New version of PLNoise: a package for exact numerical simulation of power-law noises

Edoardo Milotti *

Dipartimento di Fisica, Università di Trieste and I.N.F.N. – Sezione di Trieste
Via Valerio, 2 – I-34127 Trieste, Italy

Abstract

In a recent paper I have introduced a package for the exact simulation of power-law noises and other colored noises (E. Milotti, Comput. Phys. Commun. **175** (2006) 212): in particular the algorithm generates $1/f^\alpha$ noises with $0 < \alpha \leq 2$. Here I extend the algorithm to generate $1/f^\alpha$ noises with $2 < \alpha \leq 4$ (black noises). The method is exact in the sense that it produces a sampled process with a theoretically guaranteed range-limited power-law spectrum for any arbitrary sequence of sampling intervals, i.e., the sampling times may be unevenly spaced.

PACS: 02.50.Ey,05.40.Ca,02.70.Uu

Key words: $1/f^\alpha$ noise generation; colored noise generation; uneven sampling; Gaussian noise; $1/f$ noise; black noise; fGn; fBm

NEW VERSION PROGRAM SUMMARY

* tel. +39 040 558 3388, fax +39 40 558 3350
  Email address: `milotti@ts.infn.it` (Edoardo Milotti).

RAM: the code of the test program is very compact (about 60 Kbytes), but the program works with list management and allocates memory dynamically; in a typical run with average list length $2 \cdot 10^4$, the RAM taken by the list is 200 Kbytes.

Classification: 4.13

External routines: The package needs external routines to generate uniform and exponential deviates. The implementation described here uses the random number generation library `ranlib` freely available from Netlib [1], but it has also been successfully tested with the random number routines in Numerical Recipes [2]. Notice that `ranlib` requires a pair of routines from the linear algebra package `LINPACK`, and that the distribution of `ranlib` includes the C source of these routines, in case `LINPACK` is not installed on the target machine.

Catalogue identifier of previous version: ADXV_v1_0

Journal reference of previous version: Comput. Phys. Commun. **175** (2006) 212

Does the new version supersede the previous version?: Yes

Nature of problem: Exact generation of different types of colored noise.

Solution method: Random superposition of relaxation processes [3], possibly followed by an integration step to produce noise with spectral index > 2.

Reasons for the new version: Extension to $1/f^\alpha$ noises with spectral index $2 < \alpha \le 4$: the new version generates both noises with spectral with spectral index $0 < \alpha \le 2$ and with $2 < \alpha \le 4$.

Summary of revisions: although the overall structure remains the same, one routine has been added and several changes have been made throughout the code to include the new integration step.

Unusual features: The algorithm is theoretically guaranteed to be exact, and unlike all other existing generators it can generate samples with uneven spacing.

Additional comments: The program requires an initialization step; for some parameter sets this may become rather heavy.

Running time: running time varies widely with different input parameters, however in a test run like the one in section 3 in the long write-up, the generation routine took on average about 75 μs for each sample.

References:

[1] B. W. Brown, J. Lovato, and K. Russell: `ranlib`, available from Netlib (`http://www.netlib.org/random/index.html`, select the C version `ranlib.c`).

[2] W. H. Press, S. A. Teulkolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing, 2nd ed.* pp. 274-290 (Cambridge Univ. Press., Cambridge, 1992).

[3] E. Milotti, Phys. Rev. E **72**, 056701 (2005).

LONG WRITE-UP

1   Introduction

I have recently developed a power-law noise generator which uses a superposition of uncorrelated simple relaxation processes with a uniform distribution of relaxation rates to produce $1/f$ noise, and a power-law distribution of relaxation rates to obtain $1/f^\alpha$ noise [1,2]: the generator can be easily modified to produce complex superpositions of different power-law noises, like those observed in precise clocks [3]. This generator also has a very unusual and unique feature: the superposition mechanism takes into account the correlation between different samples, and does it so *exactly*, so that its noise output can be sampled at arbitrary sampling times. This is important whenever sampling cannot be performed at evenly spaced times: uneven sampling is quite common in astronomical observations [4], but also in other fields such as climatology [5], it shows up whenever observations have unintentional gaps and missing data [6], and sometimes bunched sampling may even be a technical need, e.g. in hard-disk controllers [7]; moreover, uneven sampling has some special properties, because it is not band limited in the same sense as ordinary, even sampling [8]. The generator does not produce exactly Gaussian noise, but the closeness to Gaussianity is tunable and the noise distribution can approach a true Gaussian to any required degree by a proper parameter setting [1].

