

Discrete-Event Simulation: Multi-Stream Lehmer RNGs

Lawrence M. Leemis and Stephen K. Park, Discrete-Event Simulation - A First Course, Prentice Hall, 2006

Hui Chen

Computer Science
Virginia State University
Petersburg, Virginia

March 1, 2017

Table of Contents

- 1 Project
- 2 Introduction
- 3 Streams
- 4 Jump Multipliers
 - Numerical Examples
- 5 SSQ with Multiple Job Types

Discussion on Project and Other Assignments

- ▶ Exercises L4-1, L4-2, L4-3, L4-4, and L4-5
- ▶ Project 1

Introduction

- ▶ Typical DES models have many stochastic components.
 - ▶ e.g., arrivals and services
- ▶ Have a unique randomness for each stochastic component
- ▶ One option: multiple RNGs
 - ▶ Often considered a *poor* option
- ▶ One RNG with multiple “streams” of random numbers
 - ▶ One stream per stochastic component
 - ▶ Considered a *better* option
 - ▶ Method: partition output from a RNG into multiple streams
 - ▶ We have been using the Lehmer RNG from the authors of the textbook

Lehmer RNG

- ▶ Lehmer RNG used in *ssq2* and *sis2*

- ▶ in C/C++: *rng.h* and *rng.c*

```
double Random(void);
void    PutSeed(long x);
void    GetSeed(long *x);
void    TestRandom(void);
```

- ▶ in Java (and C++): *Rng.java* (and *Rng.cpp* with minor syntax difference)

```
class Rng {
    [...]
    public double random() {[...]}
    public void putSeed(long x) {[...]}
    public long getSeed() {[...]}
    public void testRandom() {[...]}
}
```

Revisit Simulation Program *ssq2*

- ▶ *ssq2* has two stochastic components: arrival process and service process, e.g.,

```
double GetArrival(void) {  
    static double arrival = START;  
  
    arrival += Exponential(2.0);  
    return (arrival);  
}
```

```
double GetService(void) {  
    return (Uniform(1.0, 2.0));  
}
```

Lehmer RNG: Partition Output into Multiple Streams: 1st Approach

- ▶ First attempt: partition output from our Lehmer RNG into multiple streams
 - ▶ Method: allocate and retain internal state of the RNG for each stochastic process

Lehmer RNG: Partition Output into Multiple Streams: 1st Approach

- ▶ Allocate a different generator state variable to each process and retain it before switching to the other process
 - ▶ In the Lehmer RNG, the generator state is fully represented by the *seed*
 - ▶ Allocate to the service process its own *static* variable and initialized with a value.

GetService with Unique Seed

```
double GetService(void) {  
    double s;  
    static long x = 12345;  
    PutSeed(x);  
    s = Uniform(1.0, 2.0);  
    GetSeed(&x);  
    return (s);  
}
```


Lehmer RNG: Partition Output into Multiple Streams: 1st Approach

- ▶ Allocate to arrival process its own *static* variable and initialized with a *different* value from the service process

GetArrival with Unique Seed

```
double GetArrival(void) {  
    static double arrival = START;  
    static long x = 54321;  
    PutSeed(x);  
    arrival += Exponential(2.0);  
    GetSeed(&x);  
    return (arrival);  
}
```

where x represents the current state of the service process

Modified Arrival and Service Processes

- ▶ As modified, arrival and service times are drawn from different streams of random numbers
- ▶ Provided the streams do not overlap, the processes are uncoupled
- ▶ Although the choice of seed for each stream is deceptively simple, the choices may in fact be *poor* ones.
- ▶ Execution time cost is negligible (see Example 3.2.3 in next slide)

Exercise L6-1

Complete Example 3.2.3 as instructed as follows,

- ▶ Make a copy of *ssq2*. To ease the discussion, call the new copy *ssq2b*
- ▶ Replace *GetArrival* and *GetService* by those introduced in the *1st approach*
- ▶ Set *LAST* to 1,000,000 jobs
- ▶ Compile and run the program. Measure the execution time of new and original programs
- ▶ Compare the results with from those using *ssq2* (without any modification). Are they the same?
- ▶ Compare execution time of the new and original programs, how much slower is the new program?

