The **Synthesis** pop-up menu

Jacques Lévy Véhel 22 June 1998

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

Contents

1 Overview

Two types of signals can be generated : measures (i.e. an array of non negative data that add to one) or functions. Measures are interesting in particular when one needs to take into account the resolution in an explicit way. In each case, you may choose between deterministic or stochastic signal generation. Mostly 1D signals can be synthesized in this version of **Fraclab**, except for 2D (deterministic or stochastic) multinomial measures and 2D stationary increments fractional fields.

We describe below the sub-menus of the **Synthesis** pop-up menu, in the order they appear in **Fraclab**.

2 Functions

2.1 Stochastic

2.1.1 fBm

This functionality allows to generate a fractional Brownian motion using the Cholesky method and a Durbin Levinson algorithm. You may choose **the Hölder exponent** (a real number strictly between 0 and 1), which governs both the pointwise regularity and the shape around 0 of the power spectrum, the **sample size** (use the predefined values for powers of 2 or simply type any positive integer in the white zone), and finally the **random seed** : this is useful when you need to generate the same path several times or if you want to compare the paths of two fBm-s which correspond to the same random event but with different H. The output is a one dimensional vector named *fBm#*, where "#" is incremented by 1 from 0 each time you create a new path.

See references (1) and (2) for more on this topic.

2.1.2 mBm

This functionality allows to generate an approximation of a multifractional Brownian motion. This stochastic process is a generalization of fBm where the parameter H is allowed to vary along the path instead of remaining fixed : this means that H is now a function H(t) instead of being a real number. This is useful if you need to model a process the pointwise regularity of which varies in time, since H(t) will indeed be almost surely the regularity of the mBm at t, under certain smoothness conditions on H. Compared to the fBm generation window, the only difference is that you now want to prescribe a Hölder function H instead of a Hölder exponent ; there are two ways of doing this. Either use one of the two pre-defined choices : **piecewise constant** will generate an mBm with H=0.2 on the first half of the support and H=0.8 on the second half, and **h(t)=t** will generate an mBm with Hölder exponent t at point t. Or define your own H function, in which case you select **user defined** in the menu and simply type your function (in the same way you would do it on a Matlab command line) in the white zone below, after having erased the given example. Any continuously differentiable H function with values between 0 and 1 will yield an mBm with pointwise regularity equal to H(t) at point t almost surely. The output is a vector, called *mBm#*.

Remark : since the Cholesky Levinson algorithm is somewhat slow and this routine generates as many fBm-s as there points in the signal, computing times can get very large for paths with more than a few hundreds

of points (see reference (3)).

Try the choice **h(t)=t** to get a feeling of what it means for a process to have smoothly increasing regularity.

2.1.3 Wavelet based 1/f process

This allows to generate a "1/f" process by specifying its wavelet coefficients : at scale j, the coefficients are iid random variables following a centered Gaussian law with variance 2^j(-2H+1), where H is as usual the Hölder exponent, and coefficients at different scales are independent. You first choose the **Wavelet** type in a list, then the **Hölder exponent** between 0 and 1, the **Sample Size** and finally the **Random Seed**. The output is a vector, *Wfbm#* (reference (4)).

2.1.4 Stable motion

A stable motion is a process with i.i.d increments which follow a stable law. Stable laws are a generalization of the Gaussian, in the sense that they are the only laws that are preserved under convolutions. Except for the Gaussian, stable laws do not have a variance, and there exists a real number alpha in (0,2) such that all moments of order equal to or greater than alpha do not exist. The most well known non Gaussian stable law is the Cauchy law.

