Random Number Generators (RNG)

Advanced Seminar: Monte-Carlo Methods

David Meisel

October 28, 2013

„Gott würfelt nicht.“

(God does not play dice.)

– Albert Einstein

We ain't God, but...

... that'd be a whole lot of playing dice!

Randomness = the lack of any pattern

Many applications:

- games: dice, cards, roulette etc.
- politics: selection of members of a grand jury
- science: statistical sampling (getting statistically relevant results), simulation (Monte-Carlo) etc.
- computer science: cryptography

$\Rightarrow$ *differentiate* :

True random numbers:

- generated by physical phenomena
- examples: actual dice, radioactive decay
- typically used for technical purposes: atmospheric noise, thermal noise, electromagnetic or quantum phenomena
- generation is limited by the *speed* with which entropy harvests

Pseudo-random numbers:

- generated by mathematical/computational algorithms
- need **seed** to initialize the algorithm!
- *quasi*-random: partly rely on physical phenomena (process ID, time, etc.)
- need to be further examined before use!

Properties and test criteria:

- distribution (equally distributed, normal distribution etc.)
- sequential independence (e.g. 0 1 0 1 0 1 0 1 0 1 ...)
- **periodic length**
- different tests: χ^2, Kolmogorow-Smirnow-Test, etc.

Well-known generators: Linear congruential generator, Xorshift, Mersenne twister (periodic length: $2^{19937} - 1$)

Definition:

$$X_{n+1} \equiv (a \cdot X_n + c) \mod m$$

Explanation:

- $m, 0 < m$ is the *modulus*
- $a, 0 < a < m$ is the *multiplier*
- $c, 0 \leq c < m$ is the *increment*
- $X_0, 0 \leq X_0 < m$ is the start value (*seed*)

Periodic length: m at max, often shorter; strongly dependent on initialization values

Based on combination of bit-wise operations Shift and XOR.
Sample code:

```
uint32_t xorshift32() {
  static uint32_t x = 314159265;
  x ^= x << 13;
  x ^= x >> 17;
  x ^= x << 5;
  return x;
```

Periodic length: $2^k - 1$ (for numbers with k bits)