The **1D Exponents Estimation** pop-up menu

Jacques Lévy Véhel 22 May 2000

This text presents a brief explanation of the functionalities the **1D Exponents Estimation** pop-up menu.

Contents

1 Overview

This area of **Fraclab** is dedicated to the estimation of various exponents that arise constantly in the (multi)fractal analysis of signals. These are :

- The pointwise Hölder exponent, which characterizes the regularity of the measure/function under consideration *at* any given point.
- The local Hölder exponent, which is related to the regularity of the measure/function under consideration *around* any given point.
- The (abusively called) long range dependence exponent ; this one describes the (possible) power law behaviour of the Fourier power spectrum at frequencies close to 0.
- The four parameters of a stable motion : see the *Synthesis* pop-up menu help for a brief description of stable motions.
- The 2-microlocal exponents : these yield a finer description of the local regularity properties of a signal, which goes beyond the Hölder exponents.

Pointwise and local exponents, as well as 2-microlocal exponents, are defined at each point. Thus, when we talk of estimating one of these exponents, we really mean estimating them at all points, i.e. estimating e.g. a pointwise exponents function. In other words, the output is a new signal, rather than a number, as is the case when we compute the LRD exponent.

2 Pointwise Hölder Exponent

Many techniques have been developed to estimate pointwise (and local) Hölder exponents, none of which give satisfactory results in all cases. Estimating a local irregularity index on discrete data without any a priori assumption is indeed a difficult task. One way to obtain robust results is to use a parametric approach. Such estimators have been developed mainly in the case of fractional Brownian motion and its extensions, such as multifractional Brownian motion. One such method is implemented in **Fraclab**. Non parametric methods, which are more numerous, generally give the correct estimation only when some technical conditions are met. **Fraclab** currently includes three such estimators, which are, in order of increasing quality : a standard method based on the continuous wavelet transform, with various improvements ; a second one relying on the use of generalized iterated functions systems. And a third one based on the evaluation of the oscillations. Note finally that more robust estimates of both the pointwise and local exponents, which work under broader assumptions, are available when one computes the 2-microlocal exponents. See the help corresponding to this topic below.

2.1 Parametric estimation of the exponent of an fBm or an mBm.

For an fBm or an mBm with smooth H(t), the local and pointwise exponent are the same. The parametric estimator implemented here is based on (4), and is proved to converge almost surely as soon as one is dealing with a discrete trace of an fBm or mBm.

As in most processing tools of **Fraclab**, the first line of the sub-menu window, entitled **Input Signal**, should display the signal currently analyzed. A known bug here is that the current signal of the **Variables** list sometimes does not appear by default. When this occurs, or if you wish to switch to another signal by highlighting another selection in the Variables list, press **Refresh**. There is only parameter that needs to be chosen before you launch the estimation, which is the **window size**. The **optimal** choice, as the name indicate, is the theoretical best size, i.e. the size which minimizes the risk. As is often the case in such situations, visually more pleasing results are obtained by using a slightly larger size, typically by multiplying the optimal size by a logarithmic factor: choosing **enlarged** just does this. In many situations, you'll find that it gives a clearer picture of the meaningful variations of H, because the variance has been decreased (at the expense of course of an increase in bias). When you suspect that your analyzed signal is an fBm, i.e. the exponent is the same all along the path, a more judicious choice is to use the whole signal as a window. This is the third choice, **constant H**. Once you have decided a window size, press **Compute** as usual. The output signal, denoted *estim_Hölder_func_sig#* (where sig is your input signal), is a 1D vector which gives the exponent at each point. The two other buttons are the customary **Help** and **Close** buttons. Note finally that computing times can get very large if your input signal has more than 1024 points.

A known bug in this sub-menu is that border effects have not been satisfactorily taken into account. As a result, the estimates on the first points are sometimes weird (e.g. you get a plateau which has no real meaning).

2.2 CWT-based estimation

The Continuous-Wavelet-Transform-based non parametric estimator of the Hölder function is probably the most well known method for estimating exponents, although it is not the most precise one. It is based on a remarkable property of wavelet coefficients: They are bounded from above by a quantity that depends on scale and on the Hölder exponent. Since this bound is optimal, it can serve as a basis for an estimation method. However, the bound is practical only of one assumes that the relevant coefficients for estimating the exponent at point t are located "nearly above" t. This is equivalent to assuming that the pointwise and local exponents coincide. This condition is hard to verify in practice, and somewhat restrictive.

