3. Генератори псевдовипадкових чисел

Генерація випадкових чисел – це процес формування послідовності чисел або символів, які неможливо передбачити.

Генератори випадкових чисел (ГВЧ, англ. random number generator, RNG) можуть бути:

- апаратні (справжні) генератори випадкових чисел (англ. hardware random number generator, HRNG або true random number generator, TRNG), які для генеріраціі випадкових чисел використовують фізичні властивості довкілля (детектори подій іонізуючої радіації, дробовий шум електронних елементів, космічне випромінювання тощо);
- генератори псевдовипадкових чисел (ГПВЧ, англ. pseudorandom number generator, PRNG), які генерують числа, схожі на випадковими числа, але такими не є.

Генератор псевдовипадкових чисел – алгоритм, який породжує послідовність чисел, елементи якої майже незалежні один від одного і підкоряються заданому розподілу (зазвичай рівномірному).

У апаратних генераторів випадкових чисел існує ряд недоліків:

- наявність обладнання;
- витрати на установку і налаштування обладнання;
- відсутність уніфікації;
- дорожнеча.

Основний недолік ГПВЧ полягає в тому, що ніякий детермінований алгоритм не може генерувати повністю випадкові числа, а може тільки апроксимувати деякі їх властивості. Будь-який ГПВЧ має обмежені ресурсу, тому рано чи пізно зациклюється (починає повторювати одну і ту ж саму послідовність чисел).

Для отримання різних псевдовипадкових послідовностей, використовують різні джерела ентропії.

Як джерело ентропії в комп'ютері використовують:

- поточний час вбудованого годинника;
- лічильник тактів процесора (/dev/random в UNIX-системах);
- джерело звуку;
- стан деяких вузлів комп'ютера (комірок пам'яті, положення курсору мишки, час затримки між натисканням клавіш клавіатури, розмір вільної пам'яті тощо);
- шуми струмів / напруги (Intel).

Ентропія інтерпретується як міра невизначеності деякої системи (визначає кількість інформації по Шеннону).

Спроби створити генератор випадкових чисел відносяться до 3500 року до н.е. і пов'язані з давньоєгипетської настільною грою Сенет (використовувалися чотири плоскі палички, які пофарбовані в білий колір – з одного боку і в чорний – з іншого (аналог сучасного кубика)).

Основні вимоги, які пред'являються до ГПВЧ:

- довгий період, який гарантує відсутність зациклення послідовності в межах розв'язуваної задачі;
- ефективність – швидкість роботи алгоритму і малі витрати пам'яті;
- відтворюваність – можливість заново відтворити раніше згенеровану послідовність чисел будь-яку кількість разів;
- портуємість – однакове функціонування на різному устаткуванні і операційних системах.

Розглянемо наступні алгоритми ГПВЧ:

- метод серединних квадратів:
- лінійний конгруентний метод;
- регістр зсуву зі зворотним лінійним зв'язком (метод М-послідовності);

Метод серединних квадратів

Це один з перших алгоритмічних методів отримання рівномірно розподілених псевдовипадкових чисел. Він був запропонований Джоном фон Нейманом і полягає в наступному:

1. Вибрати початкове випадкове число X_0, яке має n-розрядне представлення.
2. Возвести це число X_i в квадрат, в результаті чого, ми отримаємо 2n-розрядне число Y_i.
3. Наступне число X_{i+1} отримаємо, склавши його n-розрядне представлення, вибравши середні n розрядів з числа Y_i.

Наприклад, якщо початкове число X_0 =3485, то Y_1 = 3485^2 = 12**1452**25, X_1 = 1452, а X_2=1083, …

Недолік цього методу – наявність кореляції між числами послідовності, а в ряді випадків випадковість взагалі може бути відсутньою. Наприклад, якщо X_0=4500, X_1=2500, X_2=2500, X_3=2500 і т.д. Крім того, даний метод має малий період і зараз представляє інтерес лише в історичному аспекті.

Лінійний конгруентний метод

Одним з простих та популярних методів зараз є лінійний конгруентний метод (ЛКМ), який запропонований Д.Г. Лехмером у 1949 році.
Для отримання послідовності випадкових значень використовують наступний рекурентний вираз:

$$X_{k+1} = (aX_k + c) \bmod m$$

де m – модуль, $m > 0$, a – множник, $0 \leq a < m$, c – приріст, $0 \leq c < m$, X_0 – початкове значення, $0 \leq X_0 < m$ (mod – ділення по модулю).