Exercise L6-1: Hints

To measure program execution time in Windows, use *PowerShell*. Below is an example that measures the execution time of a *dir* command,

Measure Program Execution Time

```
C:> powershell
PS C:> Measure-Command {dir | Out-Default}
Directory: C:\
Mode                LastWriteTime         Length Name
-----
[...]
Days                : 0
Hours               : 0
Minutes            : 0
Seconds            : 0
Milliseconds       : 129
Ticks              : 1299779
TotalDays          : 1.50437384259259E-06
TotalHours         : 3.61049722222222E-05
TotalMinutes       : 0.00216629833333333
TotalSeconds       : 0.1299779
TotalMilliseconds  : 129.9779
```

See

<https://technet.microsoft.com/en-us/library/hh849910.aspx>

Exercise L6-1: Hints

To measure program execution time in Linux, use the *time* command. Below is an example that measures the execution time of a *ls* command,

Measure Program Execution Time

```
$ time ls ssq2.cpp
ssq2.c

real    0m0.012s
user    0m0.004s
sys     0m0.008s
$
```

For more detail on the command, see its manual page (i.e., *man time*).

Streams using Multiple Seeds: Discussion

- ▶ Objective: allocate a unique stream of random numbers of each stochastic component
 - ▶ Examples of stochastic components: arrival and service processes
- ▶ Discussed approach: using *multiple seeds* of RNGs to produce *multiple unique streams* of random numbers
- ▶ Potential problem: assignment of initial seeds (or initial state)
 - ▶ Initial states should be chosen to produce *disjoint* streams
 - ▶ If states are picked at whim, no *guarantee* of disjoint streams
 - ▶ Some initial states could be just a few calls to *Random()* away from one another

Jump Multipliers

- ▶ Objective: produce multiple *disjoint* streams of random numbers
- ▶ Theorem 3.2.1 is the key to creating streams

Theorem 3.2.1

Given $g(x) = ax \pmod m$ and integer j with $j = 1, 2, \dots, m - 1$, the associated *jump function* is

$$g^j(x) = (a^j \pmod m)x \pmod m \quad (1)$$

and has the jump multiplier $a^j \pmod m$
if $(g(\cdot))$ generates x_0, x_1, x_2, \dots then $g^j(\cdot)$ generates x_0, x_j, x_{2j}, \dots

Numerical Examples

- ▶ If $m = 31$ and $a = 3$ and $j = 6$, the jump multiplier is $a^j \bmod m = 3^6 \bmod 31 = 16$
- ▶ If $x_0 = 1$ then $g(x) = 3x \bmod 31$ generates

1, 3, 9, 27, 19, 26, 16, 17, 20, 29, 25, 13,
8, 24, 10, 30, 28, 22, 4, 12, 5, 15, 14, 11,
2, 6...

- ▶ The jump function $g^6(x) = 16x \bmod 31$ generates 1, 16, 8, 4, 2, ...
i.e., the first sequence is x_0, x_1, x_2, \dots ; the second is x_0, x_6, x_{12}, \dots

Program for Numerical Example

```

#include <stdio.h>
#include <math.h>

int main() {
    long m = 31, a = 3, j = 6, jm, i, n =
        32, x = 1;

    jm = (long)pow(a, j) % m;

    printf("%2ld", x);
    for (i = 0; i < n; i++) {
        printf(" ", %2ld", x = a * x % m);
    }

    printf("\n\nJump Multiplier = %ld\n\n",
        jm);

    x = 1;
    printf("%2ld", x);
    for (i = 0; i < n/6; i++) {
        printf(" ", %2ld", x = jm * x % m);
    }
    printf("\n");

    return 0;
}