When you launch this method, you get a window entitled "Local Hölder Exponent Estimation" (recall that this will also be the pointwise one if the procedure is applicable). In this simple form, you need just decide at which point you want to compute the exponent, by entering a value in the **Estimation time** box (the default is 1, which is a bit unfortunate, since the first point will always suffer from border effects). On hitting **Compute**, the value of the estimated exponent will appear in the **Local Scaling Exponent** box. If you need more control on the computations, hit **Advanced compute**. A new window appears, entitled **Hölder Function or Local Exponent**. Check first in the **Input Signal** box that you are dealing with the desired signal, otherwise **Refresh** it in the usual way. You will then specify various parameters related to the wavelet transform: **fmin** and **fmax** let you choose the minimum and maximum frequencies of analysis. The default values are the ones yielding maximal span compatible with the size of the signal. You may change the extreme frequencies either by typing values under **fmin** and **fmax**, or by using the predefined values on the menus to the right. The **Voices** parameters governs the number of intermediate frequencies between **fmin** and **fmax** at which the continuous wavelet coefficients will be computed. Be warned that giving an excessive number of voices may result in large computing times for long signals. Checking the **Mirror** item will deal the border effects by mirroring the signal at its extremities. Otherwise, zero-padding is used. Finally, you

may choose the **Size** and **Type** of your analyzing wavelet: available wavelets are the **Mexican Hat**, and the real and analytic **Morlet** wavelet. The size may be any positive number (this parameter is not available for the Mexican hat). Once all the parameters that define the wavelet transform are chosen, hit **Compute WT**. The output signal is a matrix of size "number of voices" x "size of the original signal". It is called *cwt_signal#*, if "signal" is the name of your data, and where # is as usual an incremental parameter. It should appear in the **Input CWT** box just below the **Compute WT** button. You may want to view the continuous wavelet transform using the **View** menu. Note that **Fraclab** recognizes wavelet transforms, and display them differently form regular images. In particular, it uses a fixed aspect ratio (this is useful for instance if the number of voices is much smaller than the size of the signal), and the "jet" color-map, which often allows to highlight the important structures. If you want to view the transform as a normal image, or make other changes in the appearance, use the functionalities of the **View** menu described in the **Overview** help file.

The **Refresh** button to the left of the **Input CWT** box lets you load a wavelet transform which would already be present in the **Variables** list of the main window. This avoids computing several times the same transform. Once you are happy with your transform, move to the lower part of the window, which performs the actual computation of the exponents. Decide first if you wish to compute a **Single Time Exponent** (default), otherwise uncheck this box, and it will become **Hölder Function**. In the first case, you need to give the **Time Instant** at which the exponent will be estimated (with again the unfortunate default "1"). In the second case, this choice becomes unavailable, because you will compute the exponent at each point. In fact, this last statement is not quite true: Because the CWT-based estimator is a bit slow, it has been arbitrarily decided that only one point every four points would actually be processed. This is why the output signal is four times smaller than the input one. This fact may seem somewhat strange, because it means that the results of this menu cannot really be used in actual applications you may have. However, our justification is that the CWT-based estimation does not usually give good results. This window is thus mainly included because wavelet-based estimators are widely used, and for comparison purposes. Some of the reasons why the method often fails are explained at the end of this paragraph.

Recall that the exponent is obtained thanks to the fact that the coefficients are bounded from above by a certain quantity. As a consequence, in many cases, more relevant estimates are obtained if one chooses, at each scale, the largest coefficient among the ones which are located "nearly above" the current point t. This is the default in **Fraclab**, as is indicated by the fact that the **Local Maxima** box is checked. If you unmark this box by pressing the button to the left of **Yes** (which subsequently becomes **No**), then the program will use the coefficients which are in the column *strictly* above t. In case you want to use the local maxima, you need to tell **Fraclab** where to look for the largest coefficient. In other words, you must define precisely what is meant by "the coefficients located *nearly* above the point t". This is the purpose of the **Radius** and **Scale Depth** parameters. The default values of respectively 8 and 1 mean that, starting from the point t, the program will look for the largest coefficients at the scale immediately above (this is the "1"), and in a spatial neighbourhood of size 8. These two parameters are available for both the estimation at a single point and at all points. To the contrary, the **Specify Regression Range** choice is only meaningful for a single time exponent. When **Specify Regression Range** is chosen, you will be able to decide which range of frequencies is to be used for the estimation of the exponent (we explain how at the end of this paragraph). Otherwise, when **Full Range Regression** is selected, the estimation is performed using all coefficients between **fmin** and **fmax**. The exponents will be obtained through a regression of the logarithm of the wavelet coefficients modulus with respect to scale, and you may choose the type of regression: The

default is **Least Square Regression**. Other choices are **Weighted Least Square**, **Penalized Least Square**, **Maximum Likelihood** and **Lepskii Adaptive Procedure**. Only the last one is not well known. See (6) for details.