Наприклад, якщо початкове число $X_0=7$, $a=106$, $c=1283$, $m=6075$, то $X_1=2025$, $X_2=3308$, $X_3=1336$, і т.д.

Модуль вибирають достатньо великим, оскільки період не може містить менше m чисел. В якості m рекомендується брати найбільше просте число, яке не перевищує розрядність машинного слова.
Вибір множника визначається наступною теоремою:
лінійна конгруентна послідовність, визначена числами m, a, c і X_0 має період m тоді і лише тоді, коли виконуються наступні три умови:
1) числа c і m є взаємно простими;
2) число b = a – 1 є кратним числу p для кожного простого числа p, яке є дільником числа m;
3) число b є кратним 4, якщо число m є кратним 4.

Приклади деяких параметрів ЛКМ:
a=106, c=1283, m=6075;
a=211, c=1663, m=7875;
a=430, c=2531, m=11979;
a= 134775813, c=1, m=2^{32}.

Регістр зсуву зі зворотним лінійним зв'язком (метод М-послідовності)

Даний ГПВЧ заснований на ідеї перетворення двійкового представлення числа. Такі генератори мають деякі переваги (наприклад, швидкість генерації чисел, хороші статистичні властивості), а також можливість простої реалізації на програмному та апаратному рівнях.

Регістр зсуву – упорядкований набір бітів, що допускає операцію зміни позицій бітів на одну і ту ж величину вліво або вправо (змінна).

Регістр зсуву зі зворотним лінійним зв'язком (РЗЗЛЗ) – регістр зсуву бітових слів, у якого вхідний біт є лінійною функцією інших бітів. Обчислений біт заноситься в старшу клітинку (номер 0). Кількість комірок *p* називають довжиною регістра (розрядність змінної).

Для натурального числа p і a_{p-1}, a_{p-2}, … a_0, що приймають значення 0 або 1, визначають рекуррентную формулу:

$$x_p = x_{p-1}a_{p-1} \oplus x_{p-2}a_{p-2} \oplus \ldots \oplus x_0a_0$$

Алгоритм, який генерує послідовність, складається з наступних кроків:

1. Вміст старшої комірки визначається значенням функції зворотного зв'язку, яка є лінійною булевою функцією з заданими коефіцієнтами. Значення цього біта обчислюють за наведеною формулою.

2. Вміст кожного i-го біта переміщається в (i -1)й, $0 < i \le p$.

