Pruebas para generadores de números aleatorios

Juan Vargas

***Abstract*— Los números aleatorios tienen importantes aplicaciones que van desde la seguridad informática, el modelado, la simulación, y la estadítica hasta la música y la literatura. La aleatoriedad de un generador de números aleatorios puede ser demostrada usando varios tipos de pruebas. Este artículo presenta la implementación en Python de las pruebas de uniformidad de Kolmogorov-Smirnov y de Chi-cuadrado, y la prueba de aleatoriedad de rachas.**

I. INTRODUCCIÓN

Un número aleatorio, o más bien, una secuencia de números aleatorios, es aquella en la que cada número es escogido de un rango de posibles valores en el que cada uno tiene la misma probabilidad [1] (distribución uniforme), y la escogencia de cada número de la secuencia es estadísticamente independiente de las otras.

En diversas áreas relacionadas con la computación se necesitan números aleatorios, paradójicamente, hacer que un computador genere un número aleatorio es muy complicado ya que estos siempre siguen instrucciones, por lo que su comportamiento es completamente predecible. Existen dos enfoques principales para la generación de números aleatorios usando computadores.

El primero es usar generadores de números pseudo-aleatorios (PRNGs), estos usan fórmulas matemáticas para generar secuencias de números que parecen aleatorios, aunque en realidad no lo sean, porque son predecibles. Este tipo de generadores son eficientes, porque pueden producir muchos números en poco tiempo, y la secuencia solo empezará a repetirse cuando el número de iteraciones iguale al periodo del generador, y son determinísticos, ya que las secuencias son reproducibles si se comienza el proceso con la misma semilla , un número a partir del cual el generador hace los cálculos. Si bien la periodicidad no es una característica deseada para un generador de números aleatorios, la solución que plantean la mayoría de los generadores es establecer un periodo lo suficientemente largo como para que no afecte las aplicaciones prácticas. Los PRNGs son útiles para aplicaciones en las que se requiera una gran cantidad de números, y no sea de importancia el hecho de que la secuencia llegue a repetirse. Por lo tanto, si bien no son recomendables para aplicaciones como la encripción de datos, y los juegos de azar, en donde se necesita que los números sean realmente aleatorios, sí lo son para tareas relacionadas con el modelado y la simulación, y los videojuegos [1].

El otro enfoque consiste en usar TRNGs, que en español se llamarían generadores de números verdaderamente aleatorios. Los TRNGs usan como semilla datos generados por fenómenos físicos de la vida real, como el ruido atmosférico, el ruido térmico, la desintegración radioactiva, e incluso los cambios en el movimiento del ratón del computador. Los TRNGs normalmente son menos eficientes que los PRNGs, ya que generar cada número toma mucho más tiempo, no tienen periodo, y no son deterministas, por lo que no se puede predecir su comportamiento, y una secuencia dada no puede ser reproducida a propósito, y sus aplicaciones se ven prácticamente limitadas a aquellas en las que los PRNGs no son útiles.

II. PRUEBAS PARA GENERADORES DE NÚMEROS ALEATORIOS

Los generadores de números aleatorios que implementamos en este laboratorio fueron PRNGs, las pruebas para estos generadores se hacen para determinar si la secuencia producida, responde a algún patrón, o puede considerarse aleatoria [2]. Lo que se busca con estas pruebas es determinar la uniformidad y la aleatoriedad de la secuencia generada. La uniformidad hace referencia a que todos los números del rango en el que el generador los produzca, deben tener igual probabilidad de aparecer en la secuencia, o lo que es lo mismo, que la secuencia producida se ajuste a una distribución uniforme. La aleatoriedad hace referencia a que la secuencia producida no siga un patrón, es decir, que no se pueda predecir su comportamiento. Las pruebas de uniformidad implementadas fueron las de Kolmogorov-Smirnov y la de Chi-cuadrado. La prueba de aleatoriedad implementada fue la de rachas.