In this menu, you can synthesize a stable process by specifying the four parameters characterizing the stable law governing its increments. The **Characteristic Exponent**, alpha, between 0 and 2, controls the thickness of the tail of the process. alpha equal 1 gives the Cauchy law, and alpha lower than or equal to one yields a process without a mean. The **Skewness Parameter** is between -1 (distribution totally skewed to the left) and 1 (distribution totally skewed to the right), and controls the symmetry : the distribution is symmetric (around its location parameter) if and only if the skewness parameter is 0. The **Location Parameter**, which is simply the mean when alpha is greater than 1, is any real number. Finally, the **Scale Parameter** is a positive number related to the "size" of the distribution : multiplying the process by a positive number w results in a multiplication of the scale parameter by w. When alpha equals 2 (Gaussian case), the square of the scale parameter is simply the variance divided by 2 (see reference (5) for more).

Once the computation is over, two signals are generated : the stable process itself, named *stable_process#* and its increments, which are iid realizations of a stable law, *stable_increments#*. Both are vectors.

2.1.5 SGIFS

An SGIFS or semi-generalized Iterated Function System is a set of functions that allows to generate a fractal curve interpolating a finite number of points with control on the box dimension of the graph,and its multifractal spectrum. The graph is generated through an iterative procedure, starting from the interpolation points and applying iteratively the functions of the SGIFS. At level or iteration k, 2^k functions are applied. Within **fltool**, the number of **Interpolation points** is fixed to three. You enter the (x,y) coordinates of these points on the first line. Since the functions of the SGIFS are restricted to be affine in this implementation (although a much wider class of operators may be used in general), and because of graph continuity constraints, each function has only one free parameter, called here the **Contraction factor**. You may enter two numbers in these fields, c1 and c2, which will correspond to the mean of this factor for respectively

functions of odd and even ranks. These means are numbers with absolute value between 0.5 and 1. At each step k (the number of these steps is the parameter **Number of Iterations** on the third line), 2^k contraction factors are generated according to a normal or uniform law, depending on your choice of the **Probability law**, with mean c1 for factors of odd rank and c2 for factors of even rank, and variance j^(-var), where var is the **Variance decrease exponent**, a positive real. To each of these factors corresponds an affine function, which is applied to a part of the currently generated graph according to rules described in the references (6) and (7).

For large values of the **Variance decrease exponent**, large (close to +-1) values of c1 and c2 yield irregular graphs, while values of c1/c2 close to +-0.5 generate "smoother" graphs. Values of the **Variance decrease exponent** close to 0 add variability at small scales and "burstiness" in the graphs.

The number of points of the synthesized graph is 2^(number of iterations) + 1. The outputs consist in *sgifs_ord_#*, a vector containing the synthesized function, *sgifs_ci_#*, a vector containing the randomly chosen contraction factors, and *sgifs_#*, a structure used for internal purposes.

2.1.6 Weierstrass

The Weierstrass function is one of the first instances of a continuous nowhere differentiable function (reference (8)). It is basically a sum of damped sines with increasing frequencies. Its **Hölder exponent** H is the same at each point, and you can enter any value between 0 and 1 for H in the first line. The **Sample size** controls the length of the generated signal. The **lambda** parameter governs the spacing between adjacent frequencies : The n-th frequency is simply lambda^n. In this stochastic version, the damping factor, lambda^(-h*n), of the n-th frequency component is multiplied by a Gaussian random variable with mean 0 and variance 1. In addition a uniformly distributed random phase is added to each sine. The signal will be generated on the interval (0, **time support**). The **Sum order** is the number of terms effectively computed in the sum of sines (which theoretically contains infinitely many terms). If you plan to use Fourier tools for the analysis of this signal, it is advised to choose a value such that the synthesized signal will meet the hypotheses of the sampling theorem (this is the **Default**, so that you don't need to perform the calculation yourself). If on the contrary you will operate with time-domain analyzing tools, you'll get better results by choosing a much higher value for the sum order than the default, typically around 20. If you are not sure, go for the higher value, which will give more relevant results in general (see the help on the deterministic Weierstrass for more on this topic). Thus, in this particular case, the default should be changed in most cases. The output *Wei#* is a vector containing the synthesized function.