When you hit **Compute**, three things may happen: if **Hölder Function** was checked, the program will output a signal called em/signal_Ht#/, with the usual naming conventions. If **Single Time Exponent** was checked instead, with the option **Full Range Regression**, the program will display a graphic window showing in abscissa the log-scale, and in ordinate the log of the modulus of the coefficients. This display helps to check if the regression is meaningful, i.e. if the points on this graph are well aligned. In that view, the regression line is displayed in red. In parallel, you get the value of the estimated exponent in the box **Local Scaling Exponent**. If **Specify Regression Range** was checked instead of **Full Range Regression**, you also get a graphic window, but this time you have additional information and control: to the left of this window, you see the part of the wavelet transform which lies in the neighbourhood of the selected time instant. Note that a grey levels color-map is used. You will also see two red curves, which show the region in which local maxima are searched for when this option is activated. Finally, you should see, at each scale, i.e. on each line of the wavelet transform, a circle which indicates which coefficient was chosen at this particular scale. If no local maxima were used, this circles will lie exactly above t, while otherwise they might be distributed anywhere between the two red curves. The graph to the right of the window is the same as in the previous case, i.e. it shows in abscissa the log-scale, and in ordinate the log of the modulus of the coefficients. Now **Specify Regression Range** means that, using the large cross that appears when you point to this window, you will be able to select a range of frequencies between which the regression will be performed. Of course, you want to choose a region where the points are reasonably well aligned. Sometimes there is no such region. In other cases, there might be two or more sub-parts in the graph where linear behaviours are observed. Since we are interested here in local exponents, you should in general prefer to choose the region containing the highest frequencies. To actually select the region, click first on the left end of the chosen interval, then click again on its right end, or vice-versa. Each time you have selected a frequency range, **Fraclab** will compute an estimated exponent and show its value in the caption above the graph to the right of the window. In addition, the regression line will be displayed in red. If you want to test another region, just click again its endpoints. Once you are satisfied with a result, hit **Enter** on your keyboard, and the cross will disappear. The exponent will then appear in the **Hölder Function or Local Exponent** window in front of **Local Scaling Exponent**. You may now close the graphic window showing the evolution of the wavelet coefficients.

To understand some of the reasons why the method is not so good, try it on a deterministic Weierstrass function synthesized with the defaults options. Launch the **Advance compute** procedure, and use again the defaults, except that you estimate at the middle point of your signal, i.e. at abscissa 128. Observe how the log-log plot display oscillations: By choosing a regression range on various parts of the frequency interval, you will be able to get essentially any exponent you want, including negative ones.

2.3 GIFS-based estimation

This method is applicable only when the pointwise and local exponents coincide. In contrast with the CWT-based one, it is very fast. The principle is the following: The extension of IFS called GIFS allows to construct signals which are able to approximate (in L2 and L-infinity norms) any L2 signal with arbitrary precision. GIFS are just IFS where the number of maps and the various parameters are allowed to change at each scale

(see (2) for details). The first step in this estimation method is to compute a GIFS which approximates at best the original signal. In that view, one starts by computing the discrete wavelet transform of the signal. The parameters of the GIFS are then simply obtained as ratios of the wavelet coefficients. Once the GIFS is known, the estimation of the exponent is easy, because there is an analytical formula which gives the exponent at each point of a GIFS as a function of its parameters. Because the formula in only valid in the limit of infinite resolution, the obtained result is of course a finite size approximation.

From a practical point of view, you start as usual by checking and maybe updating the **Input** signal. You then choose which approximation procedure you want to use, i.e. the **Limit type** in a choice of **regression** and **Cesaro** (choosing one or the other option will in general hardly affect the result). Finally, you may choose which **Wavelet** to use, although in the current implementation of **Fraclab** only Daubechies 4 is available. On pressing **Compute**, you get an output signal, named *alphagifs_signal#*, which contains the estimated Hölder function. Note that this program assumes that the input data contains 2^N points. If this is not the case, only the first 2^N points will processed, where N is the largest integer such that the 2^N does not exceed the actual number of points in the signal. The remaining points will simply be ignored.

Test this method on a deterministic Weierstrass function synthesized with the defaults options.