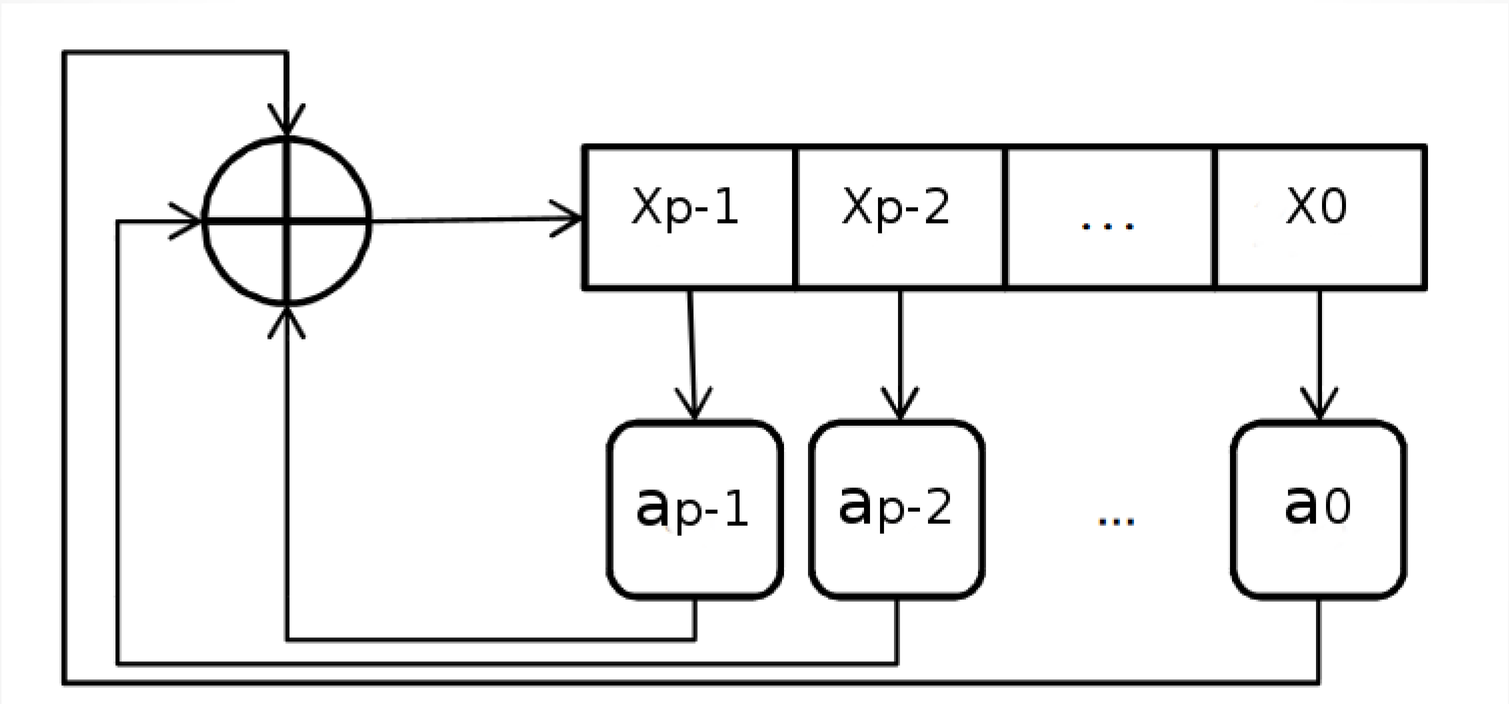


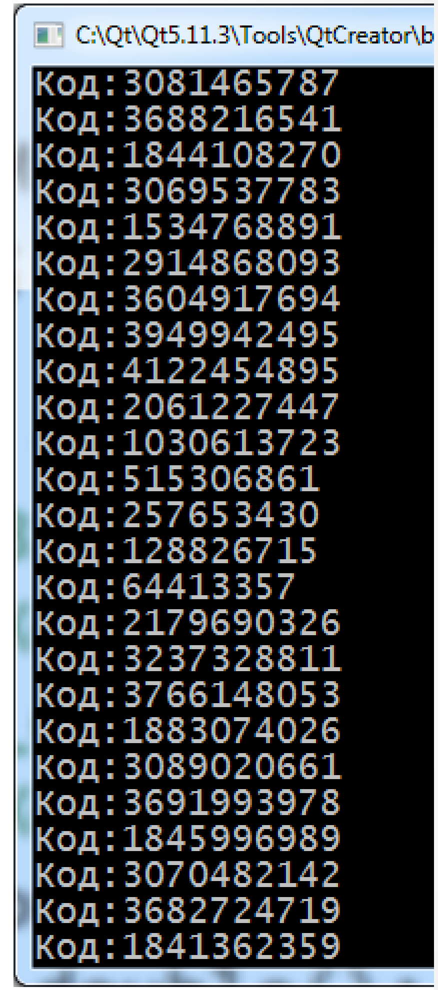
Схема роботи ГПВЧ. На базі РЗЗЛЗ

Значення a_i для генерування чисел різної розрядності p.

Число біт, p	Довжина циклу	Розряди зворотього зв'язку, a_i
16	65,535	[1,2,4,15]
17	131,071	[2,16]
18	262,143	[6,17]
19	524,287	[0,1,4,18]
20	1,048,575	[2,19]
21	2,097,151	[1,20]
22	4,194,303	[0,21]
23	8,388,607	[4,22]
24	16,777,215	[0,2,3,23]
25	33,554,431	[7,24]
26	67,108,863	[0,1,5,25]
27	134,217,727	[0,1,4,26]
28	268,435,455	[2,27]
29	536,870,911	[1,28]
30	1,073,741,823	[0,3,5,29]
31	2,147,483,647	[2,30]
32	4,294,967,295	[1,5,6,31]

