PCSC Project 9 report: Monte Carlo and non-linear operators

Gavin Lee and Huajian Qiu

December 13, 2018

This project deals with the statistical study of non-linear operators using Monte Carlo methods. At first, (pseudo) random number generators (RNGs) are implemented for a range of probability distributions. These RNGs are used to numerically calculate the expected value of a user defined, possibly non-linear function of a random variable. Statistical moments of distributions are also visualised. Finally, the central limit theorem is visually verified.

Prerequisites and how to compile the program

The visualisation functionality of this program requires Gnuplot and Boost libraries. To get these libraries on Ubuntu, use

```
sudo apt-get install gnuplot-x11
sudo apt-get install libboost-dev
sudo apt-get install libboost-system-dev
sudo apt-get install libboost-filesystem-dev
sudo apt-get install libboost-iostreams-dev
```

at the command line. The *CMakeLists.txt* specifies linking these libraries.

```
set(Boost_USE_STATIC_LIBS         ON)
set(Boost_USE_MULTITHREADED       ON)
set(Boost_USE_STATIC_RUNTIME     OFF)
find_package(Boost 1.67.0 COMPONENTS filesystem system iostreams)
if(Boost_FOUND)
include_directories(${Boost_INCLUDE_DIRS})
add_executable(main main.cpp)
target_link_libraries(monte-carlo ${Boost_LIBRARIES})
endif()
```

The program was built in *CLion 2018.2.3* and is expected to be used in this IDE. Google tests are also included in this project and can be installed using `sudo apt-get install libgtest-dev`. Class documentation can be run using the command `doxygen Doxyfile`.

Typical program execution and usage

In this project, objects underlying nearly all the functionality relies on specifying a probability distribution. At the present, the Uniform, Normal, Exponential and Geometric distributions are implemented. They can be declared with or without parameters (defaults used in the latter case). For example

```
Uniform my_uniform_distribution(10.0, 20.0);
Normal my_normal_distribution();
```

Generating (one or more) random variables from these distributions is called as follows.

```
double uniform_rv = my_uniform_distribution.generate();
std::vector<double> normal_vector(my_normal_distribution.generate(5));
```

These random numbers are used principally for two purposes. The first is to calculate Monte Carlo expectations of user defined functions of random variables. At the present these functions may only be of one variable. For example

```
double my_func(double x) { return x + pow(x1,2) - 1./x; }
Exponential my_exponential(5.5)
std::vector<double> my_exps(my_exponential.generate(100))
Expectation my_monte_carlo_expectation(&my_func);
double my_monte_carlo_expectation.calculate_sample_mean(my_exps));
```

More involved calculations of statistical moments may also be performed. Their behaviour with respect to the number of samples drawn from each distribution can also be visualised. For example

```
int order = 3;
bool centred = false;
Moments my_moment(order, centred, &my_func, &my_exponential)
std::vector<unsigned int> my_Ns{10,20,100,200};
my_moment.visualise_monte_carlo(my_Ns);
```

The second purpose is to verify the central limit theorem with these random numbers with the number of samples per trial and the number of trials specifying the numerical experiment.

```
int num_samples = 1000;
int num_trials = 200;
bool centred = false;
VerifyCLT my_verification(num_samples, num_trials, &my_exponential)
my_verification.visualise_CLT();
```

List of features

1. Generation of Uniform, Normal, Exponential and Geometric random variables. Default parameters are
   - `Uniform[a,b], a = 0, b = 1`
   - `Normal(mu,sigma), mu = 0, sigma = 1`
   - `Exponential(lambda), lambda = 1`
   - `Geometric(p), p = 0.5`
2. Expectation of user defined functions of a given input variables.
3. Calculation and visualisation of moments of a distribution with a given order and underlying distribution and user defined function in terms of its Monte Carlo estimation.
4. Visual verification of the central limit theorem by comparison to the standard normal density function or probability plot.