2.4 Oscillation-based estimation

In contrast with previous ones, this method does not assumes that the local and pointwise exponents coincide. It truly tries to estimate the pointwise one. The principle is very simple: it just uses the definition of the exponent as a measure of how the oscillation of the signal in a neighbourhood of a given point t behaves with respect to the size of the neighbourhood. As usual, you first check the **Input data name** box and **Refresh** it if needed. You then need to choose what are the minimal and maximal sizes of the neighbourhood that will be used to investigate the behaviour of the oscillations. Enter the appropriate values in the **Nmin** and **Nmax** boxes, either directly or using the menus. Any integer will do, as long as Nmin is smaller than Nmax and Nmax is compatible with the size of the signal. These values should be understood as follows: Nmax = 8, for instance, means that the largest neighbourhood will be composed of 8 points to each side of the point where one wishes to perform the estimation. In other words, the neighbourhood will be a window of size 17 sample points centered at the point of interest. Increasing the values of both Nmin and Nmax yields smoother estimates, at the expense of precision. A value of Nmin larger than 1 roughly means that a high frequency cut-off is in effect. Larger values of Nmax let you include more low frequency information in the estimation.

Finally, you may choose a **Regression Type** from the usual choice of **Least Square Regression**, **Weighted Least Square**, **Penalized Least Square**, **Maximum Likelihood** and **Lepskii Adaptive Procedure** (see reference (6)). On hitting **Compute**, you get the output *data pht_signal#*, which contains the estimated exponent at each point. Note that refinements of this oscillation based method exist. They use a Bayesian framework that allows to minimize the effect of discretization. We hope to include these extensions in a future release of **Fraclab**.

3 Local Hölder Exponent

The canonical example that highlights the difference between the two exponents is the so-called chirp **(x^a)sin(x^(-b))**, with **a,b** positive: In this case, the pointwise exponent is **a** and the local one is **a/(b+1)**. The local exponent is always smaller than the pointwise one. The situation where both exponents coincide is favorable, since it usually ease the estimation (for instance, it is a necessary condition for the basic GIFS and wavelet estimators to be valid). Such is the case for instance for the Weierstrass function, the generalized Weierstrass function with smooth h(t), the fractional Brownian motion, or the multifractional Brownian motion with smooth H(t). On the contrary, lacunary wavelet sequences such as the ones available in the synthesis menu are examples of signals which have almost everywhere almost surely different exponents. Although a wavelet-based method could easily be developed for estimating local exponents, only a oscillation-based one is implemented in **Fraclab**, as it gives generally better results. The main difference with the estimation of the pointwise exponent is that one does not compare the oscillation with the size of the neighbourhood, but with the distance between the two points where the oscillation is attained.

Again, you first check the **Input data name** box and **Refresh** it if needed. You then need to choose what are the minimal and maximal sizes of the neighbourhood that will be used to investigate the behaviour of the oscillations. Enter the appropriate values in the **Nmin** and **Nmax** boxes, either directly or using the menus. Any integer will do, as long as Nmin is smaller than Nmax and Nmax is compatible with the size of the signal. These values should be understood as follows: Nmax = 8, for instance, means that the largest neighbourhood will be composed of 8 points to each side of the point where one wishes to perform the estimation. In other words, the neighbourhood will be a window of size 17 sample points centered at the point of interest. The same remarks about the choice of values for Nmin and Nmax as in the case of the pointwise exponent apply. Compared to the case of the estimation of the pointwise exponent, there is an additional parameter here, called **Neighbourhood size**. The **Neighbourhood size** parameter mainly acts as a smoothing filter.

Finally, you may choose a **Regression Type** from the usual choice of **Least Square Regression**, **Weighted Least Square**, **Penalized Least Square**, **Maximum Likelihood** and **Lepskii Adaptive Procedure** (see reference (6)). On hitting **Compute**, you get the output data *pht_signal#*, which contains the estimated exponent at each point.

Note finally that, as in the case of the pointwise exponent, more robust estimates of the local exponent are available when one computes the 2-microlocal exponents. See the help corresponding to this topic below.

A known bug in this sub-menu is that choosing a value for Nmin larger than the default of 1 often results in an error. It is not understood yet in which situations this occurs. You might want to restart the whole process, i.e. close the window and open it again, as this often solves the problem.

It is interesting to compare the results obtained by the oscillation-based methods for computing the pointwise and local exponents: One verifies that the latter is always smaller, and that it varies much more smoothly in general.

4 LRD Exponent

As the determination of the long range dependence exponent is usually less difficult and less important in signal processing applications, there is only one method for estimating it in the current implementation of **Fraclab**. Note that LRD is a misnomer, since all this routine does is to give an exponent measuring the (possible) power law behaviour of the Fourier power spectrum at low frequencies. Actual LRD will be present only if this exponent is negative. This behaviour is assessed by looking at the wavelet coefficients at large scales. Everything in this sub-menu is so similar to the sub-menu for the CWT-based estimation of the pointwise Hölder exponent that we refer the reader to the corresponding help.

