Journal of Statistical Software

December 2005, Volume 14, Issue 18. *http://www.jstatsoft.org/*

FracSim: An R Package to Simulate Multifractional Lévy Motions

Sébastien Déjean
Université Paul Sabatier

Serge Cohen
Université Paul Sabatier

Abstract

In this article a procedure is proposed to simulate fractional fields, which are non Gaussian counterpart of the fractional Brownian motion. These fields, called real harmonizable (multi)fractional Lévy motions, allow fixing the Hölder exponent at each point. **FracSim** is an R package developed in R and C language. Parallel computers have been also used.

Keywords: fractional motions, simulation.

1. Introduction

Irregular phenomena rise up in various fields of scientific research: fluid mechanics, image processing and financial mathematics for example. From a mathematical point of view, one can measure irregularity by the Hölder exponent, which is a real number $0 < H < 1$. The lower H, the more irregular the function, whereas $H = 1$ corresponds heuristically to differentiable functions, which are considered smooth in this field. Of course, there exist many examples of Hölder functions in mathematical literature. But, non specific Hölder functions are often given by the sample path of a stochastic process.

One of the simplest models is the fractional Brownian motion B_H, in short FBM, introduced in Kolmogorov (1940) and further developed in Mandelbrot and Van Ness (1968). Then, the multifractional Brownian motion, in short MBM, has been introduced independently in Peltier and Lévy Véhel (1996) and Benassi, Jaffard, and Roux (1997) as a generalization of the FBM. The FBM and the MBM provide very powerful models in applied mathematics. Whereas the pointwise Hölder exponent of the FBM is almost surely equal to a constant, the MBM one is allowed to vary along the trajectory.

On one hand, simulation of fractional Brownian motion is now both theoretically and practically well understood (see Coeurjolly 2000; Bardet, Lang, Oppenheim, Philippe, and Taqqu

2003, for surveys on this problem). On the other hand, methods of generating sample paths of the multifractional Brownian motion are given in Peltier and Lévy Véhel (1996) and in Chan and Wood (1998).

However, it is a well known fact that in some fields of applications data do not fit Gaussian models, see for instance Mallat (1989), Simoncelli (1999), and Vidakovic (1999) for image modelling. More recently other processes that are neither Gaussian nor stable have been proposed to model Internet traffic (Cohen and Taqqu 2004; Kaj and Taqqu 2004).

Real harmonizable fractional Lévy motions, in short RHFLMs, have been defined in Benassi, Cohen, and Istas (2002) in order to obtain non-Gaussian fields that share many properties with the FBM. In particular, their sample paths are locally Hölder and their pointwise Hölder exponent is almost surely equal to a constant. Then in order to allow the pointwise Hölder exponent to vary along the trajectory, the class of real harmonizable multifractional Lévy motions, in short RHMLMs, has been introduced in Lacaux (2002). This class makes up a class of locally asymptotically self-similar fields that includes RHFLMs and the MBM.

The main aim of this paper is to propose an R package, **FracSim**, composed of R (R Development Core Team 2005) and C (Kernighan and Ritchie 1988) functions to simulate RHFLMs and RHMLMs. Such motions are described from theory and practical aspects in a first part. Then, in a second part, we give some illustrations of simulated processes in one or two dimension using **FracSim**. The third part is devoted to implementation of parallel computing that are needed for two-dimensional simulations.

2. (Multi)fractional Lévy motions

2.1. Theory

In this part we recall some theoretical facts explained in Lacaux (2004). A RHMLM X_h with multifractional function h is defined as the stochastic integral:

$$X_h(x) = \int_{\mathbb{R}^d} \frac{\mathrm{e}^{-ix\cdot\xi} - 1}{\|\xi\|^{h(x)+d/2}} M(d\xi),$$

with $M(d\xi)$ a random Lévy measure without Brownian component. See Benassi *et al.* (2002) for a precise definition of random Lévy measures.

In Lacaux (2004), RHMLMs with a Brownian part are considered. Since the Brownian part leads to an independent multifractional Brownian motion for which simulations are known we restrict ourselves to the simulation of the non-Brownian part X_h. The field X_h is an infinitely divisible field and infinitely divisible laws can be represented as a generalized shot noise series. An overview of these representations is given in Rosiński (2001). The control measure, $\nu(dz)$ of the random Lévy measure $M(d\xi)$ is in general a sigma finite measure on the set of complex numbers. If it has finite mass denoted by $\nu(\mathbb{C})$, the generalized shot noise series can be used for simulation purpose. Let us first introduce definitions of random variables used in these series. Let $(Z_n)_{n\geq 1}$, $(U_n)_{n\geq 1}$ and $(R_n)_{n\geq 1}$ be independent sequences of random variables.

- ν is supposed to be a finite measure, and $(Z_n)_{n\geq 1}$ is a sequence of i.i.d. random variables such that
$$\mathcal{L}(Z_n) = \frac{\nu(dz)}{\nu(\mathbb{C})}.$$

- $(U_n)_{n\geq 1}$ is a sequence of i.i.d. random variables such that the law of U_1 is the uniform probability on the unit sphere in $\mathbb{R}^d$.
- R_n is the nth arrival time of a Poisson process with intensity 1.
- $\xi_n = \left(\frac{R_n}{c_d\,\nu(\mathbb{C})}\right)^{1/d} U_n$, where $c_d = \frac{2\pi^{d/2}}{\Gamma(d/2)d}$ is the volume of the unit ball of $\mathbb{R}^d$.

Note that when $d = 1$, U_n is a symmetric Bernoulli random variable.

Let us now recall the theoretical result that allows us to simulate X_h. Next proposition by Lacaux (2004) used in the following is given without proof.

Proposition 2.1. *Let $K \subset \mathbb{R}^d$ be a compact set. Then almost surely, the series*

$$Y_h(x) = 2\sum_{n=1}^{+\infty} \Re\left\{ Z_n \frac{e^{-ix\cdot\xi_n} - 1}{\|\xi_n\|^{h(x)+\frac{d}{2}}} \right\}, \tag{1}$$

where $\Re$ denotes the real part of a complex number, converges uniformly on K and

$$\{X_h(x) : x \in K\} \stackrel{d}{=} \{Y_h(x) : x \in K\}.$$

where $\stackrel{d}{=}$ denotes equality in distribution.

