The **Segmentation** pop-up menu

Jacques Lévy Véhel 12 January 2001

This text presents a brief explanation of the functionalities in the **Segmentation** pop-up menu.

Contents

1 Overview

The **Segmentation** menu is currently composed of two items. As their names indicate, the first one, **1D signals WSA based segmentation** deals with 1D signals, as the second one, **Image Multifractal Segmentation** allows to process images. However, the main difference between these two sub-menus lies in the methods they use. The first one is model-based, and relies on a generalization of IFS-s called *Weakly Self Affine* functions or *SGIFS* (note that you may synthesize directly *Weakly Self Affine* functions in the **Synthesis** menu, under **Functions**/**Stochastic**/**SGIFS**). In fact this is really a *modeling* method (as such, it should appear more appropriately in a **Modeling** menu, which does not exist yet), used here for segmentation purposes. The second approach is fully non parametric, and analyses the image through various features of its multifractal spectrum. Although it is perfectly possible to use WSA functions to model and segment images as well as to use multifractal tools for the segmentation of 1D signals, these facilities are not implemented in the current version of **Fraclab**.

2 The 1D signals WSA based segmentation sub-menu

2.1 Description

As said in the introduction, this is really a modeling device, and the application to segmentation is included only as a example. As a consequence, this sub-menu should be viewed mainly as a pedagogical tool. In particular, the limitation on the position of the segmentation points to be described below can be removed with some extra-work.

The basic idea is to model a 1D signal as a *weakly self-affine* function. Such functions are generalizations of self-similar functions where the similarity ratios are allowed to vary at each scale. In other words, a weakly self-affine function is obtained through a cascading scheme as follows: assume f is defined on [0,1], and that its values at the points k/2^n, k=0, .., n are known for a given n. Then the values of f at the dyadic points k/2^(n+1) can be deduced using only two "multipliers", c(0,n) and c(1,n). These numbers allow to compute respectively the left and the right part of the dyadic tree at level n+1. Thus, the values of f at all points of the form k/2^N is completely determined from the 2N values (c(0,k),c(1,k)), k=1..N. The advantage of such a modeling are that it is reasonably compact, and that it allows to keep track of the fractal properties of the signal. Indeed, contrarily to a "classical" modeling based, e.g., on splines, the fractional dimensions, Hölder exponents and multifractal spectra assume non trivial values. Moreover, all these characteristics may be computed in an analytical way from the (c(0,k),c(1,k)), k=1..N.

In general, a given signal will not be well represented by a single weakly self-affine function. A simple procedure is then applied to segment the signal into subparts that will be well approximated. In this procedure, the dyadic tree is recursively divided until each part is closer to its weakly self-affine approximation than a given threshold in the L2 norm. We thus end up with a collection of weakly self-affine functions, defined on a partition of the dyadic tree, which gives a good representation of the original signal. As a consequence, we have obtained a segmentation of our signal into parts which are "multifractally homogeneous", in the sense that each part is obtained through a simple cascading process.

Let us now proceed to describe the various parameters involved. As usual, the first line of the sub-menu window displays the signal currently analyzed, which may be changed by highlighting another selection in the Variables list in the main window of **Fraclab**, and then pressing **Refresh**. Since the procedure that computes the weakly self-affine approximation is based on a wavelet decomposition, you may then choose the **analyzing wavelet**. In many cases, the simplest wavelet, i.e. the Haar one, will yield the best results. The next parameter is the the **Analysis depth** : as is customary in wavelet analysis, one usually does not need to analyze all the levels in the dyadic tree. This parameter lets you choose how many levels you wish to include, starting from the fine structure. The default is 5: Usually, you'll get better modeling/segmentation results by leaving aside the low frequency content of the signal, and process only the last levels. Of course, you can also try to analyze the whole signal, in which case you give the maximum depth compatible with the size of the signal: If it has 2^N points, the depth should be not greater than N-1, otherwise you will get the message "Error : The maximal analysis depth is violated" in the **Message** box of the main window. The next parameter is the **threshold** that fixes the maximal error between a part and its approximation. In other words, if the whole signal is approximated with an error lower than **threshold** by a single weakly self-affine function, no segmentation will take place. Otherwise, two subtrees will be analyzed, and further subdivided until the error becomes small enough. Thus, the smaller **threshold** is, the more segments you will get, with a more faithful representation. The last two parameters, **c_min** and **c_max** control the minimum

