Lab # 2 - Generation of Random Variates

(Implementation in C – check the lecture slides and read carefully the subject)

1) Find "**Matsumoto Home page**" and the "**last C implementation**" of the original Mersenne Twister (MT)

In the first Lab, you have implemented different techniques for the generation of random numbers. You have seen that mastering high quality random numbers is not so easy. In this lab, we will use one of the top generator proposed for Science in the 21st century (equidistributed in 623 dimensions and with a period of 2^{19937} numbers). Though not crypto secure, it will be the generator that you will use for this lab and the next ones.

You can find the current implementation in C of Mersenne Twister (MT) google to "Matsumoto Home Page", then find Mersenne Twister / 2002 version – explanation & C code. Download the .tar file with the source code + expected output and readme (take the 32 bits version).

Compile and test if you obtain the expected output (for portability & **repeatability**). In the lab questions use genrand_int32 or genrand_real(1/2) functions. Untar and unzip the archive (Unix command: "`tar zxvf yourfile.tgz`") and use the example. Compare the result you obtain locally on your computer session with the expected output (reproducibility – see the README file proposed by Matsumoto and the expected output). From now, always use a fine generator like MT or other very good generators.

Once you have tested the bitwise reproducibility of this code, you will test the next functions of this lab by adding your code before the main function of Makoto's code, and you will modify the test functions of Makoto's to test your lab functions.

2) Generation of uniform random numbers between A and B

Implementation : Using the MT function providing numbers between [0..1], propose a C function named "uniform" with 2 parameters 'a' and 'b' (real numbers) and generate pseudo-random numbers between 'a' and 'b'. Test this function for temperatures between -89,2°C and 56,7°C.

3) Reproduction of discrete empirical distributions

Suppose we have field data with 3 classes: 350 observations in class A, 450 in class B and 200 in class C., giving the following distribution probability of 3 species (A, B and C): 35% for A, 45% for B et 20% for C.

Reproduce (simulate) a population of individuals with the same distribution with MT.

a) **Implement and test** a program simulating this discrete distribution with the 3 classes A, B and C. Test it with 1 000, 10000, 100000 and 1 000 000 drawings. Cumulate the number of individual of each species in 3 variables and display the percentage obtained.

b) **Implement** a more generic function with the following input parameters: the size of an array of classes, then the array itself with the number of individuals observed in each class (see the lecture slides for an example – the HDL 'good' cholesterol and use these values to check this question).

a. First compute the corresponding array with the probability of being in each class (distribution function) and test this.
b. Then compute another array giving the cumulative probabilities. This function outputs the latter array. Test it.
c. Test the whole function with data given in the slides (and/or with your own data) and check a simulated distribution with 1000 and 1000 000 drawings.

4) Reproduction of continuous distributions

It is possible to reproduce continuous distributions by inverting the distribution function. When drawing a pseudo-random number between 0 and 1, it is possible to obtain a number distributed according to a given continuous distribution function (F) supposing that the latter is reversible.

$$x = F^{-1} \text{ (Random number drawn)}$$

This technique, named anamorphosis is not completely generic, but it can be applied to many distribution laws (Binomial, Weibull, Uniform,…). For instance, the distribution function of an exponential distribution (negative exponential law) is given in equation (8) leading to the inverse law of equation (9) which has to be implemented. We can see it as an analogue of the Poisson distribution. Actually, the time between two event in a Poisson process (intuitively: the time between two rare events) follows an exponential distribution. For instance, the time between two radioactive disintegrations.

$$F(x) = \int_0^x \frac{1}{M} e^{-\frac{1}{M}z} dz = 1 - e^{-\frac{1}{M}x} \qquad (8)$$

$$RandomNumberDrawn = 1 - e^{-\frac{1}{M}x}$$

$$\Rightarrow 1 - RandomNumberDrawn = e^{-\frac{1}{M}x}$$

$$\Rightarrow \ln(1 - RandomNumberDrawn) = -\frac{1}{M}x$$

$$\Rightarrow x = -M \ln(1 - RandomNumberDrawn) \qquad (9)$$

Uniform law between A and B : x = F^{-1} (Random number drawn) = A+(B-A) * Random number drawn

Mean = (B + A) / 2 Variance = 1/12 * $(B - A)^2$

Negative exponential law(Average) : x = F^{-1} (Random number drawn) = - Mean x Log (1 - Random number drawn)

Mean = M Variance = M

Figure 2. Inverse function of the uniform and negative exponential law

Here is what you have to code:

a. Implement the negExp function accepting the mean as a parameter.
b. Test this function with a mean of 11. Check that the average obtained after drawing 1000 (then 1000 000), it should come close to 11. This supposes using fine random numbers between 0 and 1

in equation (9) to obtain the correct distribution. Such numbers could for instance correspond to inter-arrival time between two jobs submitted to a computing cluster.

c. Check this discrete distribution (the biased dice). Use an array with 23 bins and test the frequency of numbers between 0 and 1, between 1 and 2,... Keep the last bin to cumulate the number of values above 22. For each number drawn, count in which bin it appears and cumulate this for all your drawings (1 000, 1 000 000)

```
Test22bins[ (int) negExp(11) ] ++;
```

With a Mean set to 11, you will produce many numbers between 0 and 1, a bit less between 1 and 2, etc. If you display an histogram, it can produce something that looks like figure 3 (with a different slope). You should also test that the observed mean is corresponding to the theoretical mean.