A. Prueba de Kolmogorov-Smirnov

La prueba de Kolmogorov-Smirnov es usada para determinar cuán parecida es una muestra de datos a una distribución de probabilidad de referencia, basándose en la comparación entre la función de distribució acumulada observada $S_n(x)$ y la esperada $F_X(x)$. Entre más pequea sea esta diferencia, es mayor la probabilidad de que las dos distribuciones sean iguales, o lo que es lo mismo, que la muestra de datos que está siendo evaluada provenga de dicha distribución. La función de distribución acumulada $S_n(x)$ viene dada por la siguiente expresión:

$$S_n(x) = \frac{numerodeobservaciones \leq x}{n}$$

La diferencia entre las distribuciones viene dada por el estadístico:

$$D_e = max|F_X(x) - S_n(x)|$$

Se calcula el estadístico D_e y si este es menor al valor crítico D_α encontrado en una tabla de Kolmogorov-Smirnov para cierto valor de significancia α, y un tamaño de muestra

n. La prueba de Kolmogorov-Smirnov puede resumirse en los siguientes pasos:

- Ordenar los datos de la muestra de manera ascendente.
- Calcular el estadístico: $D_e = max|F_X(x) - S_n(x)|$.
- Encontrar el valor crítico D_α en una tabla de Kolmogorov-Smirnov para cierto valor de significancia α, y un tamaño de muestra n.
- Si $D_e \leq D_\alpha$, entonces se acepta la hipótesis de que la muestra de datos se ajusta a la distribución de referencia.

En nuestro caso, la prueba compara la función de distribución de probabilidad uniforme con la distribución de una muestra de n números.

B. Prueba de Chi-cuadrado

La prueba de Chi-cuadrado χ^2 es otra prueba de uniformidad, que puede ser usada para cualquier distribución. La prueba consiste en dividir tanto la muestra de datos empírica como la muestra de datos de la distribución de referencia en k clases para elaborar un histograma a partir del cual se comparan las frecuencias obtenidas con las esperadas. El estad'istico de la prueba es el siguiente:

$$\chi^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i}$$

En donde O_i y $E_i$es son respectivamente, la frecuencia observada y la frecuencia esperada para la clase k. Se aceptará que la muestra se ajusta a la distribución de referencia con un nivel de significancia α y k grados de libertad si $\chi^2 \leq \chi^2_{\alpha,k}$, en donde el valor de $\chi^2_{\alpha,k}$ se extrae de una tabla de Chi-cuadrado (χ^2) [3].

C. Prueba de Rachas

La prueba de rachas es una prueba de aleatoriedad, que permite comprobar la hipótesis de que una muestra es aleatoria, esto es, que no sigue un patrón, y el siguiente número de la muestra siempre es independiente de los anteriores. Esta prueba se basa en el número de rachas presentes en la muestra. Una racha es una secuencia de valores muestrales que presentan una característica común, seguida de otra secuencia que no la presenta [4]. La característica implementada para este laboratorio fue determinar si la racha es creciente o decreciente, es decir, dada la secuencia $X_1, X_2, ..., X_n$, existirá una racha creciente de longitud l, mientras se cumpla que $X_i \leq X_{i+1}$, en el momento en que se deje de cumplir lo anterior, y en cambio, se cumpla que $X_i \geq X_{i+1}$, comenzará una racha decreciente, hasta que deje de cumplirse la condición inmediatamente anterior. La presencia de un número muy pequeño o muy grande de rachas en la muestra nos llevará a rechazar intuitivamente la aleatoriedad de la misma. Para comprobarlo de manera matemática, calcularemos el estadístico Z, y llamaremos R al número total de rachas.

$$Z = \frac{R - \frac{2n-1}{3}}{\sqrt{\frac{16n-29}{90}}}$$

Z sigue una distribución normal $N(0,1)$. Para un nivel de significación α, la hipótesis de aleatoriedad será rechazada si $|Z| < Z_{\alpha/2}$ [5]. Los valores de $Z_{\alpha/2}$ se extraen de la tabla de la distribución normal.

III. IMPLEMENTACIÓN

A continuación se muestra el código en Python de la implementación de estas pruebas. Para cada prueba el usuario elige, la prueba en concreto a ejecutar, el generador a probar, y la cantidad de números a generar para la prueba.

A. Prueba de Kolmogorov-Smirnov

```
        x = generadores[seleccion](n)
    # x contiene la secuencia
    #del generador
    # CDF Teorica para una distribucion
    #uniforme F(x)
    u = [1.0] * n
    # Generamos la distribucion acumulada
    #de una distribucion uniforme con
    #parametros 0 y 1, para n valores.
    x1 = np.arange(1/(n+0.0),
    1+1/(n+0.0),1/(n+0.0))
    y1 = np.cumsum(np.sort(u)/
    np.max(np.cumsum(u)))

    #Generamos distribucion acumulada
    #del generador que estamos evaluando
    #para n valores.

    x2 = np.sort(x)
    y2 = np.cumsum(np.sort(x)/
    np.max(np.cumsum(x)))
        #Calculamos la diferencia De
    De=np.absolute(y2-y1)
    #Calculamos Da
    Da = (1.36/(n**0.5)+0.0)
    if np.max(De)< Da:
        # GENERADOR UNIFORME
    else:
        # GENERADOR NO UNIFORME
```

