

PAPER • OPEN ACCESS

Performance analysis of random sampling algorithms on GNU Octave

To cite this article: Mohd Syahmi Bin Yazid *et al* 2019 *IOP Conf. Ser.: Mater. Sci. Eng.* **555** 012003

View the [article online](#) for updates and enhancements.

Performance analysis of random sampling algorithms on GNU Octave

Mohd Syahmi bin Yazid¹, Matthieu Lemaire², Mohsin bin Mohd Sies¹, Deokjung Lee²

¹School of Chemical and Energy Engineering, Faculty of Engineering, Universiti Teknologi Malaysia, 81310 Johor Bahru, Johor, Malaysia.

²Department of Nuclear Engineering, Ulsan National Institute of Science and Technology, 50 UNIST-gil, Ulsan 44919, Republic of Korea.

deokjung@unist.ac.kr

Abstract. This paper presents the performance analysis of two random sampling algorithms, the inverse-transform method and the Vose aliasing method, on GNU Octave. The Monte Carlo code MCS developed by UNIST uses random sampling methods to simulate the physics of neutron and photon transport [1]. The goal is to optimize the sampling time of MCS for cases when the probability density function is a constant function throughout the simulation. For this purpose, the runtime of the inverse-transform method and Vose aliasing method are compared for increasing input size with scripts developed on GNU Octave. To compare the execution time, the initialization and generation time of both methods are determined and discussed.

1. Introduction

The random sampling of physical parameters modelled as random variables is an essential component of Monte Carlo simulations [4]. Monte Carlo simulations are able to simulate problems that are stochastic (probabilistic) by nature, such as the transport of neutrons and photons in matter [6]. Random variables can be characterized by so-called probability distribution functions (PDFs) and there are several algorithms available in the literature to perform random sampling from a PDF. The runtime of algorithms can become the major bottleneck that limit the performances of computer simulation and selecting faster algorithms is therefore key for efficient simulations yielding results in reasonable times [3]. In this paper, an analysis of the runtime of two random sampling algorithms for discrete distributions, the inverse-transform method and the Vose aliasing method, is conducted. Both methods are implemented in the mathematical freeware GNU Octave [2] and the initialization time and generation times of the two methods is compared and discussed.

The paper is organized as follows. In section 2, the theory of random sampling, the inverse-transform method (IVM) and the Vose aliasing method (VAM) are introduced. The runtime comparison results are analysed and discussed in section 3 and conclusions are drawn out in section 4.



2. Random sampling for discrete distribution

Random sampling are techniques to generate random values of a variable x distributed in the interval $(x_{\min}; x_{\max})$ according to a given PDF, $p(x)$ [4]. In discrete distributions, the PDF $p(x)$ is quantized: this case corresponds for instance to throwing a dice or tossing a coin. Technical vocabulary associated to discrete distributions and probability terms are defined in g over the simulated histories.

Table 1. From a discrete PDF, a discrete cumulative distribution function (CDF) or a probability-alias table can be generated. This CDF / this probability-alias table can be used in conjunction with a random number generator (RNG) to sample random variables according to the initial PDF. The RNG commonly samples number uniformly in the interval (0;1). An illustration of the random sampling process is shown in Figure 1. If the number of generated random variables is large enough, quantitative information on the process being simulated (dice, neutron transport process, etc.) may be obtained by simply averaging over the simulated histories.

Table 1. Probability terms used for discrete distributions.

Terms	Notation	Definition/description
Random variable	x	Variable whose value is determine by chance
Sample space	N	Total number of possible x -values
Probability distribution functions, PDF	$PDF(x) = n/N$	Function which allocates probabilities for certain event to occur where: <ul style="list-style-type: none"> n = number of values of x that fall within interval $(x_{\min} \leq x \leq x_{\max})$ N = sample space
Cumulative distribution function, CDF	$CDF(x) = \sum_{i=1}^N p(x)$	<ul style="list-style-type: none"> Shows cumulative summation of PDF for each x-value Sum of all CDF equals 1 CDF is non decreasing function
Random number generator, rand	$Rand = \zeta$	rand is an Octave keyword to generates a random number, ξ uniformly between 0 and 1. ($0 \leq \xi < 1$).

