Generating Generalized Inverse Gaussian Random Variates by Fast Inversion

Leydold, Josef; Hörmann, Wolfgang

Published: 01/11/2009

Document Version
Publisher's PDF, also known as Version of record

Generating Generalized Inverse Gaussian Random Variates by Fast Inversion

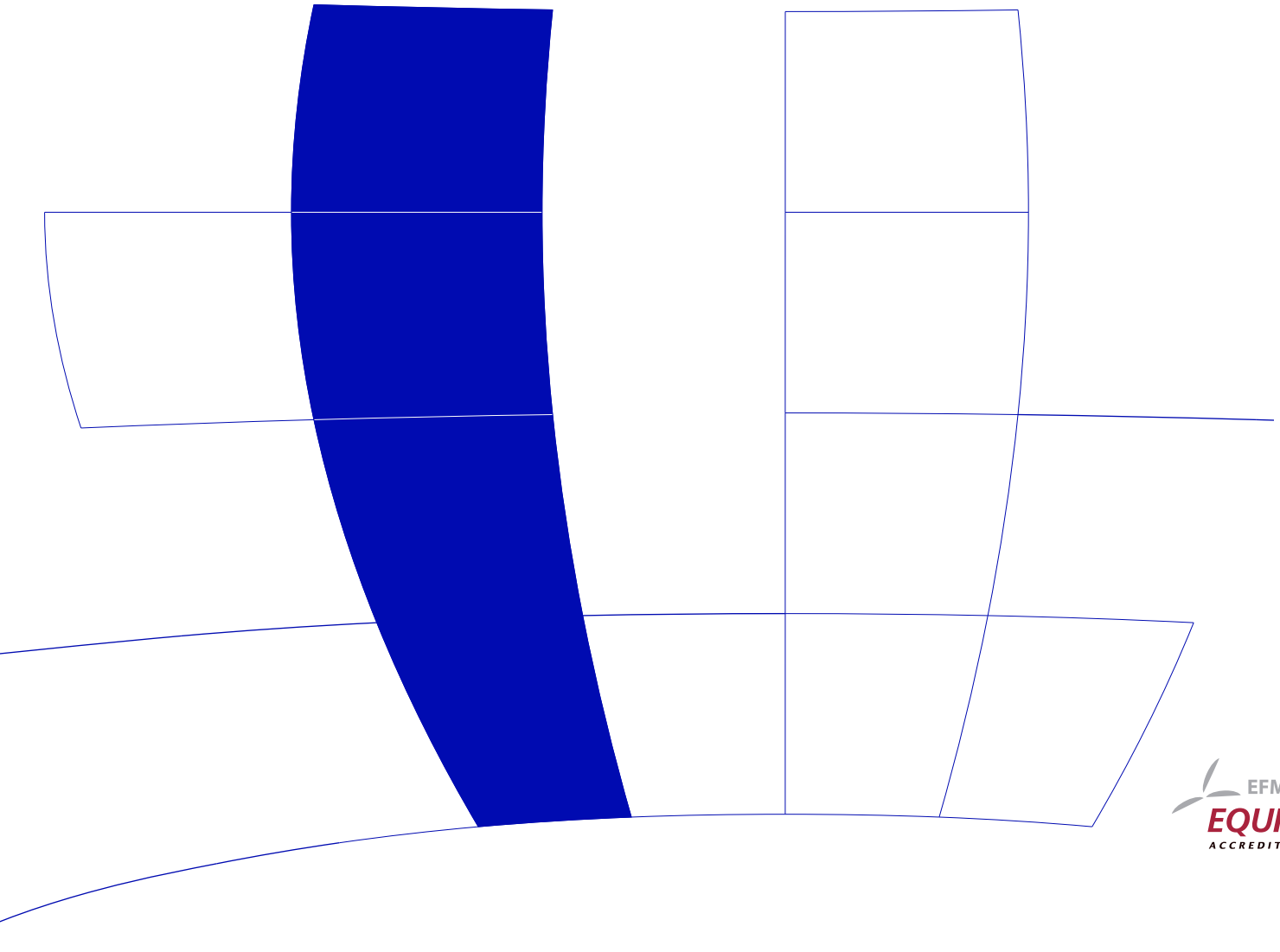
Josef Leydold, Wolfgang Hörmann

Department of Statistics and Mathematics
WU Wirtschaftsuniversität Wien

Research Report Series

Report 95
November 2009

http://statmath.wu.ac.at/



Generating Generalized Inverse Gaussian Random Variates by Fast Inversion

Josef Leydold[a,*], Wolfgang Hörmann[b]

