JHU 553.633/433: Monte Carlo Methods
J. C. Spall
10 September 2018

CHAPTER 2
RANDOM NUMBER GENERATION

- Motivation and criteria for generators
- Linear generators (e.g., linear congruential generators)
- Multiple recursive generators (e.g., Mersenne twister)
- Nonlinear, Fibonacci, ….
- Inverse transforms
- Accept–reject and related methods
- Normal distribution methods
- Multivariate distributions
- Markov chains

Uniform Random Number Generators

- Want sequence of independent, identically distributed uniform ($U(0, 1)$) random variables
  - $U(0, 1)$ random numbers of direct interest in some applications
  - More commonly, $U(0, 1)$ numbers transformed to random numbers having ***other*** distributions (e.g., in Monte Carlo simulation)
- Computer-based random number generators (RNGs) produce deterministic and periodic sequence of numbers
  - ***Pseudo*** random numbers
- Want pseudo random numbers that "look" random
  - Should be able to pass all ***relevant*** statistical tests for randomness

2

Overall Framework for Generating Random Numbers

- State at step k given transition function f_k:

$$X_k = f_k(X_{k-1}, X_{k-2}, \ldots, X_{k-r}), r \geq 1$$

- Output function, $0 \leq g \leq 1$ (or $0 < g < 1$), produces pseudo random numbers as

$$U_k = g(X_k)$$

- Output sequence of RNG is $\{U_k, k \geq 1\}$
- Period of an RNG is number of iterations before RNG output (U_k) repeats itself

3

Criteria for Good Random Number Generators

- Long period
- Strong theoretical foundation
- Able to pass empirical statistical tests for independence and distribution (next slide)
- Speed/efficiency
- Portability: can be implemented easily using different languages and computers
- Repeatability: should be able to generate same sequence from same seed
- Be cryptographically strong to external observer: unable to predict next value from past values
- Good distribution of points throughout domain (low discrepancy) (also related to *quasi-random* sequences, not covered here)

4

Criteria for Good Random Number Generators (cont'd): Statistical Tests

- Ideal aim is that no statistical test can distinguish RNG output from i.i.d. $U(0, 1)$ sequence
  - Not possible in practice due to limits of testing and limits of finite-period generators
- **Fundamental limitation:** Impossible to rule out existence of pattern just because pattern has not been found
- More realistic goal is passing only key (relevant) tests
- Null hypothesis: sequence of random numbers is realization of i.i.d. $U(0, 1)$ stochastic process
  - Almost limitless number of possible tests of this hypothesis
- Failing to reject null hypothesis improves confidence in generator but does not guarantee random numbers will be appropriate for all applications
- Bad RNGs fail simple tests; good RNGs fail only complicated and/or obscure tests

5

Types of Random Number Generators

- **Linear:** commonly used (LCG, MRG, Mersenne twister, etc.)
- **Combined:** Uses weighted combination of output of multiple generators; can increase period and improve statistical properties (L'Ecuyer, 2012)
- **Nonlinear:** structure is less regular than linear generators but more difficult to implement and analyze
- **Physical processes:** e.g., timing in atomic decay, internal system noise, atmospheric noise, quantum-based method (Bierhorst et al., 2018)
  - Not as widely used as computer-based generators due to costliness of implementation, lack of speed, and inability to reproduce same sequence
- **Digits of π:** Digits appear to pass all reasonable statistical tests (in contrast, e and $\sqrt{2}$ fail) (Dodge, 1996); research ongoing

6

Linear Congruential Generators (LCGs)

- Linear congruential generators (LCGs) produce $U(0, 1)$ numbers via

$$X_k = (aX_{k-1} + c) \bmod m$$

$$U_k = \frac{X_k}{m},$$

where a, c, and m are user-specified constants

- LCG appears to be most widely studied random number generator
  - Commonly used, but **less so than in past** due to fundamental limits in length of period and subtle correlations
- Values a, c, m, and X_0 should be carefully chosen non-negative integers:

$$0 < a < m,\ 0 \le c < m$$

$$0 < X_0 < m,\ X_k \in \{0,1,\ldots,m-1\};\ \text{period} \le m-1$$

(LCG output may be modified to avoid 0 for U_k)