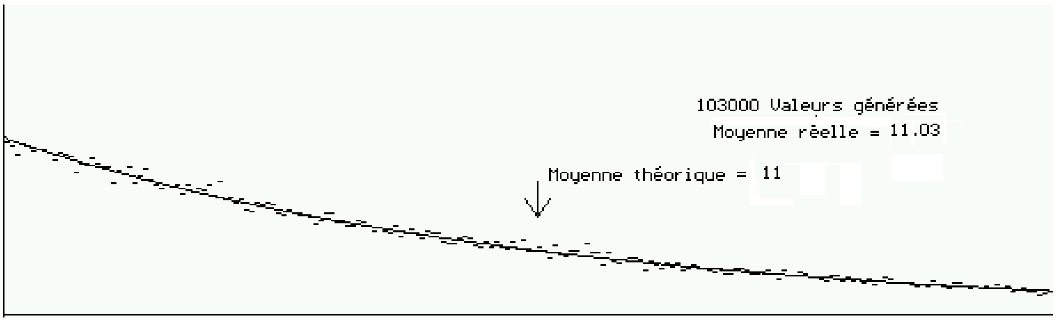

Figure 2. Simulation of the Inverse function of the uniform and negative exponential law.

5) Simulating non reversible distribution laws

In the case of non-reversible distribution laws, we can use the rejection technique, which is a Monte Carlo inspired technique. Below is a standard rejection algorithm for generating a number according to a probability distribution f(x) between 2 values MinX and MaxX (+ Min Y and MaxY which are the values providing a box around the probability distribution (density) function (PDF).

```
(1)  Generate 2 random numbers $Na_1$ and $Na_2$
(2)  Compute  X = MinX + $Na_1$ * (MaxX - MinX)
(3)  Compute Y = MaxY * $Na_2$
(4)  If Y is <= f(X)
     Then  X is considered as distributed according
           a law with f(x) as density function
     Else  reject X and goto (1) ie : draw again 2 pseudo-random numbers
           between 0 and 1, etc..
     EndIf
```

Figure 4. Generic rejection algorithm for any distributions

The Special case of the Gaussian distribution:

The density of a normal law (reduced and centred: average = 0, standard deviation = 1) is noted *N*(0,1) and given by equation (10) hereafter.

$$p(x) = \frac{1}{\sqrt{(2\pi)}} e^{-\frac{x^2}{2}} \qquad (10)$$

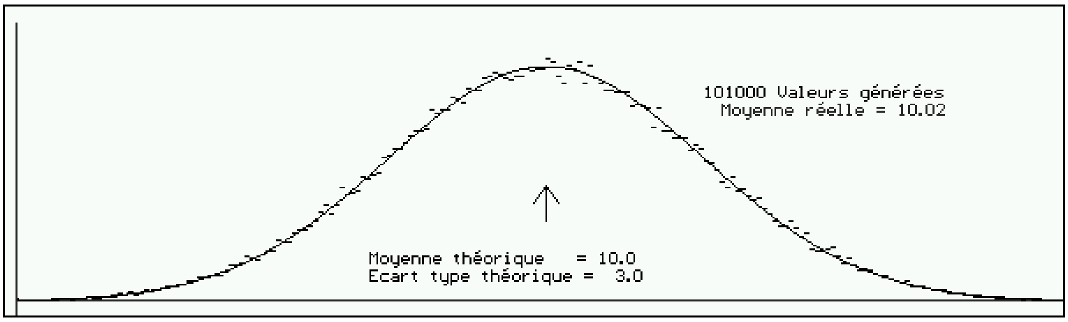

Figure 5. Generation of valued following a Gaussian distribution (average = 10, std. Dev. = 3)

5.1 First implementation:

Consider an experiment drawing 30 times a common dice. Sum the obtained results. The expected result is between 30 (as a minimum : 30 x face 1 and a potential maximum of 180 (30 x face 6) with a very low probability (1 / 6^{30})

Simulate this experiment 'many' times to obtain an approximation of the average (and of the standard deviation). You can then define statistical bins around the mean to see the (expected) bell curve (use 150 bins – an array for each possible sum and display the results with excel for instance).

5.2 Test of an analytical model of the Gaussian distribution

In 1958, Box and Muller presented an exact method without using the Central Limit theorem and using two pseudo random numbers. Equation (14) uses two random numbers Rn1 and Rn2 and produces two numbers distributed on both sides of the centred and reduced Gaussian law - *N*(0,1). Many variants exist to approximate a Gaussian distribution, some are faster, some more precise…

$$\begin{aligned} x_1 &= \cos(2\pi Rn_2)(-2\ln(Rn_1))^{\frac{1}{2}} \\ x_2 &= \sin(2\pi Rn_2)(-2\ln(Rn_1))^{\frac{1}{2}} \end{aligned} \quad (14)$$

Implementation: Test the Box and Muller functions to generate numbers around 0 following *N*(0,1). Two pseudo-random numbers give 2 numbers. Check for 1000 and 1000000 drawings how many numbers are distributed in 20 bins around -5 & 5 (between [-3..-2.5[, [-2.5, -2[,… [2..2.5[, [2.5…3[. Print your result and see if it fits with the known statistics for the Gaussian distribution?

6) Find libraries in C/C++ and Java that generate random variates like you did in the previous questions.