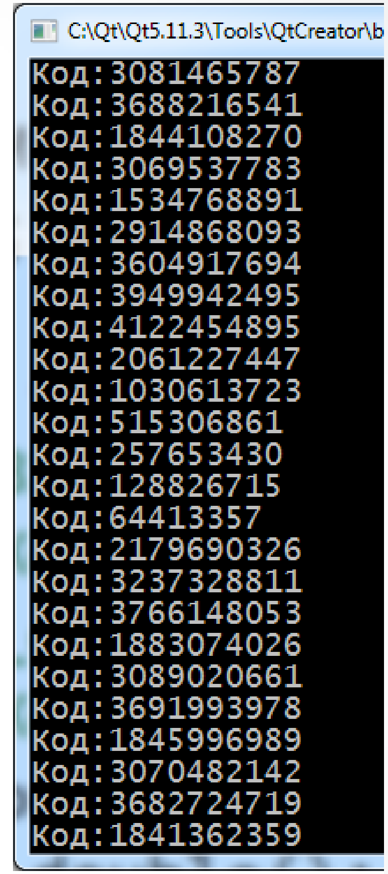
Приклад 1

```
#define BVD( x ) ((uint32_t)( 1 ) << ( x ))

uint32_t lfsr32(void) {
    // регистр сдвига
    static uint32_t lfsr = 0xDEADBEEF;

    uint8_t new_bit = 0;

    // xor всех разрядов в один бит
    if ( lfsr & BVD(1) )
        new_bit ^= 1;
    if ( lfsr & BVD(5) )
        new_bit ^= 1;
    if ( lfsr & BVD(6) )
        new_bit ^= 1;
    if ( lfsr & BVD(31) )
        new_bit ^= 1;
    // сдвиг в новый бит
    lfsr >>= 1 ;
    lfsr |= (uint32_t)( new_bit ) << 31;

    return lfsr;
}
```



Приклад 2

```c
/*
 * Подпрограмма генерации случайного числа 32-bit используя LFSR
 */
uint32_t lfsr32(void) {
    union lfsr_t {
        // регистр сдвига
        uint32_t val = 0xDEADBEEF;
        struct {
            uint32_t    :1;
            uint32_t b1 :1;
            uint32_t    :3;
            uint32_t b5 :1;
            uint32_t b6 :1;
            uint32_t    :24;
            uint32_t b31:1;
        } bval;
    };
    static lfsr_t rngval;
    // xor всех разрядов в один бит
    uint32_t new_bit = rngval.bval.b31 ^ rngval.bval.b6 ^ rngval.bval.b5 ^ rngval.bval.b1;
    // сдвиг в новый бит
    rngval.val >>= 1;
    rngval.val |= new_bit << 31;

    return rngval.val;
}
```



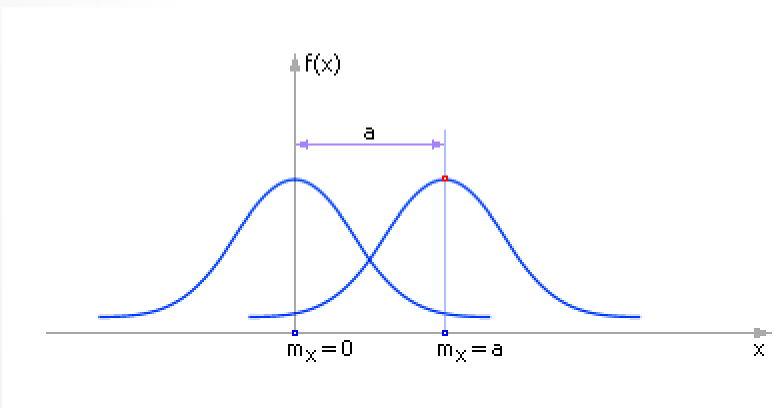
Метод генерації нормально розподілених чисел. Метод Мюллера

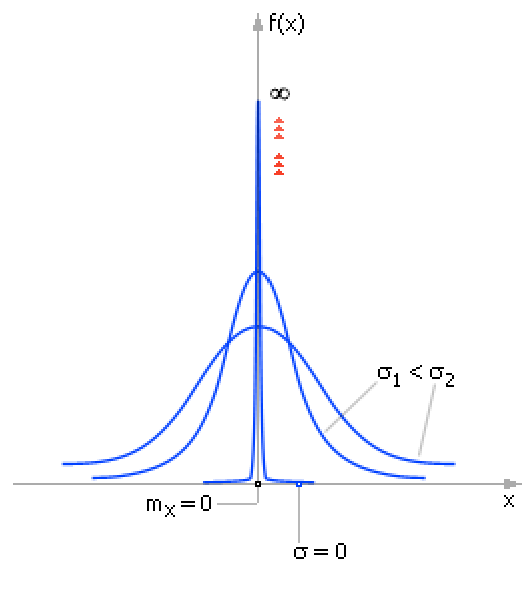
Метод Мюлера використовує ГПВЧ з рівномірним законом та перераховує у нормальний закон за виразами:

z=sqrt(-2*log(r1))*cos(2*PI*r2)

d=s*z+m.

