Vibration Analysis of Car Engine - MatDeck and Python

Description

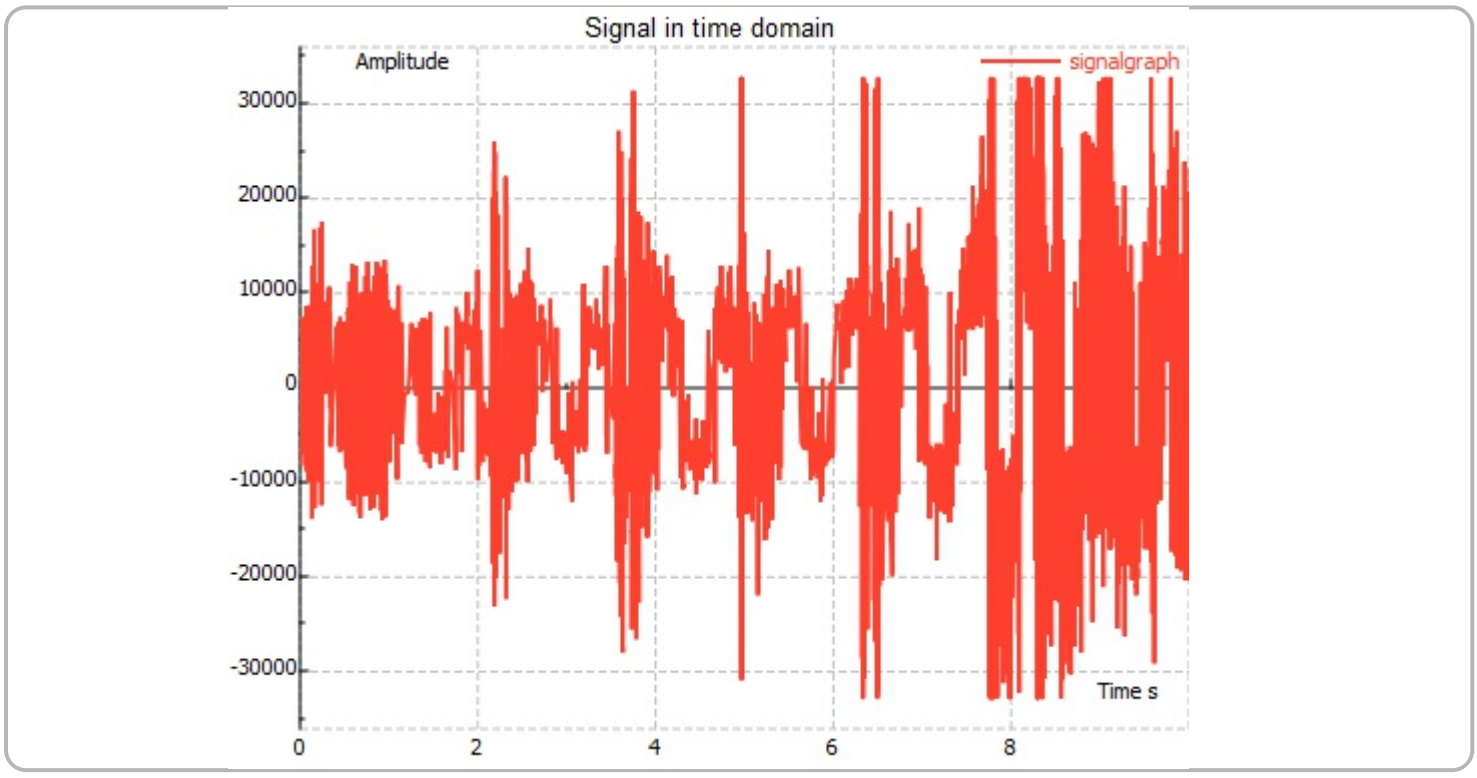
Fourier Transform -FT, and spectrograms are all common tools used for vibration analysis. In the following segment, we show how these tools are used in MatDeck to perform vibration analysis in real world examples. Furthermore, we show how Python can be used inside the MatDeck document for additional calculations.

