Lehmer RNGs: Implementation

For 32-bit systems, $2^{31} - 1$ is the largest prime
We will develop an $m = 2^{31} - 1$ Lehmer generator
  - portable, efficient
  - in ANSI C

Recall that $g(x) = ax \mod m$
ax product can be as big as $a(m-1)$

If max integer value $< m$, overflow is possible

Possible Overflow

Consider $(a, m) = (7, 13)$ for a 5-bit machine
$a(m-1) = 84 \approx 1.31 \times 2^6 \implies$ at least 7 bits

Consider $(a, m) = (48271, 2^{31} - 1)$
  - $a(m-1) \approx 1.47 \times 2^{46} \implies$ at least 47 bits
  - Even though $ax \mod m$ no more than 31 bits

Need a way to “trap” the $ax$ computation

Type Considerations

ANSI C standard guarantees 32 bits for `long`
  - 64 bits for `long long`
  - Requires overhead on 32-bit systems

Floating-point representation is inexact
An efficient integer-based implementation exists

More bits does not alleviate the problem
  - e.g., 64 bits, $m = 2^{64} - 25$: overflow possible
Algorithm Development

- Integer-based implementation, no result $> m = 2^{31} - 1$
- Recall $m$ is prime: write $m = aq + r$ where
  $$q = \left\lfloor \frac{m}{a} \right\rfloor \quad r = m \mod a$$
- Consider $(a, m) = (48 \, 271, 2^{31} - 1)$
  $$q = 44 \, 488 \quad r = m \mod a = 3399$$
- Consider $(a, m) = (16 \, 807, 2^{31} - 1)$
  $$q = 127 \, 773 \quad r = 2836$$
- Note $r < q$

Rewriting $g(x)$ To Avoid Overflow

- We have $g(x) = ax \mod m = \gamma(x) + m\delta(x)$ where
  $$\gamma(x) = a(x \mod q) + r[x/q]$$
  $$\delta(x) = [x/q] - [ax/m]$$
- Note the $ax$ product is “trapped”
- Can we avoid computing $ax$ altogether?

Rewriting $g(x)$ To Avoid Overflow

$$g(x) = ax \mod m$$
$$= ax - m[ax/m]$$
$$= ax + \left[ -m[x/q] + m[x/q] \right] - m[ax/m]$$
$$= \left[ ax - m[x/q] \right] + \left[ m[x/q] - m[ax/m] \right]$$
$$= \left[ ax - (aq + r)[x/q] \right] + \left[ m[x/q] - m[ax/m] \right]$$
$$= \left[ ax \mod q \right] - r[x/q] + m\left[ x/q \right] - [ax/m]$$
$$= \gamma(x) + m\delta(x)$$

Trapping $ax$ Product

Lemma

If $u, v \in \mathbb{R}$ with $0 < u - v < 1$, $\lfloor u \rfloor - \lfloor v \rfloor$ is either 0 or 1.

Proof.

$$0 < u - v < 1 \implies v < u < v + 1$$
$$\implies u = v + \epsilon \quad \text{with} \quad 0 < \epsilon < 1$$
- Let $k$ be largest integer $\leq v \implies \lfloor v \rfloor = k$
- Because $0 < \epsilon < 1$, either
  $$\lfloor u \rfloor = \lfloor v + \epsilon \rfloor = k \quad \text{or} \quad \lfloor u \rfloor = \lfloor v + \epsilon \rfloor = k + 1$$
- Thus, either
  $$\lfloor u \rfloor - \lfloor v \rfloor = k - k = 0 \quad \text{or} \quad \lfloor u \rfloor - \lfloor v \rfloor = (k + 1) - k = 1$$
Trapping $ax$ Product

Lemma

If $u, v \in \mathbb{R}$ with $0 < u - v < 1$, $\lfloor u \rfloor - \lfloor v \rfloor$ is either 0 or 1.

- We have $g(x) = ax \mod m = \gamma(x) + m\delta(x)$ where
  \[
  \gamma(x) = a(x \mod q) + r \lfloor x/q \rfloor \\
  \delta(x) = \lfloor x/q \rfloor - \lfloor ax/m \rfloor
  \]
- Note the ax product is “trapped”
- Can we avoid computing $ax$ altogether?
- Yes, if we can show $0 < x/q - ax/m < 1$

Show $\delta(x)$ Is Either 0 Or 1

Theorem (2.2.1)

If $m = aq + r$ is prime and $r < q$ and $x \in X_m$

\[
\delta(x) = 0 \quad \text{or} \quad \delta(x) = 1
\]

where $\delta(x) = \lfloor x/q \rfloor - \lfloor ax/m \rfloor$

Proof.