The RHMLM is then approximated by

$$Y_{h,N}(x) = 2\sum_{n=1}^{N} \Re\left\{ Z_n \frac{e^{-ix\cdot\xi_n} - 1}{\|\xi_n\|^{h(x)+\frac{d}{2}}} \right\}. \tag{2}$$

One can find in Lacaux (2004) theoretical rates of convergence for $Y_{h,N}$. Theoretical results are given on any compact set K but the algorithms are written on the interval $[0, 1]$ or on the cube $[0, 1]^2$ for sake of simplicity.

2.2. Simulation procedure

Let us first introduce the formula used for the simulation of one-dimensional RHMLM ($d = 1$). The parameters of the simulation are:

- n: the number of terms in the sum;
- k: the number of discretization points;
- $h(t)$ $(t = 1, \ldots, k)$: the value of the multifractional function at each discretization point;
- $m = \nu(\mathbb{C})$: the mass value, usually fixed at 1. Actually we choose in the simulation to take ν the uniform distribution on the unit circle in $\mathbb{C}$.

If we consider the random variables that are standard to simulate

- θ_n: n-vector generated according to a uniform distribution in $[0, 1]$, $Z_n = e^{i2\pi\theta_n}$;

- U_n: n-vector generated according to a Bernoulli distribution with equiprobability for one-dimensional simulation;
- R_n: n-vector of cumulated sums of samples from exponential distribution with parameter $\lambda = 1$,

and apply (2)

$$Y_{h,n}(t) = \sum_{l=1}^{n} \left(\frac{\cos(2\pi\theta_l) \left[\cos\left(\frac{R_l}{m} t.U_l\right) - 1 \right] + \sin(2\pi\theta_l) \left[\sin\left(\frac{R_l}{m} t.U_l\right) \right]}{R_l^{h(t)+1/2}} \right). \tag{3}$$

When $d = 2$, the formula (3) becomes

$$Y_{h,n}(x,y) = \sum_{l=1}^{n} \left(\frac{\cos(2\pi\theta_l) \left[\cos\left(\left(\frac{R_l}{\pi m}\right)^{\frac{1}{2}} (x.U_l^1 + y.U_l^2) \right) - 1 \right] + \sin(2\pi\theta_l) \sin\left(\left(\frac{R_l}{\pi m}\right)^{\frac{1}{2}} (x.U_l^1 + y.U_l^2) \right)}{R_l^{h(x,y)+1}} \right), \tag{4}$$

where k is the number of discretization points for each axis kx and ky, $c_2 = \pi$, (U_n^1, U_n^2) are uniformly distributed on the unit sphere in $\mathbb{R}^2$.

In practice, the difficulty of the simulation lies on both the number of terms in the sum n and in the calculation of scalar products between k-vectors. In our cases, n may vary between 10^3 and 10^8, and k between 100 and 1000, which may imply until 10^{14} loops for the computations in the extreme case.

3. Using FracSim

FracSim is available from the Comprehensive R Archive Network (CRAN, http://CRAN.R-project.org/ or one of its mirrors). Instructions for package installation are given by typing `help(INSTALL)` or `help(install.packages)` in R.

3.1. Implementation of FracSim

FracSim is a set of R and C functions that allows performing RHFLMs and RHMLMs simulations. Two classes of functions are implemented:

- R functions: `fracsim.1d()` and `fracsim.2d()` perform initialisations thanks to parameters set by the user (n, k and $h(t)_{t=1,\ldots,k}$), call C functions for calculation then get back results for graphical representations.
- C functions: `core_1D.c` and `core_2D.c` perform the tasks in the computation that are time consuming because of the number of loops required.

The R environment is the only user interface. The R procedure calls a C subroutine, whose results are returned to R.

In order to make it easier for the reader not used to R language, we detail the call to functions and the command used to produce graphical output in Appendix A. In the following, R commands are preceded by the prompt symbol `R>`.

3.2. Simulations

One-dimensional fractional motions

To simulate RHFLM, one needs to specify one real parameter $0 < h < 1$ or, equivalently, the multifractional function is set to a constant $h(t) = h, \forall t \in [0, 1]$.

For instance,

```
R> X05 = fracsim.1d(h=0.5,k=1000,n=5000)
```

sets the multifractional function equal to 0.5 on [0,1] and simulates the corresponding process.

To simulate on unequally spaced discretization points, the user can give a vector as input for the k parameter. For instance, to get more details between 0 and 0.2, one may define k as:

```
R> kdiscr = c(seq(from=0,to=0.2,length=800),seq(0.205,1,l=200))
```

So, respectively 800 and 200 points are used to discretise the intervals [0;0.2] and [0.205;1]. And then call the `fracsim.1d` function as: `fracsim.1d(h=0.5,k=kdiscr,n=5000)`.

The result of the function `fracsim.1d` is an R object of class `list`. It contains the following elements:

- the vector `simul.h` of the simulated regularity value;
- the vector `t` of discretization points;
- the k-vector `process` whose elements are the RHFLM value at each discretization point.

Thus to produce the graphs on the Figure 1, one has to call the function `plot` as

```
R> plot(x=X05$t,y=X05$process,type="l")
```

As previously mentioned, we can see on the Figure 1, the greater the regularity, the smoother the sample paths.