2.1.7 Generalized Weierstrass

This is an easy generalization of the classical Weierstrass function where you can control, under some conditions, the Hölder exponent at each point (reference (7)): You choose the **Hölder function**, either from one of the two pre-defined choices (**piecewise constant** or linear, i,e, **h(t)=t**), or you select **user defined** and type in your own regularity function following the example given in the dialog box. Any continuously differentiable H function with values between 0 and 1 will yield a signal with pointwise regularity equal to H(t) at point t. The other parameters are as in the "Weierstrass Function Synthesis" menu. The output is vector named *Gwei#*.

2.1.8 2D stationary increments

This sub-menu allows to generate a 2D random field with stationary increments. The first parameter is the **Hölder exponent**, which controls as usual the smoothness of the image. The **Matrix Size** parameter lets you choose the size of the field : Values larger than 256 will result in quite long computing times. Finally, the **Structure Fcn** parameter (for structure function) controls the anisotropy : at this time, only the "fBm" structure function is available, yielding an isotropic 2D fractional Brownian motion (reference (9)). The output is a square matrix, called *StInc2D#*.

2.2 Deterministic

2.2.1 IFS

An IFS or Iterated Function System is a set of functions that allows to generate a fractal curve interpolating a finite number of points with control on the box dimension of the graph and its global Hölder regularity (reference (10)). The graph is generated through an iterative procedure, starting from the interpolation points and applying iteratively the functions of the IFS. You first choose any number of **Interpolation points** by giving their (x,y) coordinates. The number of functions in the IFS will be the number N of interpolations points minus 1. Since the functions of the IFS are restricted to be either affine or sine polynomial in this implementation (although a much wider class of operators may be used in general), and because of graph continuity constraints, each function has only one free parameter, called here the **Contraction factor**. You may thus enter N-1 **Contraction factors**, which should be real numbers with absolute value between 1/(N-1) and 1. Large values result in irregular graphs, while small values gives smoother graphs. More precisely, the almost sure Hölder exponent of the graph is the base N-1 logarithm of the product of the contraction factors, divided by -N+1. The **Number of points** gives the length of the graph, and the **IFS type** lets you choose between affine functions and sine polynomials for the IFS. The output consists in *ifs_ ord_ #*, a vector containing the synthesized function, and *ifs_ #*, a structure used for internal purposes.

2.2.2 GIFS

A GIFS or Generalized Iterated Function System is a set of functions that allows to generate a fractal curve interpolating a finite number of points with control on the box dimension of the graph, its multifractal spectrum, and even its pointwise Hölder regularity (reference (7)). The graph is generated through an iterative procedure, starting from the interpolation points and applying iteratively the functions of the GIFS. At level or iteration k, 2^k functions are applied. Within **fltool**, the number of **Interpolation points** is fixed to three. You enter the (x,y) coordinates of these points on the first line. You enter next the **Hölder function**, which will rule the Hölder exponent of the graph at each point. You may use one of the three pre-defined choices : **piecewise constant**, **h(t)=t**, or **|sin(3Pi*t)|**, or define your own regularity function by selecting **user defined** and following the example given in the dialog box. The **Hölder function** should be a continuous function between 0 and 1. The parameter **Number of iterations** controls the length of the synthesized graph : 2^(Number of iterations) points will be generated. The output consists in *prescribedH_ ord_ #*, a vector containing the synthesized function, and *prescribedH_ #*, a structure used for internal purposes.