7

Supplement: Conditions for Full Period (m – 1) of LCGs

- Performance of LCG sensitive to choice of a, c, and m
- Useful to know how to pick a, c, and m to obtain full period
- Conditions below are necessary and sufficient conditions
- LCG has full period if and only if all three following conditions hold (Hull-Dobell Theorem, 1962):
  1. The only integer that divides both m and c is 1 (i.e., c is relatively prime to m);
  2. $a - 1$ is divisible by all prime factors of m;
  3. $a - 1$ is divisible by 4 if m is divisible by 4.
- Note that generator with c = 0 cannot be full period; violates condition 1 in Hull-Dobell Theorem
  - Note: c = 0 corresponds to special case of multiplicative recursive generators (next slide)

8

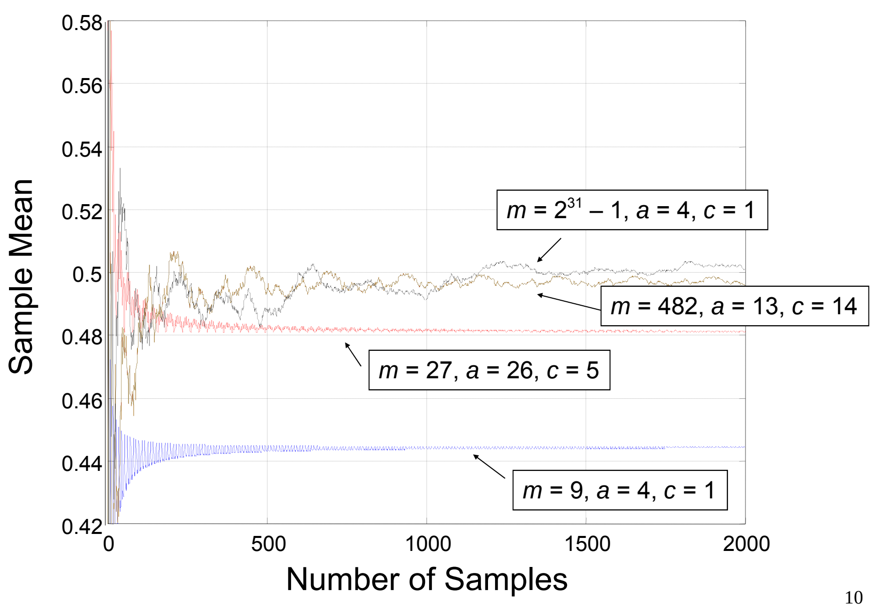
Examples of LCGs

- Famous values for *a* and *m* (assuming *c* = 0; MRG case)
  - $a = 23$, $m = 10^8 + 1$ (first LCG/MRG; original 1951 version*)
  - $a = 65539$, $m = 2^{31}$ (RANDU generator of 1960s; poor because of correlated output)
  - $a = 16807$, $m = 2^{31} - 1$ (has been discussed as minimum standard for RNGs; used in Matlab version 4)
- Example of full-period generator: $a = 4$, $m = 9$, $c = 2$ (try it!)
- ***Full period generators***: any choice X_0 produces full cycle; ***non-full period generators***: cycle length depends on X_0
- Non-full period generator: $X_k = (3X_{k-1}+2) \bmod 9$ with $X_0 = 7$
  - Then $X_1 = 5$, $X_2 = 8$, $X_3 = 8$, …, $X_k = 8$ for all $k = 3, 4, 5, 6, \ldots$

*Lehmer, D. H. (1951), "Mathematical Methods in Large-Scale Computing Units," *Annals of the Computation Laboratory of Harvard University*, no. 26, pp. 141–146.

9

Example of "Minimal" Statistical Test for LCG: Is Sample Mean Close to 0.5?