Most of the observed power-law $(1/f^\alpha)$ noises have spectral indexes $0 < \alpha \leq 2$, with an apparent clustering around $\alpha = 1$. However noises with higher spectral indexes $2 < \alpha \leq 4$ also show up in several unrelated systems [9,10] like the water level of the Nile river, economics, orchid population size [11] and local temperature fluctuations and affect precise timekeeping [3] and our ability to predict environmental and animal population variables [12]. Noises with $\alpha > 2$ also appear in the energy level fluctuations of quantum systems [13,14]. These noises correspond to generalizations of the standard one-dimensional random walk, and for this reason the corresponding process is often called *fractional Brownian motion* (fBm) (likewise Gaussian noises with $0 \leq \alpha \leq 2$ are also called *fractional Gaussian noises* (fGn)). Because of their extreme peaking behavior at low frequencies these noises are also called "black" [10], and they display marked persistence properties [9] that may lead to the mistaken identification of underlying trends in experimental data [15].

It is easy to see that there is no way to produce black noises from the superposition of simple relaxation processes, the spectral density of a train of random pulses with simple exponential response with decay rate λ is proportional to $1/(\omega^2 + \lambda^2)$ and its derivative in a log-log plot ranges from 0 to -2: thus no simple superposition can have a spectrum that decays faster than $1/f^2$ and

4

the algorithm described in [1,2] is unable to simulate this very interesting class of power-law noises. This paper addresses the practical problem of including black noises in the the noise generator introduced in [2], preserving all its good features.

2 Generation of black noises

It may be argued that the superposition argument could still be used if one takes pulses with non-exponential shapes, however in that case the theory exposed in [1] must be revised and the analysis becomes considerably harder. These difficulties can be circumvented when we note that the relationship between the spectral density $S_x(\omega)$ of a random process x and the spectral density $S_y(\omega)$ of its integral y is $S_y(\omega) = S_x(\omega)/\omega^2$ and therefore it is possible to generate a black noise taking the integral of a power-law noise obtained from simple superposition. Here, as in [1,2], I consider a noise signal $x(t)$ which is a linear superposition of many random pulses, i.e., pulses that are random in time and are associated to a memoryless process with a Poisson distribution, and such that their pulse response function is

$$h(t,\lambda) = \begin{cases} \exp(-\lambda t) & \text{if } t \geq 0 \\ 0 & \text{if } t < 0 \end{cases} \tag{1}$$

with a decay rate λ which is a positive definite random variate with probability density $g_\lambda(\lambda)$ in the range $(\lambda_{min}, \lambda_{max})$ so that

$$x(t) = \sum_k A_k h(t - t_k, \lambda_k) \tag{2}$$

where t_k is the time at which the k-th pulse occurs, A_k is its amplitude and λ_k is the decay rate of its pulse response function. In principle the amplitude is also a random variate, but here I take a fixed amplitude $A_k = A$. If n is the pulse rate, it can be shown [1] that the average signal level is

$$\langle x \rangle = nA \left\langle \frac{1}{\lambda} \right\rangle, \tag{3}$$

the variance is

$$\langle (\Delta x)^2 \rangle = \frac{nA^2}{2} \left\langle \frac{1}{\lambda} \right\rangle, \tag{4}$$

5

and the spectral density is

$$S(\omega) = \frac{nA^2}{2\pi} \int\limits_{\lambda_{min}}^{\lambda_{max}} \frac{g_\lambda(\lambda)}{\omega^2 + \lambda^2} d\lambda, \tag{5}$$

where the decay rate λ which is a positive definite random variate with probability density $g_\lambda(\lambda)$ in the range $(\lambda_{min}, \lambda_{max})$. I define the normalized zero-mean process as follows

$$x_N(t) = \frac{x(t) - \langle x \rangle}{\sqrt{\langle (\Delta x)^2 \rangle}} = \frac{A \sum_k h(t - t_k, \lambda_k) - \langle x \rangle}{\sqrt{\langle (\Delta x)^2 \rangle}}, \tag{6}$$

and the integral of $x_N(t)$ is

$$y_N(t) = \int\limits_{-\infty}^{t} x_N(t')dt' = \int\limits_{-\infty}^{t} \frac{x(t') - \langle x \rangle}{\sqrt{\langle (\Delta x)^2 \rangle}} dt' \tag{7}$$