```

```

import java.io.*;
import java.lang.Math;
import java.text.*;

public class Ex3_2_4{

    public static void main(String [] args) {
        String format = "%2d";

        long m = 31, a = 3, j = 6, jm, i, n
            = 32, x = 1;

        jm = (long)Math.pow(a, j) % m;

        System.out.format(format, x);
        for (i = 0; i < n; i++) {
            System.out.format(" ", " +
                format, x = a * x % m);
        }
        System.out.format("\n\nJump
            Multiplier = " + format +
                "\n\n", jm);

        x = 1; System.out.format(format, x);
        for (i = 0; i < n/6; i++) {
            System.out.format(" ", " +
                format, x = jm * x % m);
        }
        System.out.print("\n");
    }
}

```

Using Jumper Function

1. Compute the jump multiplier $g^j(\cdot) = a^j \pmod m$, which is a one time cost.
2. $g^j(\cdot)$ permits jumping from x_0 to x_j to x_{2j} to \dots
3. User supplies one initial seed
4. If j is chosen well, $g^j(\cdot)$ can “plant” additional initial seeds
5. Each planted seed corresponds to a different stream
6. Each planted seed is separated by j calls to *Random()*

Maximal Modulus-Compatible Jump Multiplier

Definition 3.2.1

Given a Lehmer random-number generator with prime modulus m , full-period modulus-compatible multiplier a , and a requirement for s disjoint streams as widely separated as possible, the maximal jump multiplier is $a^j \bmod m$, where j is the largest integer less than $\lfloor m/s \rfloor$ such that $a^j \bmod m$ is modulus-compatible with m .

Example 3.2.6

Jump multipliers for $(a, m) = (48271, 2^{31} - 1) = (48271, 2147483647)$ RNG

# of streams s	$\lfloor m/s \rfloor$	jump size j	jump multiplier $a^j \bmod m$
1024	$\lfloor \frac{2^{31}-1}{1024} \rfloor = 2097151$	2082675	$48271^{2082675} \bmod 2147483647 = 97070$
512	$\lfloor \frac{2^{31}-1}{512} \rfloor = 4194303$	4170283	$48271^{4170283} \bmod 2147483647 = 44857$
256	$\lfloor \frac{2^{31}-1}{256} \rfloor = 8388607$	8367782	$48271^{8367782} \bmod 2147483647 = 22925$
128	$\lfloor \frac{2^{31}-1}{128} \rfloor = 16777215$	16775552	$48271^{16775552} \bmod 2147483647 = 40509$

Jump Multiplier: A Simple Search Program

Jump Multiplier Search Program in C

```
#include <stdio.h>

long find_m_compatible(long upper, long a, long m);
long modular_pow(long long base, long long exponent, long long modulus);

int main()
{
    long a = 482711, m = 21474836471,
        nstreams[] = {10241, 5121, 2561, 1281}, i, n;

    for (i = 0; i < sizeof(nstreams)/sizeof(long); i++) {
        n = find_m_compatible(m/nstreams[i], a, m);
        printf("%ld %ld %ld %ld\n", nstreams[i], m/nstreams[i], n,
            modular_pow((long long)a, (long long)n, (long long)m));
    }

    return 0;
}
```

Jump Multiplier: A Simple Search Program

Definition 2.2.1

The multiplier a is modulus-compatible with the prime modulus m if and only if $r < q$ where $r = m \bmod a$ and $q = \lfloor m/a \rfloor$

Functions in Jump Multiplier Search Program

```

long modular_pow(long long base, long long exponent, long long modulus);

long find_m_compatible(long upper, long a, long m)
{
    long i, n, r, q;

    for (i = upper; i >= 1; i --) {
        n = modular_pow((long long)a, (long long)i, (long long)m);
        r = m % n;
        q = m / n;
        if (r < q) { /* if n is modulus-compatible with m */
            return i;
        }
    }

    return 0;
}

```

Jump Multiplier: A Simple Search Program

This function is to compute $base^{exponent} \bmod modulus$

Functions in Jump Multiplier Search Program

```
/*
 * Reference
 * http://www.sanfoundry.com/cpp-program-implement-modular-exponentiation-algorithm/
 */

long modular_pow(long long base, long long exponent, long long modulus)
{
    long long result = 1ll;

    while (exponent > 0ll) {
        if (exponent % 2ll == 1ll) {
            result = (result * base) % modulus;
        }

        exponent = exponent >> 1ll;

        base = (base * base) % modulus;
    }

    return (long) result;
}
```

