

# Comparison of Three Algorithms for Lévy Noise Generation

Matteo Leccardi

*Tethis S.r.l., via Boschetti 1, Milano, Italy*

---

## Abstract

In this paper, we describe three algorithms for the generation of symmetric Lévy noise and we discuss the relative performance in terms of time of execution on an Intel Pentium M processor at 1500 MHz. The relative performance of the three algorithm is given as a function of the Lévy stable index  $\alpha$  and of the number of produced random points.

*Keywords:* Noise Generator,  $\alpha$ -stable Distribution

*PACS:* 05.10.Ln, 89.75.Da, 05.45.Tp

---

## 1 Introduction

In the first section of this paper, we recall some definitions and basic properties of Lévy processes and  $\alpha$ -stable stochastic variables, in section 2 we describe three different algorithms for generation of  $\alpha$ -stable pseudo-random numbers, in section 3 we analyze the time performance of the algorithms and in section 4 we state our conclusions.

We can define a Lévy process as a stochastic process with both stationary and independent increments.

A distribution  $F$  is called infinitely divisible if for any  $n \geq 1$  it can be expressed as the  $n$ -th fold convolution  $G^{*n}$  of some distribution  $G$  (that depends on  $n$ ).

There exist a strong link between Lévy processes and infinitely divisible distributions, in fact it can be shown that, if  $X(t)$  is a Lévy process, then its distribution for any  $t$  is infinitely divisible and, vice versa, for any infinitely divisible distribution  $F$  there exist a Lévy process for which  $X(1) \sim F$ . [1, 2, 3, 4]

The cumulant characteristic function of Lévy processes satisfies the Lévy-Khintchine formula

$$\psi(\theta) = i\gamma\theta - \frac{1}{2}\sigma^2 + \int_{-\infty}^{+\infty} \left( \exp(i\theta x) - 1 - i\theta x 1_{\{|x|<1\}} \right) \nu(dx) \quad (1.1)$$

where  $\gamma \in \mathbb{R}$ ,  $\sigma^2 \geq 0$ , and  $\nu$  is a measure on  $\mathbb{R} \setminus \{0\}$ . [5]

These three objects are called Lévy triplet  $[\gamma, \sigma^2, \nu]$ .

A random variable  $Y$  is stable if, for each  $n \in \mathbb{N}$ , with  $Y_1, \dots, Y_n$  independent identically distributed copies of  $Y$ ,  $Y_1 + \dots + Y_n \stackrel{d}{=} bY + c$  for some constants  $b = b(n) > 0$  and  $c = c(n) \in \mathbb{R}$ .

A stable  $Y$  is called  $\alpha$ -stable,  $\alpha \in (0, 2]$ , if  $Y^{*t} \stackrel{d}{=} t^{1/\alpha} Y + c$  for  $t > 0$ , for some constant  $c = c(t) \in \mathbb{R}$ ; it can be proved that the constant  $\alpha$  is unique.  $Y$  is called strictly  $\alpha$ -stable if  $c(t) = 0$  for  $t > 0$ .

The relationship between  $\alpha$ -stable random variables and Lévy processes is determined by a theorem stating that a real random variable infinitely divisible  $Y$  is  $\alpha$ -stable if and only if it has a Lévy triplet  $[\gamma, 0, \nu]$  such that

$$d\nu(x) = \left( c_1 1_{(0, \infty)}(x) + c_2 1_{(-\infty, 0)}(x) \right) |x|^{-(\alpha+1)} dx \quad (1.2)$$

for some unique constants  $c_1, c_2 \geq 0$ .

The characteristic function is then

$$\phi(\theta) = \exp[\psi(\theta)] = \begin{cases} \exp \left\{ -c |\theta|^\alpha \left[ 1 - i\beta \operatorname{sgn}(\theta) \tan\left(\frac{\alpha\pi}{2}\right) \right] + i\theta\tau \right\}, & \alpha \neq 1 \\ \exp \left\{ -c |\theta| \left[ 1 + i\beta \frac{2}{\pi} \operatorname{sgn}(\theta) \log(|\theta|) \right] + i\theta\tau \right\}, & \alpha = 1 \end{cases} \quad (1.3)$$

where  $\alpha \in (0, 2]$  is the index of stability or characteristic exponent,  $\beta = \frac{c_1 - c_2}{c_1 + c_2} \in [-1, 1]$  is the skewness parameter,  $c > 0$  is a scale factor, and  $\tau \in \mathbb{R}$  is the location parameter (the mean value if  $\alpha \in (1, 2]$ ). [6]

For  $Y$  with the above characteristic function we write  $Y \sim S_\alpha(c, \beta, \tau)$ .