The real world example deals with the vibration analysis of a car engine which is recorded by a sensor known as a accelerometer. The data is recorded in a .wav file and the MatDeck functions which are used for reading these types of files are also taken under consideration. After that, we go through the important differences between an FFT, and spectrogram and we illustrate when it is appropriate to use each type of vibration analysis tool.

Data acquisition

As explained above, we deal with data captured with an actual accelerometer and recorded within the carengine.wav file. In order to reproduce the examples, the .wav file should be in the same folder as this document. The file ,carengine.wav, contains 10 second recording of a car engine while it was idle with the sampling rate equal to 20kHz. The signal is first analyzed in the time domain by loading it from the file and showing it graphically. The signal is also played and we can hear the sound of the vibrations before we perform deeper analysis. We use MatDeck script for the processing described above.

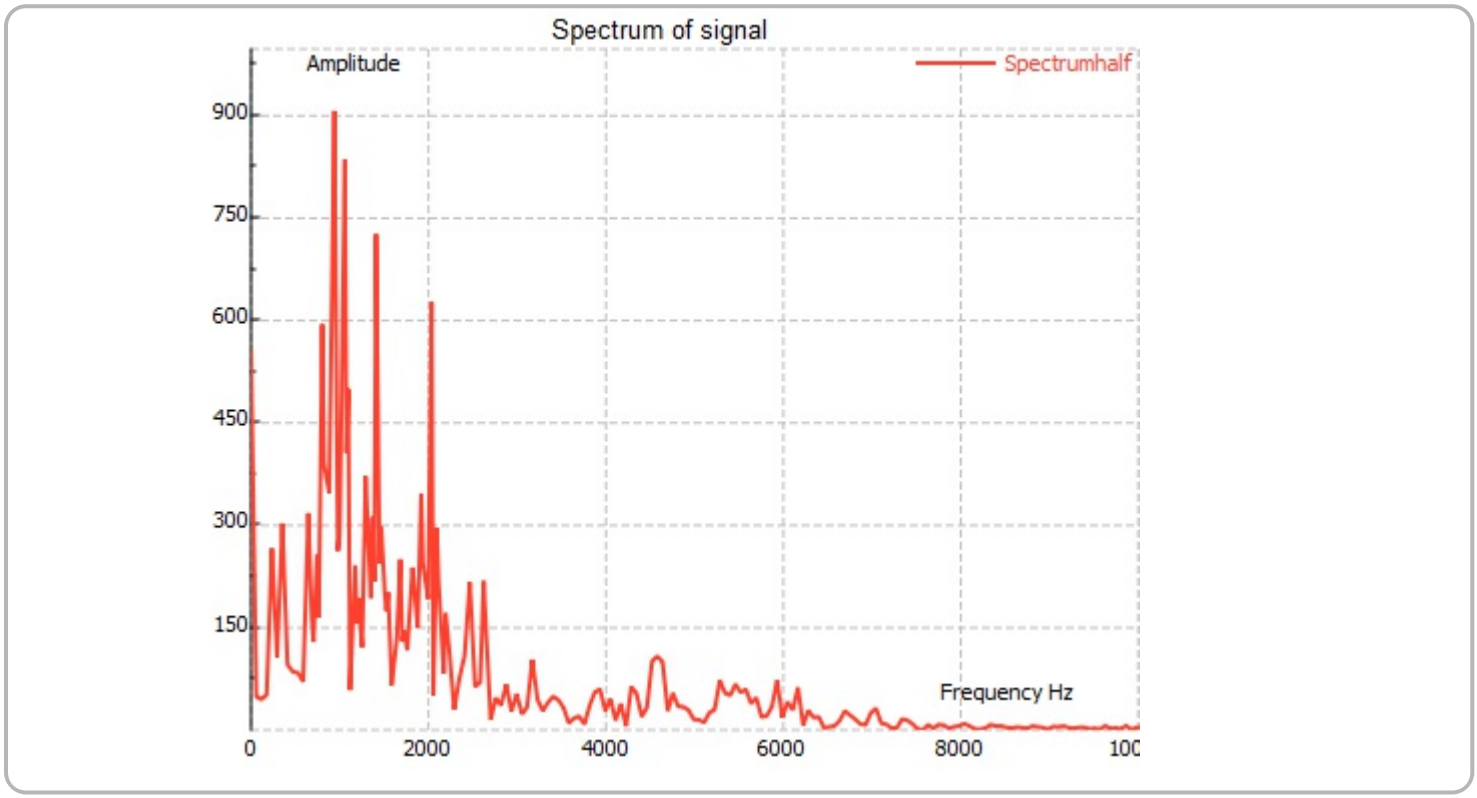
```
fileID := "carengine.wav" //file name containing the signal
props := wave_properties(fileID) //properties of the wav file
Fs := props[0] //sampling frequency
signal0 := wave_read(fileID) //read data from wav file
audio_play(Fs, props[1], signal0) //play the content
timex := ynodes(t, 0, 10, 20000) //time axis for the signal
signalgraph := join_mat_cols(timex, subset(signal0, 0, 0, 19999, 0))
```