2.2.3 Weierstrass

The Weierstrass function is one of the first instances of a continuous nowhere differentiable function (reference (8)). It is basically a sum of damped sines with increasing frequencies. Its **Hölder exponent** H is the same at each point, and you can enter any value between 0 and 1 for H in the first line. The **Sample size** controls the length of the generated signal. The **lambda** parameter governs the spacing between adjacent frequencies : the n-th frequency is simply lambda^n. Its damping factor is lambda^(-H*n). The signal will be generated on the interval (0, **time support**). The **Sum order** is the number of terms effectively computed in the sum of sines (which theoretically contains infinitely many terms). If you plan to use Fourier tools for the analysis of this signal, it is advised to choose a value such that the synthesized signal will meet the hypotheses of the sampling theorem (this is the **Default**, so that you don't need to perform the calculation yourself). If on the contrary you will operate with time-domain analyzing tools, you'll get better results by choosing a much higher value for the sum order than the default, typically around 20. If you are not sure, go for the higher value, which will give more relevant results in general. The following simple test tells you why: Generate two Weierstrass functions with exponent H = 0.2, 256 points, a time support of 1, and a large lambda, e.g. 25, with both the default sum order and a sum order of 20. Visualize and compare the two synthesized signal and notice that the first one looks very smooth. This explains why, in this particular case, the default should be changed in most cases.

The output *Wei#* is a vector containing the synthesized function.

2.2.4 Generalized Weierstrass

This is an easy generalization of the classical Weierstrass function where you can control, under some conditions, the Hölder exponent at each point (reference (7)): You choose the **Hölder function**, either from one of the two pre-defined choices (**piecewise constant** or linear, i,e, **h(t)=t)**, or you select **user defined** and type in your own regularity function following the example given in the dialog box. Any continuously differentiable H function with values between 0 and 1 will yield a signal with pointwise regularity equal to H(t) at point t. The other parameters are as in the "Weierstrass Function Synthesis" menu. The output is vector named *Gwei#*.

3 Measures

This sub-menu allows to synthesize various multinomial measures. Such measures serve as a paradigm in multifractal analysis, because they are simple to build and their possess a rich multifractal structure. Moreover, their multifractal spectra are easy to compute (at least in the deterministic case and in certain random situations). Finally, they are examples where the "multifractal formalism" holds, i.e. the Hausdorff, large deviation and Legendre multifractal spectra all coincide (see the help on the **1D Multifractal Spectra Estimation** for some details on the spectra, or references (10) and (11)). In this menu, you can construct both deterministic and random 1D and 2D measures. In addition, when the spectrum is theoretically known, you can let **Fraclab** compute it for you from the analytical formula using the parameters you chose for the measure.

All 1D multinomial measures in this menu are built in the following way: Choose a "base" b, i.e. a integer

larger than 1, and b "weights" m1, ..., mb. These weights are non negative reals which add to 1, and they may be thought of as a probability vector. Starting from the uniform measure on the interval [0,1], construct recursively a sequence of measures by splitting the support into b sub-intervals of same length and distributing the mass unevenly according to the weights. For instance, if b=2, after the first iteration, we have the two intervals [0,1/2], having mass m1, and [1/2, 1], with mass m2. The second iteration yields a measure which attributes mass m1xm1 to [0,1/4], m1xm2 to [1/4,1/2], m1xm2 again to [1/2,3/4], and m2xm2 to [3/4,1], etc... In two dimensions, the principle is the same, except we split the square [0,1]x[0,1] into sub-squares at the first step, and iterate from this. Although the mathematical results about multifractality only hold in the limit of infinite iteration we obviously have to stop at some level in practice: this is the **resolution** parameter, that you may set using the slider or by entering directly a value in the corresponding box. Note that if you choose to large a resolution, you may exceed the capacities of **Fraclab**: in this case, a error message will appear in the **Message** line of the main window. A known bug of this routine is that sometimes the error message does not disappear when you correct your parameter, so that you have to **Erase** it manually, and choose a smaller value for the resolution. To the right of the **resolution** box is the #**number of intervals** box. This is a non-editable zone, which tells you how many intervals you'll get at the end, i.e. how many different values the final measure will assume. This number depends on the **resolution**, the **dimension**, and the **base**. Next, choose to synthesize a **1d** or a **2d** measure by checking the appropriate box in front of **dimension**. Depending on your choice, the output will be a 1D signal or an image. The next parameter is the base, i.e. the number in which you split each interval at each resolution. If you chose **1d**, you need only to decide a value for **base x**, and **base y** is grayed out in this case. Otherwise, you may decide to split the original square into a different number of parts in the x and y dimensions. With the default values of 7 for the **resolution** and 3 for the **base x**, in the 1D case, **Fraclab** will iterate three times the process of splitting each interval into three sub-parts, yielding a total number of intervals of 3^7 = 2187 for the synthesized measure. The last line of the upper part of the menu lets you choose the weights vector. In 1D, just enter values in the following format: [m1 m2 m3] (if the base is 3), and in 2D, type [m1 m2;m3 m4] when **base x** = **base y** = 2. Note that, though the sum of all weights must equal 1, you may set some of them to 0: in this case, the resulting measure will be supported on a Cantor set. In particular, you'll get a signal/image where many areas are 0.

