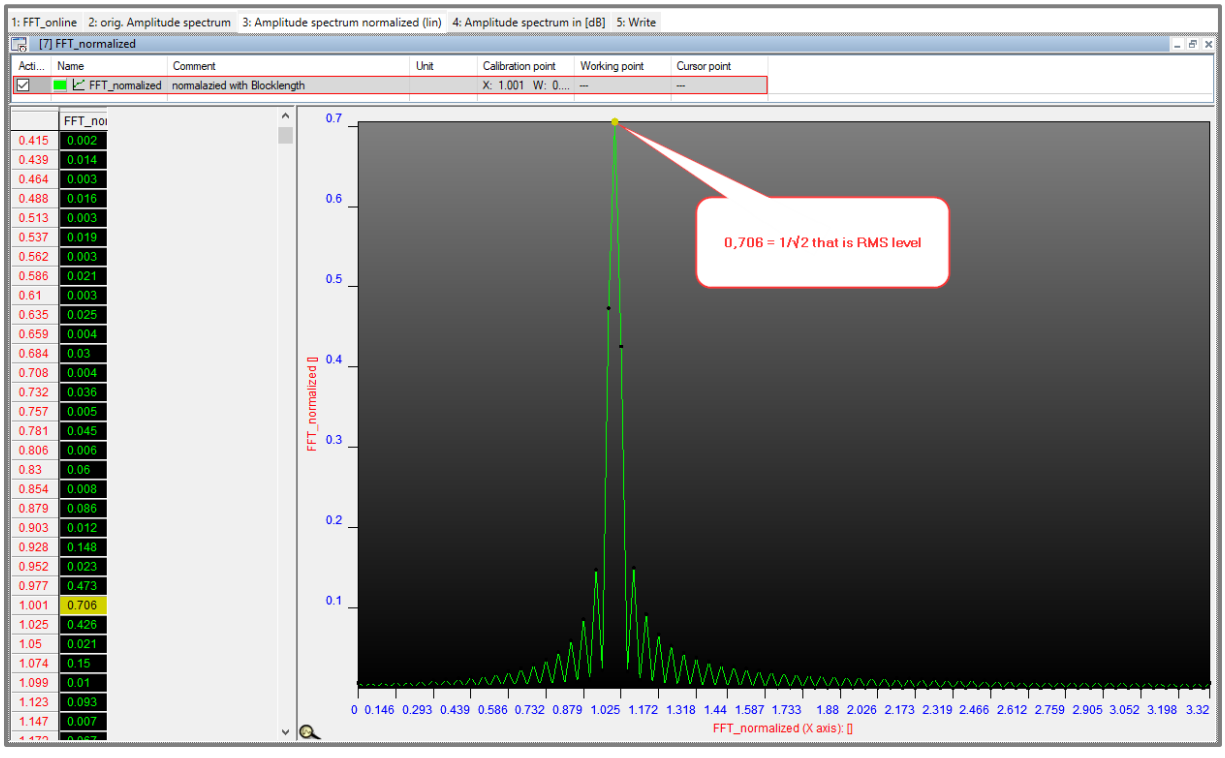

Figure 3 – Normalized FFT result

Second, in case the **BLOCK_FFT_MODE_PEAK_AMPLITUDE** flag was set for the FFT calculation, the normalized FFT result has to be multiplied with 1/√2 since the FFT spectrum result represents the RMS level at each frequency (see also Parseval's theorem for details). Third, the FFT result has to be

multiplied by 2 since only the positive side of the FFT transformation is taken into consideration. Note that FFT transformation will divide the power between the positive and negative sides. Finally, the sound pressure level is a logarithmic measure of the effective pressure of a sound relative to a reference value. It can be calculated using the following formula:

$$Lp = 10 * log_{10}\left(\frac{p^2}{{p_0}^2}\right) dB$$

The commonly used reference sound pressure p_0 in air is $=2*10^{-5}$ Pa. The result of this calculation is shown on the **Amplitude spectrum in [dB]** page (depicted in Figure 4).