List of tests

Distributions

- When initialising distributions with parameters, there are restrictions on their size or order.
  - `Uniform my_uniform(a,b): ASSERT_THROW if a<b`
  - `Normal my_normal(mu,sigma): ASSERT_THROW if sigma <= 0`
  - `Exponential my_exponential(lambda): ASSERT_THROW if lambda <= 0`
  - `Geometric my_geometric(p): ASSERT_THROW if p<=0 or p>=1`
- The random number generators should produce numbers in a specified range so there are restrictions for certain distributions
  - `Uniform(a,b): ASSERT_LE(a,min(my_uniform.generate(100))`
  - `Uniform(a,b): ASSERT_GE(b,max(my_uniform.generate(100))`
  - `Exponential(lambda): ASSERT_GE(0,min(my_exponential(lambda).generate(100))`
  - `Geometric(p): ASSERT_GET(0,min(my_geometric(p).generate(100))`

  A central component of implementing random number generators is testing whether they behave as expected. That is, whether they follow a distribution as intended. As to avoid re-inventing the wheel, we do not test the underlying Mersenne Twister algorithm. For the purpose of testing our implementation of this algorithm, we use χ^2 goodness-of-fit tests for each of the implemented distributions. As a default we generate 1000 random numbers from these distributions and test their goodness of fit to a χ^2 distribution with a 95% level of significance.
- χ^2 goodness-of-fit-tests for a default number of bins (depending on the distribution)
  - `boost::math::chi_squared my_chi2(degrees_of_freedom)`
  - `Uniform(a,b): EXPECT_LE(calculated_chisq_statistic, quantile(my_chi2,0.95)`
  - `Normal(mu,sigma): EXPECT_LE(calculated_chisq_statistic, quantile(my_chi2,0.95)`
  - `Exponential(lambda): EXPECT_LE(calculated_chisq_statistic, quantile(my_chi2,0.95)`
  - `Geometric(p): EXPECT_LE(calculated_chisq_statistic, quantile(my_chi2,0.95)`

Expectations

Sample means should lie within the minimum and maximum of the input vector. The variance of any random sample should be positive.

- `ASSERT_LE(min(random_vector), Expectation.calculate_sample_mean(random_vector))`
- `ASSERT_GE(max(random_vector), Expectation.calculate_sample_mean(random_vector)`
- `ASSERT_GT(Expectation.calculate_sample_variance(random_vector), 0)`

In the following two classes, we assert simple restrictions on the sample size, trial size and order when initialised or set by the user.

Moments

- `ASSERT_THROW if Moments.order == 0`
- `ASSERT_THROW if Moments.calculate(n == 0)`
- `ASSERT_THROW Moments.calculate(std::vector<double>.empty() == true)`
- `ASSERT_THROW Moments.visualise_monte_carlo(std::vector<unsigned int> my_n_values)` with `my_n_values containing 0`

Verify Central Limit Theorem

- `ASSERT_THROW VerifyCLT(n == 0, N == 0)`
- `ASSERT_THROW VerifyCLT.set_num_samples(n == 0)`
- `ASSERT_THROW VerifyCLT.set_num_trials(N == 0)`

TODOs and perspectives

1. The Distribution class is currently only written for an output type of `double`. In fact, the Geometric distribution, which outputs integers, are stored in doubles. This class could be templetised to take a type parameter in order to specify `double` or `int` as the return type.
2. More distributions should be implemented. A notable, illustrative example of where convergence of Monte Carlo expectations and the central limit theorem may fail is the Cauchy distribution, which does not have first or higher moments.
3. The user defined function currently only takes one variable which is limiting. A logical next step is to allow an input of n variables, all with the same distribution, and then further adding the possibility of specifying the distribution of each of those random variables.
4. The central limit theorem (CLT) can be extended to (possibly non-linear) functions of random variables. This would involve specifying a function and its derivative. The classical CLT we have used is that if $X_1, \dots, X_n \overset{\text{i.i.d.}}{\sim} X$ are random variables with finite variance and $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$, then
$$\frac{\sqrt{n}}{\sigma}\left(\bar{X} - \mu\right) \xrightarrow{d} N(0,1)$$
where μ and σ are the mean and standard deviation of X respectively. The 'Delta method' for a possibly non-linear function states that if the classical CLT holds, then
$$\frac{\sqrt{n}}{\sigma}\left(g(\bar{X}) - g(\mu)\right) \xrightarrow{d} N(0, g'(\mu)^2)$$
if $g'(\mu)$ is non-zero.
5. The Goodness of Fit testing should be put into a new class or fixture. It is currently just written for a default 1000 random variables and a 95% level of significance. Note that the 'quality' or 'randomness' of the random variables we generate has not been rigorously tested, though this is possibly outside the scope of this project. The χ^2 test implemented will fail on average 5% of the time. Other tests on the distribution of the generated numbers should be considered, for example the Kolmogorov-Smirnov or Anderson-Darling tests.