One-dimensional multifractional motions

To simulate RHMLMs, one has to give as input for h, either a vector of length k, or a function that will calculate the regularity function at each discretization point. The two following commands give examples of:

- an increasing regularity function from 0.1 to 0.9 (`Hinc`);

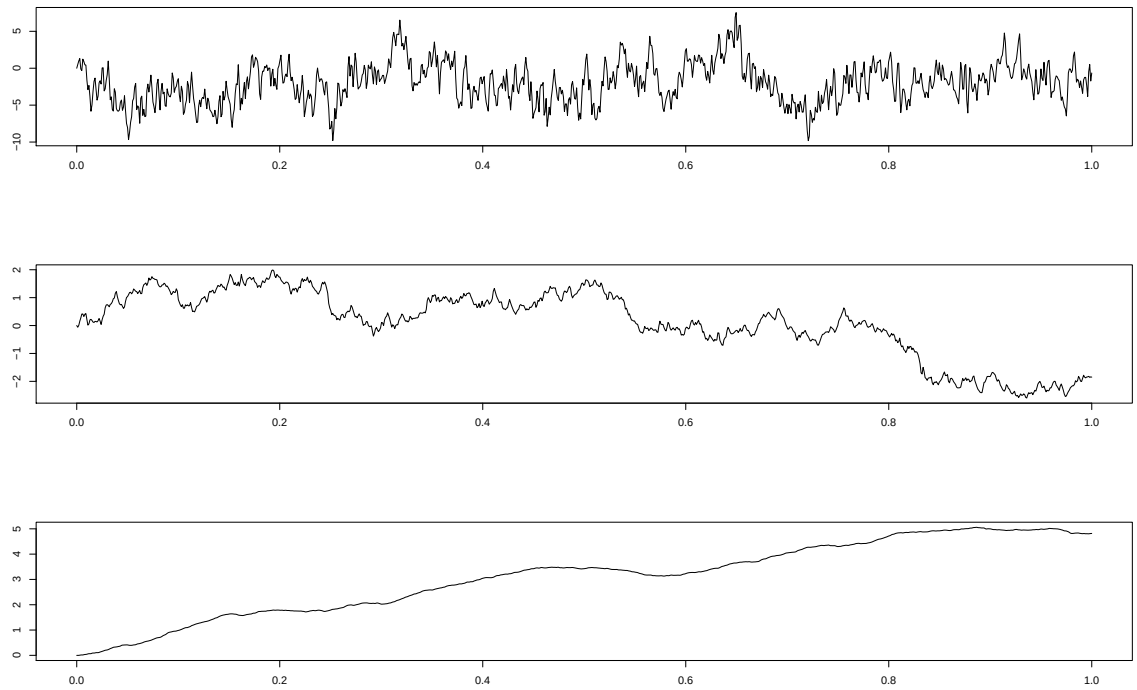

Figure 1: Examples of fractional Lévy motions: from top to bottom the regularity is set to 0.1, 0.5, 0.9.

- a sinusoidal regularity function oscillating according to a linear combination of sinus (`Hsin`);

```
R> Hinc = seq(from=0.1,to=0.9,length=1000)
R> Hsin = 0.25+0.25*sin(seq(0,1,length=1000)*(6*pi))
```

We can then call the function `fracsim.1d()` with the vectors previously defined given as parameter to the argument h:

```
R> Xinc = fracsim.1d(h=Hinc,k=1000,n=5000)
R> Xsin = fracsim.1d(h=Hsin,k=1000,n=5000)
```

An equivalent way of simulating a process according to a sinusoidal regularity function consists in using a function as input for h. Let us define the function `hsin` as below

```
R> hsin = function(a){0.25+0.25*sin(6*pi*a)}
```

The function `fracsim.1d` will calculate the regularity function at each point of the discretization, maybe user-defined as `kdiscr`:

```
R> Xsin2 = fracsim.1d(h=hsin,k=kdiscr,n=5000)
```

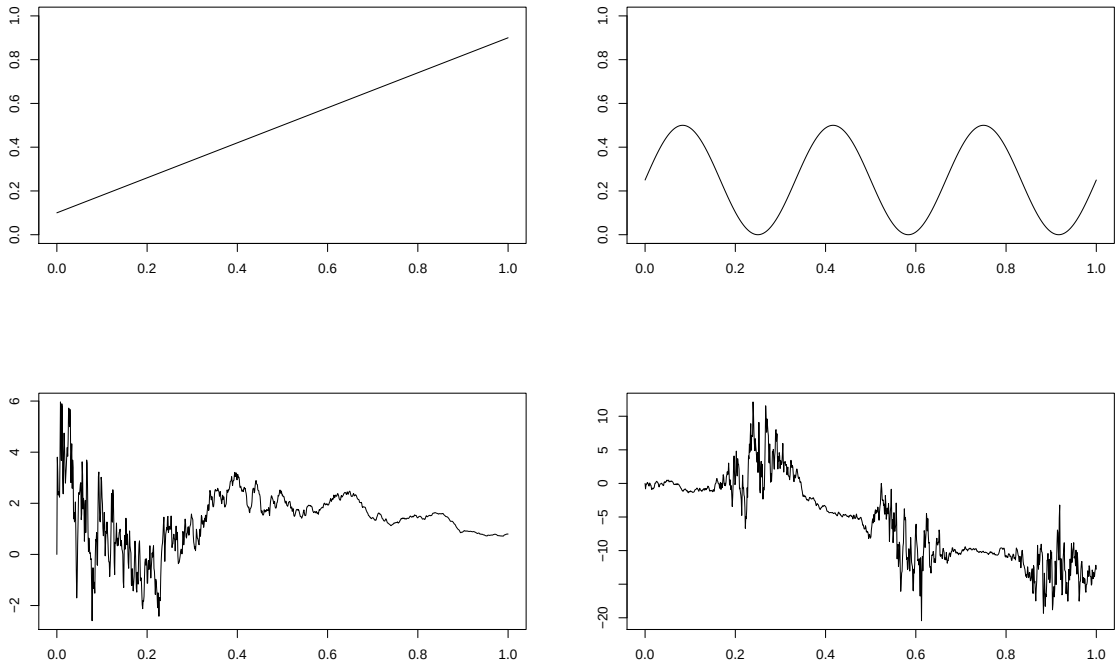

Figure 2: Two examples of multifractional Lévy motions: figures on the top represent the regularity function, bottom, the corresponding process.

To highlight the behaviour of the RHMLM according to the multifractional function, we represent both regularity functions and the corresponding sample paths on the Figure 2.

As previously mentioned for RHFLMs, the greater the regularity, the smoother the trajectory and this can be observed on one graph: for example when we look at the sinusoidal regularity, the corresponding process is much more perturbed when the regularity is close to zero.

Two-dimensional fractional motions

RHFLMs can be simulated by calling the function `fracsim.2d()` with one value given for `h`. For instance,

```
R> X2d05 = fracsim.2d(h=0.5,kx=100,ky=100,n=100000)
```

sets h constant and equal to 0.5 on a 100×100 regular grid with 10^5 terms in the sum. The user can simulate on an irregular grid by specifying `kx` and `ky` as vectors.

