Random Number Generation

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

Overall Framework for Generating Random Numbers

- State at step *k* given transition function f_k:

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

- Output function, *g*, produces pseudo random numbers as

$$U_k = g_k(J_k)$$

- 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

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)

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
- 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

Types of Random Number Generators

- **Linear:** commonly used
- **Combined:** can increase period and improve statistical properties
- **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, etc.)
  - Not as widely used as computer-based generators due to costliness of implementation, lack of speed, and inability to reproduce same sequence

Linear Congruential Generators

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

$$J_k = \left(aJ_{k-1} + c\right) \bmod m$$

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

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

- LCG appears to be most widely used and studied random number generator
- Values *a*, *c*, and *m* should be carefully chosen:

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

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

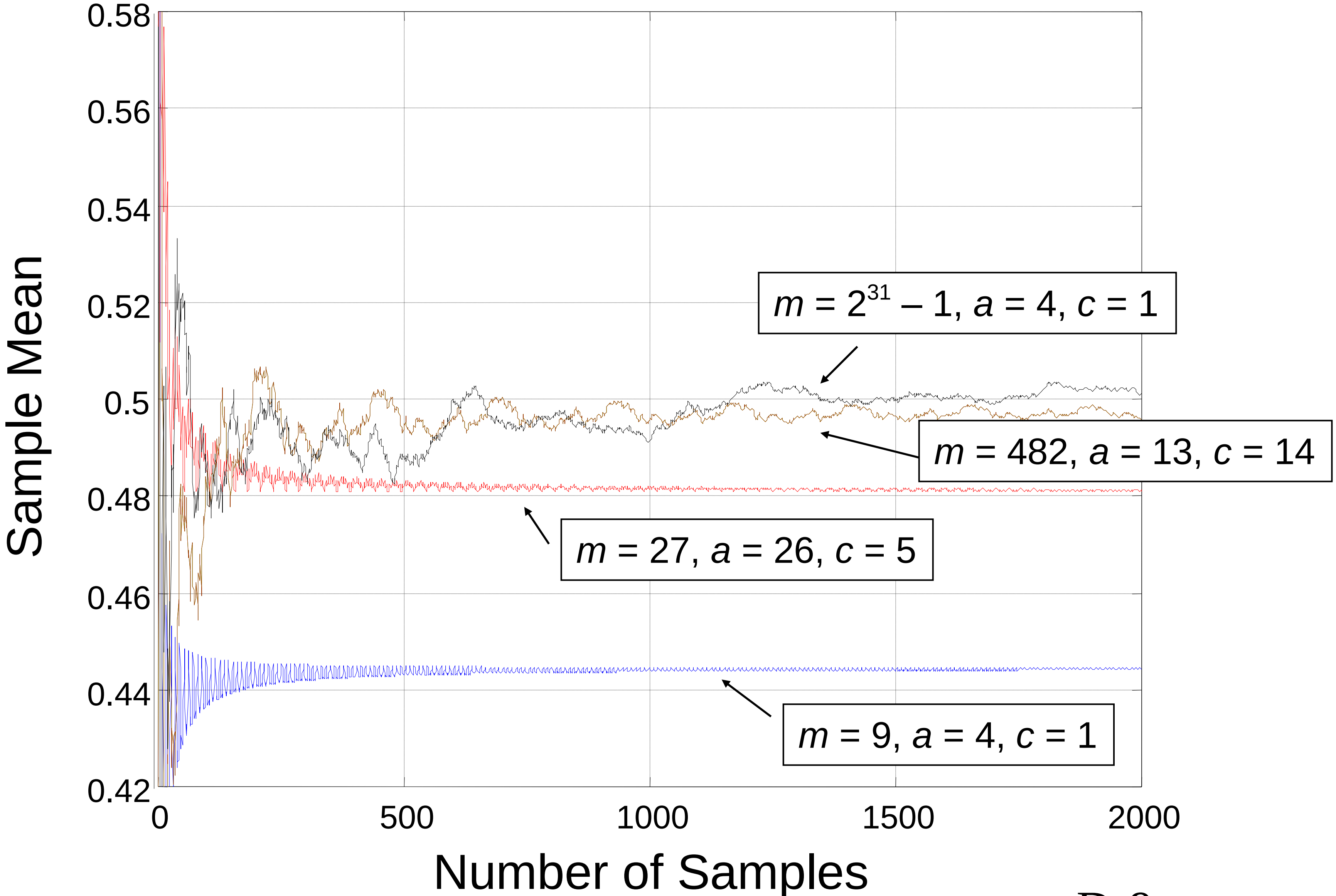
$$0 < J_0 < m, J_k \in \{0,1,\ldots,m-1\}$$