There exist an explicit formula to simulate  $\alpha$ -stable random variables, as a function of two independent variables  $w$  and  $\varphi$  with uniform distribution in the range  $(-\pi/2, \pi/2)$  and standard exponential distribution, respectively

$$c \frac{\sin \left[ \alpha \varphi + \tan^{-1} \left( \beta \tan \left( \frac{\alpha\pi}{2} \right) \right) \right] \left( \cos \left[ (1-\alpha) \varphi - \tan^{-1} \left( \beta \tan \left( \frac{\alpha\pi}{2} \right) \right) \right] \right)^{1/\alpha-1}}{\left( \cos \left[ \tan^{-1} \left( \beta \tan \left( \frac{\alpha\pi}{2} \right) \right) \right] \right)^{1/\alpha} (\cos(\varphi))^{1/\alpha} w^{1/\alpha-1}} + \tau \sim S_\alpha(c, \beta, \tau) \quad (1.4)$$

## 2 Description of the algorithms

As stated in the introduction, we tested three different algorithms for generating  $\alpha$ -stable random variables. The algorithms have been written in Matlab<sup>®</sup>.

### 2.1 Mantegna's algorithm

The first algorithm we tested (Mantegna) is described in [7]; it can produce random numbers according a symmetric Lévy stable distribution.

The function needs as input the distribution's parameters  $\alpha \in [0.3, 1.99]$  and  $c > 0$ , the number of iterations  $n$  and the number of random points to be produced; without this last input, the output consists of a single number. If an input parameter is outside the valid range, an error message is displayed and the output consists of a array of NaNs.

The algorithm can be split into three steps; the first one is to calculate

$$v = \frac{x}{|y|^{1/\alpha}} \quad (2.1)$$

where  $x$  and  $y$  are normally distributed stochastic variables. If we put

$$\sigma_x(\alpha) = \left[ \frac{\Gamma(1+\alpha)\sin(\pi\alpha/2)}{\Gamma((1+\alpha)/2)\alpha 2^{(\alpha-1)/2}} \right]^{1/\alpha} \quad (2.2)$$
$$\sigma_y = 1$$

the resulting distribution has the same behaviour of a Lévy distribution for large values of the random variable ( $|v| \gg 0$ ).

Using the nonlinear transformation

$$w = \left\{ \left[ K(\alpha) - 1 \right] e^{-|v|/C(\alpha)} + 1 \right\} v \quad (2.3)$$

the sum

$$z_{cn} = \frac{1}{n^{1/\alpha}} \sum_{k=1}^n w_k \quad (2.4)$$

quickly converges to a Lévy stable distribution. The convergence is assured by the central limit theorem.

The value of  $K(\alpha)$  can be obtained analytically

$$K(\alpha) = \frac{\alpha \Gamma((\alpha+1)/2)}{\Gamma(1/\alpha)} \left[ \frac{\alpha \Gamma((\alpha+1)/2)}{\Gamma(\alpha+1) \sin(\pi\alpha/2)} \right]^{1/\alpha} \quad (2.5)$$

while  $C(\alpha)$  is the result of a polynomial fit of the values tabulated in [7], obtained resolving the integral equation.

$$\frac{1}{\pi \sigma_x} \int_0^\infty q^{1/\alpha} \exp \left[ -\frac{q^2}{2} - \frac{q^{2/\alpha} C(\alpha)^2}{2 \sigma_x^2(\alpha)} \right] dq = \frac{1}{\pi} \int_0^\infty \cos \left[ \left( \frac{K(\alpha)-1}{e} + 1 \right) C(\alpha) \right] \exp(-q^\alpha) dq \quad (2.6)$$

The requested random variable is

$$z = c^{1/\alpha} z_{cn} \quad (2.7)$$

## 2.2 McCulloch's algorithm

The second algorithm has been encoded in Matlab by J. H. McCulloch at Ohio State University [8] and is based on the method described in [9]; the function returns an  $n \times m$  matrix of random numbers with characteristic exponent  $\alpha$ , skewness parameter  $\beta$ , scale  $c$ , and location parameter  $\tau$ . The minimum value for  $\alpha$  is 0.1 because of the non-negligible probability of overflow. As in the previous function, when an input is out of the valid range, the output is a matrix of NaNs.

We will only consider the symmetric case  $\beta=0$ .

The algorithm uses the formula (1.4), that, if  $\beta=0$ , reduces to

$$x = c \left( \frac{\cos((1-\alpha)\varphi)}{w} \right)^{\frac{1}{\alpha}-1} \left( \frac{\sin(\alpha\varphi)}{\cos(\varphi)} \right)^{\frac{1}{\alpha}} + \tau \quad (2.8)$$

Two special cases are handled separately:

$\alpha=2$  (Gaussian case)

$$x = c 2\sqrt{w} \sin(\varphi) + \tau \quad (2.9)$$

$\alpha=1$  (Cauchy case)

$$x = c \tan(\varphi) + \tau \quad (2.10)$$

## 2.3 Rejection algorithm

The third algorithm relies on the rejection method [10].