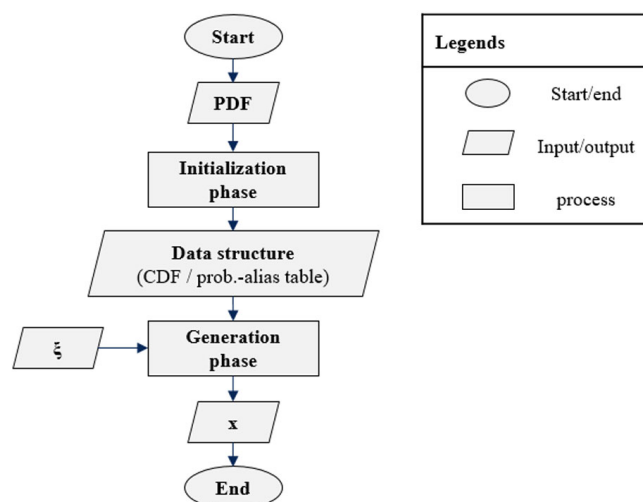


Figure 1. Random sampling for discrete distributions.

2.1 Inverse-transform method (IVM)

Inverse transform method is the conventional algorithm used in random sampling. This practical method is useful for generating random values of x using a generator of random numbers uniformly

distributed in (0,1). It exploits the fact that the CDF, because it is the cumulative summation of PDF for each x-value, is a non-decreasing function that can be inverted.

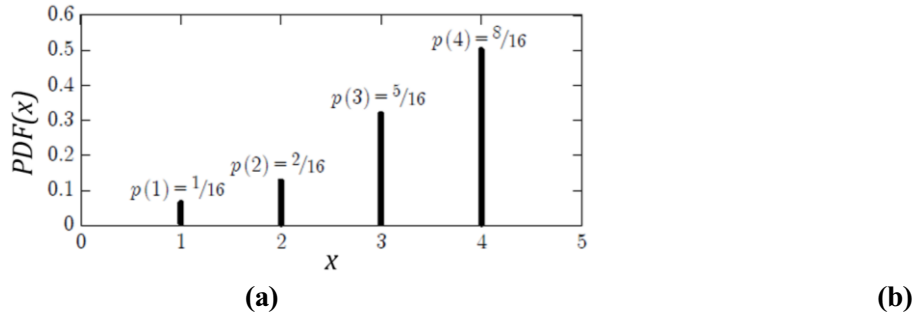


Figure 2. (a) Example of PDF and, (b) CDF for sampling with the inverse-transform method.

The output of the uniform RNG $0 < \xi < 1$ is related to CDF by Equation 1 and Equation 2 consists in inverting the CDF (hence the name inverse-transformation method). The random variable, x can therefore be sampled according to Equation 2. The sampling process is illustrated in (b)

Figure 2 [4]. The initiation phase of IVM consists in creating a CDF by summing up the PDF cumulatively. The generation phase of IVM consists in solving Equation 2 for a random number ξ sampled uniformly in (0,1) to determine the random variable, x.

$$\xi = \text{CDF}(x) \quad (1)$$

$$\text{CDF}(\xi)^{-1} = x \quad (2)$$

2.2 Vose aliasing method (VAM)

The Vose aliasing method is another approach to generate a random variable from a RNG outputting numbers uniformly between 0 and 1. The VAM relies on the use of a so-called probability-alias table instead of a CDF. The initialization phase of VAM consists in generating the probability-alias table from the specified PDF. In the probability-alias table, the row *prob* stores the probability within the column that the original PDF will be chosen whereas the row *alias* stores the index of the cut PDF (if any) [5]. This index acts as a pointer referring to the original PDF (hence the name alias). The table is characterized by a length of N and the height is 1 (100% probability) as displayed in Figure 3.

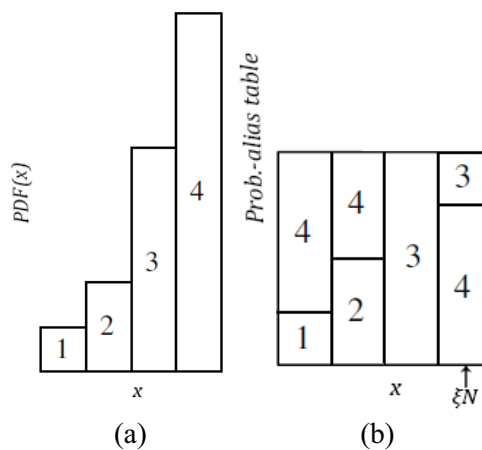


Figure 3. (a) Example of PDF and (b) *prob.-alias* table for sampling with the Vose aliasing method.