The elements of the object `X2d05` provided by the function `fracsim.2d()` are the same as those given by `fracsim.1d()`, except that, in this case, the motion values are now located on a grid and are stored in a matrix object.

To represent two-dimensional motions, we now use the R function `persp` to draw a surface:

```
R> persp(X2d05$process,shade=0.5,phi=30)
```

Arguments `shade` and `phi` are optional; they can be set to give a nice representation: `shade` modify the shade at a surface facet and `phi` is the colatitude angle defining the viewing direction.

Other functions can be used to represent two-dimensional motions, we propose a comparison of various representations at the end of this section.

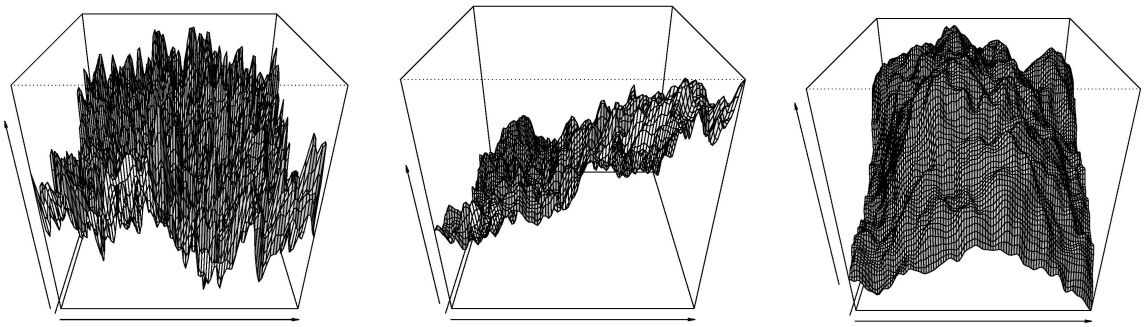

Figure 3: Examples of fractional Lévy motions: from left to right h=0.1, 0.5, 0.9.

Two-dimensional multifractional motions

To simulate two-dimensional RHMLMs, one has to give for `h`, either a matrix which dimensions are `kx` × `ky`, or, as in the one-dimensional case, a function. This function takes two parameters as input and returns a matrix whose elements are calculated on the grid defined by `kx` and `ky` vectors.

For example, we can define two matrices corresponding to the cases presented in one dimension with the following command:

```
R> Hinc = matrix(rep(seq(0,1,l=100),100),nc=100)
R> Hsin = matrix(rep(0.25+0.25*sin(seq(0,1,l=100)*(6*pi)),100),nc=100)
```

One can obtain the graph of the regularity function (Figure 4) as well by calling the function `persp` with the same optional arguments used for the representation of fractional two-dimensional motions.

Once the matrix containing the regularity function at each point of the grid is built, we can perform the simulation of the motion by calling, as in the fractional case, the function `fracsim.2d()`.

```
R> Xinc = fracsim.2d(h=Hinc,kx=100,ky=100,n=100000)
R> Xsin = fracsim.2d(h=Hsin,kx=100,ky=100,n=100000)
```

The function `fracsim.2d` can also be called with a function as input for h. Let us define the function `Hinc.diag`:

```
Hinc.diag = function(x,y){outer(x,y,function(a,b){(a+b)/(2*max(a,b))})}
```

which calculates an increasing regularity function along the diagonal of the grid.

Then `fracsim.2d` can be called as below:

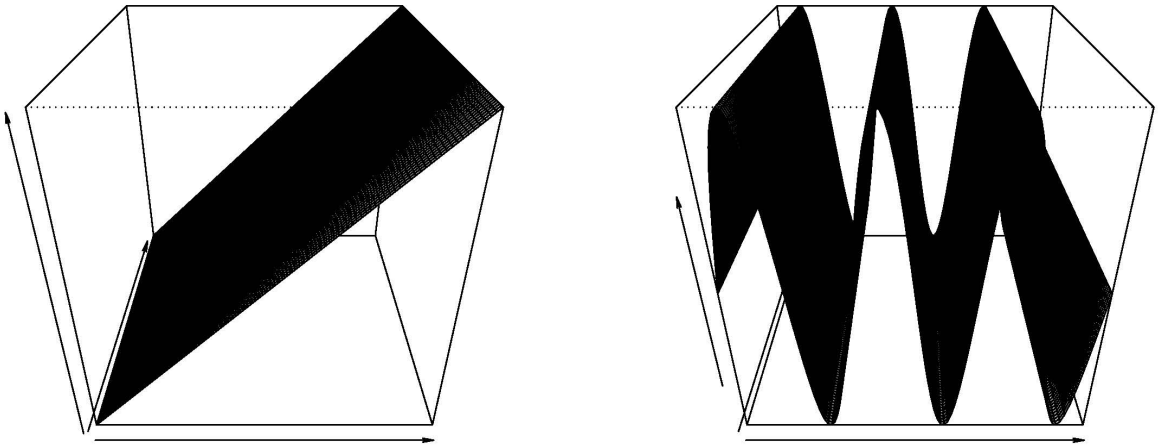

Figure 4: Two examples of regularity function for multifractional Lévy motions: increasing on the left and sinusoidal on the right.

```
Xinc.diag = fracsim.2d(Hinc.diag,n=10000,kx=100)
```

Regularity function and corresponding motion are presented further in the discussion about graphical representations (Figure 7).