Linear Congruential Generators

- Some famous values for *a* and *m* (assuming *c* = 0)
  - $a = 23$, $m = 10^8 + 1$ (first LCG; original 1951 implementation*)
  - $a = 65539$, $m = 2^{31} - 1$ (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)

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

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



Fibonacci Generators

- These are generators where current value is sum (or difference, or XOR) of two preceding elements
- Lagged Fibonacci generators use two numbers earlier in sequence

$$J_k = (J_{k-p} + J_{k-r}) \bmod m$$

$$U_k = \frac{J_k}{m}$$

p, q are the lags

Multiple Recursive Generators

- Multiple recursive generators (MRGs) are defined by

$$J_k = \left(a_1 J_{k-1} + \cdots + a_k J_{k-r}\right) \bmod m$$

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

  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
- For $r = 1$, MRG reduces to LCG

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) and in reducing discrepancy
- Two examples (L'Ecuyer, 1998)
  - Nonlinear $f = f_k$ via quadratic recursion:

$$J_k = (aJ_{k-1}^2 + bJ_{k-1} + c) \bmod m$$
$$U_k = J_k / M$$

  - Nonlinear f_k via *inversive generator* :

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

Combining Generators

- 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)

Random Number Generators Used in Common Software Packages

- Important to understand types of generators used in statistical software packages and their limitations
- MATLAB:
  - Versions earlier than 5: LCG with a = 75 = 16807, c = 0, $m = 2^{31} - 1$
  - 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 $\sim 2^{19937}$)
- EXCEL: U_k = fractional part $(9821 U_{k-1} + 0.211327)$; period $\sim 2^{23}$
- SAS (vers. 6): LCG with period $2^{31} - 1$

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

$$X = F^{-1}(U)$$

- Example: exponential distribution

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

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

Accept–Reject Method

- Let $p_X(x)$ be density function of X
- Find function $\varphi(x)$ that *majorizes* $p_X(x)$
  - Have $\varphi(x) = Cq(\mathrm{x})$, $C \geq 1$, q is density function that is "easy" to generate outcomes from
- Accept–reject method generates X by following steps:

  Generate U from $U(0,1)$ **(*)**

  Generate Y from $q(y)$, independent of U

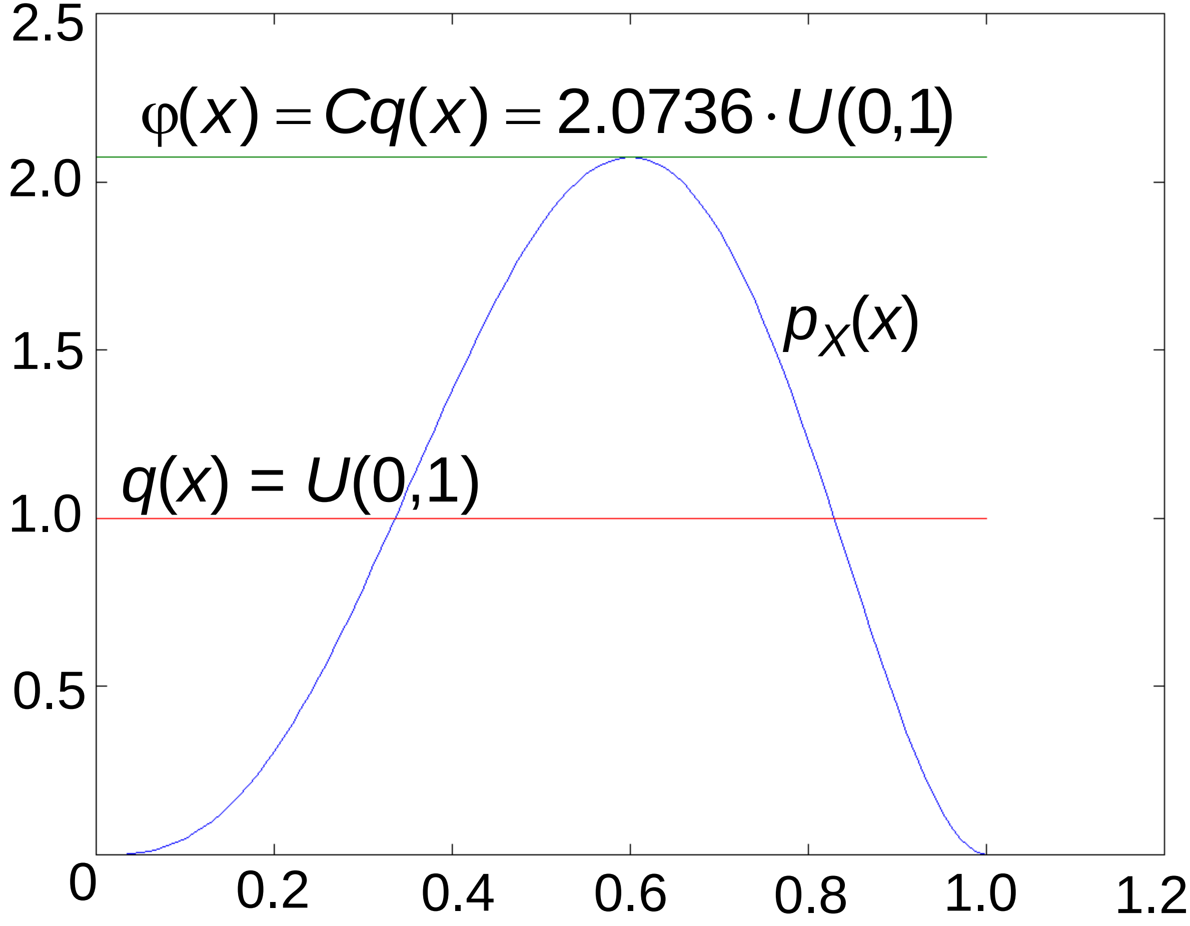
  If $U \leq \dfrac{p_X(Y)}{\varphi(Y)}$, then set $X = Y$. Otherwise, go back to **(*)**

- Probability of acceptance (efficiency) = $1/C$
- Related to Markov chain Monte Carlo (MCMC) (see Exercise 16.4 of *ISSO*)
- Example to follow next two slides ($p_X(x)$ = beta density)….

$$p_X(x) = \begin{cases} 60x^3(1-x)^2 & \text{if } 0 \le x \le 1 \\ 0 & \text{otherwise} \end{cases}$$

$$Y \sim q(y) = U(0,1)$$

$$U \le \frac{60Y^3(1-Y)^2}{2.0736}$$



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

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

$Y \sim q(y) = U(0,1)$: 0.7621, 0.4565, 0.0185, 0.8214, 0.4447, $\cdots$

$$\frac{p_X(Y)}{Cq(Y)} : 0.7249,\ 0.8131,\ 0.00018, \cdots$$

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

$X \sim p_X(x)$: ~~0.7621~~, 0.4565, ~~0.0185~~, $\cdots$

reject **accept** **reject**

Accepted values represent realization of random numbers from $p_X(x)$

References for Further Study

- 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. (2004), "Random Number Generation," in *Handbook of Computational Statistics* (J. E. Gentle, W. Härdle, and Y. Mori, eds.), Springer, Chap. II.2 (pp. 35–70).
- 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.