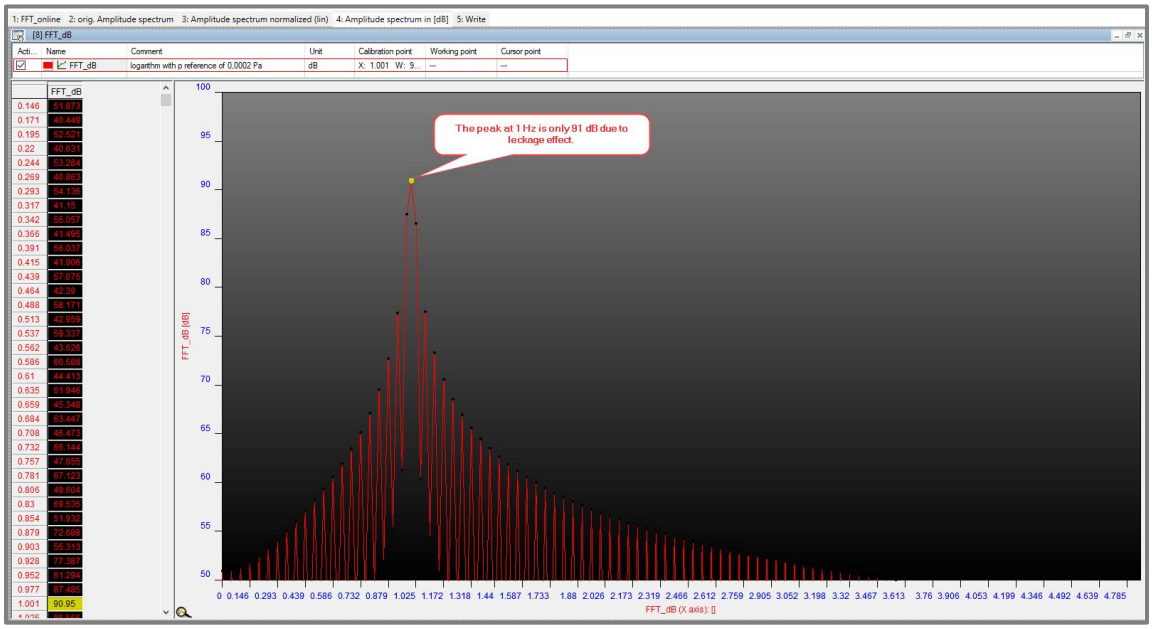

Figure 4 – FFT result in dB

However, the amplitude at 1 Hz is only 91 dB. It is 3 dB smaller than the expected level! A pure sine wave with an amplitude of 1 Pa should theoretically result in 94 dB. Also, the power in the signal seems to 'leak out' to nearby frequencies. This is the result of the so-called leakage effect that occurs when a non-integer number of periods of a signal is computed by an FFT analysis. You may verify this statement by calculating the sum over all amplitudes in the spectra, which is 94 dB. The global variable `Overall` sums up all amplitude contributions in this example. Its value after the analysis, which is indeed 94 dB, is displayed in Figure 5.

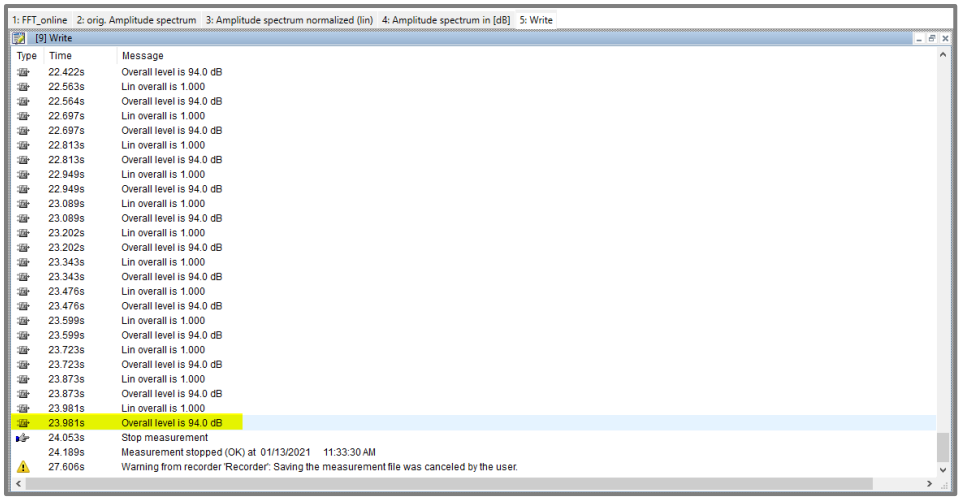

Figure 5 – Global variable Overall printed in the Write window

Caution

It takes about 20 seconds before the Overall level reaches a stable value

> This is caused by a low sampling frequency (measurement mode is 10 ms)

5.2 Offline use case

In the offline use case, the **channel 1** of the **XCPsimDemo_Sinewave_1Hz** measurement representing a 1Hz sine wave is analyzed. This use case covers averaging of the multiple FFT blocks over all measurement time. The method is called RMS (Root mean square) averaging described by the following formula:

$$FFT_{RMS} = \sqrt{\frac{1}{n}(FFT_1^2 + FFT_2^2 + \cdots + FFT_n^2)}$$

The purpose of the RMS averaging is to cancel out the random noise revealing the underlying system response with the minimum of noise. The calculation is carried out after starting the **Execute_FFT.cns** in the Task Manager (Figure 6).