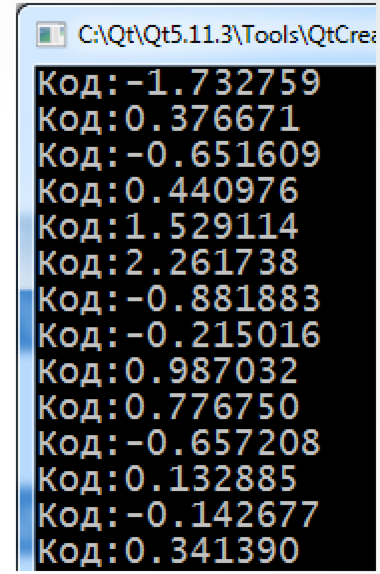
де r1, r2 – випадкові значення, s- відхилення, m – математичне сподівання.





Приклад 3

```
int main()
{
    SetConsoleCP(CP_UTF8);
    SetConsoleOutputCP(CP_UTF8);
    const double PI=3.1415;
    double d;
    double z;
    double r1, r2;
    int32_t M = 1000;

    for(int32_t i=0; i < M; i++) {
        r1 = rand_double();
        r2 = lfsr32_double();

        z=sqrt(-
2*log(r1))*cos(2*PI*r2);
        d=1*z+0; //d=2*z+7;

        printf("Код:%8.6f\n", d);

    }
    return 0;
}
```

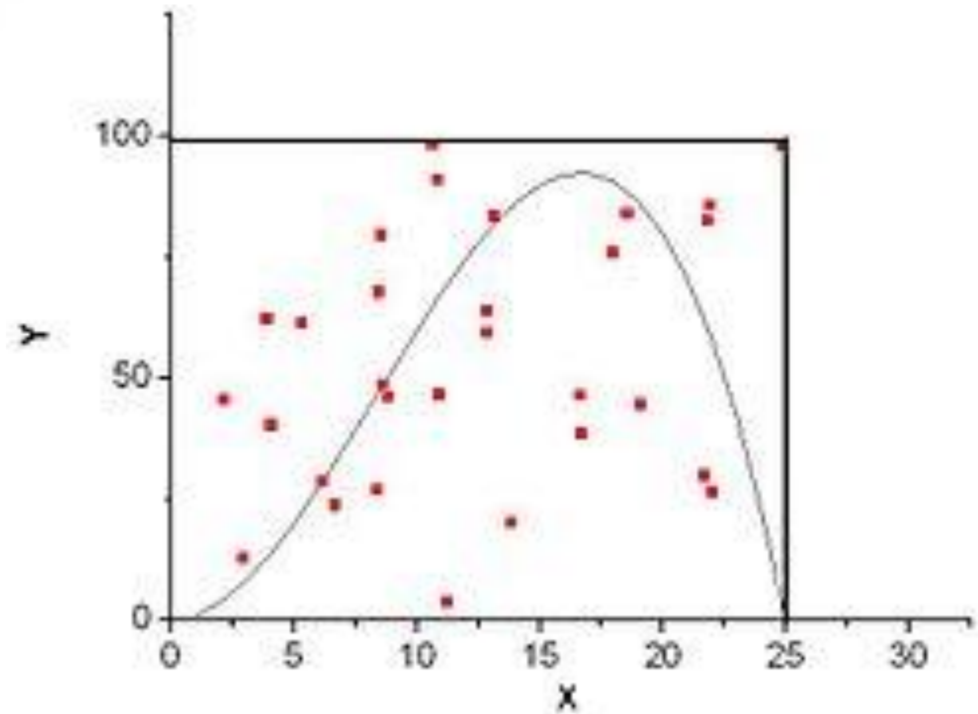



```
double lfsr32_double(void) {
    return (double)lfsr32()/UINT32_MAX;
}
double rand_double(void) {
    return (double)rand()/RAND_MAX;
}
```

Приклад застосування послідовності випадкових величин. Метод Монте-Карло

У випадках, коли аналітичні методи не можуть бути застосовні до вирішення математичних моделей, тоді можливе застосування чисельних методів. Універсальним методом математичного моделювання є метод статистичного моделювання або метод Монте-Карло. Ідея методу полягає в наступному. Замість того, щоб описувати процес за допомогою аналітичного апарату (диференціальних або алгебраїчних рівнянь), проводиться «розіграш» випадкового явища за допомогою спеціально організованої процедури, що включає в себе випадковість і дає випадковий результат. Конкретне здійснення (реалізація) випадкового процесу складається щоразу по іншому; також і в результаті статистичного моделювання («розіграшу») ми отримуємо кожного разу нову, відмінну від інших реалізацію досліджуваного процесу. Безліч реалізацій можна використовувати як якийсь штучно отриманий статистичний матеріал, який може бути оброблений звичайними методами математичної статистики.

