1

Introduction

Non-uniform random variate generation is a small field of research somewhere between mathematics, statistics and computer science. It started in the fifties of the last century in the "stone-age" of computers. Since then its development has mainly been driven by the wish of researchers to solve generation problems necessary to run their simulation models. Also the need for fast generators has been an important driving force. The main mathematical problems that have to be solved concern the distribution of transformed random variates and finding tight inequalities. Also implementing and testing the proposed algorithms has been an important part of the research work. A large number of research papers in this field has been published in the seventies and early eighties. The main bibliographical landmark of this development is the book of Devroye (1986a), that is commonly addressed as the "bible" of random variate generation. We can certainly say that random variate generation has become an accepted research area considered as a subarea of statistical computing and simulation methodology. Practically all text-books on discrete event simulation or Monte Carlo methods include at least one chapter on random variate generation; within simulation courses it is taught even to undergraduate students.

More important is the fact that random variate generation is used by lots of more or less educated users of stochastic simulation. Random variate generation code is found in spreadsheets and in expensive discrete event simulation software and of course in a variety of programming languages. Probably many of these users do not bother about the methods and ideas of the generation algorithms. They just want to generate random variates with the desired distribution. "The problems of random variate generation are solved" these people may say. And they are right as long as they are only interested in popular standard distributions like normal, gamma, beta, or Weibull distributions.

Why Universal Random Variate Generation?

The situation changes considerably if the user is interested in non standard distributions. For example, she wants to simulate models that include the generalized inverse Gaussian distribution, the hyperbolic distribution, or any other distribution that is less common or even newly defined for her purpose. Then the user had two possibilities: She could either find some code (or a paper) dealing with the generation of random variates from her distribution, or she needed some knowledge on random variate generation (or find an expert) to design her own algorithm. Today the user has a third possibility: She can find a universal generator suitable for her distribution, perhaps in our C library UNU.RAN (Universal Non-Uniform RANdom variate generation). Then she can generate variates without designing a new generator; a function that evaluates e.g. the density of the distribution or the hazard rate is sufficient.

This book is concentrating on the third possibility. We present ideas and unifying concepts of universal random variate generation, and demonstrate how they can be used to obtain fast and robust algorithms. The book is presenting the first step of random variate generation, the design of the algorithms. The second step, i.e. the implementation of most of the presented algorithms, can be found in our C library UNU.RAN (Universal Non-Uniform RANdom number generators, Leydold, Hörmann, Janka, and Tirler, 2002). As some of these algorithms are rather long it is not possible, and probably not desirable, to present all implementation details in this book as they would hide the main ideas.

What Is Non-Uniform Random Variate Generation?

Usually random variates are generated by transforming a sequence of independent uniform random numbers on $(0, 1)$ into a sequence of independent random variates of the desired distribution. This transformation needs not be one-to-one. We assume here, as it is generally done in the literature, that we have an ideal source of uniform random numbers available. An assumption which is not too unrealistic if we think of fast, modern uniform random number generators that have cycle lengths in the order 2 raised to the power of several hundreds or even thousands and equidistribution property up to several hundred dimensions, e.g., Matsumoto and Nishimura (1998) or L'Ecuyer (1999); see also the pLab website maintained by Hellekalek (2002) for further links. A collection of many published uniform random number generators – good ones and bad ones – is compiled by Entacher (2000).

Given that (ideal) source of uniform random numbers, the well known inversion, (acceptance-) rejection and decomposition methods can be used to obtain exact random variate generation algorithms for standard distributions. We do not want to give a historical overview here but it is remarkable that the rejection method dates back to von Neumann (1951). Later refinements of the

general methods were developed mainly to design fast algorithms, if possible with short code and small memory requirements. For the normal distribution compare e.g. Box and Muller (1958), Marsaglia, MacLaren, and Bray (1964), Ahrens and Dieter (1972, 1973, 1988), and Kinderman and Ramage (1976). For the gamma distribution see e.g. Cheng (1977), Schmeiser and Lal (1980), Cheng and Feast (1980), and Ahrens and Dieter (1982). In the books of Devroye (1986a) and Dagpunar (1988) you can find an impressive number of references for articles dealing with random variate generation for standard distributions. The techniques and general methods we describe in this book are very closely related to those developed for standard distributions. We know what we owe to these "pioneers" in random variate generation. Even more as we have started our research in this field with generators for standard distributions as well (see e.g. Hörmann and Derflinger, 1990). However, in this book we try to demonstrate that several of these main ideas can be applied to build universal generators for fairly large distribution families. Thus we do not concentrate on standard distributions but try to identify large classes of distributions that allow for universal generation. Ideally these classes should contain most important standard distributions. So the reader will come across quite a few standard distributions as we use them as examples for figures and timings. They are best suited for this purpose as the reader is familiar with their properties. We could have included fairly exotic distributions as well, as we have done it in some of the empirical comparisons.