. Equation (7) is a trivial analytical answer which is still useless for inclusion in the generator and it must be suitably rearranged: from equation (7) we see that $\langle y_N(t) \rangle = 0$, and that $y_N(t)$ is a kind of continuous one-dimensional random walk process

$$y_N(t + \Delta t) = y_N(t) + \int\limits_{t}^{t+\Delta t} x_N(t')dt' \tag{8}$$

$$= y_N(t) + \frac{1}{\sqrt{\langle (\Delta x)^2 \rangle}} \left(\int\limits_{t}^{t+\Delta t} x(t')dt' - \langle x \rangle \Delta t \right) \tag{9}$$

where $y_N(t)$ is the integrated process sampled at time t and $y_N(t + \Delta t)$ is sampled at time $t + \Delta t$ (notice that Δt *is not* a short integration time, but the arbitrary time interval between any two sampling times). From equation (2) we note that the integral in equation (9) can be rearranged as follows

$$\int\limits_{t}^{t+\Delta t} x(t')dt' = A \sum_k \int\limits_{t}^{t+\Delta t} h(t' - t_k, \lambda_k)dt'$$

$$= A \sum_{t_k < t} \frac{e^{-\lambda_k(t-t_k)}}{\lambda_k} \left[1 - e^{-\lambda_k \Delta t} \right] + A \sum_{t_k \in (t, t+\Delta t)} \frac{1}{\lambda_k} \left[1 - e^{-\lambda_k(t+\Delta t - t_k)} \right] \tag{10}$$

6

where the summations in (10) run over pulses that occured before t and over those that occurred within the $(t, t+\Delta t)$ time interval. Finally from equations (9) and (10) we obtain a formula for the integrated process y_N:

$$
\begin{aligned}
y_N(t + \Delta t) = y_N(t) + \frac{1}{\sqrt{\langle(\Delta x)^2\rangle}} \Bigg\{ A \sum_{t_k < t} \frac{e^{-\lambda_k(t-t_k)}}{\lambda_k} \left[1 - e^{-\lambda_k \Delta t}\right] \\
+ A \sum_{t_k \in (t, t+\Delta t)} \frac{1}{\lambda_k} \left[1 - e^{-\lambda_k(t+\Delta t - t_k)}\right] - \langle x \rangle \Delta t \Bigg\}
\end{aligned}
\tag{11}
$$

and this can be incorporated in the generator, because the summations can be evaluated using the events of the underlying Poisson process stored in a linked list as in [2]. Equation (11) is both a practical formula for the generation of the integrated process y_N and a generalization of the equivalent updating formulas in [16] (those formulas hold only for the special case of the Ornstein-Uhlenbeck process and its integral, i.e. for $\alpha = 2$ and $\alpha = 4$): the generation of the integrated process can be achieved producing and maintaining a sequence of Poisson distributed events as in [2] and using equation (11) instead of (2) to evaluate the noise process. Figures 1 and figure 2 show a pair of x_N, y_N processes obtained with the methods described above, where the integrated process has a spectral index $\alpha = 3.5$. Figure 3 shows the spectral density of the integrated process, and the numerical result is in excellent agreement with theory.

Finally notice that when the input process $x(t)$ is Gaussian (and this can be achieved with a proper choice of generator parameters as explained in [1]), the difference $y_N(t + \Delta t) - y_N(t)$ is a Gaussian variate as well.

3   Changes in the library

The changes in the library [2] are transparent for the user, and are scattered in several parts of the code. As in the first version, the package contains two header files and two code files (plus the `ranlib` files that are included for convenience, and are redistributed according to the standard Netlib rules). The first header file (`noise.h`) contains the necessary `include` statements and the structure definitions; the second header file (`noise_prototypes.h`) contains just the prototype definitions. In the present version of the generator, the structure used to share information between the generation routines (defined in the header file `noise.h`) is

```
struct info
{
```

```
double nt; /* transition rate */
double tau; /* average transition time */

double fillUpTime; /* fill-up time estimate */
double fillUpLength; /* list length estimate */

double average; /* signal average and standard deviation */
double sd;

double meaninvlambda; /* mean value of decay time */

double lambdamin; /* min and max decay rates */
double lambdamax;
double beta; /* beta input by the user */
double beta0; /* actual value of beta used in the pulse distribution */
double lmin; /* aux. variables */
double lmax;
double dl;
double binv;

};
```

This header also contains the definition of N_{decay}: `#define NDECAY 20.`; with
this definition the program is quite accurate (the average relative error after
discarding the old events is $\approx 2 \cdot 10^{-9}$), however the generation process can
be made more accurate (and slower) with a larger value of `NDECAY`, or less
accurate and faster with a smaller value.