Розглянемо спосіб, заснований на тлумаченні інтеграла як площі. Нехай підінтегральна функція не негативна і обмежена: $0 \le f(x) \le c$, а двовимірна випадкова величина розподілена рівномірно в прямокутнику D з основою $(b\text{-}a)$ і висотою c.

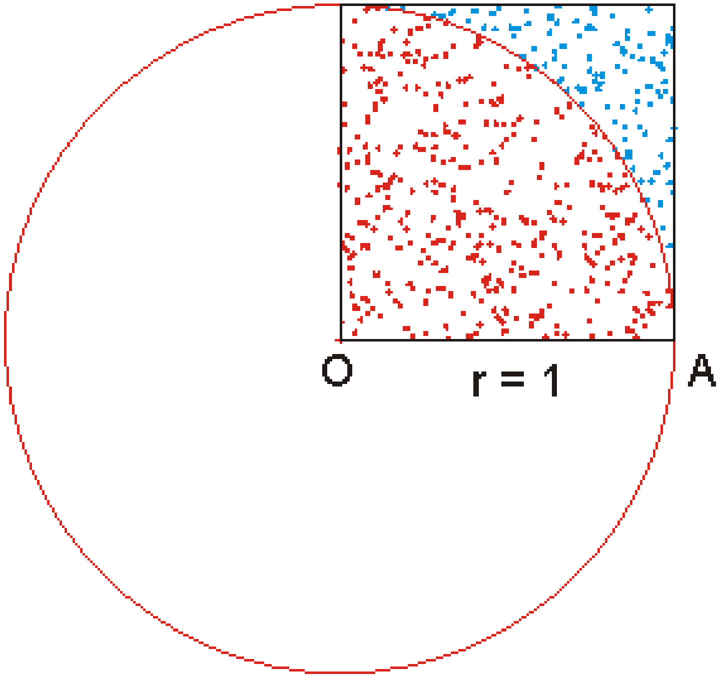


Завдання зводиться до оцінки відношення площі криволінійної трапеції, що відповідає деякому певному інтегралу до площі прямокутника D, в який цей інтеграл може бути вписаний. Ідея методу полягає в наступному. Виберемо пару випадкових чисел – координати випадкової точки в прямокутнику D, а потім виберемо наступну пару чисел – іншу випадкову точку в прямокутнику D і т.д. Коли число обраних таким чином точок стане досить великим, вони більш-менш рівномірно покриють даний прямокутник. При цьому безліч точок N, які потрапили під криву f (x), буде пропорційною площі криволінійної трапеції, а безліч всіх точок M - площі прямокутника D.
Тоді:

$$\hat{I} = S\frac{N}{M}$$

де S – площа прямокутника D.

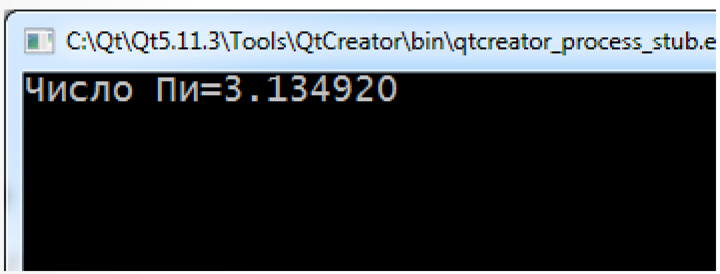
Розрахуємо число Пи за допомогою метода Монте-Карло:



?

Приклад 4

```
double lfsr32_double(void) {
    return (double)lfsr32()/UINT32_MAX;
}
double rand_double(void) {
    return (double)rand()/RAND_MAX;
}
```




```
int main()
{
    SetConsoleCP(CP_UTF8);
    SetConsoleOutputCP(CP_UTF8);

    uint32_t N=0, M = 100000;
    double x,y, yr;

    for(int i=0; i < M; i++) {
        x = rand_double();
        y = lfsr32_double();
        yr = sqrt(1-x*x);
        if(y<=yr) {
            N++;
        }
    }
    printf("Число Пи=%8.6f\n",
4.0*N/M);

    return 0;
}
```