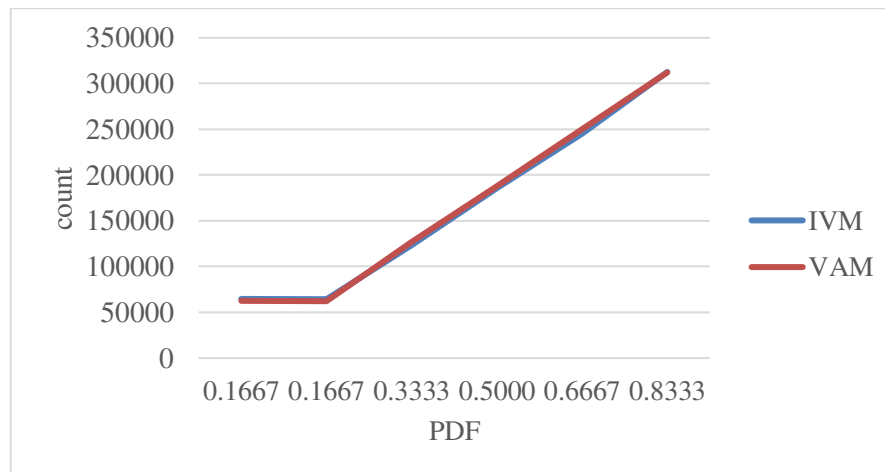
In the generation phase, a random number, ξ_1 is generated by the RNG to select a column of the probability-alias table uniformly at random. The select column is called column i . Next, another random number, ξ_2 is generated by the RNG. If $\xi_2 \leq \text{prob}[i]$, the random variable, x in $\text{prob}[i]$ is returned, else the random variable, x in $\text{alias}[i]$ is returned. In this process, 2 random numbers from the RNG are required to sample a single random variable. A trick can be used to only one random number instead of two: it is indeed possible to replace ξ_2 by ξ_1 . $N - \text{floor}(\xi_1 \cdot N)$ and still obtain a fair sampling of the random variable.

3. Results

The data collected from the validation test is summarized in Table 2 and illustrated in Figure 4. The difference between the normalized PDF, the IVM frequency and VAM frequency is very small, thus showing that both IVM and VAM methods were properly implemented in GNU Octave and are valid for further random sampling usage.

Table 2. Validation test.

Dice value	Normalized PDF	1 million samples in total			
		IVM count	IVM frequency	VAM count	VAM frequency
1	1/16 = 0.0625	6.26×10^4	0.0626	6.23×10^4	0.0623
2	1/16 = 0.0625	6.23×10^4	0.0623	6.24×10^4	0.0624
3	2/16 = 0.125	1.24×10^5	0.124	1.25×10^5	0.125
4	3/16 = 0.1875	1.87×10^5	0.187	1.87×10^5	0.187
5	4/16 = 0.25	2.49×10^5	0.249	2.50×10^5	0.25
6	5/16 = 0.3125	3.12×10^5	0.312	3.12×10^5	0.312

**Figure 4.** Validation test of implementation of random sampling methods.

The runtime data to generate one million samples from increasing PDF size is tabulated in Table 3. Figure 5 depicts the initialization time as a function of the PDF size. For both methods, the initialization time increases linearly as the PDF size increases. However, the slope for VAM is much steeper than for IVM due to a more complex initialization phase. Notice that the initialization time of both methods (runtime of one initialization phase) is very small compared with the generation time of one million samples, which makes it not significant to the total runtime.

Table 3. Runtimes of sampling methods for increasing PDF size.

		PDF size, N				
		20	40	60	80	100
Initialization time [s]	IVM	9.6×10^{-5}	2.8×10^{-5}	3.2×10^{-5}	6.2×10^{-5}	1.3×10^{-4}
	VAM	2.5×10^{-3}	3.2×10^{-3}	4.9×10^{-3}	8.8×10^{-3}	8.5×10^{-3}
Generation time [s]	IVM	209.5	348.0	478.3	616.7	746.6
	VAM	64.0	65.7	63.5	65.0	64.9

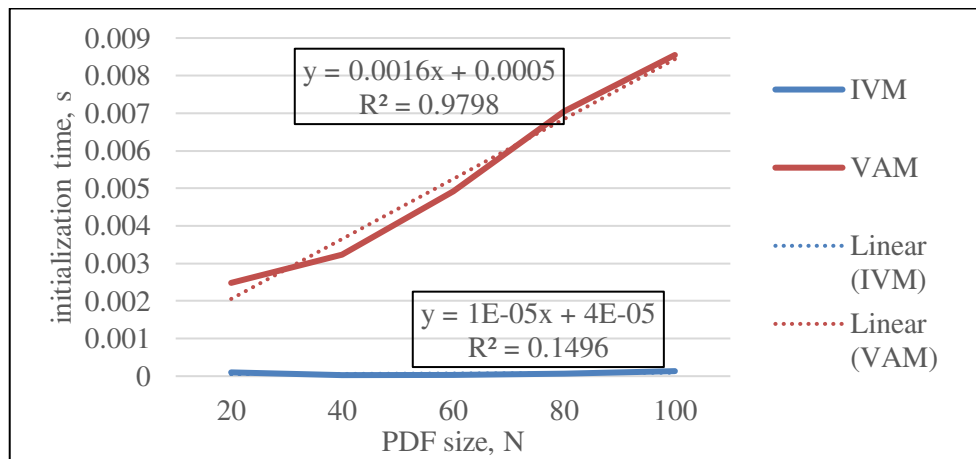


Figure 5. Initialization time [s] versus PDF size, N.