The function needs as input the distribution's parameters  $\alpha$  and  $c$ , the half-width of the interval to be considered  $M$  and the number of random points to be produced; without this last input the output

consists of a single number. If an input parameter is outside the valid range, an error message is displayed and the output consists of a array of NaNs.

We approximate the probability density of Lévy distribution with the function

$$\begin{aligned} f(x) &= L_{\alpha,c}(x_i) \\ x_i - 0.1 &\leq x \leq x_i + 0.1 \\ x_i &= -M + 0.2(i-1) \\ 1 \leq i &\leq \frac{2M}{0.2} + 1 \end{aligned} \tag{2.11}$$

where  $L_{\alpha,c}(x)$  is the probability density function (pdf) of a Lévy symmetrical stable process; the value is obtained by numerical integration of

$$L_{\alpha,c}(x) = \frac{1}{\pi} \int_0^\infty \exp(-c\theta^\alpha) \cos(\theta x) d\theta \tag{2.12}$$

Then we extract two random numbers,  $x$  and  $x_2$ , the first one belongs to a uniform distribution in the range  $[-M,M]$ ; the second follows the same distribution but lies between 0 and the probability density maximum. If  $x_2$  is less than  $f(x)$ ,  $x$  is a valid choice, otherwise it is rejected and another couple of number is extracted.

### 3 Test

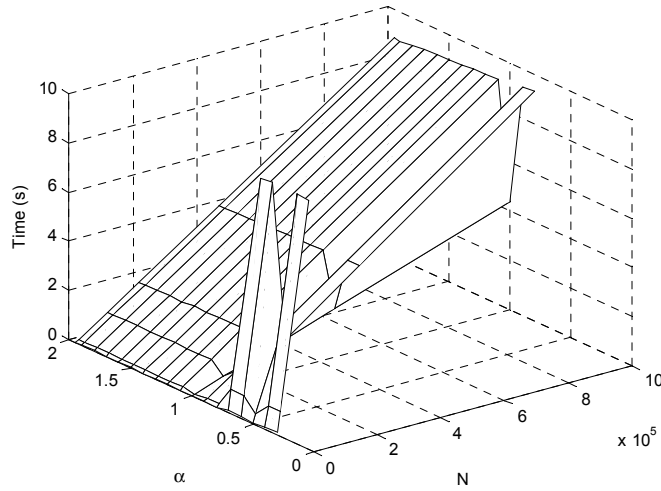
We run the tests using a laptop PC by HP<sup>®</sup>, model Compaq nx5000, equipped with 1.5 GHz Intel<sup>®</sup> Pentium<sup>®</sup> M processor, 512 MB of RAM and Windows XP home edition operating system. We set on the PC the Home/Office desk power scheme in order to obtain the highest performance state [11]. We evaluated the execution time, using the Matlab's Profiler tool, as a function of  $\alpha$  and of the number of points  $N$ ; the scale factor  $c$  was set to 1. Each value is the mean of five trials. The results are expressed in seconds and are approximate to two decimal places, because the resolution of the tool is about 15 ms.

#### 3.1 Mantegna's algorithm

We tested Mantegna's algorithm setting the number of iterations to 10, for  $\alpha$  greater than 0.75; with these values, an excellent agreement between the simulated process and the Lévy distribution is assured. With  $\alpha$  less than 0.75, we put the number of iterations to 100; this limited the maximum  $N$  to 100000, because of memory limitations.

**Table 1:** Execution times (seconds) for Mantegna's algorithm.

$\alpha \backslash N$	1000	10000	100000	200000	450000	1000000
0.30	0.11	0.93	9.20			
0.40	0.10	0.92	9.22			
0.50	0.05	0.39	3.77			
0.60	0.10	0.91	9.25			
0.70	0.10	0.92	9.18			
0.80	0.02	0.10	0.92	1.84	4.20	9.33
0.90	0.02	0.10	0.92	1.82	4.17	9.25
1.00	0.00	0.05	0.44	0.84	1.97	4.37
1.10	0.02	0.09	0.91	1.81	4.13	9.14
1.20	0.02	0.09	0.91	1.80	4.10	9.11
1.30	0.02	0.10	0.90	1.79	4.09	9.08
1.40	0.02	0.10	0.90	1.78	4.07	9.03
1.50	0.01	0.10	0.89	1.79	4.06	9.00
1.60	0.02	0.09	0.89	1.77	4.06	8.98
1.70	0.02	0.10	0.90	1.76	4.03	8.95
1.80	0.01	0.09	0.88	1.74	4.02	8.89
1.90	0.01	0.10	0.90	1.75	4.01	8.85
1.95	0.02	0.09	0.88	1.75	4.00	8.86

**Figure 1:** Surface plot of execution time for Mantegna's algorithm.

The algorithm is more efficient with  $\alpha = 0.5, 1$  because the parameter is used as a denominator of an exponent (lines 36, 38, 39, 45), and the calculation of integer powers is faster.