and maximum values of the multipliers that are to be included in the analysis. Without entering into too many details, let us say that the multipliers are obtained as ratios of wavelet coefficients at successive levels. Occasionally, such ratios may become very small or very large. In such cases, the representation by weakly self-affine functions is not well adapted, because it assumes that all the multipliers are close to their mean at a given level. Since extreme values of the multipliers do not contribute much to the fractal properties of the representation, it is best to discard them. More precisely, they are not included in the analysis, and they are processed independently. The values **c_min** and **c_max** lets you choose the range of the multipliers that will be used in the analysis. A smaller range means that only a few values will be processed. In this case, you will usually end up with what will appear as a good approximation. This is however a fallacious impression, as only a small fraction of the signal will have been processed. The box **% of the coefficients processed** at the bottom of the sub-menu lets you control how much of the signal has really been analyzed. A percentage of over 70 is significant, and you should try to reach 75%. Trying to include too many coefficients will in general result in poor approximations. To increase the number of processed coefficients, you may try to change either c_min or c_max, and see which gives best results in terms of the compromise between a good approximation and a large enough number of coefficients processed. It is also worth trying to change the analyzing wavelet, since multipliers are just ratios of wavelet coefficients, and their range is heavily dependent on the analyzing wavelet.

Once you hit compute, the **% of the coefficients processed** will be updated, and a new window will pop up displaying the original signal in blue, the weakly self-affine approximation in green and the segmentation marks as red crosses. Three new signals will have been added to the **Variables** list in the main window: *wsamod_synt#*, where # is a incremental number, is just the weakly self-affine signal. *wsamod_newci#* contains the vector of multipliers, which you may want to visualize or use for subsequent processing. In particular, you may use it to compute analytically the dimensions, multifractal spectrum and Hölder exponent of the approximation. Finally, *wsamod#* is a structure used for displaying a graph with appropriate labels and legend.

2.2 Known bugs

The Unix/Linux version of **Fraclab** running Matlab 5.3 will not let you process signals whose length is not exactly 2048 (a weird bug, indeed).

Due to the current implementation based on dyadic wavelets, the segmentation always takes place at dyadic points. Finally, if the input signal has length M, only the first 2^N points will be processed, where N is the largest integer such that 2^N is not larger than M. The remaining part will simply be ignored.

3 The Image Multifractal Segmentation sub-menu

3.1 Description.

The principle of multifractal based image segmentation is the following: it seems intuitively clear that points in an image can be classified according to their Hölder exponent. Let us take the example of points lying on contours. These points often correspond to discontinuities of the grey level map or of its derivative. They thus have in general "low" Hölder regularity. However, the exact value of the exponent will depend on the

characteristics of the image. In addition, the property of being an edge is not purely local, and one needs a global criterion in order to decide that a given point is an edge point: indeed, points lying in textured regions also have in general a low regularity, and one has to find a way to distinguish them from contours. This is where the second component of multifractal analysis enters: since edges are by definition sets of points of dimension one, we shall declare a point to lie on a contour if it has a exponent such that the associated value of the multifractal spectrum is one (see below for more on multifractal spectra). Note that, in addition to this geometrical characterization of edge points, a statistical one is possible: edge points may be defined through their probability of being hit when a pixel is randomly chosen in the image at a given resolution. The link between the geometrical and statistical characterizations is provided by the multifractal formalism. Instead of detecting edges, one can extract more complicated structures using the same principles: Starting again from the Hölder exponents, one can decide to keep those points where the spectrum has any given value. For instance, choosing a value around 1.5 will in general allow to extract very irregular contours. A value close to 2 will correspond either to smooth regions or to textures. The general procedure is the then following: One starts by computing the Hölder exponent at each point. This yields the image of Hölder exponents. The second step is to compute the multifractal spectrum. Finally, one chooses which kind of points to extract, i.e. points on smooth edges, textures, etc..., by specifying the corresponding value of the spectrum. The segmented image is then computed. At each step, several choices are available, that we proceed to describe now.

The first part of the **Multifractal Image Segmentation** menu lets you choose the **Analyzed image** as in any other menu. Second, one may specify a **Reference** image. If no reference image is given, the multifractal analysis will be performed in the usual way. In technical terms, this means that all the analysis is performed with respect to the Lebesgue measure: Exponents are computed by comparing the content of a region with its size, etc... If a reference image R is chosen, then the content of the original image will be compared to that of R. This is called *mutual multifractal analysis*. A major use of this is when one is interested in detecting changes in sequences of images: One compare each image to its predecessor in the sequence. To choose a reference image, proceed in the same way as usual: highlight an image in the **Variables** zone of the main window, and hit **Refresh** on the line of **Reference**. If you change your mind and decide not to perform a mutual analysis, just unmark the box called **use** below the item **Reference image** in the second part of the menu. If you have already computed an image of Hölder exponent that you want to use, highlight it in the main window and hit the **Refresh** button on the line of **Hölder Image**. The same holds if you want to re-use a spectrum already computed: highlight it and hit the **Refresh** button on the line of **Spectrum**. Of course, in the usual case where no Hölder image and/or spectrum exist, you just leave these boxes blank. Each time you compute a new Hölder image or a new spectrum, they will automatically be updated, so that you know which data you are using for segmentation.