Note for $u, v \in \mathbb{R}$ with $0 < u - v < 1$, $\lfloor u \rfloor - \lfloor v \rfloor$ is 0 or 1

\[
0 < \frac{x}{m} - \frac{ax}{m} = x \left( \frac{1}{q} - \frac{a}{m} \right) = x \left( \frac{m - aq}{mq} \right) = \frac{xr}{mq}
\]

and since $r < q$

\[
0 < \frac{xr}{mq} < \frac{x}{m} \leq \frac{m - 1}{m} < 1
\]

Show $\delta(x)$ Depends Only On $\gamma(x)$

Theorem (2.2.1)

With $\gamma(x) = a(x \mod q) - r \lfloor x/q \rfloor$

\[
\delta(x) = 0 \quad \text{iff.} \quad \gamma(x) \in X_m \\
\delta(x) = 1 \quad \text{iff.} \quad -\gamma(x) \in X_m
\]

Proof.

- If $\delta(x) = 0$, then $g(x) = \gamma(x) + m\delta(x) = \gamma(x) \in X_m$
  If $\gamma(x) \in X_m$, then $\delta(x) \neq 1$ otherwise $g(x) \notin X_m$
- If $\delta(x) = 1$, then $g(x) = \gamma(x) + m \in X_m \Rightarrow -\gamma(x) \in X_m$
  If $-\gamma(x) \in X_m$, then $\delta(x) \neq 0$ otherwise $g(x) \notin X_m$
Summarizing

- Write \( g(x) = ax \mod m \) as \( g(x) = \gamma(x) + m\delta(x) \) where
  \[
  \gamma(x) = a(x \mod q) + r\lfloor x/q \rfloor \\
  \delta(x) = \lfloor x/q \rfloor - \lfloor ax/m \rfloor
  \]
- Compute \( \gamma(x) \) (no potential for overflow)
  - If \( \gamma(x) > 0 \), then \( \delta(x) = 0 \)
  - If \( \gamma(x) < 0 \), then \( \delta(x) = 1 \)
- The \( ax \) product is “trapped”
- Can compute \( g(x) = ax \mod m \) without overflow

Computing \( g(x) \) Without Overflow

- Evaluates \( g(x) = ax \mod m \) with no values > \( m - 1 \)

**Algorithm 2.2.1**

\[
\begin{align*}
  t &= a \times (x \mod q) - r \times (x / q); \\
  &\quad /* t = \gamma(x) */ \\
  \text{if} \ (t > 0) \ \\
  \quad \text{return } (t); \\
  \quad /* \delta(x) = 0 */ \\
  \text{else} \ \\
  \quad \text{return } (t + m); \\
  \quad /* \delta(x) = 1 */
\end{align*}
\]

- Returns \( g(x) = \gamma(x) + m\delta(x) \)
- No overflow: \( ax \) product is “trapped” in \( \delta(x) \)

Modulus Compatibility

- Multiplier \( a \) is **modulus-compatible** with \( m \) iff. \( r < q \)
- We must have \( r < q \) in \( m = aq + r \)
  \[
  0 < \frac{xr}{mq} < \frac{x}{m} \leq \frac{m - 1}{m} < 1
  \]
- E.g., \( a = 48271 \) is modulus-compatible with \( m = 2^{31} - 1 \)
  \[
  r = 3399 \quad q = 44488
  \]

Modulus-Compatible and Full-Period

- No modulus-compatible multipliers > \( (m - 1)/2 \)
- More densely distributed on low end
- Consider (tiny) modulus \( m = 401 \):
  (Row 1: MC, Row 2: FP, Row 3: MC & FP)
Modulus-Compatibility and Smallness

- Multiplier \( a \) is "small" iff. \( a^2 < m \)
- If \( a \) is small, then \( a \) is modulus-compatible
- If \( a \) is modulus-compatible, \( a \) is not necessarily small

\( a = 48271 \) is MC with \( m = 2^{31} - 1 \) but is not "small"

Approach

- Start with a small (therefore MC) multiplier
- Search until the first FP multiplier is found (Alg. 2.1.1)

FPMC Multipliers For \( m = 2^{31} - 1 \)

- For \( m = 2^{31} - 1 \) and FPMC \( a = 7 \), there are 23093 FPMC multipliers

\[
\begin{align*}
7^1 \mod 2147483647 &= 7 \\
7^5 \mod 2147483647 &= 16807 \\
7^{113039} \mod 2147483647 &= 41214 \\
7^{188509} \mod 2147483647 &= 25697 \\
7^{536035} \mod 2147483647 &= 63295 \\
\vdots
\end{align*}
\]