The middle-part sub-window lets you decide if you want to generate a deterministic or random measure. If you choose **deterministic**, everything will happen exactly as described in the paragraph above. If you check **stochastic** instead, you get a randomized version of the construction explained above. When **deterministic** is checked, all the items below **stochastic** are grayed out. Now if you want to go for **stochastic**, you have several choices: if you check **micro-canonical**, the weights will simply be "shuffled" at all resolutions in an independent way, as indicated by the fact that the **Shuffled** box becomes active when you check **micro-canonical**. In other words, assuming for simplicity that we are working in one dimension with a base 2, each interval be be split into two halves, and each half will get m1 of the mass of the father interval with probability 1/2, and m2 with same probability. The other choice that appears when you click on **Shuffled**, i.e. **Stratified**, is not implemented in this version of **Fraclab**. If you check **canonical** instead of **micro-canonical**, the **Shuffled** box becomes grayed out, and the **perturbated** box gets active. If you synthesize the measure with these choices, at each step, the **perturbation**, that you may choose using the slider or by entering directly a numerical value, will be randomly added to some weights and subtracted from others so that, in the mean, the mass remains constant. Note that, since the weights must remain non negative, you are not allowed to enter a value for the **perturbation** larger than the smaller weight. If you do so, you'll get an error in the usual **Message** zone. Instead of **perturbated**, you may choose **uniform**. In this case,

a random number, drawn uniformly from the interval (**-perturbation**, +**perturbation**) will be added to each weight at each iteration for all intervals. Again, and for the same reasons, you are not allowed to enter a value for the **perturbation** larger than the smaller weight. Finally, you may choose **lognormal** instead of **perturbated**. In this case, the **perturbation** box becomes grayed out, and the **standard deviation** ones gets active. You may choose any real between *1e-05* and *5.0* for the **standard deviation**. A random variable X with log(X) following a normal law with mean 0 and standard deviation **standard deviation** will be added to each weight.

The bottom part of the menu allows you to compute the multifractal spectrum of the measure you have defined. Since we are talking here of analytic computation, as opposed to estimation, you'll obtain the theoretical spectrum when the formula is available. To let **Fraclab** compute the spectrum, check the box to the right of **theoretical spectrum**. This is available when you synthesize a deterministic measure, or a random one with the options **canonical** and **uniform** or **lognormal**. Note that, on some occasions, the **theoretical spectrum** box will be grayed out although you are in one of the cases above. In this case, just select again **uniform** or **lognormal**, and the box will become checkable.

Once you hit compute, your synthesized measure will be the output signal or image called *mu_n#* in the **Variables** list. If you did compute the theoretical spectrum, it will appear as *theof#*. A known bug is that the # in the *theof#* does not always increment, so that you need to be careful and save the spectrum if you need to re-use it. Also, since the spectrum is the last computed signal, it will be the highlighted one in the **Variables** list. However, since the name *theof#* was already present in this list because the # did not increment, you'll find that the selected signal is not the last one of the list, and your newly generated measure is below it. A final bug is that, on occasions, **Fraclab** will compute a void spectrum or no spectrum at all, although you requested it. Just close the multifractal measures synthesis window and try again afresh: With a bit of luck, it should work.

