Quant Finance Code Library

Random C++ Library

Authors:
Thijs van den Berg

Contributors:
Thijs van den Berg
James Hirschorn

March 13, 2012

Contents

Chapter 1

Overview

The QFCL Random C++ library covers routines used for generating samples from various probability distribution.

1.1 Engines

Engines generate random numbers with certain properties. The C++ standard library, the BOOST libraries provide various random number engines. The engines in QFCL adhere to the same interfaces and are thus interoperable with C++ and BOOST random related routines.

The Engines provided in QFCL have two major benefits:

- They are thread safe.
- These engines can generate parallel independent random streams.

Both features are requires for efficient parallel random number generation and in parallel Monte Carlo engines.

1.1.1 Interface

Engines provide the following interface

- operator() for generating random numbers.
- max() that returns the higher random number that can be generated.
- min() that returns the lowest random number that can be generated.

1.2 Distribution

todo

1.3 Generating variates

todo

```
typedef qfcl::random::cpp_rand ENG;
typedef qfcl::random::gbm_npv_vanilla_call DIST;
typedef qfcl::random::variate_generator< ENG, DIST > RNG;

double S0 = 100;
double vol = 0.25;
double yield = 0.05;
double r = 0.05;
double strike = 110;
double t = 1.5;

DIST call( S0, vol, yield, r, strike, t );
ENG eng;
RNG rng(eng,call);

for (int i=0;i<1000; ++i)
    std::cout << rng() << "\n";
```

Listing 1.1: Creating Vanilla call price sampler

Chapter 2

Engines

2.1 Mersenne Twister

todo

2.2 Counter based Philox

todo

Chapter 3

Distributions

3.1 Uniform Distributions

Description
The distribution uniform_0ex_1ex_distribution and it's variants are used to generate samples from uniform distributions. The ex and in keyword indicate exclusion of inclusion of 0 and/or 1 in the generated samples.

distribution	range
uniform_0ex_1ex	$(0, 1)$
uniform_0in_1ex	$[0, 1)$
uniform_0ex_1in	$(0, 1]$
uniform_0in_1in	$[0, 1]$

3.1.1 uniform_0ex_1ex

Synopsis

```
// Include files
#include <qfcl/random/distribution/uniform_0ex_1ex.hpp>

// Types
template<class RealType = double>  class uniform_0ex_1ex_distribution;

// convenience typedef
typedef uniform_0ex_1ex_distribution <double> uniform_0ex_1ex;
```

3.1.2 uniform_0ex_1in

Synopsis

```
// Include files
#include <qfcl/random/distribution/uniform_0ex_1in.hpp>

// Types
template<class RealType = double>  class uniform_0ex_1in_distribution;

// convenience typedef
typedef uniform_0ex_1in_distribution <double> uniform_0ex_1in;
```

3.1.3 uniform_0in_1ex

Synopsis

```
// Include files
#include <qfcl/random/distribution/uniform_0in_1ex.hpp>

// Types
template<class RealType = double>  class uniform_0in_1ex_distribution;

// convenience typedef
typedef uniform_0in_1ex_distribution <double> uniform_0in_1ex;
```

3.1.4 uniform_0in_1in

Synopsis

```
// Include files
#include <qfcl/random/distribution/uniform_0in_1in.hpp>

// Types
template<class RealType = double>  class uniform_0in_1in_distribution;

// convenience typedef
typedef uniform_0in_1in_distribution <double> uniform_0in_1in;
```

Example

```
1  #include <qfcl/random/engine/mersene_twister.hpp>
2  #include <qfcl/random/distribution/uniform_0in_1ex.hpp>
3  #include <qfcl/random/variate_generator.hpp>
4
5  int main()
6  {
7      typedef qfcl::random::cpp_rand ENG;
8      typedef qfcl::random::uniform_0in_1ex DIST;
9      typedef qfcl::random::variate_generator< ENG, DIST > RNG;
10
```

```
11      DIST u;
12      ENG eng;
13      RNG rng(eng,call);
14
15      for (int i=0;i<1000; ++i)
16          std::cout << rng() << "\n";
17      return 0;
18  }
```

Listing 3.1: Sample C++ code: function interface

3.2 Normal Distributions

Description

The distributions normal_xxx_distribution are used to generate samples from standard normal distributions. The xxx keyword indicate which method it uses to generate normal samples.

distribution	method
normal_box_muller	Box Muller transform.
normal_box_muller_polar	Box Muller transform, polar variant.
normal_inversion	Using the inverse cumulative distribution function.