10

Multiplicative Recursive Generators

- Multiplicative recursive generators (MRGs) are natural extension of LCGs with $c = 0$
- MRGs defined by

$$X_k = (a_1 X_{k-1} + \cdots + a_k X_{k-r}) \text{ mod } m$$
$$U_k = \frac{X_k}{m},$$

  where the a_i belong to $\{0,1,\ldots,m-1\}$
- Maximal period is $m^r - 1$ for prime m and properly chosen a_i
  - Potential for very large period with even modest value of m
  - Choosing $m = 2$ is popular since floating point operations (adds/multiplies) replaced by faster binary operations (XOR)
- For $r = 1$, MRG reduces to LCG with $c = 0$
- A popular form of MRG is Mersenne twister (see below)

11

Popular MRG: Mersenne Twister

- Mersenne twister (MT) is popular state-of-art method used as default generator in Matlab, R, SAS, Python, etc. based on ideas in Matsumoto and Nishimura (1998)
  - Some explanation in textbook, Sect. 2.2.2 (but not that easy to understand!)
- MT is modulo 2 ($m = 2$) generator with huge period:

$$\mathbf{2^{19937} - 1}$$

- As mod 2 generator, MT works exclusively with binary bit strings such as 1 1 0 1 1 0 0 1 0…..; based on simple XOR operations (e.g., 1 + 1 = 0)
- MT passes standard tests for statistical randomness ("Diehard" tests), but fails some more sophisticated tests
- Some criticisms—and defenses—of MT in https://cs.stackexchange.com/questions/50059/why-is-the-mersenne-twister-regarded-as-good

12

Nonlinear Generators

- Nonlinearity sometimes used to enhance performance of RNGs
  - Nonlinearity may appear in transition function f_n and/or in output function g (see earlier slide "Overall Framework for Generating Random Numbers")
  - Have some advantage in reducing lattice structure (Exercise D.2 in Spall, 2003) and in reducing discrepancy
- Two examples (L'Ecuyer, 1998)
  - Nonlinear $f = f_k$ via quadratic recursion:

$$X_k = (aX_{k-1}^2 + X_{k-1} + c) \bmod m$$
$$U_k = X_k / m$$

  - Nonlinear f_k via *inversive generator*:

$$X_k = (ak + c)^{m-2} \bmod m$$
$$U_k = X_k / m$$

13

Combining Generators to Produce *U*(0, 1) Numbers

- Used to increase period length and improve statistical properties
- Shuffling: uses second generator to choose random order for numbers produced by final generator
- Bit mixing: combines numbers in two sequences using some logical or arithmetic operation (addition and subtraction are preferred)
- Can use one RNG to initialize (seed) another RNG

14

Random Number Generators Used in Common Software Packages

- Important to understand types of generators used in statistical software packages and their limitations
- MATLAB:
  - Versions before 5: LCG with a = 75 = 16807 and $m = 2^{31} – 1$
  - Versions 5 to 7.3: lagged Fibonacci generator combined with shift register random integer generator with period $\sim 2^{1492}$ ("ziggurat algorithm")
  - Versions 7.4 and later: "Mersenne twister" (sophisticated linear algorithm with huge period $2^{19937} – 1$)
- R (default selection): Mersenne twister
- EXCEL 2010: Generate three numbers on [0,1) by MRGs (w/ r = 1), sum them and take fractional part of sum; fractional part is uniform on [0,1]; period = $2.78 \times 10^{13} = 1.58 \times 2^{44}$
  - Equivalent to ***one*** MRG with r = 1 (i.e., LCG), a = 16,555,425,264,690 and m = 27,817,185,604,309 (Zeisel, 1986)
- SAS (vers. 9; two core generators): MRG with period $2^{31} – 2$ and Mersenne twister

15

Inverse-Transform Method for Generating Non-*U*(0,1) Random Numbers

- Let $F(x)$ be distribution function of X
- Define inverse function of F by

$$F^{-1}(y) = \inf\{x : F(x) \geq y\}, 0 \leq y \leq 1$$

- Generate X by

$$\boxed{X = F^{-1}(U)} \qquad (*)$$

- Example: exponential distribution

$$F(x) = 1 - e^{-\lambda x}$$

$$X = F^{-1}(U) = -\frac{1}{\lambda}\log(1-U)$$

- Above works for discrete distributions as well:

  If $X \in \{x_1, x_2, \ldots, \}$, then (*) operates according to:

  Find smallest integer $k \geq 0$ such that $U \leq F(x_k)$; then $X = x_k$

16

Accept–Reject Method

- Let $f(z)$ be density function of Z; want to generate $Z \sim f(z)$
  - We start by generating an X and then converting it to a Z
- Find function $\varphi(x)$ that *majorizes* (dominates) $f(x)$ at all x
  - Have $\varphi(x) = Cg(x)$, $C \geq 1$, g is "proposal" density function that is "easy" to generate outcomes X from
  - Support (within $\mathbb{R}^1$) for g is *at least* as large as support for f
- Accept–reject method generates X by following steps:

> Generate X from $g(x)$. **(*)**
> Generate U from $U(0,1)$, independent of X.
> If $U \leq \dfrac{f(X)}{\varphi(X)}$, then set $Z = X$. Otherwise, go back to **(*)**.

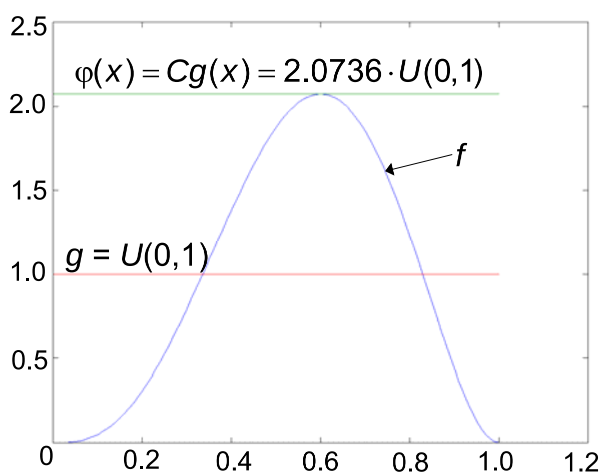
- Probability of acceptance (efficiency) = $1/C$
- Related to Markov chain Monte Carlo (MCMC) (see Exercise 16.4 of Spall, 2003)
- Example to follow next two slides ($f(z)$ = beta density)….

17

$$f(z) = \begin{cases} 60z^3(1-z)^2 & \text{if } 0 \leq z \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

$$X \sim g(x) = U(0,1)$$

$$U \leq \frac{60X^3(1-X)^2}{2.0736}$$



Note: This example adapted from Law (2007, p.438)

18

$X \sim g(x) = U(0,1)$: 0.7621, 0.4565, 0.0185, 0.8214, 0.4447, $\cdots$

$U \sim U(0,1)$: 0.9501, 0.2311, 0.6068, 0.4860, 0.8913, $\cdots$

$$\frac{f(X)}{Cg(X)} : 0.7249,\ 0.8131,\ 0.00018, \cdots$$

Accept/reject: Is U value $\leq$ above ratio?

$Z \sim f(z)$: ~~0.7621~~, 0.4565, ~~0.0185~~, $\cdots$

reject **accept** **reject**

Accepted values represent realization of random numbers from $f(z)$

19

Generating Normal Random Variables

- Generating $N(0, 1)$ random variables is critical capability
- Perhaps oldest method is to generate 12 independent $U(0, 1)$ random numbers (variance = 1/12), and then add them up and subtract 6:
  - Sum is approximately normal by CLT and has mean = 0 and variance = 1
- Box-Muller approach (textbook p. 63) is among more sophisticated and faster ways of generating $N(0, 1)$ random variables
- Modern implementations:
  - **Matlab:** Earlier versions of used polar coordinates method (modified Box-Muller method); current version of Matlab uses "ziggurat algorithm": faster and longer period than former version
  - **R**: Flexibility to specify inversion (default), Box-Muller, etc.