B. Prueba de Chi-cuadrado

```
        x = generadores[seleccion](n)
    # x contiene la secuencia
    #del generador
        # Numero de clases
        k = 20
    fi, clases, nadaimportante
    = matplotlib.pyplot.hist(x,k)
    # fi = observaciones encontradas,
    # ei= observaciones esperadas
    ei = np.ones_like(clases)*(n/k)
    chi2 = 0
```

```
    ei = n/k
    for i in fi:
        chi2 += (i-ei)**2
    #print chi2
    if chi2 <=31.41:
        #GENERADOR UNIFORME
    else:
        #GENERADOR NO UNIFORME
```

C. Prueba de rachas

```
        x = generadores[seleccion](n)
    # x contiene la secuencia
    #del generador
        creciente = True
    rCrecientes = 0
    rDecrecientes = 0
    for i in range(len(x)):
        if(i == 0):
            if(x[i]<x[i+1]):
                creciente = True
                rCrecientes += 1
            else:
                creciente = False
                rDecrecientes += 1
        else:
            if(creciente == True):
                if(x[i]<x[i-1]):
                    rDecrecientes += 1
                    creciente = False
            else:
                if(x[i]>x[i-1]):
                    rCrecientes += 1
                    creciente = True
    # R numero de rachas
    R = rCrecientes + rDecrecientes

    #Z estadistico de contraste
    Z = (R-(2*n-1)/3)/((16*n-29)/90)**0.5

    if(abs(Z) < 0.50978):
        #GENERADOR ALEATORIO
    else:
        #GENERADOR NO ALEATORIO
```

D. Generadores aleatorios usados

- Generador RANDU:

$$X_{i+1} = 65539 X_i mod 2^{31}$$

```
def randu(n):
    xList = []
    m = 2**31
    lastXn = int(time.clock())+65539
    for i in range(n):
        Xn = float((65539*lastXn)% m)
        lastXn = Xn
        random = float(Xn/m)
        xList.append(random)
    return xList
```

- Generador Sinclair ZX81:

$$X_{i+1} = 75 X_i mod 2^{16} + 1$$

```
def sinclair(n):
    xList = []
    m = 2**16+1
    lastXn = int(time.clock())+65539
    for i in range(n):
        Xn = float((75*lastXn)% m)
        lastXn = Xn
        random = float(Xn/m)
        xList.append(random)
    return xList
```

- Generador Numerical Recipes:

$$X_{i+1} = 1664525 X_i + 1013904223 mod 2^{32}$$

```
def numericalRecipes(n):
    xList = []
    m = 2**32
    lastXn = int(time.clock())+65539
    for i in range(n):
        Xn = float((1664525*lastXn+
        1013904223)% m)
        lastXn = Xn
        random = float(Xn/m)
        xList.append(random)
    return xList
```

- Generador Borland C/c++:

$$X_{i+1} = (22695477 X_i + 1) mod 2^{32}$$

```
def borland(n):
    xList = []
    m = 2**32
    lastXn = int(time.clock())+65539
    for i in range(n):
        Xn = float((22695477*
        lastXn+1)% m)
        lastXn = Xn
        random = float(Xn/m)
        xList.append(random)
    return xList
```

IV. RESULTADOS

La función principal del script pide al usuario que seleccione primero la prueba a aplicar, y luego el generador y la cantidad de números de la prueba. Los generadores sometidos a prueba fueron:

- Generador RANDU:

$$X_{i+1} = 65539 X_i mod 2^{31}$$

- Generador Sinclair ZX81:

$$X_{i+1} = 75X_i mod 2^{16} + 1$$

.

- Generador Numerical Recipes:

$$X_{i+1} = 1664525X_i + 1013904223 mod 2^{32}$$

- Generador Borland C/c++:

$$X_{i+1} = (22695477X_i + 1) mod 2^{32}$$

La semilla utilizada fue la generada por la función *time.time()* de Python, la cual retorna el valor en segundos del tiempo del procesador. Cada generador fue probado generando 100, 1000 y 10000 valores, para cada una de las tres pruebas.