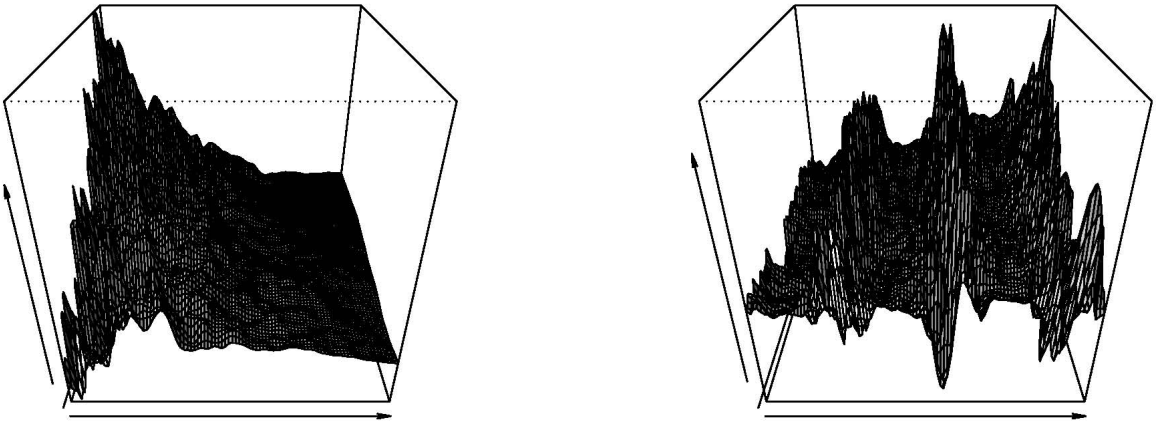

Figure 5: Two examples of multifractional Lévy motions: increasing regularity function on the left and sinusoidal regularity function on the right.

Figure 5 illustrates the behaviour of multifractional process. Considering an increasing regularity (graphic on the left of the Figure 5), the surface looks like a landscape where we go from the mountain with several peaks (low regularity) to the plain (high regularity). With a sinusoidal regularity, the 'landscape' looks like an alternation of moutains and valleys.

Graphical representations

The way we represent a two-dimensional motion has an influence on the visual impression provided. If we only use the `persp` function to draw a surface, it could be interesting to modify point of view by changing `phi` (vertical rotation) and `theta` (horizontal rotation) parameters.

However, we can use other kinds of representations such as images or contour line plots as it is done on the Figure 6.

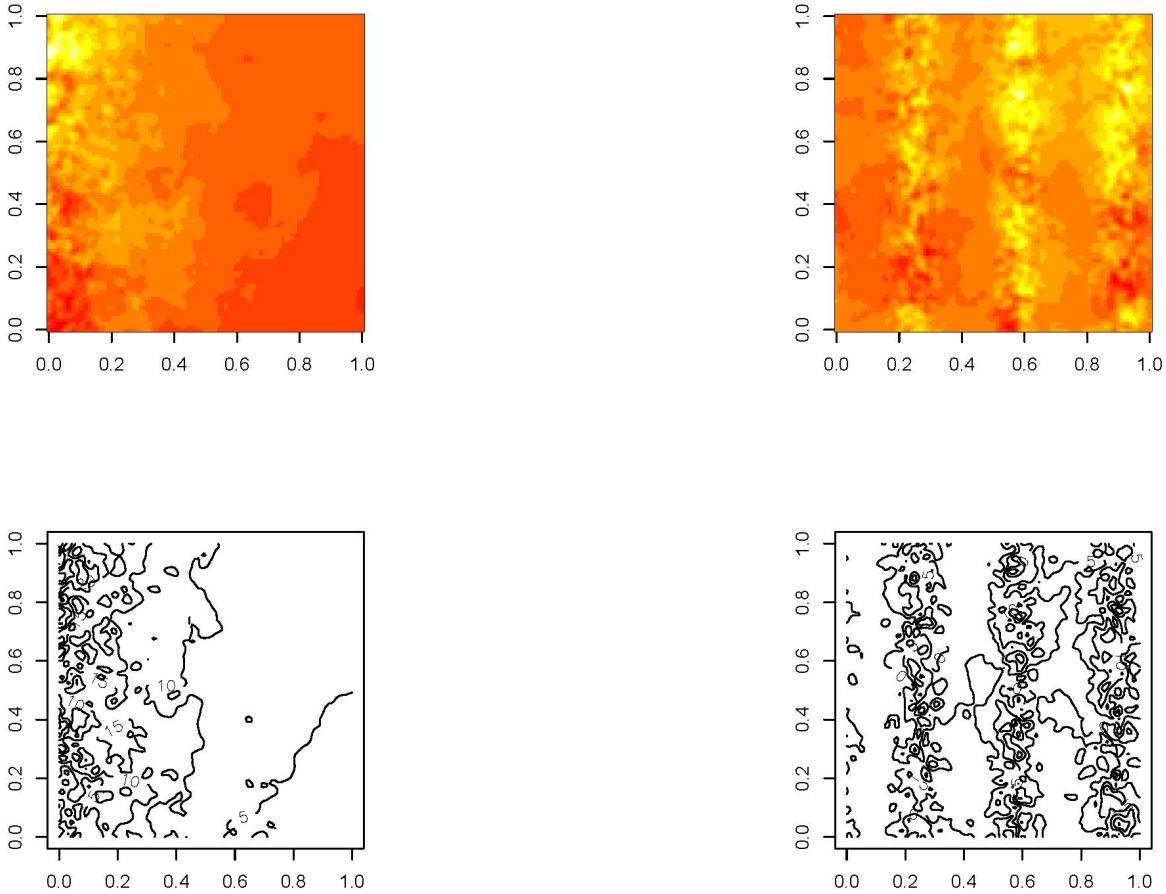

Figure 6: Image and contour representations of the simulations performed in Section 3.2.3 (Figure 5).

Amongst these various representations, it is not easy to decide which is the best; it can depend on the practice of the user. However, let's note that with a high discretization and a regularity close to zero, the visual impression given by the image is smoother than the one given by the sheet (Figure 8, obtained thanks to improvements detailed in Section 4). This can be explained by the inability to distinguish varying intensity in the same color on the image whereas the peaks appear clearly on the sheet. On the other hand, the image of a two-dimensional motion highlights more precisely the global trends of the process, which is more difficult to observe on the sheet. For instance, in the case of a motion with an increasing regularity function along the diagonal, it would be difficult to set the point of view to give a good sheet representation whereas the behaviour of the motion appears clearly on an image (Figure 7): perturbed in the bottom-left corner and smoother in the top-right one.