[a] *Department of Statistics and Mathematics, WU (Vienna University of Economics and Business), Augasse 2-6, A-1090 Wien, Austria*
[b] *Department of Industrial Engineering, Boğaziçi University, 34342 Bebek-İstanbul, Turkey*

Abstract

We demonstrate that for the fast numerical inversion of the (generalized) inverse Gaussian distribution two algorithms based on polynomial interpolation are well-suited. Their precision is close to machine precision and they are much faster than the bisection method recently proposed by Y. Lai.

Key words: generalized inverse Gaussian distribution, random variate generation, numerical inversion
2000 MSC: 65C05, 65C10

1. Introduction

There are two recent trends in the framework of financial engineering. On the one hand the development of more realistic models leads to an increased application of less frequently used distributions. On the other hand the complexity of these new models require more sophisticated numerical algorithms beyond naïve Monte Carlo methods to estimate option prices or risk measures. Among these copula based methods and using of low discrepancy sequences (so called quasi-Monte Carlo methods) are of great importance. However, these methods require a monotone one-to-one transformation of uniform $(0, 1)$ pseudo-random numbers into random variates from the requested distribution, i.e., the application of the *inversion* method is crucial.

Unfortunately, the distributions in these new models are often quite nasty to handle. In particular, computing the inverse of the cumulative distribution function (CDF) is extremely expensive. This is also the case for the *generalized inverse Gaussian* (GIG) distribution. Its density is given by

$$f_{\text{gig}}(x;\theta,\psi,\chi) = \begin{cases} \dfrac{(\psi/\chi)^{\theta/2}}{2\,K_\theta(\psi\chi)}\; x^{\theta-1} \exp\left(-\dfrac{1}{2}\left(\dfrac{\chi}{x}+\psi x\right)\right) , & x > 0 \\ 0\,, & x \le 0 \end{cases} \tag{1}$$

*Corresponding Author: Department of Statistics and Mathematics, WU (Vienna University of Economics and Business), Augasse 2-6, A-1090 Wien, Austria. Phone: +43 1 31336-4695, Fax: +43 1 31336-774

Email addresses: josef.leydold@wu.ac.at (Josef Leydold), hormannw@boun.edu.tr (Wolfgang Hörmann)

November 19, 2009

where $K_\theta(\cdot)$ denotes the modified Bessel function of third kind with index θ [1, 2]. The *inverse Gaussian* distribution (IG, also called the *Wald* distribution) is a special case of (1) with $\theta = -\frac{1}{2}$. Its density is often written as

$$f_{\mathrm{ig}}(x;\mu,\lambda) = \sqrt{\frac{\lambda}{2\pi x^3}}\,\exp\left(-\frac{\lambda(x-\mu)^2}{2\mu^2 x}\right)\,, \qquad x>0\,. \tag{2}$$

For such distributions the inverse CDF is not known explicitly. Thus Lai [3] proposed to solve the nonlinear equation $X = F^{-1}(U)$ for each sampled uniform $(0,1)$ random number U by means of the bisection method to obtain an approximate inversion algorithm for the IG distribution using the representation

$$F_{\mathrm{ig}}(x;\mu,\lambda) = \Phi\left(\sqrt{\frac{\lambda}{x}}\left(\frac{x}{\mu}-1\right)\right) + \exp\left(\frac{2\lambda}{\mu}\right)\Phi\left(-\sqrt{\frac{\lambda}{x}}\left(\frac{x}{\mu}+1\right)\right) \tag{3}$$

for its CDF. Here $\Phi(\cdot)$ denotes the standard Gaussian distribution function. He argues that Newton-Raphson method does not always converge. However, due to its slow convergence the bisection method is usually considered as a last resort.

For the case of fixed parameters we propose two much faster methods which we presented in two papers [4, 5]. These are based on polynomial approximations of the inversion CDF of the (generalized) inversion Gaussian distribution In the remaining part of this note we shortly describe these algorithms and report our computational experience.

