

Numerical Finance Reading Course

Sheet 1 - Sample Solution

Exercise 1: Linear Congruential Generators

There are many different implementations of linear congruential generators, i.e. generators of the form

$$y_{n+1} = (ay_n + b) \bmod M.$$

We want to compare the following two examples:

- a) RANDU: popular generator implemented by IBM in 1970.

$$a = 2^{16} + 3 = 65539, b = 0, y_0 \text{ odd and } M = 2^{31} = 2147483648.$$

- b) UNIX rand(): standard Unix random number generator.

$$a = 1103515245, b = 12345 \text{ and } M = 2^{31}.$$

Implement a linear congruential generator. For both examples, simulate 10000 uniformly distributed 3-dimensional pseudo-random vectors using for example $y_0 = 1$ and compare the performance of the generators by visualizing these samples in a 3D plot. Which generator would you prefer?

Solution:

```
#include <iostream>
#include <fstream>

using namespace std;

/*
 * Linear Congruential Generator
 */
class LCG {
private:
    // Parameters
    long long int a;
    long long int b;
    long long int M;

    // Internal state (i.e. last y_n)
    long long int state;

public:
```

```

// Constructor
LCG( long long int A, long long int B, long long int prime, int seed):
    a(A), b(B), M(prime), state((long long)seed) {};

// Generates the next pseudo random number ~ U[0,1]
double uni(){
    state = (a* state + b) % M;
    return (double)state / (double)M;
}

};

int main(){

    int dim = 3;
    int N = 10000;

    long long int M = 2147483648LL;
    long long int seed = 1;

    /*
     * RANDU
     */
    long long int a1 = 65539LL;
    long long int b1 = 0;
    LCG randu(a1, b1, M, seed);

    // Open text file
    ofstream randu_out("randu.txt");
    if(randu_out.is_open()){
        // For each pseudo random vector...
        for(int i = 1; i <= N; i++){
            // ... generate dim pseudo random numbers
            for(int j = 1; j <= dim; j++){
                randu_out << randu.uni() << "_";
            }
            randu_out << endl;
        }
        // Close text file
        randu_out.close();
    }
    else cout << "Unable to open file _randu.txt";

    /*
     * UNIX RAND
     */
    long long int a2 = 1103515245LL;
    long long int b2 = 12345LL;
    LCG unixrand(a2, b2, M, seed);

    ofstream unixrand_out("unixrand.txt");
    if(unixrand_out.is_open()){
        for(int i = 1; i <= N; i++){
            for(int j = 1; j <= dim; j++){
                unixrand_out << unixrand.uni() << "_";
            }
            unixrand_out << endl;
        }
        unixrand_out.close();
    }
    else cout << "Unable to open file _unixrand.txt";

    return 0;
}

```

Exercise 2: Multivariate Normal Distribution

- a) Let $X \in \mathbb{R}^n$ be a random vector with expectation vector $\mu_X \in \mathbb{R}^n$ and covariance matrix $\Sigma_X \in \mathbb{R}^{n \times n}$. Prove that for the expectation vector and the covariance matrix of a linear transformation $Z = AX \in \mathbb{R}^m$ with $A \in \mathbb{R}^{m \times n}$ it holds:

$$\begin{aligned}\mu_Z &= A\mu_X, \\ \Sigma_Z &= A\Sigma_X A^T.\end{aligned}$$

- b) Derive an algorithm for the generation of multivariate normal random vectors $Z \sim \mathcal{N}(\mu, C)$ ($Z, \mu \in \mathbb{R}^n, C \in \mathbb{R}^{n \times n}$). Recall that standard normally distributed pseudo-random numbers $X \sim \mathcal{N}(0, 1)$ can be obtained for example with the Box-Muller algorithm.

Solution:

- a) We have:

$$\mu_Z = \mathbb{E}[Z] = \mathbb{E}[AZ] = A\mathbb{E}[Z] = A\mu_X$$

and

$$\begin{aligned}\Sigma_Z &= \mathbb{E}[(Z - \mu_Z)(Z - \mu_Z)^T] \\ &= \mathbb{E}[A(X - \mu_X)(A(X - \mu_X))^T] \\ &= A\mathbb{E}[(X - \mu_X)(X - \mu_X)^T]A^T \\ &= A\Sigma_X A^T.\end{aligned}$$

- b) If $X = (X_1, \dots, X_n) \sim N(0, I)$, then $Z := \mu + AX \sim N(\mu, AA^T)$. Hence an algorithm to compute multivariate normal random vectors has the following form:
- 1) Compute $A \in \mathbb{R}^{n \times n}$ such that $AA^T = C$, e.g. with the Cholesky decomposition.
 - 2) Generate $X_1, \dots, X_n \sim N(0, 1)$ with the Box-Muller algorithm.
 - 3) Compute $Z := \mu + A \cdot (X_1, \dots, X_n)^T$.