A Proposal to Add Random-Number Distributions to C++0x

Document number: WG21/N1914 = J16/05-0174
Date: 2005-10-21
Revises: None
Project: Programming Language C++
Reference: ISO/IEC IS 14882:2003(E)
Reply to: Marc Paterno <paterno@fnal.gov>
Mark Fischler <mf@fnal.gov>
Walter E. Brown <wb@fnal.gov>
Jim Kowalkowski <jbk@fnal.gov>
CEPA Dept., Computing Division
Fermi National Accelerator Laboratory
Batavia, IL 60510-0500

Contents

---

1 Introduction

In [Pat04] we presented an argument in favor of a cohesive (and thus defensible) selection of random-number distributions in C++0x, and identified a corresponding superset of the distributions described in [Mau03]. Our argument was favorably received by the Library Working Group, and we were encouraged by the LWG to propose wording for our additional distributions. This paper provides the proposed wording for these distributions.

2 Base document

This document uses TR1 [Aus05] as its base document. Wording for the next Standard is provided under the assumption that clause 5.1 `[tr.rand]` has already been voted into the Working Paper.

3 Proposed Text

Note: the wording presented in this section describes additions to the `<random>` header.

3.1 New entries for Clause 5.1.2

Add the following text after the synopsis of class template `gamma_distribution`:

```
// [5.1.7.10] Class template negative_binomial_distribution
template<class IntType = int, class RealType = double>
  class negative_binomial_distribution;

// [5.1.7.11] Class template weibull_distribution
template<class RealType = double>
  class weibull_distribution;

// [5.1.7.12] Class template extreme_value_distribution
template<class RealType = double>
  class extreme_value_distribution;

// [5.1.7.13] Class template lognormal_distribution
template<class RealType = double>
  class lognormal_distbution;

// [5.1.7.14] Class template chi_squared_distribution
template<class IntType = int, class RealType = double>
  class chi_squared_distribution;

// [5.1.7.15] Class template cauchy_distribution
template<class RealType = double>
  class cauchy_distribution;

// [5.1.7.16] Class template fisher_f_distribution
template<class IntType = int, class RealType = double>
  class fisher_f_distribution;

// [5.1.7.17] Class template student_t_distribution
template<class IntType = int, class RealType = double>
  class student_t_distribution;

// [5.1.7.18] Class template discrete_distribution
template<class IntType = int>
  class discrete_distribution;

/ [5.1.7.19] Class template piecewise_constant_distribution
template<class RealType = double>
  class piecewise_constant_distribution;

// [5.1.7.20] Class template piecewise_smooth_distribution
template<class RealType = double>
  class piecewise_smooth_distribution;
```

3.2 New subclauses for Clause 5.1.7

Add the following text after clause 5.1.7.9 `[tr.rand.dist.gamma]`:

5.1.7.10 Class template `negative_binomial_distribution`

1 A `negative_binomial_distribution` random distribution produces random integers $i \geq 0$ distributed with probability

$$P(i) = \binom{k+i-1}{i} p^k (1-p)^i \ ,$$

where k and p are the parameters of the distribution.

```
template<class IntType = int, class RealType = double>
class negative_binomial_distribution
{
public:
  // types
  typedef RealType input_type;
  typedef IntType  result_type;

  // constructor and member functions
  explicit negative_binomial_distribution(IntType k = 0,
                                          const RealType& p = 0.5);
  IntType k() const;
  RealType p() const;
  void reset();
  template <class UniformRandomNumberGenerator>
    result_type operator()(UniformRandomNumberGenerator& urng);
};
```

```
  negative_binomial_distribution(IntType k = 0,
                                 const RealType& p = 0.5);
```

2 *Requires:* $0 \leq$ `p` ≤ 1 and `k` ≥ 0.

3 *Effects:* Constructs a `negative_binomial_distribution` object; `k` and `p` are the parameters of the distribution.

```
  IntType k() const;
```

4 *Returns:* The `k` parameter of the distribution.

```
  RealType p() const;
```

5 *Returns:* The `p` parameter of the distribution.