FFT analysis in MatDeck

The result of the FFT analysis is the acceleration/vibration amplitude as a function of the frequency, which allows us to perform analysis in the *frequency domain* (or spectrum) to gain a deeper understanding of our vibration profile. Most vibration analysis is typically done in the frequency domain. The number of discrete frequencies that are calculated using fft() as part of a Fourier transform is directly proportional to the number of samples in the original signal. However, the FFT analysis can be calculated for a smaller number of frequency points as well. The user has to decide the trade-of between spectrum precision and the number of operations. If N is the length of the calculated spectrum, the frequency samples are given in intervals of Fs/N from 0Hz up to (N-1)Fs/N. Real world signals have a conjugate symmetric spectrum, which means that the amplitude spectrum is symmetric and the phase spectrum is anti-symmetric around the origin. Therefore, it is very common to show the amplitude spectrum from 0Hz up to the half of the sampling rate.

```
8  N0 := 1024 //Number of frequency points for FFT
9  freq := ynodes(f, 0, N0 * Fs / (N0 - 1), N0) //Frequency axis
10 Spectrum := fft1n(signal0, N0) //Calculation of fft
11 Spectrumgraph := join_mat_cols(freq, abs(Spectrum) / N0)
12 Spectrumhalf := subset(Spectrumgraph, 0, 0, N0 / 2, 1)
```



FFT analysis in Python

The same analysis can be done in Python, by coding directly inside the MatDeck document. Python and MatDeck can exchange variables easily as illustrated in the code chunk. First, we create the variable Spectrum P in Matdeck, and we then assign it's value in Python.