Jump Multiplier: A Simple Search Program

Jump Multiplier Search Program in Java

```
public class JumpMultiplier {
    public static void main(String [] args) {
        JumpMultiplier jm = new JumpMultiplier ();

        long a = 482711, m = 21474836471,
            nstreams [] = {10241, 5121, 2561, 1281}, n;

        for (int i = 0; i < nstreams.length; i++) {
            n = jm.find_m_compatible(m/nstreams[i], a, m);
            System.out.format("%d %d %d %d\n", nstreams[i], m/nstreams[i], n,
                jm.modular_pow(a, n, m));
        }
    }
}
```

.....

Jump Multiplier: A Simple Search Program

Jump Multiplier Search Program in Java

.....

```
private long modular_pow(long base, long exponent, long modulus) {
    long result = 1l;

    while (exponent > 0l) {
        if (exponent % 2l == 1l) {
            result = (result * base) % modulus;
        }

        exponent = exponent >> 1l;

        base = (base * base) % modulus;
    }

    return result;
}
```

.....

Jump Multiplier: A Simple Search Program

Jump Multiplier Search Program in Java

.....

```
private long find_m_compatible(long upper, long a, long m) {
    long i, n, r, q;

    for (i = upper; i >= 1; i --) {
        n = modular_pow(a, i, m);
        r = m % n;
        q = m / n;
        if (r < q) { /* if n is modulus-compatible with m */
            return i;
        }
    }

    return 0;
}
```

Exercise L6-2

- ▶ Use the program discussed above to compute the jump multiplier table similar to slide 19.
 - ▶ You will enter, compile, and run the programs.

Library *rngs*

- ▶ *rngs* is an upward-compatible multi-stream replacement for *rng*
- ▶ By default, provides 256 streams, indexed 0 to 255 (0 is the default)
- ▶ Only one stream is active at any time
- ▶ Six available functions:
 - ▶ `Random(void)`
 - ▶ `PutSeed(long x)`: superseded by `PlantSeeds`
 - ▶ `GetSeed(long *x)`
 - ▶ `TestRandom(void)`
 - ▶ `SelectStream(int s)`: used to define the active stream
 - ▶ `PlantSeeds(long x)`: “plants” one seed per stream
- ▶ Henceforth, *rngs* is the library of choice

Example 3.2.7: *ssq2* Revisited

- ▶ Use *rngs* functions for *GetArrival*, *GetService*
- ▶ Include *rngs.h* and use *PlantSeeds(12345)*

GetArrival Method

```
double GetArrival(void) {  
    static double arrival = START;  
    SelectStream(0);  
    arrival += Exponential(2.0);  
    return (arrival);  
}
```

GetService Method

```
double GetService(void) {  
    SelectStream(2);  
    return (Uniform(1.0, 2.0));  
}
```

Exercise L6-3

Complete Example 3.2.7 as instructed as follows,

- ▶ Make a copy of *ssq2*. To ease the discussion, call the new copy *ssq2c*
- ▶ Replace *GetArrival* and *GetService* by those using library rngs. Find library rngs in Blackboard.
- ▶ Set *LAST* to 1,000,000 jobs
- ▶ Compile and run the program. Measure the execution time of new and original programs
- ▶ Compare the results with from those using *ssq2* (without any modification) and those using *ssq2b*(Exercise L6-1). Are they the same?
- ▶ Compare execution time of the new and original programs, how much slower is the new program?

