RANDOM NUMBERS GENERATION

MIGUEL BECERRA, TATIANA COY & STHEFANÍA PINTO
UNIVERSIDAD INDUSTRIAL DE SANTANDER

INTRODUCTION

Random numbers are often used in the design of simulations for many phenomenons, related to different areas of science, in which is important to have reliable and reproducible results.
Using random numbers allows scientists to prove the reproducibility of their results in a determined algorithm for the modelling situation. Therefore, the generation of random numbers becomes quite an important part of the model's developing.

PROBLEM

Create an algorithm for the generation of random numbers that is simple, fast and easy to run.

APPROACH AND SOLUTION METHOD

The linear congruential method produces a sequence of pseudo-random integers according to the following recursive relationship:

$$X_{n+1} = (ax_n + b) mod m$$

- The initial value)
- a is called the constant multiplier. a < m
- b is the increment b < m
- m is the modulus. 0 < m

The selection of a, c, m and Xo drastically affects the statistical properties such as mean and variance, and the cycle length.

After making an algorithm for the solution, using this mathematical method, the next step was to translate it into a code. The code was made in C language, supported by GUANE cluster, from Industrial University of Santander. The program goes like this:

```
int xo,x1,xs; /*xs=semilla, xo=valor anterior al que se quiere generar,
x1=número siguiente que va a ser generado */
int a,b,m; /*a=cte multiplicador, c=cte incremento, m=módulo */
int i,n; /*i para el loop, n cantidad mínima de valores a generar */
int array[50]; /*para almacenar los valores obtenidos */
int Periodo = 0; /*Para llevar cuenta del periodo de cada configuracion */
```

Fig. 1. Declaring of the needed variables.

This code is interactive, of course. The user must input the initial conditions for the algorithm to generate the numbers. Afterwards, the actual coding begins, as a **for** loop is designed and the variables are operated.

```
xo=xs; /*definimos xo=xs para que el primer número con
el que se genera los siguientes sea el de la semilla*/

for(i=0;i<n;i++) /* loop para generar los números */
{
    x1=(a*xo+b) %m; /*Forma recursiva para obtener los números aleatorios*/

    if (x1 != xs){
    array[i]=x1;
    xo=x1;
    Periodo=Periodo + 1;    /*De esta manera llevamos un conteo del Periodo*/
    } else{                 /*Si el valor obtenido es igual al de la semilla, deja de generar números*/
        array[i]=x1;
        break;
    }
}
```

Fig. 2. Developing of the loop

Finally, the algorithm returns the values for the generated random numbers, and the proccess is finished.

```
    printf("Los números generados son: ");
    for(i=0;i<Periodo+1;i++)
    {
        printf("%d\n",array[i]);        /*Se muestran los números generados*/
        /*printf("%d", Periodo);*/
        printf("\t");
    }
    if (Periodo==n){    /*Si el pedidoeriodo es mayor que el número pedido de datos a generar, se pide
    ingresar un valor mayor, para de esta manera poder hallar el periodo*/
        printf("Aún no se cúal es el periodo, ingresa un valor mayor a la cantidad de números a generar.");
    } else {
        printf("Tu periodo es de: %d\n", Periodo); /*de lo contrario, se muestra el periodo obtenido*/
    }

    getch();
    return(0);
}
```

Fig. 3. Loop results/Algorithm execution results

The approximate time that the code takes to be executed is 0.004 seconds.

```
real    0m0.004s
user    0m0.000s
sys     0m0.002s
```

COMPLEXITY ANALYSIS

After an extensive complexity analysis, it could be noticed that the algorithm works under a linear behavior in all three posible cases: the best, the worst and the average behavior.
Therefore, the algorithm proved itself to be useful and optimal enough to be considered suitable as a solution for the problem.

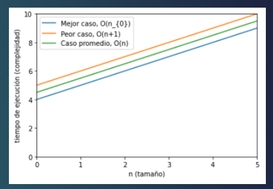

Fig. 4. Algorithm's behavior for best, worst and average case of execution time

FUTURE IMPROVEMENT

The proccess showed that, using the linear congruential method, the generated items wouldn't be completely random, but pseudo-random numbers, and this effect is caused by the initial conditions of the seed. It could be said that a way to improve this solution is using a modified method, in which the previous generated number has no influence in the generation of the next number.
For example, instead of using just one seed, a list of seeds could be used, as an array that uses the same algorithm for each seed at the same time, generating a new array of random numbers.

This would allow the program to use more parallelization in its structure, which would make the algorithm more efficient.

REFERENCES

- J.J, Pamies-Teixeira & C, Rodriguez & Correia, Mário & C.C, Ferreira & R, Toledo. (2005). La Importancia de la Simulación en la Investigación Científica del Maquinado.
- Mañas, J. A. (2017, 10 febrero). Análisis de algoritmos - complejidad. Departamento de Ingeniería de Sistemas Telemáticos -Universidad Politécnica de Madrid. Recuperado 7 de marzo de 2022, de https://www.dit.upm.es/~pepe/doc/adsw/tema1/Complejidad.pdf