What Is a Universal Generator?

A *universal* (also called *automatic* or *black-box*) generator is a computer program that can sample from a large family of distributions. The distributions are characterized by a program that evaluates (e.g.) the density, the cumulative distribution function, or the hazard rate; often some other information like the mode of the distribution is required. A universal generator typically starts with a *setup* that computes all constants necessary for the generation, e.g. the hat function for a rejection algorithm. In the *sampling* part of the program these constants are used to generate random variates. If we want to generate a large sample from a single distribution, the setup part is executed only once whereas the sampling is repeated very often. The average execution time of the sampling algorithm to generate one random variate (without taking the setup into account) is called *marginal execution time*. Clearly the *setup time* is less important than the marginal execution time if we want to generate a large sample from a single distribution.

Fast universal generators for discrete distributions are well known and frequently used. The indexed search method (Chen and Asau, 1974) and the alias method (Walker, 1974, 1977) can generate from any discrete distribution with known probability vector and bounded domain. For continuous distributions the design of universal generators started in the eighties and is mainly linked with the name of Luc Devroye. During the last decade research in this direc-

tion was intensified (see e.g. Gilks and Wild, 1992; Hörmann, 1995; Ahrens, 1995; Leydold, 2000a) and generalized to random vectors (see e.g. Devroye, 1997a; Leydold and Hörmann, 1998). However it seems that these developments are little known and hardly used. When we have started our UNU.RAN project in 1999 we checked several well known scientific libraries: IMSL, Cern, NAG, Crand, Ranlib, Numerical recipes, and GSL (Gnu Scientific Library). Although all of them include quite a few random variate generation algorithms for continuous standard distributions we were not able to find a single universal generator for continuous distributions in any of them. And the situation is similar for random-vector generation methods. This has been a main motivation for us to start our project with the aim to write both a book and a C library to describe and realize theory and practice of universal random variate and random vector generation.

Why We Have Written this Book?

We are convinced that universal random variate generation is a concept of greatest practical importance. It also leads to nice mathematical theory and results. Neither the mathematics nor the implemented algorithms have been easily accessible up to now, as there is – up to our knowledge – no book and no software library available yet that includes the main concepts of universal random variate generation.

Implementation of Universal Algorithms

One main problem when implementing universal algorithms for continuous distributions in a software library is certainly the application programming interface (API). Considering generators for a fixed distribution everything is simple. Any programmer can easily guess that the following C statements are assigning a realization from a uniform, a standard normal, and a gamma(5) random variate to the respective variables.

```
x  = random();
xn = randnormal();
xg = randgamma(5.);
```

For a universal method the situation is clearly more complicated. The setup often is very expensive compared to the marginal generation time and thus has to be separated from the sampling part of a universal algorithm. Moreover, to run such a setup routine we need a lot more parameters. In a routine like `randgamma(5.)` all required information is used to build the algorithm and is thus contained implicitly in the algorithm. For black-box algorithms we of course have to provide this information explicitly. This may include – depending on the chosen algorithm – the density, its derivative and its mode, its cumulative distribution function, its hazard rate, or similar data

and functions to describe the desired distribution. Furthermore, all the black-box algorithms have their own parameters to adjust the algorithm to the given distribution. All of this information is needed in the setup routine to construct a generator for the distribution and to store all necessary constants. The sampling program then uses these constants to generate random variates.

There are two very different general solutions for the implementation of automatic algorithms. First, we can make a library that contains both a setup routine and a sampling routine using an object-oriented design. The setup routine creates an instance of a *generator object* that contains all necessary constants. The sampling routine is using this generator object for generation. If we want to sample from several different distributions in one simulation we can create instances of such generator objects for all of them and can use them for sampling as required. In our library UNU.RAN we have implemented this idea. For further details see Sect. 8.1.

Our experiences with the implementation of universal algorithms in a flexible, reliable, and robust way results in rather large computer code. As the reader will find out herself, the complexity of such a library arises from the setup step, from parts performing adaptive steps, and (especially) from checking the data given by the user, since not every method can be used for every distribution. The sampling routine itself, however, is very simple and consists only of a few lines of code. Installing and using such a library might seem too tedious for "just a random number generator" at a first glance, especially when only a generator for a particular distribution is required. As a solution to this problem we can use universal methods to realize the concept of an *automatic code generator for random variate generation*. In this second approach we use the constants that are computed in the setup to produce a single piece of code in a high level language for a generator of the desired distribution. Such a code generator has the advantage that it is also comparatively simple to generate code for different programming languages. Moreover, we can use a graphical user interface (GUI) to simplify the task of obtaining a generator for the desired distribution for a practitioner or researcher with little background in random variate generation. We also have implemented a proof of concept study using a web based interface. It can be found at `http://statistik.wu-wien.ac.at/anuran/`. Currently, program code in C, FORTRAN, and Java can be generated and downloaded. For more details see Leydold, Derflinger, Tirler, and Hörmann (2003).