Uncoupling Stochastic Processes

- ▶ Per modifications, arrival and service processes are uncoupled
- ▶ Consider changing the service process to $Uniform(0.0, 1.5) + Uniform(0.0, 1.5)$
- ▶ Without uncoupling, arrival process sequence would change!
- ▶ With uncoupling, the service process sees exactly the same arrival sequence
- ▶ Important variance reduction technique

Single-Server Service Node with Multiple Job Types

- ▶ Extend the single-server service node model from Chapter 1
 - ▶ Consider multiple job types, each with its own arrival and service process
 - ▶ Examples 3.2.8 and 3.2.9: Suppose there are two job types
 1. Exponential(4.0) interarrivals, Uniform(1.0, 3.0) service
 2. Exponential(6.0) interarrivals, Uniform(0.0, 4.0) service
- Use rngs to allocate a different stream to each stochastic process

Arrival Process for Multiple Job Types

The arrival process generator in program `ssq2` can be modified as follows,

Example 3.2.8: Arrival Process

```

double GetArrival(int *j) {           /* returns job type in j */
  const double mean[2] = {4.0, 6.0}; /* two job types */
  static double arrival[2] = {START, START};
  static int init = 1;
  double temp;
  if (init) {                          /* initialize the arrival array */
    SelectStream(0);
    arrival[0] += Exponential(mean[0]);
    SelectStream(1);
    arrival[1] += Exponential(mean[1]);
    init = 0;
  }
  if (arrival[0] <= arrival[1])
    *j = 0;                             /* next arrival is job type 0 */
  else
    *j = 1;                             /* next arrival is job type 1 */
  temp = arrival[*j];                  /* next arrival time to be returned */
  SelectStream(*j);
  arrival[*j] += Exponential(mean[*j]); /* arrival after next arrival */
  return (temp);
}

```


Service Process for Multiple Job Types

The service process generator in program `ssq2` can be modified as follows,

Example 3.2.9: Service Progress

```
double GetService(int j)
{
    const double min[2] = {1.0, 0.0};
    const double max[2] = {3.0, 4.0};
    /*
     * Two RNG streams, i.e., streams 0 and 1 are used in the arrival
     * process generator. We now use streams 2 and 3 for the service
     * process generator. In the following, j should be either 0 or 1.
     */
    SelectStream(j + 2);
    return (Uniform(min[j], max[j]));
}
```

Service Process for Multiple Job Types

- ▶ Index j matches service time to appropriate job type
- ▶ All four simulated stochastic processes are uncoupled
- ▶ Any process could be changed without altering the random sequence of others!

Consistency Check

- ▶ Additional modification to *ssq2*
 - ▶ job-type-specific statistics-gathering needs to be added in the *main* method/function
- ▶ How do we know if our modifications are correct? *Use consistency check to increase confidence.*
 - ▶ $\bar{w} = \bar{d} + \bar{s}$
 - ▶ $\bar{l} = \bar{q} + \bar{x}$
 - ▶ How about average service time of both job types?
Since $(1.0 + 3.0)/2 = (0.0 + 4.0)/2 = 2.0$, we expect $\bar{s} = 2.0$
 - ▶ How about the net arrival rate of both job types?
Since the arrival rates of job types 0 and 1 are $1/4$ and $1/6$, respectively, we expect the net arrival rate should be $1/4 + 1/6 = 5/12$, i.e., $\bar{r} = 1/(5/12) = 12/5 = 2.4$.
 - ▶ The steady-state utilization should be the ratio of the arrival rate to the service rate, i.e., $(5/12)/(1/20) = 5/6 \approx 0.83$

Exercise L6-4

Modify program *ssq2* to support two job types as discussed.

- ▶ Make a copy of *ssq2*. To ease the discussion, call the new copy *ssq2d*
- ▶ Modify program *ssq2d* as suggested in Examples 3.2.8 and 3.2.9.
- ▶ Modify the *main* method/function to include job-type-specific statistics
- ▶ Answer the following questions,
 - ▶ What portion of processed jobs are of type 0?
 - ▶ What are \bar{w} , \bar{d} , \bar{s} , \bar{l} , \bar{q} , and \bar{x} for each job type?
 - ▶ What did you do to convince yourself that your results are valid (hint: consistency check)?
 - ▶ Why are \bar{w} , \bar{d} , and \bar{s} the same for both job types, while \bar{l} , \bar{q} , and \bar{x} are different?

Summary

- ▶ Multiple stream RNGs
 - ▶ Generate disjoint random number streams
 - ▶ Want the streams are far apart
- ▶ More simulation examples to be discussed