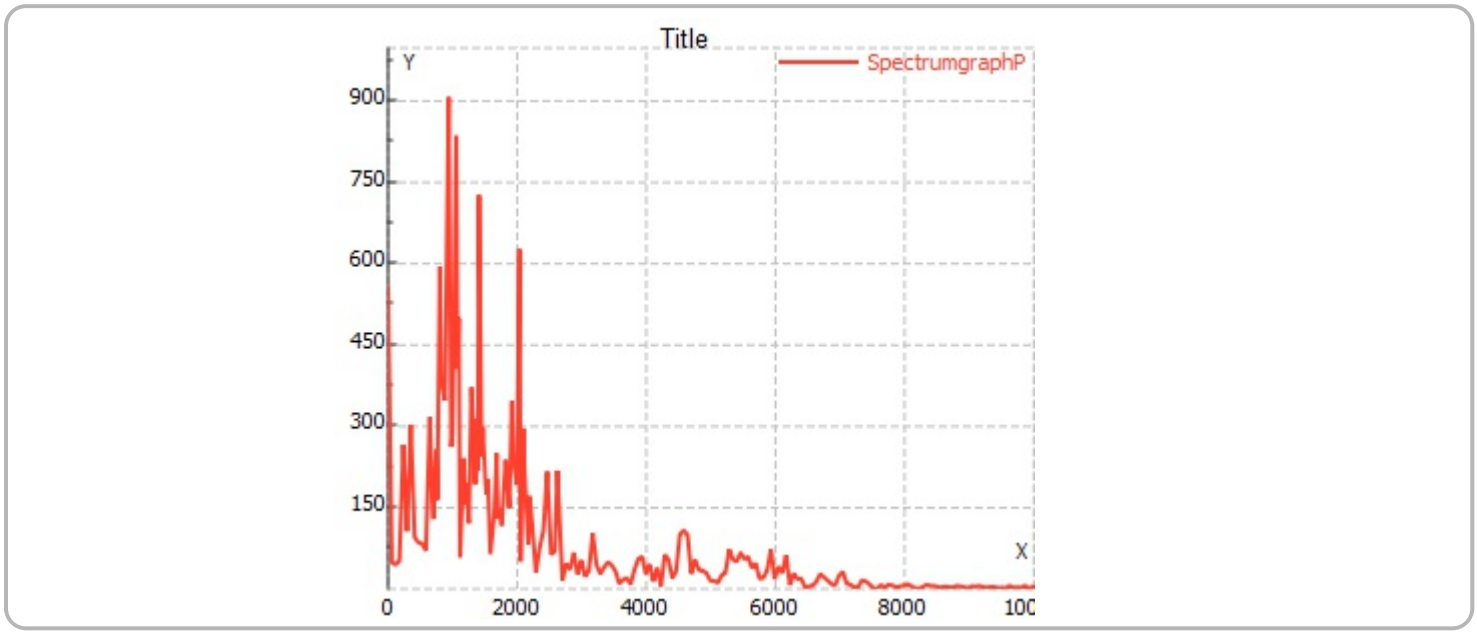
```
13 Spectrump := 0
```

Python code used to calculate FFT is given in the figure below. The FFT function uses MatDeck variables.

```
#py
from scipy.fftpack import fft

Temp = fft(signal0, N0)

Spectrump = Temp.tolist()

###
```

We can plot the amplitude spectrum in MatDeck using the value obtained in Python.