It should be clear that the algorithm design of the setup and the sampling routine remain the same for both possible implementation concepts. Thus we will not consider the differences between these two approaches throughout the book. The only important difference to remember is that the speed of the setup is of little or no concern for the code generator whereas it may become important if we use generator objects.

Theoretical Concepts in the Book

We have spoken quite a lot about algorithms and implementation so far but most of the pages of the book are devoted to the theoretical concepts we need for designing random variate generation algorithms. Typically we are concerned with the following problems:

- We have to show that the algorithms generate variates from the desired distribution.
- For rejection algorithms we use inequalities to design upper and lower bounds for densities.
- The automatic construction of hat functions for rejection methods requires design points. Hence we also need algorithms to find such points. These algorithms are either simple and fast, or provide optimal solutions (or both).
- We need properties of the distribution families associated with the universal algorithms.
- We need simple (sufficient) conditions for the (large) class of distributions for which an algorithm is applicable. When the condition is not easily computable either for the routines in the library or for the user of such a library, a black-box algorithm is of little practical use.
- The complexity of an algorithm is the number of operations it requires for generating a random variate. For most algorithms the complexity is a random variate itself that depends on the number of iterations I till an algorithm terminates. For many universal algorithms we can obtain bounds for the expected number of iterations $\mathrm{E}(I)$.
- We want to have some estimates on the "quality" of the generated random variates. In simulation studies streams of *pseudo*-random numbers are used, i.e. streams of numbers that cannot be distinguished from a stream of (real) random numbers by means of some statistical tests. Such streams always have internal structures (see e.g. L'Ecuyer (1998) for a short review) and we should take care that the transformation of the uniform *pseudo*-random numbers into non-uniform *pseudo*-random variates do not interfere with the structures of this stream.

Chapters of the Book

Chapter 2 presents the most important basic concepts of random variate generation: *Inversion*, *rejection*, and *composition* for continuous random variates. Thus it is crucial to the rest of the book as practical all of the algorithms presented in the book depend on one or several of these principles. Chapter 3 continues with the basic methods for generating from discrete distributions, among them inversion by sequential search, the indexed search and the alias method.

Part II of the book deals with continuous univariate distributions. Chapters 4, 5, and 6 present three quite different approaches to design universal

algorithms by utilizing the rejection method. Chapter 7 realizes the same task using numerical inversion. Chapter 8 compares different aspects of the universal algorithms presented so far including our computational experiences when using the UNU.RAN library. It also describes the main design of the UNU.RAN programming interface that can be used for generating variates from discrete distributions and from random vectors as well. Part II closes with Chap. 9 that collects different special algorithms for the case that the density or cumulative distribution function of the distribution is not known.

Part III consists only of Chap. 10. It explains recent universal algorithms for discrete distributions, among them the indexed search method for distributions with unbounded domain and different universal rejection methods.

Part IV, that only consists of Chap. 11, presents general methods to generate random vectors. It also demonstrates how the rejection method can be utilized to obtain universal algorithms for multivariate distributions. The practical application of these methods are restricted to dimensions up to about ten.

Part V contains different methods and applications that are closely related to random variate generation. Chapter 12 collects random variate generation procedures for different situations where no full characterization of the distribution is available. Thus the decision about the generation procedures is implicitly also including a modeling decision. Chapter 13 (co-authored by M. Hauser) presents very efficient algorithms to sample Gaussian time series and time series with non-Gaussian one-dimensional marginals. They work for time series of length up to one million. Markov Chain Monte Carlo (MCMC) algorithms have become frequently used in the last years. Chapter 14 gives a short introduction into these methods. It compares them with the random vector generation algorithms of Chap. 11 and discusses how MCMC can be used to generate iid. random vectors. The final Chap. 15 presents some simulation examples for financial engineering and Bayesian statistics to demonstrate, at least briefly, how some of the algorithms presented in the book can be used in practice.

Reader Guidelines

This book is a research monograph as we tried to cover – at least shortly – all relevant universal random variate generation methods found in the literature. On the other hand the necessary mathematical and statistical tools are fairly basic which should make most of the concepts and algorithms accessible for all graduate students with sound background in calculus and probability theory. The book mainly describes the mathematical ideas and concepts together with the algorithms; it is not closely related to any programming language and is generally not discussing technical details of the algorithm that may depend on the implementation. The interested reader can use the source code of UNU.RAN to see possible solutions to the implementation details for most of the important algorithms. Of course the book can also be seen as the

"documentation" explaining the deeper mathematics behind the UNU.RAN algorithms. Thus it should also be useful for all users of simulations who want to gain more insight into the way random variate generation works.

