Documentation for Data Structures Project

Niko Ilomäki[1, a)]
University of Helsinki

(Dated: January 4, 2015)

I. DESCRIPTION

Subject of the project is Mersenne Twister (MT), the most commonly used modern pseudorandom number generator (prng; `http://en.wikipedia.org/wiki/Mersenne_twister`). Implementation uses Python 3. Implementation includes a number of variants of MT, including at least MT19937 and MT19937-64 (see link above). One or more older prng's are used as reference points.

Implementation includes functions for generating numbers from the most common probability distributions, i.e. uniform, normal, exponential, binomial and beta distributions etc. Some visualization tools for the distributions will be provided. A number of randomness tests are also part of the project. These may include some of the so-called diehard tests (`http://en.wikipedia.org/wiki/Diehard_tests`).

MT only uses basic arrays, so no special data structures will be implemented as part of the project. Since MT is a prng, its only parameters are the size of the random number array and the used seed value. In this project, different distributions are provided as well in addition to uniform. Time usage of MT is $\mathbb{O}(n)$ where n is the size of the requested random number array.

II. WEEK 1

I'll properly start the project next week; therefore this week didn't provide much new. Nothing seems unreasonably unclear at the moment and in case problems arrive I will turn to IRC first. The program itself hasn't progressed at all yet; I've researched the algorithms, but haven't coded anything. However, by next Sunday I'm aiming for a preliminary complete program. Next thing I'm going to do is implementing MT19937 and some of the distribution transform functions with Python 3.

III. WEEK 2

Lately I've been studying both Mersenne twister implementation and different probability distributions. Generators MT19937 and MT19937-64 have now been implemented in the program. A few more will follow for comparison purposes, perhaps a lagged Fibonacci generator (simpler and less random) and the Blum Blum Shub generator (very slow, but a lot better for cryptographical purposes). Most of the planned probability distributions have also been implemented. Their implementation uses quantile functions (inverse functions of the respective cumulative distribution functions) on input numbers between 0 and 1 from the generators. This is a well-known technique in Monte Carlo simulation. Output visualization and testing are priorities for next week.

[a)]niko.ilomaki@helsinki.fi