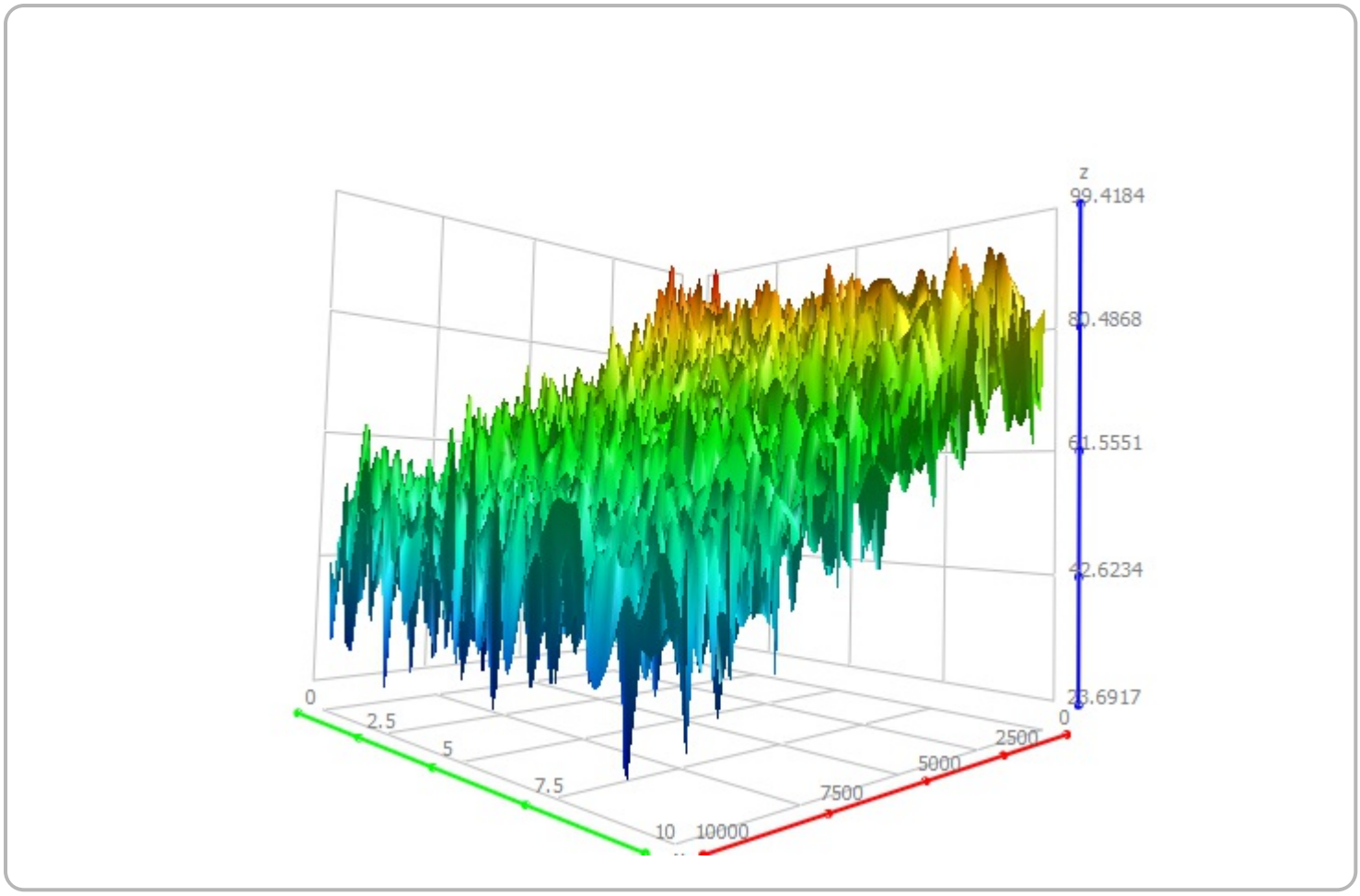
```
SpectrumgraphP := join_mat_rows(freq, abs(Spectrump) / N0)
SpectrumgraphP := subset(SpectrumgraphP, 0, 0, 1,  N0 / 2)
```



Spectrogram

From the frequency spectrum of the signal, we can see the dominant frequency components of the signal. However, we can't see the time domain position when certain frequency components occur. In this example, and whenever the vibration frequency changes with time, we need a spectrogram. A spectrogram works by breaking the time domain data into a series of chunks and taking the periodogram of these time periods. These series of periodograms are then overlapped to visualize how both the amplitude and frequency of the vibration signal changes with time. We need a three dimensional graph to represent the spectrogram against time and frequency. MatDeck contains a function called spectrogram() which shows the results in a 3D graph. MatDeck's function spectrogram() has the following arguments: input the signal as a vector for which the power spectral density is estimated, window function is determined by the string name of the window used and the number of samples used to generate the result which is also the number of samples used to perform fft() within each block in time. The last two arguments are: the block length in samples, and number of overlapping samples between consecutive blocks. These two arguments define the overall number of points in the time taken to calculate the spectrum.

```
block := 10000 //block length
nover := 5000 //Overallping samples between segments
// Spectrogram, rectangular window is used
Spect := spectrogram(signal0, "rectangular", N0, block, nover)
// Prepare data for 3D graph
Sp := data3d(10 * log(Spect), xx, 0, Fs / 2, y, 0, 10)
gr1 := graph3d(0, Sp) //the 3D graph is prepared
set_size(gr1, 550, 450) // set size of 3D graph
```



In this example, the engine was revved for a short while during the experiment. The spectrogram shown above illustrates how the dominate frequencies change with time in relation to when the car engine was idle and revved. Using a spectrogram, we can get a deeper analysis of the vibration profile and how it changes with time.