The second part of this menu deals with the computation of the **Pointwise Hölder exponent**. You first choose the **Capacity** from a choice of **sum, lpsum, min, max, iso, adaptive iso**. These corresponds to different ways of measuring the content of a given region in the image: by choosing **sum**, you associate to each region the sum of the grey level of the pixels in it. **lpsum** computes the Lp norm, i.e. the 1/p-power of the sum of the p-powers of the individual grey levels (in the current release of **Fraclab**, p is fixed to 2). **min** measure the region content by the minimum of the grey levels of its pixels, and **max** of course by the maximum of the grey levels. Finally, **iso** assigns to a region the cardinal of the largest subset of pixels having the same grey level. For instance, if a region is composed of N pixels all having different grey levels, its iso capacity will be one. If all pixels have the same grey level, the iso capacity is N, etc.. The **adaptive iso** is a

refinement of this taking into account a possible noise in the image. The most important capacities are the sum, max, and iso ones. The choice of one capacity rather than another one should be performed on a trial and error basis. As a general rule, max, min and (adaptive) iso capacities give more robust and meaningful results. In any case, you should experiment with different capacities and look at the result before you decide which one you choose: Different capacities will often highlight different aspects of your image.

Since the Hölder exponent at any point (x,y) will be computed using regressions on windows centered at (x,y) with increasing sizes, you may as usual decide the range of sizes of these windows. **min size** and **max size** lets you choose the minimum and maximal sizes of the involved windows. Finally, recall to check or not the **use** box if you wish to perform a mutual analysis. When you're done, hit **Compute Hölder**, and the image of Hölder exponents, called *hld2dCoef_image#*, will appear both in this menu on the line **Hölder image**, and in the **Variables** list of the main window. You may want to view this image, as it often contains the most interesting segmentation information.

The next step is to compute a multifractal spectrum. You need first to choose which spectrum you want to use, using the list facing the **Spectrum** item. There are three of them. The **Hausdorff** spectrum gives geometrical information pertaining to the dimension of sets of points in the image having a given exponent. Basically, you will compute some sort of box dimensions (not quite in fact) for these sets of points. You thus need to tell **Fraclab** the minimum and maximum sizes of the boxes used to estimate the dimension. Use for this the **min boxes** and **max boxes** facilities. Be warned that excessive **max boxes** sizes (over 64) will result in long computation times. Increasing the **min boxes** yields smoother but less precise spectra. When you hit **Compute spectrum**, a new 1D signal, the spectrum, will be added to the main window, the name of which is of the form *hSpectrum_image_fd2d_alpha#*, where "image" is the name of your original data and # is an incremental number. This spectrum is a curve where abscissa represent all the Hölder exponents that occur in your image, and the ordinate is the dimension of the sets of pixels with a given exponent. This signal also appears on the line **Spectrum** of this menu.

The second spectrum is the **large deviation** spectrum. This spectrum yields a statistical information, related to the probability of finding a point with a given exponent in the image (or more precisely, how this probability behaves under changes of resolution). The computation is based on techniques used in density estimation, and uses a kernel. You need to tell **Fraclab** how the size of the kernel will be chosen, by selecting, in front of the item **adaptation**, one of **maxdev**, **diagonal**, **double kernel** and **manual**. Only **maxdev** and **manual** are implemented in the current version of **Fraclab**. Choosing **maxdev** will make **Fraclab** use an optimal, signal dependent, size computed from some empirical statistical criterion. If you rather want to go for **manual**, then the system will use a fixed value, that has been found to be acceptable in a number of situations. Note that in this simplified version of the spectrum computation, you cannot enter a numerical value in the **manual** mode, contrarily to what happens in the **1D signals Multifractal Spectra** estimation menu.