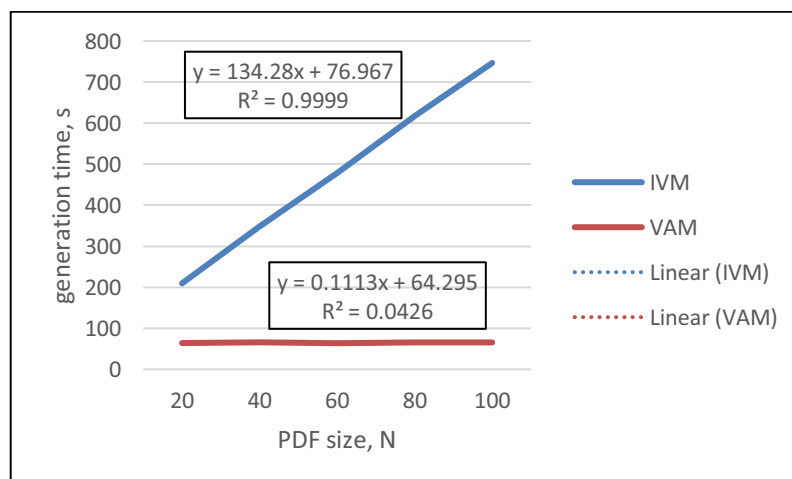


Figure 6. Generation time [s] versus PDF size, N.

In Figure 6, a proportional increase of the generation time as a function of the PDF size is observed. This relationship occurs because in the generation phase of IVM, a sequential search of the CDF is performed to output a sample, whose runtime increases linearly with the PDF size. Meanwhile,

VAM shows an almost constant generation time close to ± 64.5 s regardless of the PDF size. This speed advantage is due to the smart data structure of the prob.-alias tables. Therefore, as long as the initialization phase is only performed once, this analysis shows that the VAM has a better runtime performance than the IVM.

4. Conclusions

The IVM and VAM have been implemented in GNU Octave and the complexity analysis of both algorithms have been performed. VAM is shown to have a better generation runtime performance compared to IVM but a worse initialization runtime. For random variables with constant PDFs, the initialization is only called once: VAM is therefore a good alternative to IVM to optimize the sampling time of random variables characterized by constant PDF. An example of constant PDFs can be for instance the electron shell transition probabilities in Monte Carlo codes simulating the transport of photons and the creation of fluorescence photons through atomic relaxation processes. Implementing the VAM instead of IVM for such a case will result in a more complex initialization phase and coding but a net gain of computing time. VAM instead of IVM may be used in other circumstances as well as long as the PDF governing the random variables is constant and the initialization phase is only called once.

5. References

- [1] Lee D, lemaire M, 2017 MCS Photon Physics Retrieved from CORE | UNIST:http://reactorcore.unist.ac.kr/wpcontent/uploads/sites/229/2016/03/20180131_MCS_photon_transport.pdf
- [2] Eaton J, 2018 About and History of GNU Octave Retrieved from GNU Octave: <https://www.gnu.org/software/octave/about.html>
- [3] Sedgewick R, 2013 *An Introduction To The Analysis of Algorithms, Second Edition* . Pearson Education, Inc.
- [4] Salvat F 2014 *PENELOPE-2014: A Code System for Monte Carlo Simulation of Electron and Photon Transport* (Barcelona, Spain: Organisation for Economic Co-operation and Development (OECD), Nuclear Energy Agency (NEA)).
- [5] Schwarz K 2011 Darts, Dice, and Coins: Sampling from a Discrete Distribution Retrieved from <http://www.keithschwarz.com/darts-dice-coins/>
- [6] Vujic J L 2008 Monte Carlo Sampling Algorithms (Berkeley: Nuclear Engineering Department, University of California).

Acknowledgement

This project was conducted as a two-month undergraduate internship under the supervision of the UNIST CORE staff members who provided relevant advice all the way to complete this paper. The publication of this paper is financially funded by the Malaysian Nuclear Society (MNS). Their contributions are highly appreciated.