The first code file (`list_routines.c`) contains the code for the list routines:

- `Append`: this is a variant of the usual `Push` routine;
- `Process_List`: this routine processes the list to get rid of the old elements
  that can no longer influence the output signal;
- `Print_List`: this routine prints the list;
- `Response`: this routine computes the noise signal for a spectral index between
  0 and 2;
- `IntegratedResponse`: this routine computes the integrated noise signal and
  is called when the user asks for a spectral index between 2 and 4;
- `Get_List_Length`: this routine returns the length of the list.

these are internal routines, they are not meant to be called by the user, and
are listed here for completeness; `IntegratedResponse` is new in this version.

The second code file (`generator.c`) contains the user-callable routines. These
routines have the same names and calling sequences as the first version of the
generator [2], even though they have been modified in several places.

4 Changes in the test program

The user interface and the output file structure of the test program included in the package is the same as in the previous version [2]. The important difference is that now the allowed values of the spectral index α are in the range $0 \leq \alpha \leq 4$.

The output file begins with a header, which is a single line with the following values (separated by tabs):

(1) `dt` = sampling interval;
(2) `nsamp` = number of samples;
(3) `tmax` = time of last sample;
(4) `noise_info.nt` = transition rate;
(5) `noise_info.tau` = average time between transitions;
(6) `noise_info.lambdamin` = λ_{min};
(7) `noise_info.lambdamax` = λ_{max};
(8) `noise_info.beta` = $\beta = \alpha - 1$;
(9) `noise_info.meaninvlambda` = $\langle 1/\lambda \rangle$;
(10) `noise_info.fillUpTime` = fill-up time estimate;
(11) `noise_info.fillUpLength` = list length estimate;
(12) `noise_info.average` = average output amplitude;
(13) `noise_info.sd` = standard deviation of noise signal;

The rest of the file is a set of records, each with the following tab-separated values:

(1) `kk` = record number;
(2) `t` = actual time;
(3) `tt` = time of last transition event;
(4) `listLength` = list length;
(5) `signal` = signal (for $0 \leq \alpha \leq 2$) or integrated response (for $2 < \alpha \leq 4$);
(6) `norm_signal` = normalized signal value;

If $0 \leq \alpha \leq 2$ `signal` is

$$\texttt{signal} = \sum_{t_k < t} \exp\left(\lambda_k (t - t_k) \right) \tag{12}$$

and `norm_signal` $= x_N$, while if $2 < \alpha \leq 4$ `signal` is the output of the `IntegratedResponse` routine

$$\texttt{signal} = \sum_{t_k < t} \frac{e^{-\lambda_k(t-t_k)}}{\lambda_k} \left[1 - e^{-\lambda_k \Delta t} \right] + \sum_{t_k \in (t, t+\Delta t)} \frac{1}{\lambda_k} \left[1 - e^{-\lambda_k(t+\Delta t - t_k)} \right] \tag{13}$$

and `norm_signal` = y_N.

The code distribution also contains two *Mathematica* notebooks to analyze and display the program output; the notebook `display.nb` is used for spectral index $0 \le \alpha \le 2$, while `display2.nb` is used for spectral index $2 < \alpha \le 4$.

The test program is used to generate the example discussed in the next section.

5 Test run

In the example shown below the spectral index is $\alpha = 3.5$ (i.e. the program generates $1/f^{3.5}$ noise, and $\beta = 2.5$), $\lambda_{min} = 0.0001$ and $\lambda_{max} = 1$ (the power-law region spans approximately 4 orders of magnitude), so that this corresponds very closely to the example given in [2]. And indeed, the program returns a signal which is the time integral of the sequence generated in the example given in [2], and the underlying sequence is the same (the random number generator is the same as that in [2], and uses the same seed for the random number sequence).

```
*** PLNoise ***

1. terminal input of control variables

Enter sampling time interval (dt): 1
Enter number of samples: 4194304
--> Total sampling time: 4.1943e+06
Enter transition rate: 0.1
--> Average transition time: 10
Enter lambda_min: 0.0001
Enter lambda_max (0 = single relax. rate, lambdamax = lambdamin): 1
Enter alpha (spectral index in 1/f^alpha): 3.5

2. initialization

Initialization time: 0.050000 seconds

Noise parameters:

 -- Spectral shape:
Min decay rate: 0.0001
Max decay rate: 1
Beta: 2.5 (spectral index is 1+beta = 3.5)

 -- Poisson process:
Transition rate: 0.1
```

```
Average transition time: 10

 -- Algorithmic variables:
Fill-up time: 200000
Fill-up length: 20000
Mean value of decay time (<1/lambda>): 100
Mean list length: 200
Signal average: 10
Signal variance: 5
Signal standard deviation: 2.23607
Signal skewness: 0.298142
Rule of thumb for Gaussianity: n<1/lambda> = 10 >= 10, noise is Gaussian

 -- Internal parameters:
lmin: 0.01
lmax: 1
dl: 0.99
binv: 2

List length after initialization: 203

... initialized ...
... starting now ...


3. main generation loop

|-----------------------------------------------|
 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Generation time: 313.780000 seconds

4. statistics

Statistics of generated sequence:
Average: 10.0659
Variance: 4.91088
Standard deviation: 2.21605
Skewness: 0.318761

5. end
```

