UNIVERSITATEA DE STAT DIN MOLDOVA
FACULTATEA MATEMATICĂ ŞI INFORMATICĂ
DEPARTAMENTUL INFORMATICĂ

CEMÎRTAN CRISTIAN

Lucrare de laborator nr. 1

La disciplina **Probabilități și statistică**

Coordonator: Topală Oleg, dr., conf. univ.

Chişinău, 2022

CUPRINS

I. CERINȚA DE REALIZAT

86. Din mulțimea {0, 1, 2, …, N}, unde N este constanta 100, se extrage câte un număr cu întoarcere până când sunt extrase toate numerele pare. ξ – numărul de numere impare care nu sunt extrase nici o dată. A = {1 este scos cel puțin de două ori}. Să se calculeze P(A) și Mξ.

Conform suportului de laborator, metoda Monte Carlo se bazează pe calculul valorilor medii, deci avem formulele:

$$M\xi \approx \frac{\sum_{i=1}^{n} x_i}{n}$$

$$P(A) \approx \frac{|A|}{n}, unde\ A\ este\ mulțimea\ experimentelor\ efectuate$$

În cadrul programului, variabila n este nr. maxim de experimente, specificat de utilizator. Știind că sarcina nr. 86 are schema de tip „fără întoarcere”, și că putem extrage un număr cel mult infinit de ori, atunci putem zice că |Ω| este infinit. Ca consecință, cu cât mai mare este n, cu atât mai precis vor fi calculate P(A) și Mξ.

II. DESCRIEREA ALGORITMULUI

1. $N \leftarrow 100$;
2. $A \leftarrow 0$;
3. $\xi \leftarrow 0$;
4. n ← un număr introdus de utilizator de la tastatură, ce reprezintă nr. de experimente;
5. set ← un set, inițial gol, ce va conține experimentele non-duplicate;
6. Cât timp se satisface: $|set| \neq n$, **notăm bucla (1)**:
    1. arr ← un tablou din $N + 1$ numere întregi; ***Memorizează contorul pentru orice număr extras***;
    2. $arr_0 \ldots arr_N \leftarrow 0$; ***Indexarea se începe cu 0***;
    3. ex ← un vector dinamic, inițial gol, ce conține, în mod secvențial, numerele extrase;
    4. even ← se trunchiază: $1 + (N / 2)$; **precizare**: $|\{x \in [0, N] \mid x \bmod 2 = 0\}|$;
    5. odd ← se trunchiază: $N - (N / 2)$; **precizare**: $|\{x \in [0, N] \mid x \bmod 2 = 1\}|$;
    6. $isA \leftarrow 0$;
    7. Cât timp se satisface: $even \neq 0$:
        1. j ← un număr aleatoriu (random) *distribuit uniform* în intervalul $[0, N]$;
        2. Se anexează j în vectorul ex;
        3. Dacă $arr_j = 0$:
            - Dacă $j \bmod 2 = 1$:
                - $odd \leftarrow odd - 1$.

            Altfel:

                - $even \leftarrow even - 1$.

            Altfel, dacă $j = 1$, $arr_j = 1$:

            - $isA \leftarrow 1$.

            Altfel:

            - Se repetă bucla (1).
        4. $arr_j \leftarrow arr_j + 1$.
        5. Se repetă bucla (1).
    8. Dacă a avut loc succes anexarea lui ex în set (nu este duplicat):
        1. Dacă $isA = 1$:
            - $A \leftarrow A + 1$;

2. $\xi \leftarrow \xi + \text{odd}$.

7. $P(A) \leftarrow A / n$;
8. $M\xi \leftarrow \xi / n$;
9. Se afișează la ecran valorile lui: $P(A)$, $M\xi$;
10. Programul se termină cu succes.

III. DESCRIEREA ALGORITMULUI (VERSIUNEA COMPACTĂ)

Știind că mulțimea Ω este nenumărabilă, am elaborat încă un program ce memorizează doar contorul experimentelor ce satisfac condiției A, admițând experimentele duplicate.

1. $N \leftarrow 100$;
2. $A \leftarrow 0$;
3. $\xi \leftarrow 0$;
4. $i \leftarrow 0$;
5. Cât timp se satisface: $i < n$, **notăm bucla (1)**:
    1. arr ← un tablou din $N + 1$ numere întregi;
    2. $arr_0 \ldots arr_N \leftarrow 0$; ***Indexarea se începe cu 0***;
    3. even ← se trunchiază: $1 + (N / 2)$; precizare: $|\{x \in [0, N] \mid x \bmod 2 = 0\}|$;
    4. odd ← se trunchiază: $N - (N / 2)$; precizare: $|\{x \in [0, N] \mid x \bmod 2 = 1\}|$;
    5. Cât timp se satisface: $even \neq 0$:
        1. $j \leftarrow$ un număr aleatoriu (random) *distribuit uniform* în intervalul $[0, N]$;
        2. Dacă $arr_j = 0$:
            - Dacă $j \bmod 2 = 1$:
                - $odd \leftarrow odd - 1$.