The execution time slightly decreases when  $\alpha$  increases.

**Table 2:** Detailed analysis of execution time of a single test; the report is generated by the Profiler tool.

Input parameters: $\alpha = 1.5, c = 1, n = 10, N = 1000000$				
Line number	Code	Calls	Total time	% time
38	$v = \text{sigx} * \text{randn}(n, N) ./ \text{abs}(\dots)$	1	7.63 s	82%
43	$w = ((\text{kappa} - 1) * \exp(-\text{abs}(v \dots$	1	1.50 s	16%
45	$z = (1/n^{\text{invalpha}}) * \text{sum}(w) \dots$	1	0.14 s	2%
49	$z = g^{\text{invalpha}} * z;$	1	0.05 s	1%

41	$p = [-17.7767 \ 113.3855 \ \dots]$	1	0.00 s	0%
All other lines			0.00 s	0%
Totals			9.32 s	100%

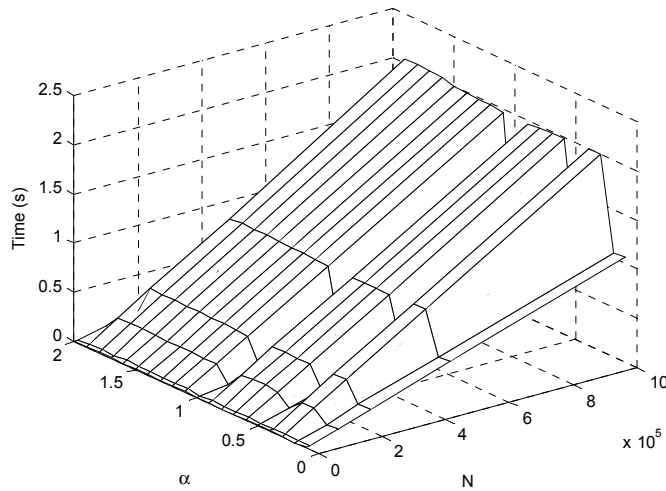
The detailed analysis in table 2 shows that more than 80% of the total time is spent to execute the first step in the algorithm, the calculation of  $v$  (see (2.1)).

### 3.2 McCulloch's algorithm

We used McCulloch's algorithm to create one-dimensional arrays and set location parameter  $\tau$  to 0.

**Table 3:** Execution times (seconds) for McCulloch's algorithm.

$\alpha \backslash N$	1000	10000	100000	200000	450000	1000000
0.10	0.01	0.02	0.11	0.23	0.49	1.07
0.20	0.00	0.01	0.10	0.22	0.48	1.05
0.30	0.00	0.02	0.21	0.42	0.90	2.00
0.40	0.01	0.03	0.20	0.40	0.91	2.00
0.50	0.01	0.01	0.09	0.19	0.44	0.96
0.60	0.00	0.02	0.20	0.40	0.88	1.98
0.70	0.00	0.02	0.21	0.40	0.90	1.98
0.80	0.00	0.02	0.21	0.41	0.89	1.98
0.90	0.01	0.03	0.20	0.40	0.88	1.97
1.00	0.00	0.01	0.05	0.12	0.26	0.57
1.10	0.01	0.02	0.20	0.40	0.89	1.97
1.20	0.01	0.02	0.20	0.40	0.90	1.99
1.30	0.01	0.02	0.20	0.40	0.90	1.98
1.40	0.01	0.03	0.20	0.40	0.90	1.99
1.50	0.00	0.03	0.20	0.41	0.91	2.00
1.60	0.01	0.03	0.21	0.41	0.91	2.03
1.70	0.01	0.02	0.22	0.42	0.91	2.04
1.80	0.00	0.02	0.21	0.42	0.92	2.06
1.90	0.00	0.02	0.21	0.42	0.91	2.05
2.00	0.00	0.01	0.08	0.15	0.32	0.74



**Figure 2:** Surface plot of execution time for McCulloch's algorithm.

The execution time is almost negligible for  $N \leq 10000$ .

The algorithm is more efficient for  $\alpha = 0.1, 0.2, 0.5$  because this parameter is used as a denominator of an exponent (line 127), and the calculation of integer powers is faster.

With  $\alpha = 2$  and  $\alpha = 1$  the function uses different formulae (lines 116, 125).