2. Inversion by Polynomial Interpolation

The basic idea of that approach is quite simple: Evaluate the CDF F at a couple of points x_i and interpolate the nodes $(u_i = F(x_i), x_i)$ by means of polynomials. This is done in the *setup* part of the algorithm where all required coefficients are computed and stored in a table. Notice that there is no necessity to compute $F^{-1}(u_i)$ in the setup. The table is then used in the *sampling* part to compute the approximate inverse CDF for a given U.

There exist several variants for this task. In [4] we argued in favor of (cubic) Hermite interpolation:

- For smooth CDFs the approximation error can be bounded.
- It allows to estimate the approximation error during the setup "on the fly". Thus the maximal tolerated error can be controlled by the user.
- It is possible to adjust the interpolant stepwise by adding additional points without recomputing all polynomials. Thus we start with a rough approximation with only a few nodes and recursively add more nodes wherever we need improvements.
- There are simple conditions on the coefficients of the polynomials that ensure monotonicity of the interpolant.

Imai and Tan [6] report that this method performs very well for the generalized hyperbolic distribution and the normal-inverse Gaussian distribution. However, they mention a slow setup as they use numerical integration to compute the CDF.

In [5] we suggest a new algorithm that combines Newton's interpolation formula with adaptive Gauss-Lobatto integration to evaluate the differences of the CDF. Thus only the probability density function (PDF) of the target distribution is required. The resulting setup is much faster than using the algorithm of [4] together with a packed quadrature routine like QUADPACK [7] to compute the CDF. Moreover, we observed that it is numerically more robust when the maximal tolerated approximation error is close to machine precision.

A main concern of any numerical inversion algorithm must be the control of the approximation error, i.e., the deviation of the approximate inverse CDF F_a^{-1} from the exact function F^{-1}. We are convinced that the *u-error* defined by

$$\varepsilon_u(u) = |u - F(F_a^{-1}(u))| \tag{4}$$

is well-suited for this task. In particular it can be computed during the setup and it can be interpreted with respect to the resolution of the underlying uniform pseudo-random number generator or low discrepancy set (see [5] for details). In fact goodness-of-fit tests like the Kolmogorov-Smirnov test or the χ^2 test look exactly at that deviation. We call the maximal tolerated u-error the *u-resolution* of the algorithm in the sequel.

The number of required nodes (and thus the table size and setup time) depend on the requested u-resolution and on the target distribution. The marginal generation time is however very fast and (almost[1]) independent from the target distribution.

3. Computational Experiments

We have worked out all required details and implemented these two algorithms in our C library UNU.RAN [8]. We used the names `HINV` for the Hermite interpolation algorithm of [4] and `PINV` for the Newton interpolation with Gauss-Lobatto integration algorithm of [5]. The UNU.RAN library is also accessible from the R statistical programming language [9] by using our package *Runuran* [10].

The following example shows how method `PINV` can be used in R to draw a sample from an IG distribution. Notice that setup and sampling part are separated. The tables for the interpolant are stored in object `gen`. Thus there is no necessity to rerun the setup every time. The argument `uresolution` is optional.

```
> ## load 'Runuran' library
> library(Runuran)
> ## define PDF for inverse Gaussian
> ig.pdf <- function(x,mu,lambda) {
>               sqrt(1/x^3) * exp(-(lambda*(x-mu)^2)/(2*mu^2*x)) }
> ## run setup (create generator object for mu=3 and lambda=2)
> gen <- pinv.new(pdf=ig.pdf, lb=0, ub=Inf, center=1, uresolution=1.e-10,
                  mu=3, lambda=2)
```

[1] The runtime is only influenced by the table size due to cache effects.

```
> ## draw a sample of size 5
> ur(gen,5)
[1] 0.6151017 3.5961721 0.7396323 0.9170969 1.7725783
> ## compute (approximate) inverse CDF for u=0.9
> uq(gen,0.9)
[1] 6.84101
```

Remark 1. Our algorithms are designed as black-box algorithms, i.e., the user can provide a function that evaluates the PDF together with a "typical" point of the target distribution, and a maximal tolerated approximation error. Thus our example could be modified for quite arbitrary distributions. (Of course it does not work for all distributions but for bounded smooth densities with moderate or low tails we have never observed problems.)