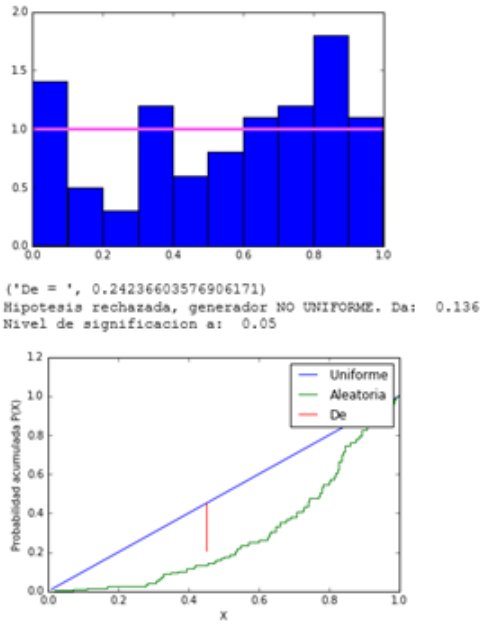


Fig. 1. Prueba de Kolmogorov-Smirnov del generador RANDU con n = 100

A. Generador RANDU

- Prueba de Kolmogorov-Smirnov: El generador RANDU, no superó la prueba de Kolmogorov-Smirnov con ninguno de los tres valores de n, aunque la uniformidad puede verse mejorando en el histograma producido a medida que aumenta el tamaño de la muestra. Esto puede verse en Fig. 1, Fig. 2 y Fig. 3.
- Prueba de Chi-cuadrado: El generador superó la prueba para n = 100, aunque para n igual a 1000 y a 1000 no lo hizo, a pesar de eso, la uniformidad puede verse mejorando en el histograma con el aumento en el tamaño de la muestra. Esto puede verse en Fig. 4.
- Prueba de rachas: El generador superó la prueba de rachas para n = 100, pero no lo hizo para n igual a 1000 y a 10000. Los resultados pueden verse en Fig. 5, Fig. 6 y Fig. 7.

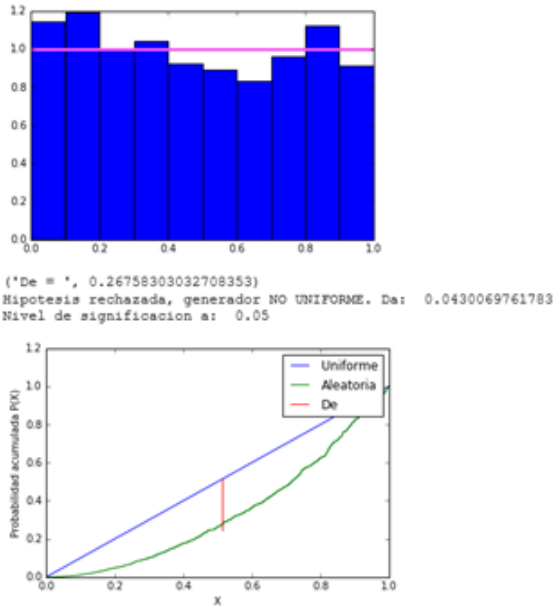


Fig. 2. Prueba de Kolmogorov-Smirnov del generador RANDU con n = 1000

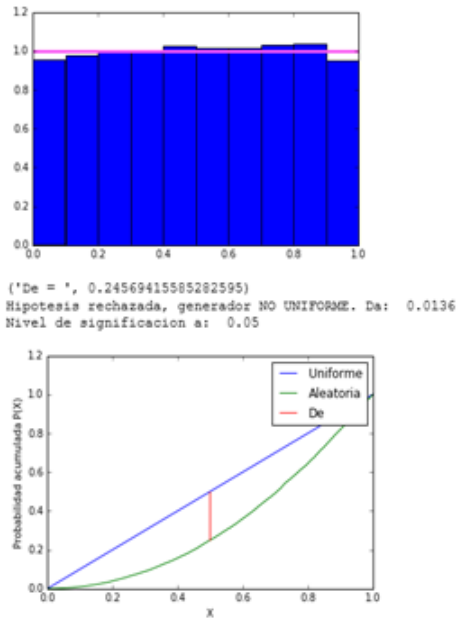
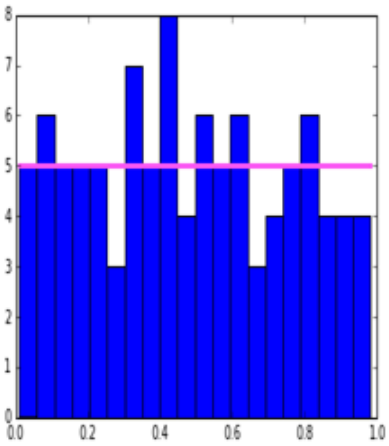