**Table 4:** Detailed analysis of a single test.

Input parameters: $\alpha = 1.5, c = 1, N = 1000000$				
Line number	Code	Calls	Total time	% time
127	$x = ((\cos((1-\alpha)*\phi)) \dots$	1	1.58 s	78%
112	$w = -\log(\text{rand}(m,n));$	1	0.27 s	13%
113	$\phi = (\text{rand}(m,n) - .5) * \pi;$	1	0.11 s	5%
152	$x = \text{delta} + c * x;$	1	0.02 s	1%
126	else	1	0.02 s	1%
All other lines			0.02 s	1%
Totals			2.02 s	100%

In this case the analysis shows that less than 20% of the total time is spent to extract the random variables, and the time spent in the rest of calculations is comparable to the corresponding time in Mantegna's algorithm, 1.62 s for McCulloch's and 1.69 s for Mantegna's.

### 3.3 Rejection algorithm

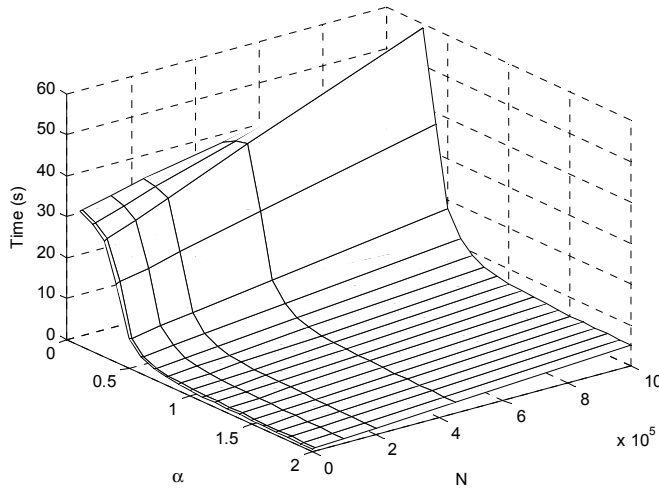
We extracted random numbers with rejection algorithm in the range  $[-10, 10]$ .

**Table 5:** Execution times (seconds) for rejection algorithm.

$\alpha \backslash N$	1000	10000	100000	200000	450000	1000000
0.10	32.67	32.95	34.41	36.42	40.22	48.96
0.20	30.95	30.98	33.21	36.08	41.66	54.36
0.30	28.28	28.60	31.33	34.63	42.51	59.34



0.40	19.21	19.39	20.97	22.86	27.38	37.06
0.50	7.00	7.11	8.08	9.20	11.94	17.92
0.60	3.93	4.00	4.71	5.52	7.53	11.92
0.70	2.49	2.55	3.14	3.82	5.44	9.04
0.80	1.92	1.96	2.48	3.07	4.72	7.66
0.90	1.59	1.61	2.10	2.65	3.96	6.84
1.00	1.28	1.34	1.79	2.28	3.53	6.24
1.10	1.24	1.29	1.71	2.21	3.39	5.97
1.20	1.18	1.22	1.64	2.12	3.26	5.78
1.30	1.14	1.19	1.58	2.05	3.16	5.63
1.40	1.07	1.11	1.52	1.97	3.06	5.48
1.50	1.02	1.05	1.45	1.91	2.97	5.35
1.60	0.91	0.95	1.34	1.77	2.86	5.19
1.70	0.90	0.93	1.32	1.75	2.83	5.18
1.80	0.88	0.91	1.30	1.73	2.79	5.11
1.90	0.82	0.85	1.24	1.67	2.74	5.06
2.00	0.74	0.79	1.16	1.60	2.66	4.98



**Figure 3:** Surface plot of execution time for rejection algorithm, the  $\alpha$  axis is inverted to ease reading.

With low values for  $\alpha$  Matlab produces the warning: Maximum function count exceeded in quadl.m, during the calculations for the probability density of the Lévy distribution; for the tests we suppressed this warning.

The execution time shows a strong dependence on the value of  $\alpha$ , the main contribution comes from the calculation of Lévy pdf, for example increasing  $\alpha$  from 0.2 to 1.8 reduces the time from about 30 s to less than one second (see tables 6 and 7). The execution time is also affected by the ratio of accepted points to the total produced random numbers.

With less than 100000 points, the time spent to extract random numbers is almost negligible compared to the calculation of Lévy pdf.