Durations

For one-dimensional simulations, calculations last about few seconds on a 'current use' machine (SGI Origin 2000, RAM: 3840 MO, processors: 195 MHz - Technical informations

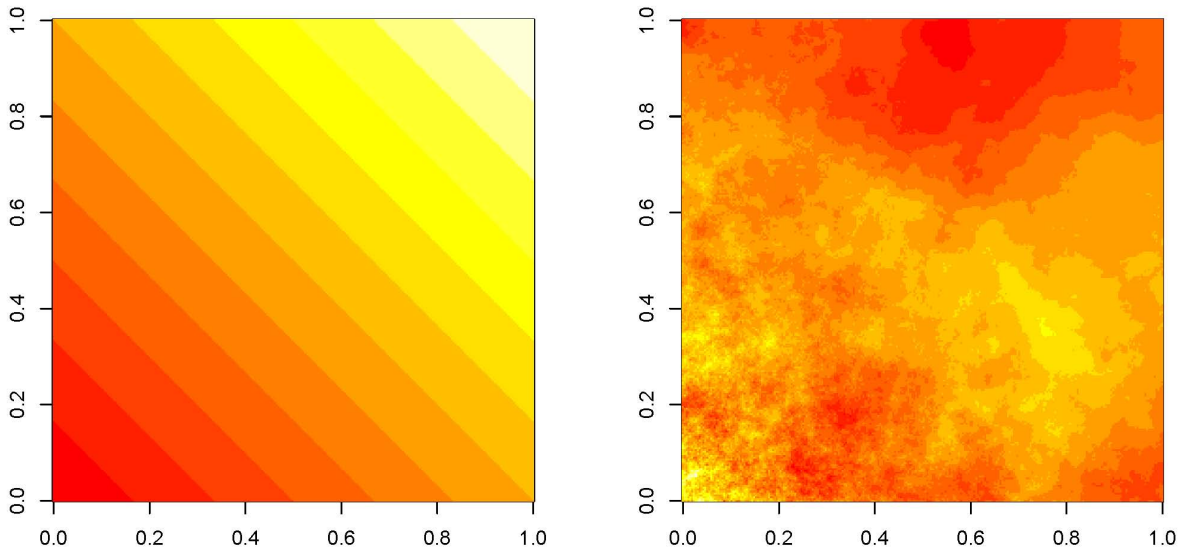

Figure 7: Regularity function produced by `Hinc.diag()` (on the left) and two-dimensional motion associated (on the right). Values increase from red to yellow.

are described at http://www.cict.fr/sys/Serveurs/ondine.html) for the configuration we performed, and we did not have any trouble with too long calculations.

For two-dimensional motions, the calculation durations order of magnitude can be linearly extrapolated from the basic configuration $\{kx = ky = 100, n = 10000\}$. Such a configuration, with 10^8 loops, lasts about 2 minutes on the machine previously mentioned. We can estimate the potential duration for the extreme configuration we would like to perform $\{kx = ky = 500, n = 10^8\}$ at about one year. In Section 4, we described out first steps in parallel computing to try to reach this extreme configuration in a 'reasonable' duration.

4. Parallel computing

In order to go further in our experimental design, we use a supercomputer via the scientific group Calmip (Calcul en Midi-Pyrénées http://www.calmip.cict.fr/). This machine is exclusively dedicated to intensive computation. Its is composed of 68 processors (1.5 GHz) with 136 Go of RAM memory in shared configuration. Technical characteristics are detailed in CALMIP (2005).

To make better use of this machine, we adapted our programs to exploit multi-threadings possibilities via parallel computing. We used an implicit parallelization (OpenMP 2005) that seemed to be a good compromise between automatic and explicit parallelization. The former is easier to implement (just add an argument to the compilation command) but less efficient. The latter, using for example Message Passing Interface, (MPI 2005), is more efficient but more demanding: the programmer has to give each processor a specific task and to ensure inter-processors communications. With OpenMP, one just has to add few lines to precise the piece of code that has to be shared between the processors and then performed in a multi-threading way.

To do so, we convert our function for two-dimensional simulations into one main C program.

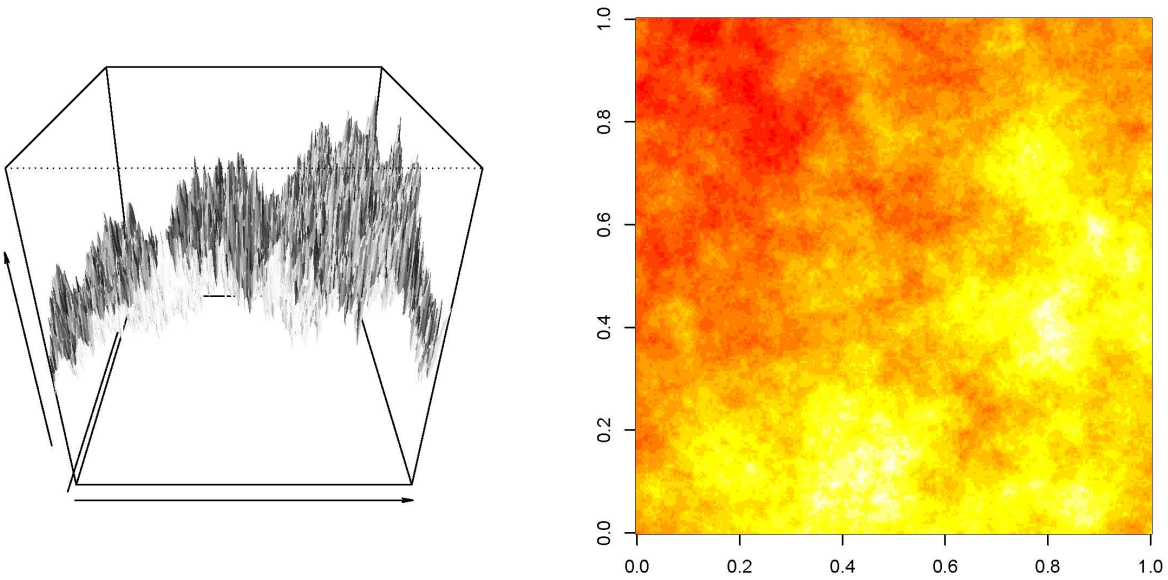

Figure 8: Surface and image representations of a RHFLM (h=0.4) on a 256×256 grid.

It is composed of three parts:

- initializations and random number generation;
- computations;
- output to a file.

The first and the third part were previously dealt in R; the second one is equivalent to the C part of the package. This task, which contains the most demanding part of the computation, requires three nested loops: the rows, the columns and the terms of the sum to be caculated in each cell.