The program has been compiled and run with the same compiler and on the same machine as the example in [2] (i.e., the compiler `gcc 4.0.1` with optimization flag `-O3` has been used, and the program has been run on an Apple Powerbook G4 - 1.5 GHz with the Mac OS X 10.4.6 UNIX flavor): a comparison between the generation times shows that in this case the integration step

leads to a 75% increase of generation time with respect to the unintegrated case.

References

[1] E. Milotti, Phys. Rev. E **72** (2005) 056701.

[2] E. Milotti, "PLNoise: a package for exact numerical simulation of power-law noises", Comp. Phys. Comm., in press.

[3] See, e.g., J. A. Barnes et al., IEEE Trans. on Instr. and Meas. **IM-20** (1971) 105.

[4] H. M. Adorf, in ASP Conf. Ser. 77, *Astronomical Data Analysis Software and System IV*, ed. R. A. Shaw, H. E. Payne, and J. J. Hayes, ASP, San Francisco, 1995, p. 460.

[5] D. Henslop and M. J. Dekkers, Physics of the Earth and Planetary Interiors **130** (2002) 103.

[6] P. S. Wilson, A. C. Tomsett, and R. Toumi, Phys. Rev. **E68** (2003) 017103.

[7] S.C. Wu and M. Tomizuka, in Proceedings of the 2003 American Control Conference, 2003, vol. 5, IEEE, Piscataway, p. 4347.

[8] F. J. Beutler, SIAM Rev. **8** (1966) 328,

[9] B. B. Mandelbrot, *The Fractal Geometry of Nature*, pp. 247-255, (W. H. Freeman & Co., New York, 1983).

[10] M. Schroeder, *Fractals, Chaos, Power Laws: minutes from an infinite paradise*, pp. 121-133, (W. H. Freeman & Co., New York, 1991).

[11] M. P. Gillman and M. E. Dodd, Bot. J. Linn. Soc. **126** (1998) 65.

[12] K. M. Cuddington and P. Yodzis, Proc. R. Soc. Lond. **B266** (1999) 969.

[13] A. Relaño, J. M. G. Gomez, R. A. Molina, J. Retamosa, and E. Faleiro, Phys. Rev. Lett. **89** (2002) 244102.

[14] L. Salasnich, Phys. Rev. E **71** (2005) 047202.

[15] G. Rangarajan and M. Ding, Phys. Rev. E **61** (2000) 4991.

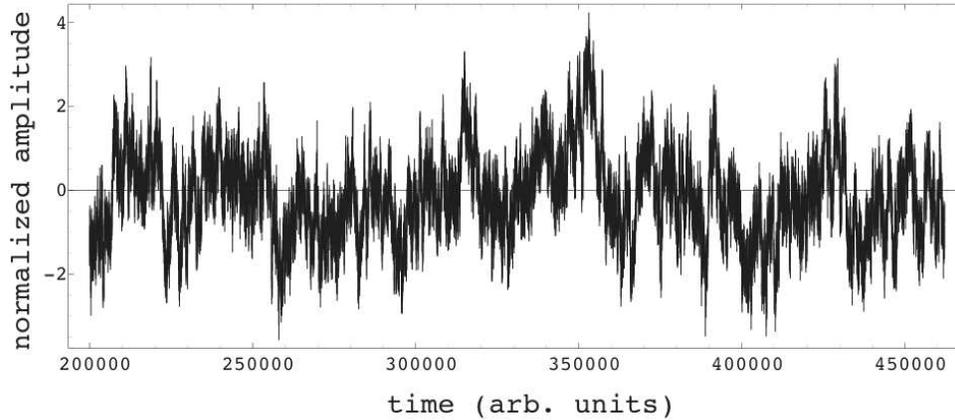[16] D. T. Gillespie, Phys. Rev. E **54** (1996) 2084.