5.1.7.11 Class template `weibull_distribution`

1 A `weibull_distribution` random distribution produces random numbers $x \geq 0$ distributed with probability density function

$$p(x) = \frac{a}{b} \left(\frac{x}{b}\right)^{a-1} \exp\left(-\left(\frac{x}{b}\right)^a\right) \ ,$$

where a and b are the parameters of the distribution.

```
template<class RealType = double>
class weibull_distribution
{
public:
  // types
  typedef RealType input_type;
  typedef RealType result_type;

  // constructor and member functions
  explicit weibull_distribution(const RealType& a = 1 ,
                                const RealType& b = 1)

  RealType a() const;
  RealType b() const;
  void reset();
  template <class UniformRandomNumberGenerator>
    result_type operator()(UniformRandomNumberGenerator& urng);
};


  weibull_distribution(const RealType& a = 1, const RealType& b = 1);
```

2 *Requires:* `a` > 0 and `b` > 0.

3 *Effects:* Constructs a `weibull_distribution` object; `a` and `b` are the parameters of the distribution.

```
  RealType a() const;
```

4 *Returns:* The `a` parameter of the distribution.

```
  RealType b() const;
```

5 *Returns:* The `b` parameter of the distribution.

5.1.7.12 Class template `extreme_value_distribution`

1 An `extreme_value_distribution` random distribution produces random numbers x distributed with probability density function

$$p(x) = \frac{1}{b} \exp\left(\frac{x-a}{b} - \exp\left(\frac{x-a}{b}\right)\right) ,$$

where a and b are the parameters of the distribution.

```
template<class IntType = int, class RealType = double>
class extreme_value_distribution
{
public:
```

```
  // types
  typedef RealType input_type;
  typedef RealType result_type;

  // constructor and member functions
  explicit extreme_value_distribution(const RealType& a = 0,
                                      const RealType& b = 1);
  RealType a() const;
  RealType b() const;
  void reset();
  template <class UniformRandomNumberGenerator>
    result_type operator()(UniformRandomNumberGenerator& urng);
};


  extreme_value_distribution(const RealType& a = 0,
                             const RealType& b = 1)
```

2 *Requires:*

`b` > 0.

3 *Effects:* Constructs an `extreme_value_distribution` object; `a` and `b` are the parameters of the distribution.

```
  RealType a() const;
```

4 *Returns:* The `a` parameter of the distribution.

```
  RealType b() const;
```

5 *Returns:* The `b` parameter of the distribution.

5.1.7.13 Class template `lognormal_distribution`

1 A `lognormal_distribution` random distribution produces random numbers x distributed with probability density function

$$p(x) = \frac{1}{sx\sqrt{2\pi}} \exp\left(-\frac{(\ln x - m)^2}{2s^2}\right),$$

where m and s are the parameters of the distribution.

```
template<class RealType = double>
class lognormal_distribution
{
public:
  // types
  typedef RealType input_type;
  typedef RealType result_type;
```

```
  // constructor and member functions
  explicit lognormal_distribution(const RealType& m = 0 ,
                                  const RealType& s = 1);

  RealType m() const;
  RealType s() const;
  void reset();
  template <class UniformRandomNumberGenerator>
    result_type operator()(UniformRandomNumberGenerator& urng);
};
```

```
  lognormal_distribution(const RealType& m = 0, const RealType& s = 1);
```

2 *Requires:* `s` > 0.

3 *Effects:* Constructs a `lognormal_distribution` object; `m` and `s` are the parameters of the distribution.

```
  RealType m() const;
```

4 *Returns:* The `m` parameter of the distribution.

```
  RealType s() const;
```

5 *Returns:* The `s` parameter of the distribution.

5.1.7.14 Class template `chi_squared_distribution`

1 A `chi_squared_distribution` random distribution produces random numbers $x > 0$ distributed with probability density function

$$p(x) = \frac{x^{(n/2)-1}\exp(-x/2)}{\Gamma(n/2)2^{n/2}} ,$$

where n is the parameter of the distribution.

```
template<class IntType = int, class RealType = double>
class chi_squared_distribution
{
public:
  // types
  typedef RealType input_type;
  typedef RealType result_type;

  // constructor and member functions
  explicit chi_squared_distribution(IntType n = 1);
```

```
  IntType n() const;
  void reset();
  template <class UniformRandomNumberGenerator>
    result_type operator()(UniformRandomNumberGenerator& urng);
};

  chi_squared_distribution(IntType n = 1);
```

2 *Requires:* `n` > 0.

3 *Effects:* Constructs a `chi_squared_distribution` object; `n` is the parameter of the distribution.

```
  IntType n() const;
```

4 *Returns:* The `n` parameter of the distribution.