20

Random Vector Generation

- Consider each setting on case-by-case basis. Important special cases:
  - Accept–reject method (next slide)
  - Multivariate normal (below)
  - Generating on simplex (convex hull containing n+1 points not lying on same hyperplane in $\mathbb{R}^n$); all points on line segments connecting any two points in set must also lie in set)
  - Generating points on or in an n-dimensional hypersphere (direct method or A-R method [next slide])
- **Example of multivariate normal.** First, generate vector of n i.i.d. scalar $N(0, 1)$ random variables X_i. Then, form $\boldsymbol{Z} \sim N(\mu, \Sigma)$ according to

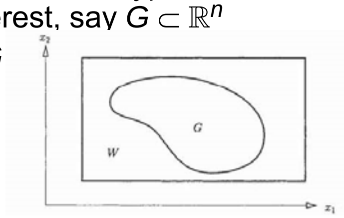
$$\boldsymbol{Z} = \boldsymbol{BX} + \mu,$$

where $\boldsymbol{B}$ is ***any*** square matrix such that $\Sigma = \boldsymbol{BB}^T$ (textbook mentions special case of $\boldsymbol{B}$ = Cholesky factorization)

21

Vector Accept–Reject Method

- Case-by-case for generating random vector $\boldsymbol{Z}$
- One case of practical interest is in generating points ***uniformly*** in some bounded subspace of $\mathbb{R}^n$
- Two step procedure A-R procedure:
  - Generate $\boldsymbol{X}$ uniformly in some hypercube $W \subset \mathbb{R}^n$ that contains region of interest, say $G \subset \mathbb{R}^n$
  - Set $\boldsymbol{Z} = \boldsymbol{X}$ if $\boldsymbol{X}$ lies in G



(Source: Fig. 2.7 in Rubinstein and Kroese, 2017)

- Special case of practical interest is in generating points *in* or *on* a hypersphere in n dimensions
  - Unfortunately, extreme inefficiency for n >> 1. E.g., for "in":

$$\#\text{accepted points} = \frac{\text{vol(hypersphere)}}{\text{vol(hypercube)}} = O\left(\left(\frac{\pi}{4}\right)^{n/2} \frac{1}{(n/2)!}\right)$$

- An alternative method based on generating normal random variables discussed in text, Sect. 2.5.4

22

Generating from Markov Chains

- Wish to generate sequence $\{X_t\}$ satisfying Markov property (Sect. 1.12 in textbook)
- Algorithm for generating from Markov chains is straightforward:
  1. Generate $X_0 \in \{1, 2,\ldots,m\}$ from $\pi^{(0)}$
  2. Given $X_t = i$, generate X_{t+1} from the ith row of $\boldsymbol{P}$
  3. Increment t to t+1 and repeat step 2

23

References for Further Study

- Bierhorst, P. et al. (2018), "Experimentally Generated Random Numbers Certified by the Impossibility of Superluminal Signaling," *Nature*, April 12, pp. 223–226.
- Dodge, Y. (1996), "A Natural Random Number Generator," *International Statistical Review*, vol. 64, pp. 329–344.
- Law, A. M. (2007), *Simulation Modeling and Analysis* (4th ed.), McGraw-Hill, New York, Chap. 8.
- L'Ecuyer, P. (1998), "Random Number Generation," in *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice* (J. Banks, ed.), Wiley, New York, Chap. 4.
- L'Ecuyer, P. (2012), "Random Number Generation," in *Handbook of Computational Statistics* (J. E. Gentle, W. Härdle, and Y. Mori, eds.), Springer, Chap. 3 (pp. 35–71).
- Matsumoto, M. and Nishimura, T. (1998), "Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator", *ACM Transactions on Modeling and Computer Simulation* , vol. 8, pp. 3–30.
- Moler, C. (2004), *Numerical Computing with MATLAB* (Chap. 9: Random Numbers), SIAM, Philadelphia (online at www.mathworks.com/moler/chapters.html).
- Neiderreiter, H. (1992), *Random Number Generation and Quasi-Monte Carlo Methods*, SIAM, Philadelphia.
- Rubinstein, R. Y. and Kroese, D. P. (2017), *Simulation and the Monte Carlo Method* (3rd ed.), Wiley, New York. **[Course text]**
- Zeisel, H. (1986), "A Remark on Algorithm AS 183. An Efficient and Portable Pseudo-Random Number Generator," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 35, p. 89.

24