Our problem is quite easy to parallelize because of the independence between the calculation performed on each cell of the grid. Thus we can divide the grid in several blocks and give each processor a block to perform computation on. Technically, the loops intructions are nested in an OpenMP environment written in C as follows:

```
#pragma omp parallel private(tmp,tmp1,tmp2,tmp3,l)
 {
  #pragma omp for
  for (i=0;i<kx;i++) /* loop over the rows */
  #pragma omp parallel shared(i)
    {
    #pragma omp for
    for (j=0;j<ky;j++) /* loop over the columns */
      {
        for (l=0;l<n;l++) /* loop over the terms of the sum */
          {
```

```
            /*
             * Computations
             */
          } /* End of loop l */
      } /* End of loop j */
    } /* End of loop i */
} /* End of parallel area */
```

OpenMP instructions must be preceded by the `#` character. Details on how to run such a program are given in Appendix B.

First simulations performed in this way are promising. We can divide the computation duration by at least the number of processors involved in the computations. We use from 8 to 16 processors. Furthermore, one processor of the supercomputer is faster than those of the current machine previously used.

So a simulation with $kx = ky = 512$ and n=10^7 last about 25 hours using (only) 8 processors instead of an estimated time around 2 months (1500 hours) with our previous configuration (R and C functions).

5. Conclusion

In this article we have proposed an efficient way to produce non Gaussian fields and we hope it can help to model various irregular phenomena. Of special interest is the use of parallel computers that shows that the algorithm is flexible enough. On the other hand, other non-Gaussian models exist and could be simulated in a similar way (Cohen, Lacaux, and Ledoux 2005) and we plan to generalize our techniques to those fields. Moreover statistical estimations of the Hölder exponent (Coeurjolly 2000) are possible for fractional Brownian motions and we plan to apply this to non-Gaussian fields.

We maintain a web page about RHFLMs and RHMLMs. It will always contain the latest release of the **FracSim** package: http://www.lsp.ups-tlse.fr/FracSim/.

Acknowledgements

Authors would like to thanks Céline Lacaux for interesting discussions and the scientific grouping CALMIP (Calcul en Midi-Pyrénées) for giving us access to the supercomputer used for parallel computing.

References

Bardet JM, Lang G, Oppenheim G, Philippe A, Taqqu MS (2003). "Generators of Long-Range Dependent Processes: A Survey." In "Theory and Applications of Long-Range Dependence," pp. 579–623. Birkhäuser Boston, Boston, MA.

Benassi A, Cohen S, Istas J (2002). "Identification and Properties of Real Harmonizable Fractional Lévy Motions." *Bernoulli*, **8**(1), 97–115.

Benassi A, Jaffard S, Roux D (1997). "Gaussian Processes and Pseudodifferential Elliptic Operators." *Revista Mathematica Iberoamericana*, **13**(1), 19–89.

CALMIP (2005). "Groupement Scientifique CALMIP." URL http://www.calmip.cict.fr/.

Chan G, Wood A (1998). "Simulation of Multifractional Brownian Motion." *Proceedings in Computational Statistics*, pp. 233–238.

Coeurjolly JF (2000). "Simulation and Identification of the Fractional Brownian Motion: A Bibliographical and Comparative Study." *Journal of Statistical Software*, **5**(8).

Cohen S, Lacaux C, Ledoux M (2005). "A General Framework for Simulation of Fractional Fields." Preprint. URL http://www.lsp.ups-tlse.fr/Fp/Cohen/.

Cohen S, Taqqu M (2004). "Small and Large Scale Behavior of the Poissonized Telecom Process." *Methodology and Computing in Applied Probability*, **6**(4), 363–379.

Kaj I, Taqqu M (2004). "Convergence to Fractional Brownian Motion and to the Telecom Process: The Integral Representation Approach." Preprint, URL http://www.math.uu.se/%7Eikaj/preprints/kajtaqqu040712.pdf.

Kernighan BW, Ritchie DM (1988). *The C Programming Language*. Prentice-Hall, Englewood Cliffs, New Jersey.

Kolmogorov A (1940). "Wienersche Spiralen und einige andere interessante Kurven im Hilbertschen Raum." *Comptes Rendus (Doklady) de l'Académie des Sciences de l'URSS*, **26**, 115–118.

Lacaux C (2002). "Real Harmonizable Multifractional Lévy Motions." Preprint, URL http://www.lsp.ups-tlse.fr/Fp/Lacaux/maths.html.

Lacaux C (2004). "Series Representation and Simulation of Multifractional Lévy Motions." *Advances in Applied Probability*, **36**(1), 171–197. ISSN 0001-8678.

Mallat S (1989). "A Theory of Multiresolution Signal Decomposition: The Wavelet Representation." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **11**, 674–693.

Mandelbrot B, Van Ness J (1968). "Fractional Brownian Motion, Fractional Noises and Applications." *Siam Review*, **10**, 422–437.

MPI (2005). "Message Passing Interface." URL http://www.mpi-forum.org/.

Peltier R, Lévy Véhel J (1996). "Multifractional Brownian Motion: Definition and Preliminary Results." URL http://www-syntim.inria.fr/fractales/.

OpenMP (2005). "OpenMP Architecture Review Board." URL http://www.OpenMP.org/.

R Development Core Team (2005). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-00-3, URL http://www.R-project.org/.

Rosiński J (2001). "Series Representations of Lévy Processes from the Perspective of Point Processes." In "Lévy processes," pp. 401–415. Birkhäuser Boston, Boston, MA.

Simoncelli E (1999). "Bayesian Denoising of Visual Images in the Wavelet Domain." In "Lecture Notes in Statistics," volume 141, pp. 291–308. Springer-Verlag.

Vidakovic B (1999). *Statistical Modeling by Wavelets*. John Wiley and Sons, Inc, N. Y.

A. Description of R script

In this appendix, we give the code used to produce figures of the article.