5.1.7.15 Class template `cauchy_distribution`

1 A `cauchy_distribution` random distribution produces random numbers x distributed with probability density function

$$p(x) = \left(\pi b \left(1 + \left(\frac{x-a}{b}\right)^2\right)\right)^{-1} ,$$

where a and b are the parameters of the distribution.

```
template<class RealType = double>
class cauchy_distribution
{
public:
  // types
  typedef RealType input_type;
  typedef RealType result_type;

  // constructor and member functions
  explicit cauchy_distribution(const RealType& a = 0 ,
                               const RealType& b = 1);

  RealType a() const;
  RealType b() const;
  void reset();
  template <class UniformRandomNumberGenerator>
    result_type operator()(UniformRandomNumberGenerator& urng);
};

  cauchy_distribution(const RealType& a = 0, const RealType& b = 1);
```

2 *Requires:* `b` > 0.

3 *Effects:* Constructs a `cauchy_distribution` object; `a` and `b` are the parameters of the distribution.

```
RealType a() const;
```

4 *Returns:* The `a` parameter of the distribution.

```
RealType b() const;
```

5 *Returns:* The `b` parameter of the distribution.

5.1.7.16 Class template `fisher_f_distribution`

1 A `fisher_f_distribution` random distribution produces random numbers $x \geq 0$ distributed with probability density function

$$p(x) = \frac{\Gamma\big((m+n)/2\big)}{\Gamma(m/2)\Gamma(n/2)} \left(\frac{m}{2}\right)^{m/2} x^{(m/2)-1} \Big(1 + \frac{mx}{n}\Big)^{(m+n)/2},$$

where m and n are the parameters of the distribution.

```
template<class IntType = int, class RealType = double>
class fisher_f_distribution
{
public:
  // types
  typedef RealType input_type;
  typedef RealType result_type;

  // constructor and member functions
  explicit fisher_f_distribution(IntType m = 1, IntType n = 1);

  IntType m() const;
  IntType n() const;

  void reset();
  template <class UniformRandomNumberGenerator>
    result_type operator()(UniformRandomNumberGenerator& urng);
};
```

```
  fisher_f_distribution(IntType m = 1, IntType n = 1);
```

2 *Requires:* `m` > 0 and `n` > 0.

3 *Effects:* Constructs a `fisher_f_distribution` object; `m` and `n` are the parameters of the distribution.

```
  IntType m() const;
```

4 *Returns:* The `m` parameter of the distribution.

```
IntType n() const;
```

5 *Returns:* The `n` parameter of the distribution.

5.1.7.17 Class template `student_t_distribution`

1 A `student_t_distribution` random distribution produces random numbers x distributed with probability density function

$$p(x) = \frac{1}{\sqrt{n\pi}} \frac{\Gamma\big((n+1)/2\big)}{\Gamma(n/2)} \left(1 + \frac{x^2}{n}\right)^{-(n+1)/2} ,$$

where n is the parameter of the distribution.

```
template<class IntType = int, class RealType = double>
class student_t_distribution
{
public:
  // types
  typedef RealType input_type;
  typedef RealType result_type;

  // constructor and member functions
  explicit student_t_distribution(IntType n = 1);

  IntType n() const;
  void reset();
  template <class UniformRandomNumberGenerator>
    result_type operator()(UniformRandomNumberGenerator& urng);
};
```

```
  student_t_distribution(IntType n = 1);
```

2 *Requires:* `n` > 0.

3 *Effects:* Constructs a `student_t_distribution` object; `n` is the parameter of the distribution.

```
  IntType n() const;
```

4 *Returns:* The `n` parameter of the distribution.

5.1.7.18 Class template `discrete_distribution`

1 A `discrete_distribution` random distribution produces random integers $0 \le i < n$ distributed with probability

$$P(i) = w_i \ ,$$

where the n weights w_i are specified via the parameters of the distribution.

```
template<class IntType = int>
class discrete_distribution
{
public:
  // types
  typedef double input_type;
  typedef IntType result_type;

  // constructor and member functions
  template <class InputIterator>
    discrete_distribution(InputIterator first, InputIterator last);
  discrete_distribution();
  void reset();
  template <class UniformRandomNumberGenerator>
    result_type operator()(UniformRandomNumberGenerator& urng);
  result_type min() const;
  result_type max() const;
};
```

```
  template <class InputIterator>
    discrete_distribution(InputIterator first, InputIterator last);
```

2 *Requires:*

a. `first` shall meet the requirements of input iterator, as shall `last`;
b. `[first, last)` shall form a sequence of length $n > 0$;
c. `*first` shall return a value convertible to `double`;
d. `*i` ≥ 0 for all iterators `i` in the sequence `[first, last)`; and
e. `*i` > 0 for at least one iterator `i` in the sequence `[first, last)`.