In general the chapters collect algorithms that can be applied to similar random variate generation problems. Only the generation of continuous one-dimensional random variates with known density contains so much material that is was partitioned into the Chaps. 4, 5, and 6. Several of the chapters are relatively self contained. Nevertheless, there are some dependencies that lead to the following suggestions: For readers not too familiar with random variate generation we recommend to read Chap. 2 and Chap. 3 before continuing with any of the other chapters. Chapters 4 to 7 may be read in an arbitrary order; some readers may want to consult Chap. 8 first to decide which of the methods is most useful for them. Chapters 9, 12, and 13 are self contained, whereas some sections of Chaps. 10 and 11 are are generalizations of ideas presented in Chaps. 4 and 6. Chapter 14 is self contained, but to appreciate its developments we recommend to have a look at Chap. 11 first. In Chap. 15 algorithms of Chaps. 2, 4, 11 and 12 are used for the different simulations.

Course Outlines

We have included exercises at the end of most of the chapters as we used parts of the book also for teaching courses in simulation and random variate generation. It is obvious that the random variate generation part of a simulation course will mainly use Chaps. 2 and 3. Therefore we tried to give a broad and simple development of the main ideas there. It is possible to add the main idea of universal random variate generation by including for example Sect. 4.1. Selected parts of Chap. 15 are useful for a simulation course as well.

A special course on random variate generation should start with most of the material of Chaps. 2 and 3. It can continue with selected sections of Chap. 4. Then the instructor may choose chapters freely, according to her or the students' preferences: For example, for a course with special emphasis on multivariate simulation she could continue with Chaps. 11 and 14; for a course concentrating on the fast generation of univariate continuous distributions with Chaps. 5, 7, and 8. Parts of Chap. 15 can be included to demonstrate possible applications of the presented algorithms.

What Is New?

The main new point in this book is that we use the paradigm of universal random variate generation throughout the book. Looking at the details we may say that the rigorous treatment of all conditions and all implications of transformed density rejection in Chap. 4 is probably the most important contribution. The second is our detailed discussion of the universal generation of random vectors. The discussion of Markov chain Monte Carlo methods for

generating iid. random vectors and its comparison with standard rejection algorithms is new.

Throughout the book we present new improved versions of algorithms: Among them are variants of transformed density rejection, fast numerical inversion, universal methods for increasing hazard rate, indexed search for discrete distributions with unbounded domain, improved universal rejection algorithms for orthomonotone densities, and exact universal algorithms based on MCMC and perfect sampling.

Practical Assumptions

All of the algorithms of this book were designed for practical use in simulation. Therefore we need the simplifying assumption of above that we have a source of truly uniform and iid. (independent and identically distributed) random variates available. The second problem we have to consider is related to the fact that a computer cannot store and manipulate real numbers.

Devroye (1986a) assumes an idealized numerical model of arbitrary precision. Using this model inversion requires arbitrary precision as well and is therefore impossible in finite time unless we are given the inverse cumulative distribution function. On the other hand there are generation algorithms (like the series method) that require the evaluation of sums with large summands and alternating signs, which are exact in the idealized numerical model of arbitrary precision.

If we consider the inversion and the series method implemented in a modern computing environment (e.g. compliant with the IEEE floating point standard) then – using bisection – inversion may be slow but it is certainly possible to implement it with working precision close to machine precision. On the other hand, the series method may not work as – due to extinction – an alternating series might "numerically" converge to wrong values. We do not know the future development of floating point arithmetic but we do not want to use an idealized model that is so different from the behavior of today's standard computing environments. Therefore we decided to include numerical inversion algorithms in this book as long as we can easily reach error bounds that are close to machine precision (i.e. about 10^{-10} or 10^{-12}). We also include the series method but in the algorithm descriptions we clearly state warnings about the possible numerical errors.

The speed of random variate generation procedures is still of some importance as faster algorithms allow for larger sample sizes and thus for shorter confidence intervals for the simulation results. In our timing experiments we have experienced a great variability of the results. They depended not only on the computing environment, the compiler, and the uniform generator but also on coding details like stack variables versus heap for storing the constants etc. To decrease this variability we decided to define the *relative generation time* of an algorithm as the generation time divided by the generation time for the exponential distribution using inversion which is done by

$X \leftarrow -\log(1 - \text{random}())$. Of course this time has to be taken in exactly the same programming environment, using the same type of function call etc. The relative generation time is still influenced by many factors and we should not consider differences of less than 25 %. Nevertheless, it can give us a crude idea about the speed of a certain random variate generation method.

We conclude this introduction with a statement about the speed of our universal generators. The relative generation time for the fastest algorithms of the Chaps. 4, 5 and 7 are not depending on the desired density and are close to one; i.e. these methods are about as fast as the inversion method for exponential variates and they can sample at that speed from many different distributions.