WIME toolbox for Scilab

Adrien Deliège, Samuel Nicolay

March 10, 2017

1 WIME for beginners

WIME stands for "Wavelet-Induced Mode Extraction", which is a method designed to extract oscillating components from a given signal. This technique relies on the continuous wavelet transform and can be used in many applications. At the time of writing this README document, an article about WIME is under review for publication in Physical Review E. However, you can request a copy of the draft of the paper via [1] (click on "Article-Submitted.pdf"). Alternatively, a beamer on WIME and a comparison with the "Empirical Mode Decomposition" can be found at [2] (select the pdf "BeamerLLN.pdf").

There are multiple ways to use WIME in your research. In this section, we review the most intuitive and easy-to-use functions that are implemented and that you can use without worrying about technical details. Please make sure that the fftw3 library is installed on your computer before using this toolbox.

1.1 *WIME* function

The basic function that extracts oscillating components from a signal following the ridges of maxima is naturally called *WIME*. You can have a look at *demo1.sce* and *demo2.sce* for examples. Its full syntax is the following:

```
comps=WIME(signal,visu,wavpar)
```

where *comps* are the components extracted by the algorithm, *signal* is your data, *visu* is an optional parameter that switches on/off the visualization of the process and *wavpar* is an optional argument that deals with the parameters of the wavelet transform and the number of iterations of the algorithm. The default value of *visu* is 0 (off). The *wavpar* vector is structured as

`wavpar=[nOct, nVoices, alpha, beta, omega, nIter, thresh]`

where *nOct* is the number of octaves that will be used, *nVoices* is the number of voices between the octaves, *alpha, beta, omega* are parameters for the wavelet, *nIter* is the maximum number of iterations (maximum number of components to extract) and *thresh* is a threshold (between 0 and 1) of energy under which the extraction of components automatically stops. The default values are

`wavpar=[-1, 40, 1, 2, 5.336446, 10, 0.05].`

Note that the value -1 for *nOct* indicates that you let the program compute the largest number of octaves that should be used given the size of the data. You can customize the *wavpar* vector only partially if you want: just use -1 where you want the default values to be used. For example,

`wavpar=[-1, 20, 2, -1, 5.336446, -1, 0.3]`

is equivalent to

`wavpar=[-1, 20, 2, 2, 5.336446, 10, 0.3].`

Consequently, the simplest way to call *WIME* is:

```
comps = WIME(signal)
```

which will extract components automatically. It can also be worth visualizing the successive TF planes, ridges and spectra, which can be done by simply calling

`comps = WIME(signal,1).`

Note that the vertical axis of the TF planes and the horizontal axis of the spectra are labeled "Period" and not "Frequency". They do not take the sampling into account, i.e. if you have e.g. 50 measurements per second then the period "75" corresponds to a period of 1.5 (=75/50) second. Also, they are in logarithmic scale. The labels on the axes can be modified with basic Scilab manipulations. See also the *spectrum* function below for a linearization of the horizontal axis of the spectrum.

1.1.1 A few words on *wavpar*

The number of octaves, *nOct*, is directly linked with the periods you want to investigate. The larger *nOct*, the larger the periods. You can choose a low value of *nOct* to examine only short periods if you are interested in e.g. denoising procedures, which allows you to gain time compared to the default case. If you enter a value of *nOct* which is too large, then all the wavelet coefficients will be impacted by border effects, which can be seen as the fact that the periods to be investigated are too large and cannot be efficiently detected; you will receive a message giving the number of octaves that has been used instead. If you really want to force *WIME* to consider larger periods/octaves, then you should artificially extend the signal by adding enough zeroes at the beginning and/or at the end of the signal. This padding is done by default anyway, but giving a longer signal fools the code and allows you to go beyond recommended limitations. Be aware that the results obtained for these larger octaves can be seriously flawed by border effects. Also, do not forget to post-process the components (cut them) so that they have the same length as the initial signal. Although this kind of procedure can be useful, we advise that you perform it carefully.

The number of voices, *nVoices*, is related to the "frequency resolution" that you want. In other words, it denotes the number of intermediate scales/frequencies/periods that have to be considered between each octave, so that the time-frequency representation of the signal is "continuous" along the vertical direction.

The parameters *alpha, beta, omega* allow to define a family of analyzing wavelets defined by their Fourier transform as:

$$\hat{\psi}(\xi) = \sin^{\alpha}\left(\frac{\pi\xi}{2\Omega}\right) e^{\frac{-(\xi-\Omega)^{\beta}}{2}}.$$

The default values make it close to the Morlet wavelet and is well-suited for time-frequency analysis.

The algorithm of *WIME* as described in [1] is an iterative process which requires a stopping criterion. If the considered signal is purely AM-FM and long enough so that all the relevant components (i.e. including those at large periods) can be extracted, then the energy of the successive rests will tend to zero and so an energy-based stopping criterion will be met. In the present case, it corresponds to the moment where the energy of the current rest is lower than *thresh*$*$energy of the initial signal. However, for real-life signals, it can appear that trends, noise and too short signals can prevent the algorithm from converging in this way. Therefore, as a security measure, *nIter* limits the number of iterations and thus of components to extract.

WIME stops when either the number of iterations exceeds *nIter* OR when the energy-based criterion is reached.

1.2 *WIME_flat* function

In some cases, it can be more convenient to forget about following the ridges of maxima and just extract components along a fixed scale (the ridges are thus horizontal lines). The function *WIME_flat* performs this task, in the same spirit as the *WIME* function (see [3]). More precisely, at each iteration, the spectrum is computed and a component is extracted at the period corresponding to the highest maximum of the spectrum. The syntax is left unchanged:

```
comps=WIME_flat(signal, visu, wavpar)
```

and the parameters are the same as previously.