5 Stable Processes

This routines allows to estimate the four relevant parameters in case your signal is a stable Levy process, i.e. a process with i.i.d. increments having stable marginal law. More details about stable processes and the parameters that characterize them are available in the **Synthesis** menu. Two well-known estimation methods are implemented (see (5)), the Mac Culloch and Koutrouvelis ones.

5.1 Mac Culloch Method

Check as usual your **Input** signal, and just hit **Compute**. The estimated values of the **Characteristic Exponent**, the **Skewness**, **Location** and **Scale** parameter will appear, along with estimates of their standard deviation (in the column **std**). Be careful that if you want to estimate the parameters of the stable motion X, you should input here the signal Y which contains the increments of X. In other words, this procedure estimates the parameters of the process, the increments of which are the input signal.

5.2 Koutrouvelis Method

This is exactly the same as above, except this time no estimates are available for the standard deviations. The same remark about the increments applies.

6 2-microlocal Exponent

2-microlocal analysis extends the usual regularity analysis based on Hölder exponents by associating to each point in the signal an infinite number of exponents. It is not possible to describe the details in this help (see reference (3)). We will only mention that 2-microlocal analysis defines, at each point t, a "2-microlocal frontier", which is a concave decreasing curve in an abstract plane whose coordinates are denoted (s, s'). The intersection of the frontier with the s-axis is precisely the local Hölder exponent, while, under mild assumptions, the intersection with the second bisector is the pointwise exponent. In addition, 2-microlocal analysis describes completely what happens under integro-differentiation of the signal: the frontier is simply translated along the s-axis. This allows to predict the changes in regularity under various transformations,

most notably the Hilbert transform. Classical definitions of 2-microlocal analysis involve either a Littlewood-Paley analysis, or a wavelet analysis. There also exist purely time domain definitions, which however only yield a sub-part of the 2-microlocal frontier. These time domain definitions are advantageous because they allow to design robust estimation procedures. These in turn yield estimators of both the pointwise and local exponents, which are in many cases better than direct methods. Two such methods are currently implemented in **Fraclab**.

6.1 Oscillation-based (1)

Hit first **Refresh** so that the **name** and **size** boxes get updated. The **window** parameter will, as usual, define the number of sample points that will be used for estimating the exponents at any given point t : exactly 0.5 **window** points will be taken into account on each side of t. The **step** parameter lets you decide the spacing between consecutive points were the exponents will be computed. Thus, choosing **step** equal to 1 means that you wish to perform the estimation at each point. More precisely, the routine will compute the estimations at the following points, assuming your signal is (t(1), ..., t(n)): The first point to be considered will be t(0.5 **window** + 1). Then all points of the form t(0.5 **window** + 1 + **step**) will be dealt with, until the index (0.5 **window** + 1 + **step**) exceeds n - 0.5 **window** -1. Let us take the example of the default values which are **window** = **step** = n. In this case, the computations will be performed only at the middle point 0.5n of your input signal, using the whole signal as a window. In general, you will want to choose a large value for **window**, i.e. at least 100, and preferably 200. Note however that a large window size implies long computing times (for **window** = **step** = n = 256, you should wait about one minute).The actual number of points where the estimation will be performed will then depend on both the **step** parameter (choosing **step** not smaller than n implies that only one point will be considered) and the length n of your signal.

You then need to choose what to **estimate**: The default is **frontier**. When you hit **Compute**, you will, if you are patient enough, get three output signals: *Gfrd_sig#* is a graph structure containing the abscissa and ordinates in the (s, s') plane of the 2 microlocal frontier. This is the signal you may want to visualize. The second output, *frd_sig#*, is the plain frontier. Finally, *lend_sig#* is a graph that lets you check the quality of the estimation. Briefly, the method works by locating the abscissa s of the plateau with maximal length in a certain graph. In order for the procedure to be robust, there should be a well defined maximal length plateau in *lend_sig#*. Since there is a different graph for each value of s', *lend_sig#* will contain several piecewise constant lines, and the estimation will work well if each of these lines has a well defined maximal length plateau. You may check this by viewing this output (see (7) for more on this algorithm). In general, the frontier *frd_sig#*, being an estimated one, will not be concave and decreasing as predicted by the theory. The **Make the frontier convex** part of the menu allows to force this, and thus to obtain more meaningful results. Select *frd_sig#* in the Variables list, then hit **Refresh** (in the **Make the frontier convex** part of the window) so that the **name** of the input signal becomes *frd_sig#*. You may then adjust the **s' cut-off** : this is the ordinate of the first point that will be processed. More precisely, only the part of the estimated frontier whose points have coordinates (s, s') with s' *smaller* than the chosen cut-off will be considered and convexified. As a consequence, the output obtained on pressing **Convexify**, called **conv_frd_sig#**, will be the convex envelope of the sub-part of the frontier with s' < cut-off. The default cut-off of 0 is usually a relevant choice, because the estimation method only guarantees to have meaningful results for the part of the frontier between the s axis and the second bissector s' = -s. Moreover, the two main values we are interested in are the intersection of the frontier with the s axis (local Hölder exponent) and with the second

bisector (pointwise exponent).