            Altfel:

                - $even \leftarrow even - 1$.

            Altfel, dacă $j = 1$, $arr_j = 1$:

            - $A \leftarrow A + 1$.

            Altfel:

            - Se repetă bucla (1).
        3. $arr_j \leftarrow arr_j + 1$.
        4. Se repetă bucla (1).
    6. $\xi \leftarrow \xi + odd$.
6. $P(A) \leftarrow A / n$;
7. $M\xi \leftarrow \xi / n$;
8. Se afișează la ecran valorile lui: $P(A)$, $M\xi$;
9. Programul se termină cu succes.

IV. COD SURSĂ

```
#include <iostream>
#include <random>
#include <deque>
#include <set>
static constexpr size_t N = 100;

int main()
{
    std::mt19937 rd(std::random_device{}());
    std::uniform_int_distribution<> di(0, N);

    size_t A = 0, xi = 0, n;
    std::cin >> n;

    using Experiment = std::deque<uint8_t>;
    std::set<Experiment> set;

    while (set.size() != n)
    {
        uint8_t arr[N + 1]{};
        Experiment ex;

        auto even = 1 + (N >> 1);
        auto odd = N - (N >> 1);
        bool isA = false;

        while (even != 0)
        {
            auto j = di(rd);
            ex.emplace_back(j);

            if (arr[j] == 0)
                --(j & 1 ? odd : even);
            else if (j == 1 && arr[j] == 1)
                isA = true;
            else
                continue;

            ++arr[j];
        }

        if (set.emplace(ex).second)
        {
            if (isA)
                ++A;

            xi += odd;
        }
    }

    std::cout << "P(A) = " << static_cast<double>(A) / n
        << "\nMxi = " << static_cast<double>(xi) / n;
}
```

V. COD SURSĂ (VERSIUNEA COMPACTĂ)

```
#include <iostream>
#include <random>
static constexpr size_t N = 100;

int main()
{
    std::mt19937 rd(std::random_device{}());
    std::uniform_int_distribution<> di(0, N);

    size_t A = 0, xi = 0, n;
    std::cin >> n;

    for (size_t i = 0; i < n; ++i)
    {
        uint8_t arr[N + 1]{};
        auto even = 1 + (N >> 1);
        auto odd = N - (N >> 1);

        while (even != 0)
        {
            auto j = di(rd);

            if (arr[j] == 0)
                --(j & 1 ? odd : even);
            else if (j == 1 && arr[j] == 1)
                ++A;
            else
                continue;

            ++arr[j];
        }

        xi += odd;
    }

    std::cout << "P(A) = " << static_cast<double>(A) / n
        << "\nMxi = " << static_cast<double>(xi) / n;
}
```

VI. REZULTATELE PROGRAMULUI

Rulând programul de 20 de ori, cu n = 1000000, am obținut:

Tabela 1

	Mξ	**P(A)**
	0,958575	0,912436
	0,959291	0,912508
	0,959869	0,912527
	0,960139	0,912533
	0,960333	0,912578
	0,960517	0,912626
	0,960775	0,912628
	0,960956	0,912659
	0,961106	0,912735
	0,961224	0,912766
	0,961259	0,912823
	0,96127	0,912875
	0,961462	0,912905
	0,961864	0,912921
	0,961885	0,912985
	0,962129	0,913028
	0,9622	0,913064
	0,962427	0,913099
	0,963788	0,913216
	0,964322	0,913326
Mediana	**0,9612415**	**0,912795**

VII. REZULTATELE PROGRAMULUI COMPACT

Rulând programul de 20 de ori, cu n = 1000000, am obținut:

Tabela 2

	Mξ	**P(A)**
	0,960118	0,912297
	0,960198	0,912431
	0,960212	0,912484
	0,960465	0,912522
	0,960794	0,912563
	0,960923	0,912595
	0,961026	0,912666
	0,961069	0,912699
	0,961219	0,912702
	0,961299	0,912747
	0,961341	0,912786
	0,961361	0,912812
	0,961462	0,912823
	0,961672	0,912851
	0,961809	0,912862
	0,96233	0,91298
	0,96269	0,913033
	0,963328	0,913177
	0,963385	0,9132
	0,963679	0,913214
Mediana	**0,96132**	**0,912767**

Observând tabelul, putem spune că datele de ieșire nu diferă mult în comparație cu versiunea precedentă a programului.

CONCLUZII

Acest program a fost elaborat în limbajul de programare C++17, utilizând doar bibliotecile prevăzute de membrii comitetului C++. Pentru a aplica metoda Monte Carlo în practică, am utilizat următorul generator de numere pesudo-aleatorie: **Mersenne Twister**[1], dezvoltat în anul 1997, de către cercetătorii *Makoto Matsumoto* și *Takuji Nishimura*.

[1] https://en.wikipedia.org/wiki/Mersenne_Twister