Fig. 1. Normalized process $x_N(t)$ with power-law spectrum $1/f^{1.5}$ produced by the generator described in [1,2]. This record contains $2^{18} = 262144$ samples, and the generation parameters are $n = 10$, $\lambda_{min} = 0.0001$, $\lambda_{max} = 1$, $\beta = 0.5$ (see [1] for a detailed explanation of these parameters). Time does not start from 0 because the initial part of the noise record is used for initialization, and the relaxation rates λ are given in arbitrary frequency units related to the arbitrary time units used in the figure.
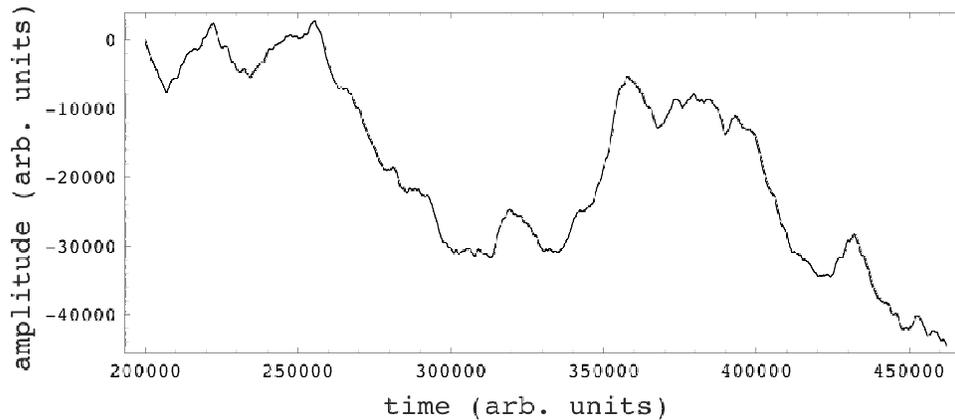

Fig. 2. Plot of the process $y_N(t)$ obtained from the time integration of the process $x_N(t)$ shown in figure 1, as explained in the text. The apparent global linear trend is characteristic of black noise, and any such trend disappears or is replaced by another different trend in longer records.
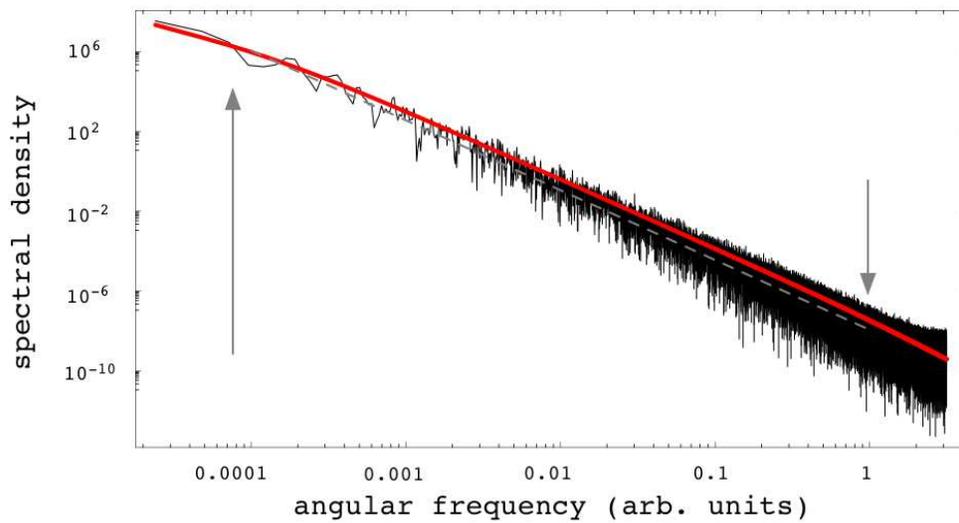
Fig. 3. Spectral density of the process $y_N(t)$ shown in figure 2. The linear trend must be removed to avoid artifacts, either by detrending or by windowing: here I used a Hanning window. The arrows mark the positions of the minimum and maximum relaxation rates λ_{min} and λ_{max}, the thick line is the expected average spectrum, corrected for the window incoherent gain, and the dashed line shows the expected slope of a $1/f^{3.5}$ spectral density. The slight upward bend at high frequency is due to uncorrected aliasing.