If you have chosen to estimate a frontier, it generally makes sense to perform the estimation at one or only a few points. Otherwise, the outputs will be quite messy, with different lines crossing in a complicated manner. However, all the computations, including the convexification, work in the same way in the case of multiple frontiers. Of course, the *lend_sig#* output becomes very difficult to read.

Instead of estimating a frontier, you may estimate the **local** or **pointwise** exponent. In that view, click on the default **frontier** to the right of **estimate**, and choose either **local** or **pointwise**. **Fraclab** will then estimate the part of the frontier around the s axis and the second bisector, and will extract the values of the local and pointwise exponents for you. The output will then be the graph of the chosen exponent, or, more precisely, a 1D signal giving the estimated exponents at the points where the computations were performed. In these cases, you will in general want to estimate with a **step** size of 1 on order to get the exponents at each point whose index lies between : 0.5 **window** + 1 and : n - 0.5 **window** - 1. For both **local** or **pointwise**, the lower part of the window (the **Make the frontier convex** part) is grayed out. You just need to click the **Compute** button, and you'll get two outputs : *Gloc_sig#* and *lenlocd_sig#* for the local exponent, and *Gpt_sig#* and *lenptd_sig#* for the pointwise one. *Gloc_sig#* and *Gpt_sig#* contain the estimated Hölder exponents, while *lenlocd_sig#* and *lenptd_sig#* are again plots of the plateaus, which allow to check that a meaningful maximal plateau was detected.

6.2 Oscillation-based (2)

For this routine, we have chosen to have two separate sub-menus, one for the estimation at one point and one for the estimation on an interval. This is mainly for interface presentation purpose, as the algorithms are the same.

6.3 One point estimation

Hit first **Refresh** so that the **Input Signal** gets updated. In the part **2-microlocal frontier estimation**, choose the **Point to analyze** by entering its index in the box facing it. The default is the point right at the middle of your signal. You may then choose the **Discretization** of the frontier, i.e. the number of values s' at which the corresponding s will be estimated. Be warned that this method can be quite lengthy, and that its time complexity is linear in the number of values of s'. The remaining part of the window deals with the type and presentation of the **Results**. First, decide if you wish to see an **Example of estimation** in the **Graphs to be plotted** part. As in the previous method, the algorithm works by detecting significant straight lines in a certain abstract space. Choosing **Yes** will let **Fraclab** display a graph in which you may want to check that a significant linear part is present (this linear part is materialized by a blue line, see reference (8) for details). Then, you need to decide if you want the algorithm to compute the local and pointwise exponents. In the **Regularity exponents** part, choose **Yes** or **No** to the right of **Compute the exponents**. If you choose **No**, you will only get the frontier, a signal called *Graph_frt_sig#*, which will be both saved in the **Variables** list and displayed on the screen (another signal is sent to the **Variables** list, *frt_sig#*, which contains only the s-values in the frontier). *Graph_frt_sig#* is really a graph structure: *Graph_frt_sig#.data1* contains the s-values of the frontier, while *Graph_frt_sig#.data2* contains the s'-values). On the display, the frontier appears as a set of small triangles connected by straight segments. In addition the two lines s' = -s and s' = 1 -s are plotted. This is because this estimation procedure is only

valid for the part of the frontier between these two lines. In addition, displaying the second bisector helps locating the pointwise Hölder exponent.

If you choose **Yes** for **Compute the exponents**, you'll get in addition the estimated values of the **Pointwise exponent**, as obtained with two slightly different methods, and of the **Local exponent**, which will appear in the boxes in front of the corresponding labels. Also, these values will be saved in the **Variables** list, under the names *pw1Exp_sig#*, *pw2Exp_sig#*, and *locExp_sig#* (these are 1x1 matrices). In this sub-menu, only one point is processed, and you get one value for each exponent.

Finally, since some computations are lengthy, you may decide to see the **Waitbar on screen** by choosing **Yes** in the lower part of the window.