3 *Effects:* Constructs a `discrete_distribution` object that will deliver discrete `result_type` values $0 \le$ `i` $<$ `n`. The values are distributed according to the n weights

$$w_k = \frac{*\mathtt{i}_k}{\sum_{j=0}^{n-1} *\mathtt{i}_j} \ .$$

```
  discrete_distribution();
```

4 *Effects:* Constructs a `discrete_distribution` object that will always deliver the value zero.

```
  result_type min() const;
```

5 *Returns:* zero.

```
  result_type max() const;
```

6 *Returns:* $\mathtt{n} - 1$.

5.1.7.19 Class template `piecewise_constant_distribution`

1 A `piecewise_constant_distribution` random distribution produces random numbers $x_0 \leq x < x_n$ distributed according to a piecewise constant probability density function

$$p(x) = \begin{cases} 0 & \text{if} \quad x < x_0 \\ \rho_i & \text{if} \quad x_i \leq x < x_{i+1},\ i = 0 \ldots n-1 \\ 0 & \text{if} \quad x \geq x_n \end{cases} ,$$

where the densities ρ_i and their corresponding interval boundaries x_i are specified via the parameters of the distribution.

```
template<class RealType = double>
class piecewise_constant_distribution
{
public:
  // types
  typedef double input_type;
  typedef RealType result_type;

  // constructor and member functions
  template <class InputIteratorX, InputIteratorW>
    piecewise_constant_distribution
      (  InputIteratorX first, InputIteratorX last, InputIteratorW weights);
  piecewise_constant_distribution();
  void reset();
  template <class UniformRandomNumberGenerator>
    result_type operator()(UniformRandomNumberGenerator& urng);
  result_type min() const;
  result_type max() const;
};


  template <class InputIteratorX, InputIteratorW>
  piecewise_constant_distribution
    (InputIteratorX first, InputIteratorX last, InputIteratorW weights);
```

2 *Requires:*

a. `first` shall meet the requirements of input iterator, as shall `last` and `weights`;
b. `[first, last)` shall form a sequence x of length $n+1$ where $n > 0$, with $x_0 = *\mathtt{first}$;
c. the sequence `[first, last)` shall be non-decreasing;
d. `*first` shall return a value convertible to `result_type`;
e. `*weights` shall return a value convertible to `double`;
f. the length of the sequence w starting from `weights` shall be at least n, with $w_0 = *\mathtt{weights}$, and any w_k for $k \geq n$ shall be ignored by the distribution;
g. $w_k \geq 0$ for $0 \leq k < n$; and
h. $w_k > 0$ for at least one k where $0 \leq k < n$.

3 *Effects:* Constructs a `piecewise_constant_distribution` object that will deliver `result_type` values $x_0 \le x < x_n$. These values are distributed such that the probability

$$P(x_k \le x < x_{k+1}) = \frac{w_k}{\sum_{j=0}^{n-1} w_j} \ .$$

Further, the values x are distributed uniformly within each subinterval $x_k \le x < x_{k+1}$. [*Note:* This implies that for subintervals $x_k \le x < x_{k+1}$ of non-zero width $\ell_k = x_{k+1} - x_k$, the density value ρ_k is proportional to w_k/ℓ_k. If $x_k = x_{k+1}$ the probability of delivering x_k is $w_k / \sum_j w_j$. —*end note*]

```
result_type min() const;
```

4 *Returns:* x_0.

```
result_type max() const;
```

5 *Returns:* x_n.