3.2.1 normal_box_muller

Synopsis

```
// Include files
#include <qfcl/random/distribution/normal_box_muller.hpp>

// Types
template<class RealType = double> class normal_box_muller_distribution;

// convenience typedef
typedef normal_box_muller_distribution<double> normal_box_muller;
```

3.2.2 normal_box_muller_polar

Synopsis

```
// Include files
#include <qfcl/random/distribution/normal_box_muller_polar.hpp>

// Types
template<class RealType = double> class normal_box_muller_polar_distribution;

// convenience typedef
typedef normal_box_muller__polar_distribution<double> normal_box_muller_polar;
```

3.2.3 normal_inversion

Synopsis

```
// Include files
#include <qfcl/random/distribution/normal_inversion.hpp>

// Types
template<class RealType = double> class normal_inversion_distribution;

// convenience typedef
typedef normal_inversion_distribution<double> normal_inversion;
```

Example

```
1  #include <qfcl/random/engine/mersene_twister.hpp>
2  #include <qfcl/random/distribution/normal_inversion.hpp>
3  #include <qfcl/random/variate_generator.hpp>
4
5  int main()
6  {
7      typedef qfcl::random::cpp_rand ENG;
8      typedef qfcl::random::normal_inversion DIST;
9      typedef qfcl::random::variate_generator< ENG, DIST > RNG;
10
11     DIST n;
12     ENG eng;
13     RNG rng(eng,n);
14
15     for (int i=0;i<1000; ++i)
16         std::cout << rng() << "\n";
17     return 0;
18 }
```

Listing 3.2: Sample C++ code: function interface

3.3 Geometric Brownian Motion type Distributions

3.3.1 gbm_at_fixed_time

Description

Sample from a fixed time distribution of geometric Brownian motion.

todo

3.3.2 gbm_interval_high

todo

3.3.3 gbm_interval_low

todo

3.3.4 gbm_first_hitting_time

todo

3.3.5 gbm_npv_vanilla_call

Synopsis

```
// Include file
#include <qfcl/random/distribution/gbm_npv_vanilla_call.hpp>

// Types
template<class RealType = double>
class gbm_vanilla_call_distribution;

// convenience typedef
typedef gbm_vanilla_call_distribution<double> gbm_vanilla_call;
```

Description

The distribution gbm_npv_vanilla_call_distribution is used to generate samples from a vanilla call option price. The underlying process is assumed to be a geometric Brownian Motion. The prices are npv-ed (net present valued).

Example

```
1  #include <qfcl/random/engine/mersene_twister.hpp>
2  #include <qfcl/random/distribution/gbm_npv_vanilla_call.hpp>
3  #include <qfcl/random/variate_generator.hpp>
4
5  int main()
6  {
7      typedef qfcl::random::cpp_rand ENG;
8      typedef qfcl::random::gbm_npv_vanilla_call DIST;
9      typedef qfcl::random::variate_generator< ENG, DIST > RNG;
10
11     double S0 = 100;
12     double vol = 0.25;
13     double yield = 0.05;
14     double r = 0.05;
15     double strike = 110;
16     double t = 1.5;
17
18     DIST call( S0, vol, yield, r, strike, t );
19     ENG eng;
20     RNG rng(eng,call);
21
```

```
22      for (int i=0;i<1000; ++i)
23          std::cout << rng() << "\n";
24      return 0;
25  }
```

Listing 3.3: Sample C++ code: function interface

3.3.6 gbm_npv_vanilla_put

todo

Chapter 4

Tutorial

A tutorial that introduces the fundamental concepts required to use Qfcl.Random, and shows how to use Qfcl.Random to develop simple programs.

4.1 Headers and Namespaces

All the code in this library is inside namespace qfcl::random. You can to bring distribution names into scope perhaps with a using namespace qfcl::random; declaration, but it's recommended that you use declarations like using qfcl::random::normal_inversion;

In order to generate samples from a distribution some_special_distribution you will need to include either the header <qfcl/random/distributions/some_special.hpp> or the "include everything" header: <qfcl/random/distributions.hpp>.

```
#include <qfcl/random/distributions/some_special.hpp>
```

Listing 4.1: Including a "single distribution" header

4.2 Distribution names and aliases

All distribution are defined as templates with the real type as argument and have names ending with "_distribution". E.g. the normal_inversion_distribution, is an distribution sampler that uses the inversion method to generate normal variates.

```
// include file
#include <qfcl/random/distributions/normal_inversion.hpp>

// declaring a distribution
qfcl::random::normal_inversion_distribution<float> my_dist;
```

The real type of distribution defaults to double, and all distributions have a typedef omitting the _distribution in the name for the double real type. The following declarations are all identical

```
// verbose version
qfcl::random::normal_inversion_distribution<double> my_dist;

// using the default template type (double)
qfcl::random::normal_inversion_distribution<> my_dist;

// using the convenience typdef
qfcl::random::normal_inversion my_dist;
```