6.4 Interval estimation

Basically, this is the same as above, except now you may perform estimation as several consecutive points of your original signal. The only difference are:

- In the **2-microlocal frontier estimation** part, you choose the **Starting point of the analysis** and **Ending point of the analysis** instead of the **Point to analyze**. By defaults, these are the points n/2 -10 and n/2 +10 if n is the length of your input signal, i.e. 21 points will be analyzed in the middle of the signal.
- The sub-part **Regularity exponents** is now titled **Plotting the Regularity exponents**. Because you will get three exponents at each point (two different estimates of the pointwise exponent, plus a local exponent), the routine will not display numerical values as in the case of the **One point estimation** routine. Instead, it will save the Hölder functions in the **Variables** list, and will also display them on the screen.

When the computations are over, three graphic windows appear on the screen: the two first ones display the pointwise exponents as estimated with two slightly different procedures, and the third one display the local Hölder function. In abscissa, you can check that the exponents were estimated between the **Starting point of the analysis** and **Ending point of the analysis**. In addition, four signals have been added to the **Variables** list: *Frt_sig#* are the estimated frontiers are each analyzed point. *Pw1Exp_sig#* and *Pw2Exp_sig#* are the pointwise Hölder functions, and *LocExp_sig#* is the estimated local one.

6.5 Known bugs

In the first oscillation-based method for estimating 2 microlocal exponents, the # in the *lend_sig#* output does not increment. Thus you need to be careful and rename this signal if you plan to use it afterwards. Also, the currently highlighted signal in the Variables window is *lend_sig#* after the computation is completed. Thus you'll find that your newly estimated frontiers are at the bottom of the list as usual, while your selection will be located above them.

In the second oscillation-based method for estimating 2 microlocal exponents, the # in all output signals does not always increment. The same remarks apply. In addition, in the case of estimation at multiple points, only the s-values are saved in the frontiers. This is not very convenient. However, you will generally

want to estimate frontiers at single points. Otherwise, use the first method **Oscillation based (1)** that does not have this drawback.

7 Homework

We have already used some of the routines of the estimation menu in the homework section of the **Overview and main functionalities** of this help.

We will here compare the various methods for estimating the local and pointwise exponent *functions* (i.e. the exponents at all points of the signal) on a test signal, namely a generalized Weierstrass function with linearly increasing regularity. In that view, first synthesize this signal: Go to **Synthesis/Functions/Deterministic/Generalized Weierstrass**. Choose **h(t)=t** as a **Hölder Function**. Set the **Sample size** to 1024, leave **lambda** unchanged, and set the **Sum Order** to 30. Hit **Compute** to obtain the signal *Gwei0*. **View** it in the usual way, and notice how the regularity visibly increases from left to right.

The signal *GWei0* has, at all point t in (0,1), both local and pointwise exponent equal to t. We shall see that estimating the regularity exponents is not an easy task in general, and that some methods give an acceptable result on this particular signal. We shall first test the CWT method: Go to **1D Exponents Estimation/Pointwise Holder Exponent/CWT based estimation**. In the window that appears, click on **Advanced compute**. Now compute the CWT of *GWei0* as explained above: Check first that the **Input Signal** is indeed *GWei0*, otherwise **Refresh** it in the usual way. Keep the default parameters for **fmin**, **fmax**, **Voices**, **Mirror**, **Size** and **Type** of the analyzing wavelet. Hit **Compute WT**. The wavelet transform *cwt_ Wei00* should appear in the **Input CWT** box. Uncheck the box **Single Time Exponent**, so that it becomes **Hölder Function**. Leave the other parameters unchanged and hit **Compute**. After a short while, you'll get a new signal *GWei0_Ht0*, that contains the estimated pointwise exponent. **View** this signal. As you'll see, this is very far from being the expected y=x line. In order to understand why we are so much off, we shall now try and estimate the exponent at a single point. In that view, click on **Hölder Function** which now turns to **Single Time Exponent**. In the **Time instant** box, enter, e.g., 300. Leave the other parameters unchanged, and hit **Compute**. A graphic window appears: On its left part you can see the points in the scale/space plane that were selected as local maxima. On the right part is displayed a log-log plot of the magnitude of the wavelet coefficients at these points with respect to scale. The slope, currently indicated by the red line, is supposed to give the Hölder exponent. With the cross that appears when you point to this window, select a regression range (by default, this is the whole available range). Selecting the range between points 4 and 7, you'll get an estimated exponent of around 18. Between points 2 and 4, you'll get something like -1. As a matter of fact, you can obtain almost any exponent by clicking in an adequate region. The problem here is that the exponent is governed by the extremal values of the wavelet coefficients. A robust method should then compute the regression only considering those extremal values. Unfortunately, it is not easy to detect which values are indeed extremal.