Coding the PDF can be avoided as the package already has built-in distributions. Thus `udig(mu=3, lambda=2)` creates an object that contains all required information (including the PDF) about the IG distribution.

```
> ## run setup (create generator object for IG with mu=3 and lambda=2)
> gen <- pinvd.new( udig(mu=3, lambda=2), uresolution=1.e-12 )
> ## draw a sample of size 5
> ur(gen,5)
[1] 8.2701337 0.9978058 1.1816979 0.4788929 2.0146778
```

It is not astonishing that this method is also suitable for sampling from the Generalized Inverse Gaussian distribution using density (1).

```
> gen <- pinvd.new( udgig(theta=-1, psi=3, chi=2) )
> ur(gen,5)
[1] 0.1987231 0.8583419 1.4482379 0.2322391 1.6078295
```

Our extensive tests with the IG distribution and CDF (3) for many different parameters show that the actually observed u-error never exceeds a requested u-resolution of 10^{-10} or greater. For a u-resolution of 10^{-12} we came across a few parameter settings where the maximal observed u-error was slightly too large, but never larger than $1.16 \cdot 10^{-12}$.

We also measured both the setup time and the marginal generation time for our methods `PINV` and `HINV` as well as for the bisection method `BIS`. In addition we added a modified regula falsi algorithm `RF` [11] that uses bisection when convergence is too slow. For both `RF` and `BIS` we used the 10th and 90th percentile for the starting interval as well as a table of size 100 (a table of size 1000 did not show much improvement). All methods are available in our UNU.RAN library.

Table 1 summarizes our timing results. The reported timings are relative to sampling one exponential distributed random variate using inversion (i.e., `-log(1-U)`). As can easily be seen, the methods based on polynomial interpolation have very fast marginal generation times (faster than generating an exponential random variate) but require some setup. This is worthwhile for moderate (or larger) sample sizes; the break-even point is somewhere between 500 and 10.000, depending on the parameters, algorithms and resolution. Regula falsi is faster only when samples smaller than that break-even point

Method	ε_u	IG(0.45, 0.3)	IG(0.5, 18)	IG(4.3, 0.6)	GIG(−5, 2, 3)
PINV-5	10^{-8}	0.53 (12393)	0.53 (8142)	0.52 (15528)	0.53 (11109)
	10^{-10}	0.53 (19421)	0.52 (15346)	0.53 (22440)	0.52 (18898)
	10^{-12}	0.53 (32838)	0.53 (28688)	0.53 (37003)	0.53 (35991)
HINV-3	10^{-8}	0.65 (4672)	0.65 (3992)	0.66 (4532)	–
	10^{-10}	0.66 (13214)	0.66 (12438)	0.67 (13145)	–
	10^{-12}	0.74 (42148)	0.73 (39916)	0.77 (40387)	–
RF-2	10^{-8}	24.52 (114)	19.99 (114)	22.71 (92)	–
	10^{-10}	26.54 (170)	21.32 (131)	24.41 (96)	–
	10^{-12}	27.90 (138)	22.62 (131)	25.66 (93)	–
RF-100	10^{-8}	16.04 (2101)	13.22 (1771)	14.34 (1786)	–
	10^{-10}	17.46 (2261)	14.94 (1931)	15.35 (1998)	–
	10^{-12}	19.10 (2486)	16.10 (2067)	17.18 (2092)	–
BIS-2	10^{-8}	83.63 (134)	72.67 (142)	69.68 (56)	–
	10^{-10}	106.89 (127)	92.96 (98)	88.94 (118)	–
	10^{-12}	130.28 (131)	113.21 (150)	108.12 (83)	–
BIS-100	10^{-8}	66.57 (2121)	57.47 (1754)	55.69 (1761)	–
	10^{-10}	89.96 (2257)	77.84 (1921)	75.09 (2032)	–
	10^{-12}	113.30 (2456)	98.20 (2052)	94.28 (2076)	–