5.1.7.20 Class template `piecewise_smooth_distribution`

1 A `piecewise_smooth_distribution` random distribution produces random numbers $x_0 \le x < x_n$ distributed with probability

$$\begin{array}{lcll} P(x < x_0) & = & 0 & \\ P(x_k \le x < x_{k+1}) & = & w_k & (0 \le k < n) \\ P(x \ge x_n) & = & 0 & \end{array} \ .$$

The interval boundaries x_k, $0 \le k \le n$, and the bin probabilities w_k, $0 \le k < n$, are the parameters of the distribution. [*Note:* These parameters are calculated as described below. —*end note*]

```
template<class RealType = double>
class piecewise_smooth_distribution
{
public:
  // types
  typedef double input_type;
  typedef RealType result_type;

  // constructor and member functions
  template <class InputIteratorX, InputIteratorW>
    piecewise_smooth_distribution
      (InputIteratorX first, InputIteratorX last, InputIteratorW weights);
  template <class InputIteratorX, class Func>
    piecewise_smooth_distribution
     (InputIteratorX first, InputIteratorX last, Func pdf);
  piecewise_smooth_distribution();
  void reset();
  template <class UniformRandomNumberGenerator>
    result_type operator()(UniformRandomNumberGenerator& urng);
  result_type min() const;
  result_type max() const;
};
```

```
template <class InputIteratorX, InputIteratorW>
piecewise_smooth_distribution
  (InputIteratorX first, InputIteratorX last, InputIteratorW weights);
```

2 *Requires:*

a. `first` shall meet the requirements of input iterator, as shall `last` and `weights`;
b. `[first, last)` shall form a sequence x of length $n+1$ where $n > 0$, with $x_0 =$ `*first`;
c. the sequence `[first, last)` shall be non-decreasing;
d. `*first` shall return a value convertible to `result_type`;
e. `*weights` shall return a value convertible to `double`;
f. the length of the sequence w starting from `weights` shall be at least n, with $w_0 =$ `*weights`, and any w_k for $k \geq n$ shall be ignored by the distribution;
g. $w_k \geq 0$ for $0 \leq k < n$; and
h. $w_k > 0$ for at least one k where $0 \leq k < n$.

3 *Effects:* Constructs a `piecewise_smooth_distribution` object that will deliver values $x_0 \leq x < x_n$. The values are distributed such that the probability

$$P(x_k \leq x < x_{k+1}) = \frac{w_k}{\sum_{j=0}^{n-1} w_j}$$

unless $x_k = x = x_{k+1}$ in which case the probability of delivering precisely x is

$$P(x = x_k) = \frac{w_k}{\sum_{j=0}^{n-1} w_j} \ .$$

```
template <class InputIteratorX, class Func>
piecewise_smooth_distribution
  (InputIteratorX first, InputIteratorX last, Func pdf);
```

4 *Requires:*

a. `first` shall meet the requirements of input iterator, as shall `last`;
b. `[first, last)` shall form a sequence x of length $n+1$ where $n > 0$, with $x_0 =$ `*first`;
c. the sequence `[first, last)` shall be non-decreasing;
d. `*first` shall return a value convertible to `result_type`;
e. `pdf` shall be callable with one argument of type `double`, and shall return a type convertible to `double`;
f. `pdf(`x`)` ≥ 0 for all $x_0 \leq x < x_n$; and
g. `pdf(`x_k`)` > 0 for at least one k, $0 \leq k \leq n$.

5 *Effects:* Constructs a `piecewise_smooth_distribution` object is if it had been constructed by `piecewise_smooth_distribution(first, last, weights)`, where `weights` is the beginning iterator of a sequence w of length n such that

$$w_k = \int_{x_k}^{x_{k+1}} \mathrm{pdf}(x)\, dx \ .$$

[*Note:* The resulting distribution has a probability distribution function that is smooth on $(x_0 \leq x < x_n)$. If `pdf`(x) is a smooth function, this distribution can be used as an excellent approximation to the distribution defined by normalizing `pdf`. —*end note*]

```
result_type min() const;
```

6 *Returns:* x_0.

```
result_type max() const;
```

7 *Returns:* x_n.

Bibliography

[Aus05] Matt Austern. (Draft) technical report on standard library extensions. Paper N1836, ISO/IEC SC22/JTC1/WG21, June 24 2005. Online: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2005/n1836.pdf; same as ANSI NCITS/J16 05-0096.

[Mau03] Jens Maurer. A proposal to add an extensible random number facility to the standard library (revision 2). Paper N1452, ISO/IEC SC22/JTC1/WG21, April 10 2003. Online: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2003/n1452.html; same as ANSI NCITS/J16 03-0035.

[Pat04] Marc Paterno. On random-rumber distributions for C++0x. Paper N1588, ISO/IEC SC22/JTC1/WG21, February 13 2004. Online: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2004/n1588.html; same as ANSI NCITS/J16 04-0028.