DSC 2003 Working Papers
(Draft Versions)
http://www.ci.tuwien.ac.at/Conferences/DSC-2003/

Automatic Nonuniform Random Variate Generation in R

Günter Tirler and Josef Leydold

Institut für Statistik, WU Wien
Augasse 2-6, A-1090 Vienna, Austria, EU

Abstract

Random variate generation is an important tool in statistical computing. Many programms for simulation or statistical computing (e.g. `R`) provide a collection of random variate generators for many standard distributions. However, as statistical modeling has become more sophisticated there is demand for larger classes of distributions. Adding generators for newly required distribution seems not to be the solution to this problem. Instead so called automatic (or black-box) methods have been developed in the last decade for sampling from fairly large classes of distributions with a single piece of code. For such algorithms a data about the distributions must be given; typically the density function (or probability mass function), and (maybe) the (approximate) location of the mode. In this contribution we show how such algorithms work and suggest an interface for R as an example of a statistical library.

1 Introduction

Random variate generation is an important tool in statistical computing. Many programms for simulation or statistical computing (e.g. `R`) provide a collection of random variate generators for many standard distributions. There exists a vast literature on generation methods for standard distributions; see, for example, the books by Devroye [4], Dagpunar [3], Gentle [5], or Knuth [8]. These books are usually the source for algorithms implemented in software. These algorithms are often especially designed for a particular distribution and tailored to the features of each probability density function. The designing goals for these methods are fast

generators and/or simple code. However, as statistical modeling has become more sophisticated there is demand for larger classes of (non-standard) distributions. Adding generators for newly required distribution seems not to be the solution to this problem.

In the last decade so called automatic (also called universal or black-box) methods have been developed for sampling from fairly large classes of distributions with a single piece of code. For such algorithms a data about the distributions must be given; typically the density function (or probability mass function), and (maybe) the (approximate) location of the mode. Obviously these universal methods need some setup step, in opposition to special generators, e.g., to the Box-Muller method [2]. Nevertheless, we always can select between a fast setup step and slow marginal generation times or (very) fast marginal generation times at the expense of a time consuming setup step. Some of the algorithms can be adjusted by a single parameter to the needs of the current situation. Although originally motivated to generate from non-standard distributions these universal methods have advantages which makes their usage attractive even for standard distributions. For univariate continuous distribution there are methods like *Transformed Density Rejection* [6], or algorithms based on a variant of the ratio-of-uniforms method [9] or on piecewise constant hat functions [1]. They have the following properties in common [see 10, for details]:

- Only one piece of code, well implemented and debugged only once, is required.
- By a simple parameter it is possible to choose between fast setup with slow marginal generation time and vice versa.
- It can sample from truncated distributions.
- The algorithms can be made as close to inversion as requested.
- The marginal generation time does not depend on the density function and is faster than many of the specialized generators (even for the normal distribution).
- It can be used for variance reduction techniques.
- The quality of the generated random numbers only depends on the underlying uniform random number generator.

For more details on these and many other universal methods see the forthcoming monograph by Hörmann, Leydold, and Derflinger [7].

2 UNU.RAN

Universal methods are usually harder to implement since there is a setup step where the necessary constants for the generation steps have to be precomputed. Moreover, it might be necessary whether a particular method works with the given distribution. Thus we have implemented many of these automatic algorithms using ANSI C in

a library called UNU.RAN (*Universal Non-uniform RANdom variate generators*). Our main goal was to get a portable, flexible and robust program, see Leydold et al. [11]. It is implemented using an object oriented programming paradigma. Using this library first a generator object has to be created that then can be used to sample from the given distribution. Thus it is easy to exchanged distributions in simulations. Moreover each generator object may have its own pseudo-random number generator or share one with other generators.

There exist two application programming interfaces: A "traditional" API where the generator object is created via `new` call and where replacement functions are used to replace default parameters by user defined. A second interface uses a string which describes both the desired distribution and the choosen generation method.

3 An R interface

We have proposed an R interface for this library, called *Runuran*. This extends the usual functionallity in R for random variate generation in several ways. First it is easy to sample from non-standard distributions. Secondly it is possible to choose different methods for a particular distribution, which is not yet possible in R except for normal distributions.

The object oriented approach of UNU.RAN is reproduced using S4 classes. This provides nearly all features of UNU.RAN and is very simple to use. In the following chapter we will describe the main ideas of our implementation and give examples of how to use UNU.RAN in R.