Once you're finished with testing various regression ranges, hit **return** on your keyboard and close this window. We shall now try the GIFS based estimation. In the **Variables** list of the main window, select *GWei0*, and go to **1D Exponents Estimation/Pointwise Holder Exponent/GIFS based estimation**. Verify that the **Input** is what you want, and hit **Compute** with the default parameters. The estimation, called *alphagifs_ GWei00*, quickly appears in the **Variables** list. On viewing it, you'll see that the general

trend is what we were expecting, with a number of small oscillations around it. The slope is somewhat smaller than 1, but we do get the impression of a signal with steadily increasing regularity.

Our third method is the oscillation based one. Again, select *GWei0* in the **Variables** list, and go to **1D Exponents Estimation/Pointwise Holder Exponent/oscillation based method**. Check your **Input data**, and hit **Compute** with the default parameters. You get the estimation in the signal called *pht_ GWei00*: We obtain again a roughly correct general trend, with more oscillations than in the GIFS based estimation, but with a better slope (you can verify this by viewing both *pht_ GWei00* and *alphagifs_ GWei00* on the same graph, using the **hold** facility in the **View** menu). You can reduce a bit the oscillations by choosing **Nmin**= 4 and **Nmax**= 64

In the fourth experiment, we shall compute the local exponent: select *GWei0* in the **Variables** list, and go to **1D Exponents Estimation/Local Holder Exponent/oscillation based method**. Check your **Input data**. Hit **Compute** with the default parameters. The output is called *pht_ GWei01*. The same remark as above applies, i.e. we get a lot of oscillations with a roughly correct slope. Again, you can minimize the oscillations by choosing **Nmin**= 4 and **Nmax**= 64. It is interesting to check that the local estimation is, at (almost...) all points, a lower bound to the pointwise one, as predicted by the theory.

Finally, we'll see what happens when using a more advanced method based on 2-microlocal analysis. Select *GWei0* in the **Variables** list, and go to **1D Exponents Estimation/2-microlocal Exponent/Oscillation based (1)**. In the window that appears, hit **Refresh** so that the **name** of the Input data becomes *GWei0*. Choose a **window** size of 100 instead of the default 1024. Since the computations are a bit long with this method, we'll perform the estimation only every tenth point. Thus, we set the **step** parameter to 10. Finally, click on **frontier** and select **local** instead. Hit **Compute**. You will probably wait for less than a couple of minutes until the outputs, called *lenlocd_ GWei00* and *Gloc_ GWei00* appear. The estimated local exponent is *Gloc_ GWei00* and this is the one we shall view. As above, we get a curve that increases in the mean between 0 and 0.8, although this time there is a larger number of points where the local exponent is grossly underestimated. We shall also estimate the pointwise exponent with this method: click again on **local** and this time select **pointwise** instead. Hit **Compute**. View the output called *Gpt_ GWei00*. This estimate is a little bit better that the local one, with fewer underestimated values. Recall however that, in both cases here, the fact that we estimate only every tenth exponent make the underestimated areas look more important. If you are very patient, you may check this by launching the estimation with a **step** of 1 instead of 10.

8 References

(1) S. Jaffard, *Lacunary wavelet series*, Annals of Applied Probability (to appear).

(2) J. Lévy Véhel and K. Daoudi, *Generalized IFS for Signal Processing*, IEEE DSP Workshop, Loen, Norway. September 1-4, 1996.

(3) B. Guiheneuf and J. Lévy Véhel, *2-Microlocal Analysis and Application in Signal Processing*, Proceedings of International Wavelets Conference, Tangier, April 1998.

(4) A. Benassi, S. Cohen, and J. Istas, *Identifying the multifractional function of a Gaussian process*, Stat. Proba. Letters , Vol. 39, p. 337–345, 1998.

(5) L. Belkacem, *alpha-SDE and Option Pricing Model*, Fractals in Engineering (J. Lévy Véhel, E. Lutton and C. Tricot Eds.), Springer Verlag, 1997.

(6) C. Canus, *Robust Large Deviation Multifractal Spectrum Estimation*, Proceedings of International Wavelets Conference, Tangier, April 1998.

(7) K. Kolwankar, J. Lévy Véhel, *A time domain characterization of the fine local regularity of signals*, http://www-rocq.inriq.fr/fractales/Publications/

(8) S. Seuret, J. Lévy Véhel, *A time domain characterization of 2 microlocal spaces*, http://www-rocq.inriq.fr/fractales/Publications/

(9)A. Ayache, S. Cohen, J. Lévy Véhel, *The covariance structure of multifractional Brownian motion, with application to long range dependence*, ICASSP, June 2000.