**Table 6:** Detailed analysis of a single test.

Input parameters: $\alpha = 0.2$ , $c = 1$ , $M = 10$ , $N = 1000$				
Line number	Code	Calls	Total time	% time
31	<code>fx = levypdf([-M:0.2:M], ...</code>	1	29.98 s	99.9%
32	<code>x2M = levypdf(0, alpha, g...</code>	1	0.02 s	0.1%
40	<code>end</code>	34474	0.00 s	0%
38	<code>x(k) = 2*M*rand-M;</code>	34474	0.00 s	0%
39	<code>x2 = rand*x2M;</code>	34474	0.00 s	0%
All other lines			0.00 s	0%
Totals			30.00 s	100%

**Table 7:** Detailed analysis of a single test.

Input parameters: $\alpha = 1.8$ , $c = 1$ , $M = 10$ , $N = 1000000$				
Line number	Code	Calls	Total time	% time
40	<code>end</code>	5676623	1.08 s	21%
31	<code>fx = levypdf([-M:0.2:M], ...</code>	1	0.83 s	16%
38	<code>x(k) = 2*M*rand-M;</code>	5676623	0.37 s	7%
39	<code>x2 = rand*x2M;</code>	5676623	0.27 s	5%
37	<code>while(x2 &gt; fx(1+floor(...</code>	1000000	0.19 s	4%
All other lines			2.34 s	46%
Totals			5.08 s	100%

## 4 Conclusions

McCulloch's algorithm is the fastest for every value of  $N$  and  $\alpha$ , this is due to the low number of random numbers used to calculate each output value: it uses two uniform random variables, while Mantegna's uses  $2n$  normal variables (where  $n$  is the number of iterations); with the rejection algorithm the ratio of accepted points to the total produced random numbers is strongly dependent on the value of  $\alpha$  and on the width of the considered interval, for the cases we studied it approximately varies from  $1/35$  to  $1/6$  (see tables 6 and 7).

The rejection algorithm is faster than Mantegna's for large values of  $N$  and  $\alpha$ , it could be more efficient using a different method to calculate the Lévy probability density, instead of numerical integration.

Mantegna's algorithm can be a valid choice for hardware implementation, because it relies on simpler calculations than McCulloch's; it is faster when  $\alpha$  is in the range  $[0.75, 1.95]$ .

## 5 Acknowledgements

The author would like to acknowledge support by Italian MIUR project “High Frequency Dynamics in Financial Markets”