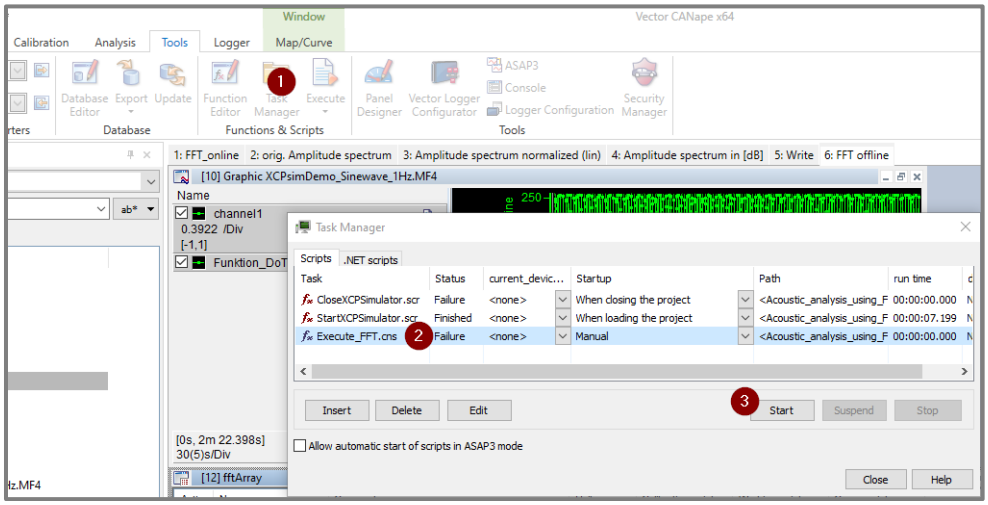

Figure 6 – Start the Execute_FFT.cns

The result is displayed on 'FFT' page, see Figure 7. Additionally, further information on the calculated FFT is printed out in the write window. First, the **counter** indicates the number of FFT blocks that were used for averaging. This parameter can be modified by the local parameter **inkr (**see Figure 9) of the **Funktion_DoTheFFT_offline** function. The **inkr** parameter specifies the number of input samples after which the FFT calculation is performed. The higher the value for the **inkr** parameter, the smaller the **counter**. The specification of the **inkr** parameter and the FFT block length have an impact on the calculation of the so-called Moving Average (see Figure 7 for details).

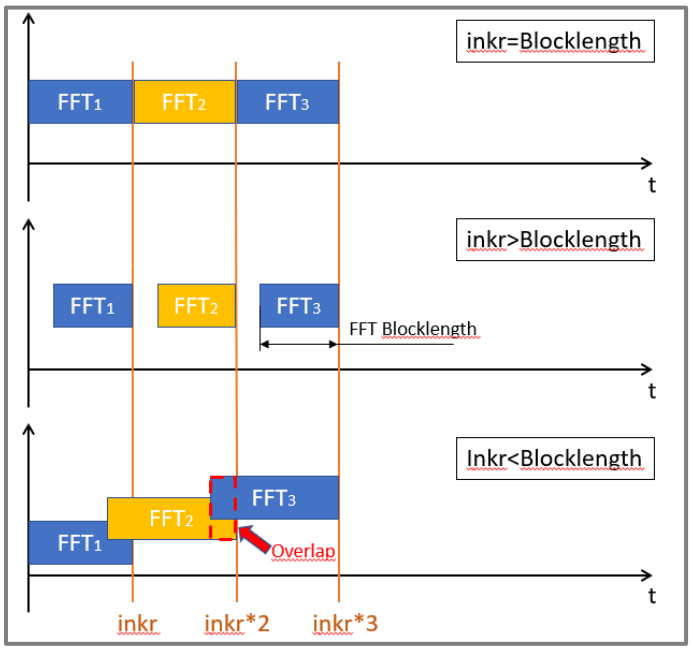

Figure 7 – Moving Average

The **inkr** parameter can have any value. However, the same value as for the FFT block length or smaller is recommended.
Second, the overall_lin gives the sum of all amplitude contributions of each FFT block that's equivalent to the online use case, see chapter 5.1.