4 Homework

There is not much of a homework for this section of the help. You may just play around and synthesize the various signals, testing how the parameters will affect the outputs. Here are however some combinations worth trying:

Weierstrass function

If you have not done so already, try the test mentioned in the description of the **Weierstrass** sub-menu. This experiment highlights some specific difficulties one meets when dealing with fractal signals. These are due to the fact that such signals are by definition not band-limited. The classical sampling theory specifies that the signals should then be low-pass filtered before processing. This is not always relevant or even possible in our case. As a matter of fact, a whole new sampling theory is needed for the kind of signals of interest in multifractal analysis. Indeed, the basic assumption is that (a part of) the pertinent information in the signal lies in its regularity structure, e.g. its Hölder function. Low-pass filtering the signal will transform it into an infinitely smooth one, thus loosing essential features. It is beyond our scope here to discuss further this important matter, and we will content ourselves with a simple experiment: Generate two deterministic Weierstrass functions with exponent H = 0.2, 256 points, a time support of 1, and a large lambda, e.g. 25, with both the default sum order and a sum order of 20. Visualize and compare the two synthesized signals and notice that the first one (let's call it *Wei0*) looks very smooth. This is clear, since the highest frequency

allowed in *Wei0* will be such that only one term will be included in the summation: *Wei0* is thus simply a sine function, and as a consequence it has no fractal features. For instance,it would not make sense to try and estimate a fractional dimension of *Wei0*. On the other hand, the signal generated without caring about the sampling theorem does look irregular. This simple experiment shows that new rules are needed when one processes "fractal" signals.

mBm and generalized Weierstrass function

If your machine is sufficiently powerful, try synthesizing some mBm-s with 512 samples and various Hölder functions. This will help you understand what exactly the Hölder exponent controls. If synthesizing 512 samples traces of mBm takes ages, do not worry: You can perform the same kind of experiments using Generalized Weierstrass functions, either deterministic or stochastic. Try first the default Hölder functions $\mathbf{h(t) = t}$. Observe the smooth evolution of the pointwise regularity along the graph, and get a feeling of what it means to have regularity t at each point t. Enter then your own functions and make additional tests.

5 References

(1) T. Lundahl , W.J. Ohley, S.M. Kay, R. Siffert, *Fractional Brownian motion: A Maximum Likelihood Estimator and Its Application to Image Texture*, IEEE Transactions on medical imaging, vol MI-5,3,pp 152-161,September,1986.

(2) K.J. Falconer, J. Lévy Véhel, *Horizons of fractional Brownian surfaces*, Proc. of the Royal Math. Soc. To appear.

(3) R. Peltier, J. Lévy Véhel, *Multifractional Brownian Motion : definition and preliminary results*, Inria Research Report 2645

(4) G. Wornell, *Wavelet-Based Representation for the 1/f Family of Fractal Processes*, Proc. IEEE, Vol 81, pp 1428-1450, Oct. 1993

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

(7) K. Daoudi, J. Lévy Véhel, Y. Meyer, *Construction of continuous functions with prescribed local regularity*, Constructive Approximation, 014(03), pp. 349-385, 1998.

(8) G.H. Hardy, *On Weierstrass' Non-Differentiable Function*, Trans. Am. Math. Soc., 17:301-325, 1916.

(9) B. Pesquet Popescu, P Larzabal, *2D Self Similar processes with Stationary Fractional Increments*, Fractals in Engineering (J. Lévy Véhel, E. Lutton and C. Tricot Eds.), Springer Verlag, 1997.

(10) M. Barnsley, *Fractal Functions and Interpolation*, Constructive Approximation, 1985.

(11) B.B. Mandelbrot, *A class of multinomial multifractal measures with negative (latent) values for the dimension*, Fractals physical origin and properties, Proc. Erice meeting, L. Pietronero, Ed., Plenum Press, 3-29, 1989.

(12) G. Brown, G. Michon and J. Peyrière, *On the multifractal analysis of measures* , J. Stat. Phys. T.66, pp.775-790, 1992.