## Appendix: Matlab programs

```
1      % MANTEGNA.M
2      % Stable random number generator
3
4      % Based on the method of R. N. Mantegna "Fast, accurate algorithm for
5      % numerical simulation of Lévy stable stochastic processes"
6      % Physical Review E 49 4677-83 (1994)
7
8      function z = mantegna(alpha, c, n, N)
9
10     % Errortraps:
11     if (alpha < 0.3 | alpha > 1.99)
12         disp('Valid range for alpha is [0.3;1.99].')
13         z = NaN * zeros(1,N);
14         return
15     end
16     if (c <= 0)
17         disp('c must be positive.')
18         z = NaN * zeros(1,N);
19         return
20     end
21     if (n < 1)
22         disp('n must be positive.')
23         z = NaN * zeros(1,N);
24         return
25     end
26     if nargin<4
27         N=1;
28     end
29     if (N <= 0)
30         disp('N must be positive.')
31         z = NaN;
32         return
33     end
34
35     invalpha = 1/alpha;
36     sigx = ((gamma(1+alpha)*sin(pi*alpha/2))/(gamma((1+alpha)/2)...
37         *alpha*2^((alpha-1)/2)))^invalpha;
38     v = sigx*randn(n,N)./abs(randn(n,N)).^invalpha;
39     kappa = (alpha*gamma((alpha+1)/(2*alpha)))/gamma(invalpha)...
40         *((alpha*gamma((alpha+1)/2))/(gamma(1+alpha)*sin(pi*alpha/2)))^invalpha;
41     p = [-17.7767 113.3855 -281.5879 337.5439 -193.5494 44.8754];
42     c = polyval(p, alpha);
43     w = ((kappa-1)*exp(-abs(v)/c)+1).*v;
44     if(n>1)
45         z = (1/n^invalpha)*sum(w);
46     else
47         z = w;
48     end
49     z = c^invalpha*z;
50
51 % STABRND.M
52 % Stable Random Number Generator (McCulloch 12/18/96)
53
54 function [x] = stabrnd(alpha, beta, c, delta, m, n)
55
56 % Returns m x n matrix of iid stable random numbers with
57 % characteristic exponent alpha in [.1,2], skewness parameter
58 % beta in [-1,1], scale c > 0, and location parameter delta.
59 % Based on the method of J.M. Chambers, C.L. Mallows and B.W.
60 % Stuck, "A Method for Simulating Stable Random Variables,"
61 % JASA 71 (1976): 340-4.
62 % Encoded in MATLAB by J. Huston McCulloch, Ohio State
63 % University Econ. Dept. (mcculloch.2@osu.edu). This 12/18/96
64 % version uses 2*m*n calls to RAND, and does not rely on
65 % the STATISTICS toolbox.
```

```

16 % The CMS method is applied in such a way that x will have the
17 % log characteristic function
18 %     log E exp(ixt) = i*delta*t + psi(c*t),
19 %     where
20 %     psi(t) = -abs(t)^alpha*(1-i*beta*sign(t)*tan(pi*alpha/2))
21 %             for alpha ~= 1,
22 %             = -abs(t)*(1+i*beta*(2/pi)*sign(t)*log(abs(t))),
23 %             for alpha = 1.
24 % With this parameterization, the stable cdf S(x; alpha, beta,
25 % c, delta) equals S((x-delta)/c; alpha, beta, 1, 0). See my
26 % "On the parametrization of the afocal stable distributions,"
27 % _Bull. London Math. Soc._ 28 (1996): 651-55, for details.
28 % When alpha = 2, the distribution is Gaussian with mean delta
29 % and variance 2*c^2, and beta has no effect.
30 % When alpha > 1, the mean is delta for all beta. When alpha
31 % <= 1, the mean is undefined.
32 % When beta = 0, the distribution is symmetrical and delta is
33 % the median for all alpha. When alpha = 1 and beta = 0, the
34 % distribution is Cauchy (arctangent) with median delta.
35 % When the submitted alpha is > 2 or < .1, or beta is outside
36 % [-1,1], an error message is generated and x is returned as a
37 % matrix of NaNs.
38 % Alpha < .1 is not allowed here because of the non-negligible
39 % probability of overflows.
40
41 % If you're only interested in the symmetric cases, you may just
42 % set beta = 0 and skip the following considerations:
43 % When beta > 0 (< 0), the distribution is skewed to the right
44 % (left).
45 % When alpha < 1, delta, as defined above, is the unique fractile
46 % that is invariant under averaging of iid contributions. I
47 % call such a fractile a "focus of stability." This, like the
48 % mean, is a natural location parameter.
49 % When alpha = 1, either every fractile is a focus of stability,
50 % as in the beta = 0 Cauchy case, or else there is no focus of
51 % stability at all, as is the case for beta ~=0. In the latter
52 % cases, which I call "afocal," delta is just an arbitrary
53 % fractile that has a simple relation to the c.f.
54 % When alpha > 1 and beta > 0, med(x) must lie very far below
55 % the mean as alpha approaches 1 from above. Furthermore, as
56 % alpha approaches 1 from below, med(x) must lie very far above
57 % the focus of stability when beta > 0. If beta ~= 0, there
58 % is therefore a discontinuity in the distribution as a function
59 % of alpha as alpha passes 1, when delta is held constant.
60 % CMS, following an insight of Vladimir Zolotarev, remove this
61 % discontinuity by subtracting
62 %     beta*c*tan(pi*alpha/2)
63 % (equivalent to their -tan(alpha*phi0)) from x for alpha ~=1
64 % in their program RSTAB, a.k.a. RNSTA in IMSL (formerly GGSTA).
65 % The result is a random number whose distribution is a contin-
66 % uous function of alpha, but whose location parameter (which I
67 % call zeta) is a shifted version of delta that has no known
68 % interpretation other than computational convenience.
69 % The present program restores the more meaningful "delta"
70 % parameterization by using the CMS (4.1), but with
71 % beta*c*tan(pi*alpha/2) added back in (ie with their initial
72 % tan(alpha*phi0) deleted). RNSTA therefore gives different
73 % results than the present program when beta ~= 0. However,
74 % the present beta is equivalent to the CMS beta' (BPRIME).
75 % Rather than using the CMS D2 and exp2 functions to compensate
76 % for the ill-condition of the CMS (4.1) when alpha is very
77 % near 1, the present program merely fudges these cases by
78 % computing x from their (2.4) and adjusting for
79 % beta*c*tan(pi*alpha/2) when alpha is within 1.e-8 of 1.
80 % This should make no difference for simulation results with
81 % samples of size less than approximately 10^8, and then
82 % only when the desired alpha is within 1.e-8 of 1, but not
83 % equal to 1.
84 % The frequently used Gaussian and symmetric cases are coded
85 % separately so as to speed up execution.
86
87 % Additional references:
88 % V.M. Zolotarev, _One Dimensional Stable Laws_, Amer. Math.
89 % Soc., 1986.
90 % G. Samorodnitsky and M.S. Taqqu, _Stable Non-Gaussian Random
91 % Processes_, Chapman & Hill, 1994.
92 % A. Janicki and A. Weron, _Simulaton and Chaotic Behavior of
93 % Alpha-Stable Stochastic Processes_, Dekker, 1994.
94 % J.H. McCulloch, "Financial Applications of Stable Distributons,"