You then choose the shape of the kernel, by selecting in front of **kernel** one of **gaussian**, **boxcar**, **epanechnikov**, **mollifier** and **triangle** (consult any book on density estimation to know more about these kernels). Finally, you need to decide at which resolution the spectrum will be estimated. High resolution (small value for the parameter **coarse graining resolution**) results in higher accuracy, while low resolution (large value for the parameter **coarse graining resolution**) yields more robust results. When you hit **Compute spectrum**, the 1D signal *gSpectrum_image_fg2d_alpha#* is created, where "image" is the name of your original data and # is an incremental number. This spectrum is a curve where abscissa represent all the Hölder exponents that occur in your image, and the ordinate is some sort of probability of hitting a pixel

with a given exponent.

The third spectrum is called the **Legendre** spectrum. It is just a concave approximation to the large deviation spectrum. Its main interest is that it usually yields much more robust estimates, though at the expense of a loss of information. No further parameters are necessary for computing this spectrum. The output is called *lSpectrum_image_fl2d_alpha#*. Note that the Legendre spectrum always assumes that you are working with the sum capacity. In other words, whatever your choice of the capacity is, **Fraclab** will always compute the Legendre spectrum as though the Hölder exponent had been computed with respect to the sum capacity.

Once you have obtained an image of Hölder exponents and a spectrum, you may proceed to the **Segmentation** part of the menu. The principle is simple: you just need to choose a range of dimensions, between **min dim.** and **max dim.**. Hitting **Compute seg.** will then extract from the original image those points the exponent of which have a corresponding value of spectrum that falls inside this range of dimensions. **Fraclab** will output a binary image, called *seg_image#*, where the extracted points are in white and everything else is black. As said above, choosing approximately [0.75,1.25] will hopefully result in many cases in a reasonable edge detection. Setting the range to, say, [1.5,2], will yield a "fat" binary image (i.e. of dimension 2) containing smooth regions and/or textures.

Finally, the last line of this sub-menu includes the familiar **Compute all** (pressing this will launch the computation of all phases of the segmentation), **Help** and **Close** buttons.

Since this menu contains a whole lot a parameters the influence of which is not always easy to understand, every effort has been made to choose default values which give reasonable results in a variety of situations. Thus you should not be too much concerned with tuning finely the various parameters. As a rule, only the choice of the capacity and associated minimum and maximum sizes, plus the **min dim.** and **max dim.** values should be extensively investigated.

A last comment: the shape of the spectra for a typical image is very different depending on the capacity: for the sum capacity, it generally has an approximately bell shape. For the max capacity, it looks more like a segment of the form y = 2 - ax, with a > 0, as for the iso one, it would resemble y= ax, again with a >0.

3.2 Known bugs

When you launch the **Multifractal Image Segmentation** menu, the currently highlighted signal in the main window is not selected by default in the **Analyzed** box. Sometimes no signal at all is selected, other times you get previously processed images. You thus need to press **Refresh** to select the signal you wish to process.

The computation of the spectra is quite unstable in this version, specially the large deviation and Legendre ones. You may thus either stick to the Hausdorff spectrum, or use the defaults for the large deviation spectrum, or be prepared to launch several times the computation before **Fraclab** accepts to compute a meaningful spectrum. Sometimes it is a good idea to close the window and start afresh. You may also try and change various parameters such as the kernel shape and the maxdev/manual choice. Usual signs that a problem occurred are a) an error message in your matlab windows, or b) the "NaN" symbols appearing in the **min dim.** and **max dim.** boxes. Recall also that the Legendre spectrum is always computed assuming that the used capacity is the sum one.

4 Homework

An example of the use of the **Image multifractal segmentation** menu has been presented in the general help **Overview and main functionalities of Fraclab**, so we'll concentrate here on the use of WSA functions as a modeling device. We'll see that this tool is versatile enough to adapt to a large variety of signals, as long as they have significant variability and irregularity. As a warm-up, we consider first the modeling/segmentation of synthetic signals. Of course, since WSA functions are generalizations of IFS, it would be cheating to try the algorithm on IFS, GIFS or SGIFS. Instead, let us see what happens for stable motions. Synthesize thus a stable motion with **Characteristic Exponent** equal to 1.3, and the default values for the remaining parameters, except that you fix the sample size to 2048 points (recall that in the current version of **Fraclab**, the WSA-based segmentation tool will clip signals to their closest lower power of 2). Since the selected signal after synthesis is the increments process *stable_increments#*, recall to click on *stable_process#* in the **Variables** list to make it the current signal. Then launch the WSA-based segmentation window, and hit **Compute** again with the default values. Since your input signal is random, I cannot predict exactly what will happen, but chances are that you'll obtain a reasonably correct modeling of your stable process. Typically, you'll get that approximately two thirds of the coefficients have been processed, with a segmentation into four parts (i.e. you'll see 5 red crosses on the graph). In addition, the WSA model (the green curve) will be close to the original signal (in blue) at most points, except in general for a couple of regions where you'll observe a high discrepancy. Now let us refine our segmentation. In that view, choose a smaller **threshold**, say 1 instead of the default 10, and hit **Compute** again. Probably the same proportion of coefficients have been processed, but you have now typically 8 segments with a close agreement except maybe in one zone. You can diminish the threshold again or play with the other parameters to see how they affect the modeling.