Table 1: Marginal generation time and setup time (in parenthesis) for some IG(μ, λ) and an GIG(θ, ψ, χ) distribution and u-resolutions ε_u. `PINV-5`: Newton interpolation of order 5. `HINV-3`: cubic Hermite interpolation. `RF-2` & `RF-100`: modified regula falsi with a table of starting points of size 2 and 100. `BIS-2` & `BIS-100` bisection method with a table of starting points of size 2 and 100. The timings are relative to sampling one exponential distributed random variate using inversion (i.e., `-log(1-U)`).
Time unit is 0.105 μsec.; Intel Core Duo 2.0 GHz, Linux 2.6.26, GCC 4.3.4.

are required. We could not see any benefit from using the bisection method as it was always clearly slower than regula falsi. We did not include the timings for GIG for the methods that require the CDF. We tried numerical integration to obtain its CDF using the GNU Scientific library [12] but the runtimes where discouraging slow.

Remark 2. Our computational experiments (as well as those of [6]) make clear that the problems of `HINV` ("HL method") reported by Lai [3] in his last two paragraphs can only be attributed to a problem in his implementation. Indeed parameter δ is used for the initial rough approximation and has hardly any influence on the accuracy of the resulting interpolant.

4. Conclusion

Our algorithms are well-suited for generating random variates from the (generalized) inverse Gaussian distribution by numerical inversion. In all our experiments the maximal observed u-error was always within acceptable deviation from the requested u-resolution. After a moderate setup our algorithms are faster than the standard method for generating exponential variates and much faster than numerical root finding algorithms such as, e.g., the bisection method that has been suggested by Lai [3]. The speed up factors we observed are between 50 and 220. Our algorithms are available as library UNU.RAN and as R package *Runuran* which can both be downloaded from our web site.

References

[1] B. Jørgensen, Statistical Properties of the Generalized Inverse Gaussian Distribution, vol. 9 of *Lecture Notes in Statistics*, Springer-Verlag, New York-Berlin, 1982.

[2] M. Abramowitz, I. A. Stegun (Eds.), Handbook of mathematical functions, Dover, New York, 9th edn., ISBN 0486-612-724, 1972.

[3] Y. Lai, Generating inverse Gaussian random variates by approximation, Computational Statistics & Data Analysis 53 (10) (2009) 3553–3559, doi:10.1016/j.csda.2009.03.007.

[4] W. Hörmann, J. Leydold, Continuous Random Variate Generation by Fast Numerical Inversion, ACM Trans. Model. Comput. Simul. 13 (4) (2003) 347–362.

[5] G. Derflinger, W. Hörmann, J. Leydold, Random Variate Generation by Numerical Inversion when only the Density Is Known, ACM Trans. Model. Comput. Simul. To appear. Preprint available as Research Report Series of the Department of Statistics and Mathematics at WU Vienna, No. 90, June 2009. `http://epub.wu.ac.at/`, `oai:epub.wu-wien.ac.at:epub-wu-01_f41`.

[6] J. Imai, K. S. Tan, an accelerating quasi-Monte Carlo method for option pricing under the generalized hyperbolic Lévy process, SIAM J. Sci. Comput. 31 (3) (2009) 2282–2302, doi: 10.1137/080727713.

[7] R. Piessens, E. de Doncker-Kapenga, C. W. Überhuber, D. K. Kahaner, QUADPACK. A subroutine package for automatic integration, vol. 1 of *Springer Series in Computational Mathematics*, Springer-Verlag, Berlin, 1983.

[8] J. Leydold, W. Hörmann, UNU.RAN – A Library for Non-Uniform Universal Random Variate Generation, Version 1.5, Department of Statistics and Mathematics, WU Wien, A-1090 Wien, Austria, `http://statmath.wu-wien.ac.at/unuran/`, 2009.

[9] R Development Core Team, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, URL `http://www.R-project.org`, ISBN 3-900051-07-0, 2009.

[10] J. Leydold, W. Hörmann, Runuran – R interface to the UNU.RAN random variate generators, Version 0.11, Department of Statistics and Mathematics, WU Wien, A-1090 Wien, Austria, `http://cran.r-project.org/`, 2009.

[11] A. Neumaier, Introduction to numerical analysis, Cambridge University Press, Cambridge, 2001.

[12] M. Galassi, et al., GNU Scientific Library, Version 1.13, `http://www.gnu.org/software/gsl/`, 2009.