- \( a = 16807 \) is a "minimal" standard [link to PDF]
- \( a = 48271 \) provides (slightly) more randomness [link to PDF]

Generate All FPMC Multipliers

- Find one full-period modulus-compatible (FPMC) multiplier
- Extend Alg. 2.1.2 to generate all others

Algorithm 2.2.2

\[
i = 1; \\
x = a; \\
\text{while (} x \neq 1 \text{) }
\]

\[
\begin{align*}
&\quad \text{if } ((m \% x < m / x) \text{ and } \gcd(i, m-1) == 1)) \\
&\quad \quad /* x is full-period & modulus-compatible */ \\
&\quad \quad i++; \\
&\quad \quad x = g(x); /* use Alg. 2.2.1 to evaluate } g(x) */
\end{align*}
\]

ANSI C Implementation

Lehmer RNG in ANSI C with \((a, m) = (48271, 2^{31} - 1)\)

```c
double Random(void)
{
    static long state = 1;
    const long A = 48271; /* multiplier */
    const long M = 2147483647; /* modulus */
    const long Q = M / A; /* quotient */
    const long R = M % A; /* remainder */

    long t = A * (state % Q) - R * (state / Q);
    if (t > 0)
        state = t;
    else
        state = t + M;
    return ((double) state / M);
}
```
A Good RNG Library

- Defined in the source files `rng.h` and `rng.c`
- Based on the implementation considered in this lecture
  ```c
  double Random(void)
  void PutSeed(long seed)
  void GetSeed(long *seed)
  void TestRandom(void)
  ```
- Initial seed can be set directly, via prompt, or by system clock
- `PutSeed()` and `GetSeed()` often used together
- \( a = 48271 \) is the default multiplier

Randomness

- Choose the FPMC multiplier that gives “most random” sequence
- Many statistical tests of randomness
- In 2-space, \( (x_0, x_1), (x_1, x_2), \ldots \) form a lattice structure
- For any integer \( k \geq 2 \), the points
  \[
  (x_0, x_1, \ldots, x_{k-1}), (x_1, x_2, \ldots, x_k), (x_2, x_3, \ldots, x_{k+1}), \ldots
  \]
  form a lattice structure in \( k \)-space
- Numerically analyze uniformity of the lattice
  *E.g., Knuth’s spectral test*

Random Numbers Falling In The Planes

![Random Numbers Falling In The Planes](image)

Using the RNG

- The following generates 400 2-space points at random
  ```c
  seed = 123456789; // or 987654321
  PutSeed(seed);
  x_0 = Random();
  for (i = 0; i < 400; i++)
  {
      x_{i+1} = Random();
      Plot(x_i, x_{i+1});
  }
  ```

Generating 2-Space Points
Observations on Randomness

- Appearance of randomness is an illusion
- If all \( m - 1 \) points were generated, lattice would be evident
- Distinction between ideal and good RNGs

Zoom in on Randomness

- "Zoom in" to square with corners \((0, 0)\) and \((0.001, 0.001)\)
  - \(seed = 123456789\)
  - \(PutSeed(seed)\)
  - \(x_0 = Random()\)
  - \(for (i = 0; i < 2147483646; i++ ) \{
      x_{i+1} = Random();
      if ((x_i < 0.001) and (x_{i+1} < 0.001))
      Plot(x_i, x_{i+1});
  \}
- Any tiny square range should appear (approx.) the same

Scatter Plots for \( m = 2^{31} - 1 \)

- Further justification for using \( a = 48271 \) over \( a = 16807 \)

Other Considerations

Consider if 20 RNs were needed, seed \( x_0 = 109869724 \):

\[
0.64 \ 0.72 \ 0.77 \ 0.93 \ 0.82 \ 0.88 \ 0.67 \ 0.76 \ 0.84 \ 0.84 \\
0.74 \ 0.76 \ 0.80 \ 0.75 \ 0.63 \ 0.94 \ 0.86 \ 0.63 \ 0.78 \ 0.67
\]
- Nothing "wrong" with this sequence
- Do not throw away "outliers"

**Cycling:** generating more than \( m - 1 \) random values
- Cycling should be avoided within a single simulation
In-class Work: Replicate These Figures

- Use `rngs.h, rngs.c` under “Code” link on course web page
- In separate C/C++ program, implement slide 44
- In R:
  - `output_16807 <- read.table("OUTPUT_16807.txt")`
  - `head(output_16807)`
  - `plot(output_16807$V1, output_16807$V2, pch = '.')`