- Simulations of 1D-RHFLMs with $h = 0.1$, $h = 0.5$ and $h = 0.9$. The results are stored in the three objects `X01`, `X05` and `X09`.

```
R> X01 = fracsim.1d(h=0.1,k=1000,n=5000)
R> X05 = fracsim.1d(h=0.5,k=1000,n=5000)
R> X09 = fracsim.1d(h=0.9,k=1000,n=5000)
```

- Graphical representations of 1D-RHFLMs (Figure 1)

```
R> par(mfrow=c(3,1))
R> plot(x=X01$t,y=X01$process,type="l")
R> plot(X05$t,X05$process,type="l")
R> plot(X09$t,X09$process,type="l")
```

  The first line divides the graphical window horizontally in three part to put graphics one under each other. Then, we produce plots giving the time discretization vector as abscissa and the process values as ordinate. The option `type="l"` draws a line between each point.

- Simulations of 1D-RHMLMs with increasing and sinusoidal multifractional function

```
R> Hinc = seq(from=0.1,to=0.9,length=1000)
R> Hsin = 0.25+0.25*sin(seq(0,1,length=1000)*(6*pi))
```

  We define an increasing multifractional function with the function `seq` in order to get 1000 (`length`) values regularly spaced between 0.1 (`from`) and 0.9 (`to`). The same function is used in combination with the sinus function to simulate a sinusoidal multifractional function.

- Graphical representations of 1D-RHMLMs (Figure 2)

```
R> par(mfrow=c(2,2))
R> plot(Xinc$t,Hinc,type="l",ylim=c(0,1))
R> plot(Xsin$t,Hsin,type="l",ylim=c(0,1))
R> plot(Xinc$t,Xinc$process,type="l")
R> plot(Xinc$t,Xsin$process,type="l")
```

  The graphical window is divided into four parts (2×2 grid). The top row contains the plots of the multifractional functions; the vertical axis is set to [0,1] (`ylim`). The bottom row contains the trajectory of the RHMLMs.

- Simulations of 2D-RHFLMs with $h = 0.1$, $h = 0.5$ and $h = 0.9$.

```
R> X2d01 = fracsim.2d(h=0.1,kx=100,ky=100,n=100000)
R> X2d05 = fracsim.2d(h=0.5,kx=100,ky=100,n=100000)
R> X2d09 = fracsim.2d(h=0.9,kx=100,ky=100,n=100000)
```

In these calls, the argument `ky` is optional; if missing, its value is taken to be equal to `kx`.

- Graphical representations of 2D-RHFLMs (Figure 3)

```
R> par(mfrow=c(1,3))
R> persp(X2d01$process,shade=0.5,phi=30)
R> persp(X2d05$process,shade=0.5,phi=30)
R> persp(X2d09$process,shade=0.5,phi=30)
```

The function `persp` produces the plot on a grid determined by the size of the matrix given as the first parameter (`X2d0*`). We used `shade=0.5` to define the shade at a surface facet in order to provide an approximation to daylight illumination as explained in the R help for the `persp` function. The option `phi=30` implies a vertical rotation that sets the point of view above the surface.

- Simulations of 2D-RHMLMs with increasing and sinusoidal multifractional function

```
R> Hinc = matrix(rep(seq(0,1,l=100),100),ncol=100)
R> Hsin = matrix(rep(0.25+0.25*sin(seq(0,1,l=100)*(6*pi)),100),ncol=100)
```

The two lines above produce two matrices that are the discretization of the multifractional functions in two cases: 1) the regularity increases column by column, 2) the regularity is sinusoidal according to the columns of the grid. By default, the R function `matrix` produces a matrix from a vector filling by column.

```
R> Xinc = fracsim.2d(h=Hinc,kx=100,ky=100,n=100000)
R> Xsin = fracsim.2d(h=Hsin,kx=100,ky=100,n=100000)
```

Once defined the regularity matrix, the call to `fracsim.2d()` is the same as for RHFLMs.

- Graphical representations of multifractional functions for 2D-RHMLMs (Figure 4)

```
R> persp(Hinc,shade=0.5,phi=30)
R> persp(Hsin,shade=0.5,phi=30)
```

- Graphical representations of 2D-RHMLMs (Figure 5)

```
R> persp(Xinc$process,shade=0.5,phi=30)
R> persp(Xsin$process,shade=0.5,phi=30)
```

- Image and contour plots (Figure 6)

```
par(mfrow=c(2,2))
R> image(Xinc$process)
R> image(Xsin$process)
R> contour(Xinc$process)
R> contour(Xsin$process)
```

- Regularity function increasing along the diagonal and corresponding 2D-motion represented as an image (Figure 7)

```
R> Hinc.diag =
+ function(x,y){outer(x,y,function(a,b){(a+b)/(2*max(a,b))})}
R> Xinc.diag = fracsim.2d(Hinc.diag,n=10000,kx=100)

R> par(mfrow=c(1,2))
R> image(Hinc.diag(seq(0,1,l=100),seq(0,1,l=100)))
R> image(Xinc.diag)
```

B. Run parallel program

The compilation is done using the option `-openmp` to deal with the OpenMP command preceded by '#'.

```
> icc -openmp -O2 -o exec-2d main-2d-openmp.c
```

The option `-O2` specifies the level of optimisation. This produces an executable file (`exec-2d`). Details about the compiler used can be found at http://www.intel.com/cd/software/products/asmo-na/eng/compilers/clin/219831.htm.

In order to optimise the execution, we use the Portable Batch System (PBS, http://www.OpenPBS.org/). A file (`exec-2d.cmd`) containing the following line is created.

```
## PBS data
## Number of processors
#PBS -l ncpus=8

## Memory size
#PBS -l mem=4gb

## Maximal CPU time
#PBS -l cput=100:00:00

export OMP_NUM_THREADS=8

./exec-2d
```

The submission is done thanks to the command: `qsub exec-2d.cmd`. The execution is delayed until the number of required processors is available.

Affiliation:

Sébastien Déjean
Laboratoire de Statistique et Probabilités
Université Paul Sabatier
31062 Toulouse Cedex 9, France
E-mail: sebastien.dejean@math.ups-tlse.fr
URL: http://www.lsp.ups-tlse.fr/Fp/Dejean/

Serge Cohen
Laboratoire de Statistique et Probabilités
Université Paul Sabatier
31062 Toulouse Cedex 4, France
E-mail: serge.cohen@math.ups-tlse.fr
URL: http://www.lsp.ups-tlse.fr/Fp/Cohen/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 14, Issue 18 *Submitted:* 2005-10-06
December 2005 *Accepted:* 2005-12-08