1.3 *WIME_flat_all* function

The last possibility that we discuss for extracting components from the signal is the function *WIME_flat_all*. Its principle is the same as *WIME_flat* but here, at a given iteration, all the local maxima of the spectrum engender a component, simultaneously. In other words, if at the first iteration, the spectrum displays 2 peaks, then 2 components will be extracted, at the corresponding periods (again the ridges are horizontal lines). These components are then subtracted from the signal and the process is repeated. The syntax is:

```
comps=WIME_flat_all(signal, visu, wavpar)
```

In this case, *nIter* still limits the number of iterations but will generally be different from the number of components extracted. This approach is very similar to *WIME_flat* and will often lead to similar results. It has the advantage of being faster since there are potentially less wavelet transforms to compute, but some of the components extracted can be irrelevant and/or noise-related, depending of the localizations of the maxima of the spectrum.

A variant of this technique is used in [4], where the periods at which to extract components are memorized at the first step and re-used for all the following iterations. This is mainly done to sharpen the components obtained in the first place and improve the reconstruction of the signal at the borders. See [4] for more details.

1.4 A last comment

At this stage, you have probably figured out that there are many ways to perform a wavelet-induced mode extraction. Consequently, we strongly encourage you to design your own codes tailored for your data and for the goals pursued. For that purpose, you only need to know a few details that are explained in the next section.

2 More details for easy customization

Once you have understood how to use *WIME* and the continuous wavelet transform to perform your own experiments, you might want to implement your own codes to refine your results. This section gives a quick overview of the few important functions that you would require and that are already written. You can also have a look at *demo3.sce* for an example with an electrocardiogram signal. The functions in question are the following:

- *wmd_cwt* is the sole function of the toolbox written in C. It is the key function that performs the continuous wavelet transform of a signal. Note that it uses the fftw3 library, which needs to be installed on your computer. The syntax of *wmd_cwt* is

  ```
  [mod,arg]=wmd_cwt(signal, nOct, nVoices, alpha, beta, omega)
  ```

  where *mod* is the modulus of the wavelet transform and *arg* its argument (in the present context, it is more convenient to use the polar representation of complex numbers than real and imaginary parts). They are matrices with *nOct*nVoices* rows and N =*length of signal* columns, where the first rows correspond to the highest frequencies/lowest periods. The visualization of the TF-frequency plane is actually a function that plots *mod*. The first rows of *mod* are often set to zero (or not displayed) for clearer TF-planes (cfr line "$a(1:nVoices,:) = min(a)$;" in the *WIME* function).

- *spectrum* is the function that computes the wavelet spectrum, i.e. the row-wise mean of *mod*. Its syntax is

  ```
  [sp,xsp]=spectrum(mod, wavpar)
  ```

  where *sp* is the spectrum, *xsp* and *wavpar* are optional arguments. If *wavpar* is given, then *xsp* corresponds to the abscissas of the spectrum converted into periods, which allows to plot the spectrum

```
plot(xsp,sp)
```

with a linear scale on the horizontal axis instead of the initial logarithmic scale. The conversion of the k^{th} abscissa to the corresponding period p is given by

```
p=2^((k-1)/nVoices)*2*%pi/omega.
```

- *VisualizeCWT* performs the TF representation of the signal under consideration. It can be used to plot *mod* and *arg*. Its syntax is

  ```
  VisualizeCWT(M, wavpar, nbcolors)
  ```

  where M is the matrix to be plotted and *nbcolors* is an optional argument indicating the number of colors to be used. The colormap used is the *jetcolormap* which ranges from blue (low values) to red (large values). The argument *wavpar* is only required for the conversion of the labels into periods for the vertical axis (see *spectrum*).

- *freqband* encompasses the abscissa of the highest local maxima of the spectrum between its two neighboring local minima:

  ```
  fb=freqband(spectrum).
  ```

  The structure of *fb* is

  ```
  fb=[fbmin1,fbmax,fbmin2].
  ```

- *findridge* computes the ridge of maxima given *mod* and *fb*:

  ```
  [ridge,bi,bs]=findridge(mod, fb),
  ```

  where *bi* and *bs* are the lower and upper bounds of the frequency band at each position/time.

- *Component* extracts the oscillating component corresponding to a given ridge:

  ```
  c=Component(mod, arg, ridge)
  ```

  and is simply computed as

```
c(t)=2*mod(ridge(t),t)*cos(arg(ridge(t),t));
```

for each time t. **IMPORTANT:** the multiplication by 2 is needed and justified by theoretical results ([1, 2]).

- *energy* simply computes the energy of a signal from the signal analysis point of view:

```
e=energy(signal),
```

i.e.

```
e=sum(abs(signal).^2).
```

That's it! Basically, all you need to understand is the *wmd_cwt* function, the conversion from indices to periods and the extraction of components. Do not hesitate to send your feedback or ask questions! Good luck with your research and enjoy *WIME*!

References

[1] https://orbi.ulg.ac.be/handle/2268/207133.

[2] https://orbi.ulg.ac.be/handle/2268/207123.

[3] Adrien Deliège and Samuel Nicolay. *A New Wavelet-Based Mode Decomposition for Oscillating Signals and Comparison with the Empirical Mode Decomposition*, chapter Information Technology: New Generations: 13th International Conference on Information Technology, pages 959–968. Springer International Publishing, Cham, 2016.

[4] Adrien Deliège and Samuel Nicolay. Analysis and indications on long-term forecasting of the Oceanic Niño index with wavelet-induced components. *Pure and Applied Geophysics*, pages 1–12, 2017.