```

```

95 % _Handbook of Statistics_ Vol. 14, forthcoming early 1997.
96
97 % Errortraps:
98 if alpha < .1 | alpha > 2
99     disp('Alpha must be in [.1,2] for function STABRND.')
100     alpha
101     x = NaN * zeros(m,n);
102     return
103 end
104 if abs(beta) > 1
105     disp('Beta must be in [-1,1] for function STABRND.')
106     beta
107     x = NaN * zeros(m,n);
108     return
109 end
110
111 % Generate exponential w and uniform phi:
112 w = -log(rand(m,n));
113 phi = (rand(m,n)-.5)*pi;
114
115 % Gaussian case (Box-Muller):
116 if alpha == 2
117     x = (2*sqrt(w) .* sin(phi));
118     x = delta + c*x;
119     return
120 end
121
122 % Symmetrical cases:
123 if beta == 0
124     if alpha == 1 % Cauchy case
125         x = tan(phi);
126     else
127         x = ((cos((1-alpha)*phi) ./ w) .^ (1/alpha - 1) ...
128             .* sin(alpha * phi) ./ cos(phi) .^ (1/alpha)); ...
129     end
130
131 % General cases:
132 else
133     cosphi = cos(phi);
134     if abs(alpha-1) > 1.e-8
135         zeta = beta * tan(pi*alpha/2);
136         aphi = alpha * phi;
137         alphi = (1 - alpha) * phi;
138         x = ((sin(aphi) + zeta * cos(aphi)) ./ cosphi) ...
139             .* ((cos(alphi) + zeta * sin(alphi)) ...
140                ./ (w .* cosphi)) .^ ((1-alpha)/alpha);
141     else
142         bphi = (pi/2) + beta * phi;
143         x = (2/pi) * (bphi .* tan(phi) - beta * log((pi/2) * w ...
144             .* cosphi ./ bphi));
145         if alpha ~= 1
146             x = x + beta * tan(pi * alpha/2);
147         end
148     end
149 end
150
151 % Finale:
152 x = delta + c * x;
153 return
154 % End of STABRND.M

```

```

1 % REJFAST.M
2 % Stable random number generator, based on rejection method
3
4 function x = rejfast(alpha, c, M, N)
5
6 % Errortraps:
7 if (alpha <= 0 | alpha > 2)
8     disp('Valid range for alpha is (0;2].')
9     x = NaN * zeros(1,N);
10    return
11 end
12 if (c <= 0)
13     disp('c must be positive.')
14     x = NaN * zeros(1,N);
15     return
16 end
17 if (M <= 0)
18     disp('M must be positive.')

```

```

19         x = NaN * zeros(1,N);
20
21     end
22     if nargin < 4
23         N=1;
24     end
25     if (N <= 0)
26         disp('N must be positive.')
27         x = NaN;
28         return
29     end
30
31     step=0.2;
32     fx = levypdf([-M:step:M], alpha, c);
33     x2M = levypdf(0, alpha, c);
34     x = zeros(1,N);
35     for k = 1:N;
36         x2 = x2M+1;
37         x(k) = 0;
38         while(x2 > fx(1+floor((x(k)+step/2+M)/step)))
39             x(k) = 2*M*rand-M;
40             x2 = rand*x2M;
41         end
42     end

```

## References

- 1 K. Sato, Levy processes and infinitely divisible distributions, Cambridge University Press, Cambridge, 1999.
- 2 J. Bertoin, Lévy processes, Cambridge University Press, Cambridge, 1996.
- 3 W. Feller: An introduction to probability theory and its applications, John Wiley & Sons, Chichester, 1968.
- 4 O. Kallenberg: Foundations of modern probability, Springer-Verlag, New York, 2002.
- 5 W. Schoutens, Lévy Processes in Finance, John Wiley & Sons, Chichester, 2003.
- 6 A. Janicki, A. Weron, Simulation and chaotic behavior of  $\alpha$ -stable stochastic processes, Marcel Dekker, New York, 1994.
- 7 R.N. Mantegna, Fast, accurate algorithm for numerical simulation of Levy stable stochastic processes, Physical Review E, vol. 49, 4677-4683, 1994.
- 8 <http://economics.sbs.ohio-state.edu/jhm/jhm.html>
- 9 J.M. Chambers, C.L. Mallows, and B.W. Stuck, A method for simulating stable random variables, Journal of the American Statistical Association, vol. 71, 340-344, 1976
- 10 W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, Numerical Recipes in C, Cambridge University Press, Cambridge, 1993.
- 11 Hewlett-Packard Company, Software Guide HP Compaq Notebook Series, document number 347403-001, 2003.