As we can see in the following simple example it is very easy to create non uniform random numbers for complicated distributions, e.g. the hyperbolic distribution:

```
> hyp = new("unur","cont;pdf=\"1/sqrt(1+x^2)*exp(-2*sqrt(1+x^2)+x)\"")
> x<-sample.unur(hyp,10000)
> hist(x,breaks=50)
```

Besides this default string interface of UNU.RAN there is also a second interface available with several strings. The first one describes the distribution, the second one the method and the third one the parameters of the method.

```
> gen = new("unur",distr="normal();domain=(0,inf)",
                   method="arou", methodpars="max_sqhratio=0.9")
> x<-sample.unur(gen,10000)
> hist(x,breaks=50)
```

The default values are the empty strings. The default method depends on the distribution and is documented in [12]. We define in R an special S4-class named `unur` with two slots

```
 > setClass("unur",representation(string="character",p="externalptr"),
                  prototype = list(string=character(), p="externalptr"))
```

In the slot `string` we save all necessary information about the distribution, used method etc. to generate our 'generator object' in UNU.RAN. A full description can be found in Leydold et al. [11]. The second slot contains an external pointer which refers to a generator object created by the C-code.

With the definition of function `initialize` for our class `unur` we ensure that after the creation of a new instance of the class `unur` we have a pointer to the generation object created and handeled by the C-ode of UNU.RAN.

```
> setMethod("initialize","unur",
           function(.Object,x=character())
            { ...
              .Object@p <-.Call("R_unur_init",x)
              .Object
            }
         )
```

The setup step of black box algorithms is hidden behind this initialization. If it failed the user is informed by an error message. This can be happen for example if the choosen method does not work for a special distribution.

The user does not need to know anything about the created generator object unless that it contains all information to create the random variates very fast and efficiently. Of course UNU.RAN allocates memory which should be deallocated in `R` . `R` provides an ideal function named `R_RegisterCFinalizer` which ensures that the memory will be deallocated with the command `gc()`.

Our first version uses only the same built in uniform RNG for all generators although UNU.RAN can use multiple streams of uniform random number generators. With the functions

```
> seed.unur(1234)
> reset.unur()
```

we can set and reset the seed.

The following examples should show some interesting features of our interface. We can use different algorithms for a large class of distributions

```
> gen = new("unur",distr="beta()",method="tdr")
> x<-sample.unur(gen,10000)
```

or we can sample from truncated functions

```
> gen = new("unur",distr="normal();domain=(-1,1)")
> x<-sample.unur(gen,10000)
```

or we can sample from a kernel density estimate with kernel smoothing

```
> gen = new("unur",distr="cemp;data=(-0.1,0.05,....)",
            method="empk", methodpars="smothing=0.8")
> x<-sample.unur(gen,10000)                                    .
```

4 Conclusion

Our interface provides the possibility to use a lot of algorithms in R to generate non uniform variates for large classes of distributions. Due to the implementation of S4-classes the handling is very easy. A R package with automatic installation and documentation is in preparation. A closer relationship between R and UNU.RAN should be possible and easy to realize. For example to define a distribution function in R and use algorithms of UNU.RAN to generate random numbers. But this makes only sense if we first define a distribution object in R. This is already in planning by a group in Bayreuth [12]. A lot of contributed packages of R use random numbers of special distributions but everybody uses his own code and variables. Therefore we think that a standard description of distribution objects including generation of random variates will be helpful for a lot of code developers in R.

References

[1] J. H. Ahrens. A one-table method for sampling from continuous and discrete distributions. *Computing*, 54(2):127–146, 1995.

[2] G. E. P. Box and M. E. Muller. A note on the generation of random normal deviates. *Annals of Math. Stat.*, 29(2):610–611, 1958.

[3] J. Dagpunar. *Principles of Random Variate Generation*. Clarendon Oxford Science Publications, Oxford, U.K., 1988.

[4] L. Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag, New-York, 1986.

[5] J. E. Gentle. *Random Number Generation and Monte Carlo Methods*. Statistics and Computing. Springer, New York, 1998.

[6] W. Hörmann. A rejection technique for sampling from T-concave distributions. *ACM Trans. Math. Software*, 21(2):182–193, 1995.

[7] W. Hörmann, J. Leydold, and G. Derflinger. *Automatic Non-Uniform Random Variate Generation*. Springer-Verlag, Berlin Heidelberg, 2003. accepted for publication.

[8] D. E. Knuth. *The Art of Computer Programming. Vol. 2: Seminumerical Algorithms*. Addison-Wesley, 3rd edition, 1998.

[9] J. Leydold. Automatic sampling with the ratio-of-uniforms method. *ACM Trans. Math. Software*, 26(1):78–98, 2000. URL http://www.acm.org/pubs/citations/journals/toms/2000-26-1/p78-leydold/.

[10] J. Leydold and W. Hörmann. Universal algorithms as an alternative for generating non-uniform continuous random variates. In G. I. Schuëller and P. D. Spanos, editors, *Monte Carlo Simulation*, pages 177–183. A. A. Balkema, 2001. Proceedings of the International Conference on Monte Carlo Simulation 2000.

[11] J. Leydold, W. Hörmann, E. Janka, and G. Tirler. *UNU.RAN – A Library for Non-Uniform Universal Random Variate Generation*. Institut für Statistik, WU Wien, A-1090 Wien, Austria, 2002. available at `http://statistik.wu-wien.ac.at/unuran/`.

[12] private communications with R. Ruchdaschel and M.Kohl.