Programmazione Avanzata per il Calcolo Scientifico
Advanced Programming for Scientific Computing
Lecture title: Random numbers and distributions

Luca Formaggia

MOX
Dipartimento di Matematica
Politecnico di Milano

A.A. 2014/2015

C++11 support for statistical distributions

C++11 provides (finally) an extensive support for random number generators and univariate statistical distribution. You need the header `<random>`. The chosen design is based on two types of object

- **Engines** serve as a *stateful source of randomness* They are function objects that provide random unsigned values uniformily distributed in a range;
- **Distributions**. They specify how values generated by the engine have to be transformed to generate a sequence with prescribed statistical properties.

The design separates the *pseudo random number* generators, implemented in the engine, from their use to generate a specific distribution.

Engines

Random number engines generate pseudo-random numbers using seed data as entropy source. Several different classes of pseudo-random number generation algorithms are implemented as templates that can be customized. We have a set of basic engines which implement well know pseudo-random generators. Are class templates (I do not indicate here the template parameters for brevity)

linear_congruential_engine	implements linear congruential algorithm
mersenne_twister_engine	implements Mersenne twister algorithm
subtract_with_carry_engine	implements subtract with carry (a lagged Fibonacci) algorithm

However, for simplicity of use, the library provides some *predefined engines* which are in fact instances of the above with predefined values of the template argument.

Predefined engines

The main one is default_random_engine, which is in fact implementation dependent and should guarantee a good compromise between efficiency and quality of the (pseudo) random numbers.
The others are minstd_rand0, minstd_rand, mt19937 mt19937_64, ranlux24_base,ranlux48_base, ranlux24, ranlux48 and knuth_b.

The library provides also a non deterministic random number generator adapter, called random_device, in case you hardware provides a non-deterministic random number generator (all other engines are deterministic, thus pseudo-random, generators)

Details may be found in in this web page.

How to use a predefined engine

It is rather simple. You generate an object of the chosen class either with the default constructor or providing a seed (a part random_device, which does not require a seed).

```
std::default_random_engine rd1;
// with a seed
std::default_random_engine rd2(1566770);
```

Seeds are unsigned integers that you may use to introduce entropy. If you use the same seed the sequence of pseudo random number will be the same every time you execute the program (the default_random_engine guarantees this property only on the same architecture).

Note: Random engines should never used alone, but always together with a distributions.

Distributions are template functions or classes (again I omit to indicate the template parameters because the default values are usually ok, look at the indicated reference if you want details). They implement a call operator () that accepts in input an engine object: they transform the random sequence into the wanted distribution.

```
#include <random>
#include <iostream>
int main(){
  std::random_device gen;
  std::uniform_int_distribution<> dis(1, 6);
  for(int n=0; n<10; ++n)std::cout << dis(gen) << ' ';
  std::cout << std::endl;
}
```

Here uniform_int_distribution<> dis(1, 6) defines an integer uniform distribution in the range $(1, 6)$.

Distributions

The arguments of the constructor of a distribution (if it is implemented as a class) vary depending on the distribution. However, all distributions have default values. The list of distributions available is very long. We list only some

Uniform	uniform_int_distribution, uniform_real_distributions
Bernoulli	bernoulli_distribution, binomial_distribution
Poisson	poisson_distribution, exponential_distribution, gamma_distribution
Normal	normal_distribution, lognormal_distribution,cauchy_distribution
Sampling	discrete_distribution, piecewise_linear_distribution

A full list in this web page

An utility

std::seed_seq consumes a sequence of integer-valued data and produces a requested number of unsigned integer values i, $0 \leq i < 2^{32}$, based on the consumed data. The produced values are distributed over the entire 32-bit range even if the consumed values are close.

It provides a way to seed a large number of random number engines or to seed a generator that requires a lot of entropy, given a small seed or a poorly distributed initial seed sequence.

```
std::seed_seq seq({1,2,3,4,5});
std::vector<std::uint32_t> seeds(10);
seq.generate(seeds.begin(), seeds.end());
```

Examples

In the directory
RandomDistributions
you find a simple example of various distributions.