To perform a more interesting test, we shall try the WSA modeling tool on the file *satdrum.dat*, that you will find in the DATA directory that comes with **Fraclab**. This is a small excerpt of a recording of a highly saturated drum part. An interesting feature of this kind of numerical effects is that they add a lot of energy in the high frequencies. As a result, the data have a power spectral density that decay very slowly. This is reminiscent of what happens for "fractal signals", and, indeed, pure IFS have power spectral density that decay approximately linearly in log-log coordinates. To check this, synthesize an IFS (**Synthesis**/**Functions**/**Deterministic**/**IFS**) with the default values, except that you require that, e.g., 4096 sample points are generated, just to ease the computation of the FFT. This will generate *ifs_ord_0*. Then type *spectrum(ifs_ord_0)* in the matlab window. Check on the figure that you have the expected behaviour, i.e. an approximately linear behaviour (recall that the axes are bi-logarithmic, and that the two external curves are the 0.95 confidence interval). We shall now compute the spectrum of our drum part: First load it into **Fraclab** in the usual way: Hit **Load** in the main window, then locate the signal called *satdrum.dat*. Click on it so that its name appears in the **Name** box. This is a text file, so you need to select **ASCII** in front of **Load as**. Hit **Load** and **Close** this window. You get a signal called *fsatdrum* in your **Variables** list. To obtain the spectral density, type *spectrum(fsatdrum)* in the matlab window. Notice how the power spectral density is roughly linear in log-log coordinates.

To model *fsatdrum* with a WSA function, open the **Segmentation** pop-up menu, and choose **1D signals WSA based segmentation**. In the window that appears, check that the **Input Signal** is *fsatdrum*, otherwise select it in the **Variables** window and hit **Refresh**. Select **Daubechies 4** as an **Analyzing Wavelet**, 5 as the **Analyzing depth**, and set the **threshold** to 2 (this is not crucial in this experiment).

Finally, choose **cmin = 0.1** and **cmax = 1**. Hit **Compute**. You should get that approximately two thirds of the coefficients are processed, which indicates that the WSA modeling is meaningful. Check in the graphic window that appears that the approximating WSA function, wsamod_synt0, is indeed close to the original signal. Of course, in such a situation, the real test is not visual resemblance but auditive fidelity. We will thus listen to the two signals. In that view, type *wavplay(fsatdrum,44100); pause(1); wavplay(wsamod_synt0)* if you are operating under Windows. If you are running Unix/Linux, replace the command *wavplay* by *soundsc*. This should play the two signals at their correct rate of 44100 Hz, provided your computer have sound capabilities. If all went well, you will not hear any obvious difference between the original and the model. You may also compute the spectrum of *wsamod_synt0*, and check that it indeed looks very much like the one of *fsatdrum*.

As a final test, you should try to model *fsatdrum* with a "classical" method, e.g. using a spline interpolation or any other method you like. To make a fair comparison, the classical model should contain a number of parameters of the order of one third of the number of samples in *fsatdrum*, since this is basically what the WSA model needed (more precisely : one third of the number of samples + 2*log(two thirds of the number of samples)). You'll see that, although you can get a visually good agreement, you can not match easily the spectral content of *fsatdrum* in the high frequencies, nor obtain a faithful auditive reproduction. Using spline interpolation, for instance, you can get a roughly equivalent L2 error with *fsatdrum* as the one achieved by *wsamod_synt0*, but the power spectral density falls off much more rapidly at high frequencies. Also, the sound of the spline interpolation is much softer, loosing quite a lot of the saturation effect. As expected, WSA modeling is thus useful when there is relevant information in the high frequencies.

5 References

(1) K. Daoudi and J. Lévy Véhel, *Signal Representation and Segmentation based on Multifractal Stationarity*, http://www-rocq.inriq.fr/fractales/Publications/

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

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

(4) J. Lévy Véhel, *Numerical Computation of the Large Deviation Multifractal Spectrum*, CFIC, Rome, 1996.

(5) J. Lévy Véhel, *Introduction to the Multifractal Analysis of Images* Fractal Image Encoding and Analysis, Y. Fisher Editor, Springer Verlag, 1996.