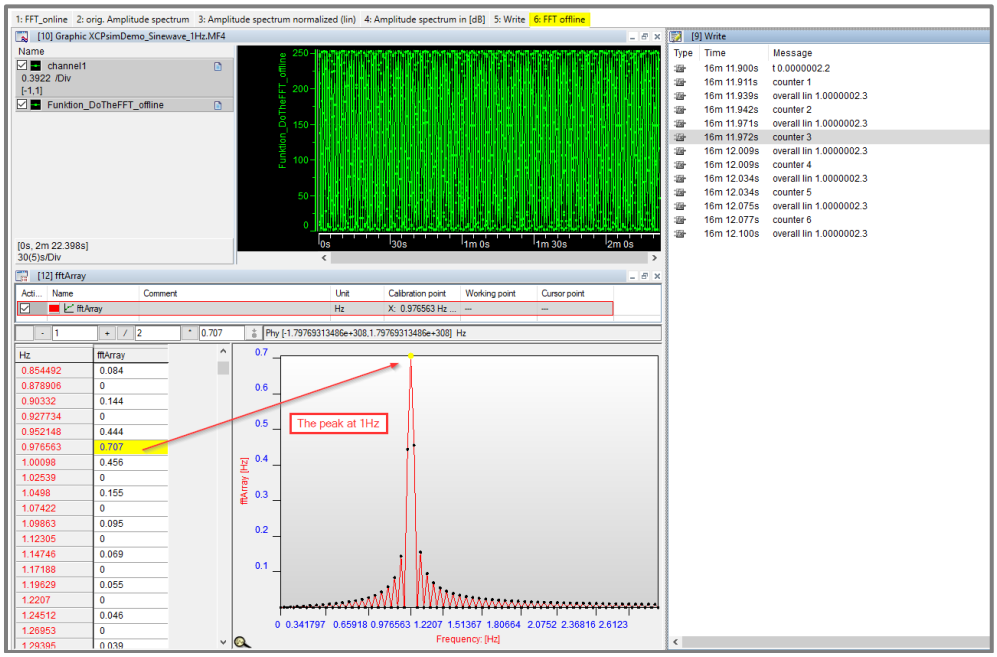

Figure 8 – The result of RMS averaging

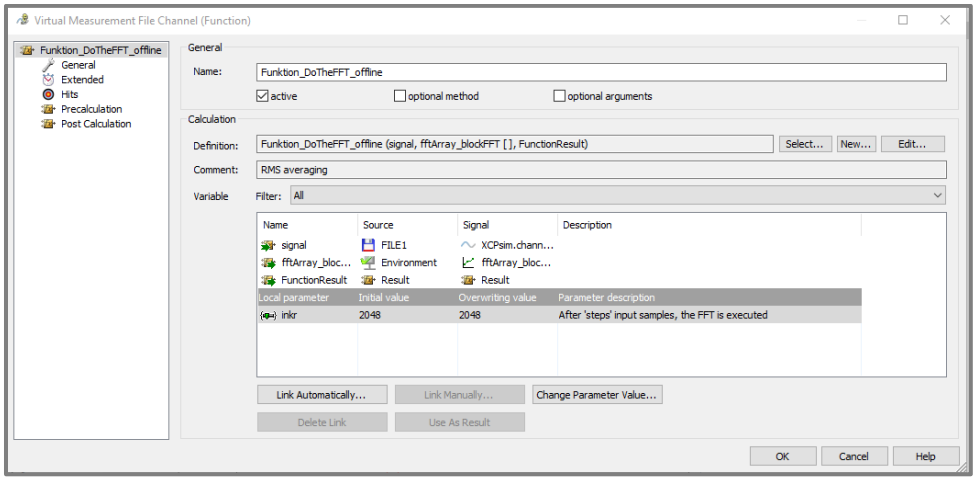

Figure 9 – Virtual Measurement File Channel settings for the Funktion_DoTheFFT_offline function

6 Associated files

The source code and/or example project files (SN-IMC-1-097-Acoustic_Analysis_Using_FFT_Function.zip) can be found in the ZIP archive in the corresponding KnowledgeBase Article.

7 Contacts

For support related questions please address to the support contact for your country https://www.vector.com/int/en/company/contacts/support-contact/.