Fig. 3. Prueba de Kolmogorov-Smirnov del generador RANDU con n = 10000

B. Generador Sinclair ZX81

- Prueba de Kolmogorov-Smirnov: El generador Sinclair ZX81 no pasó la prueba de Kolmogorov-Smirnov con ningún valor de n, aún así, la uniformidad aumenta con la cantidad de números generados. Los resultados se muestran en Fig. 8, Fig. 9 y Fig. 10.
- Prueba de Chi-cuadrado: Este generador no superó la prueba de Chi-cuadrado con ningún valor de n. Los histogramas de las secuencias producidas se muestran en Fig. 11, y Fig. 12.



```
Hipotesis aceptada. Generador UNIFORME. Estadistico x2:  30.0
Nivel de significacion a:  0.05
```

Fig. 4. Prueba de Chi-cuadrado del generador RANDU con n = 100

- Prueba de rachas: Este generador solamente superó la prueba de rachas para n = 10000. El nmero de rachas crecientes fue muy cercano al nmero de rachas decrecientes, en las tres pruebas. La relacin entre el tamao total de la muestra y la cantidad de rachas presente en la misma fue de 62%, 65.8%, y 66.51% para las muestras 100, 1000 y 10000 valores respectivamente. Los resultados de las tres pruebas pueden verse en Fig. 13, Fig. 14, y Fig. 15.

C. *Generador Numerical Recipes*

- Prueba de Kolmogorov-Smirnov: El generador Numerical Recipes no superó la prueba de Kolmogorov-Smirnov, el valor del estadstico de diferencia fue de aproximadamente 0.22, 0.26, y 0.25, aún así, a medida que se aumentaba el tamaño de la muestra el límite establecido por D_α era cada vez más bajo.
- Prueba de Chi-cuadrado: Este generador no superó la prueba de Chi-cuadrado. Como en los casos anteriores, en el histograma se puede ver que la uniformidad de la

```
Inserte la cantidad de numeros a generar n: 100
Rachas Crecientes:  34
Rachas Decrecientes:  33
Total de Rachas R:  67
Nivel de confianza a:  0.05
El generador probado es ALEATORIO. Hipotesis aceptada. Z:  0.242535625036
```

Fig. 5. Prueba de rachas del generador RANDU con n = 100

```
Inserte la cantidad de numeros a generar n: 1000
Rachas Crecientes:  341
Rachas Decrecientes:  341
Total de Rachas R:  682
Nivel de confianza a:  0.05
El generador probado es NO ALEATORIO. Hipotesis rechazada. Z:  1.2026336448
```

Fig. 6. Prueba de rachas del generador RANDU con n = 1000

```
Inserte la cantidad de numeros a generar n: 10000
Rachas Crecientes:  3346
Rachas Decrecientes:  3345
Total de Rachas R:  6691
Nivel de confianza a:  0.05
El generador probado es NO ALEATORIO. Hipotesis rechazada. Z:  0.59305680665
```

Fig. 7. Prueba de rachas del generador RANDU con n = 10000

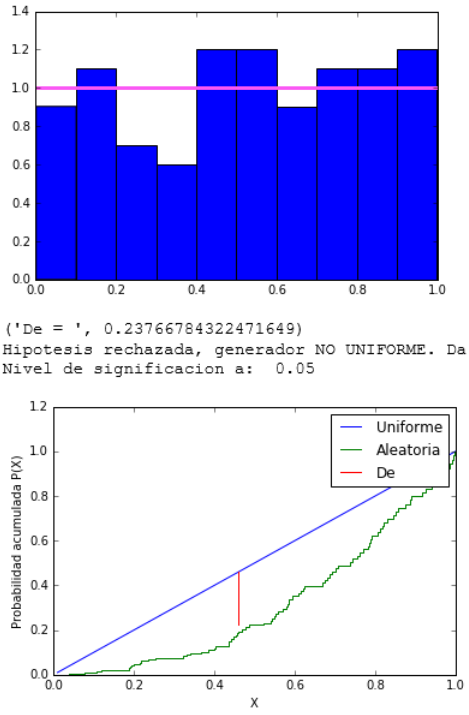


Fig. 8. Prueba de Kolmogorov-Smirnov del generador Sinclair Zx81 con n = 100

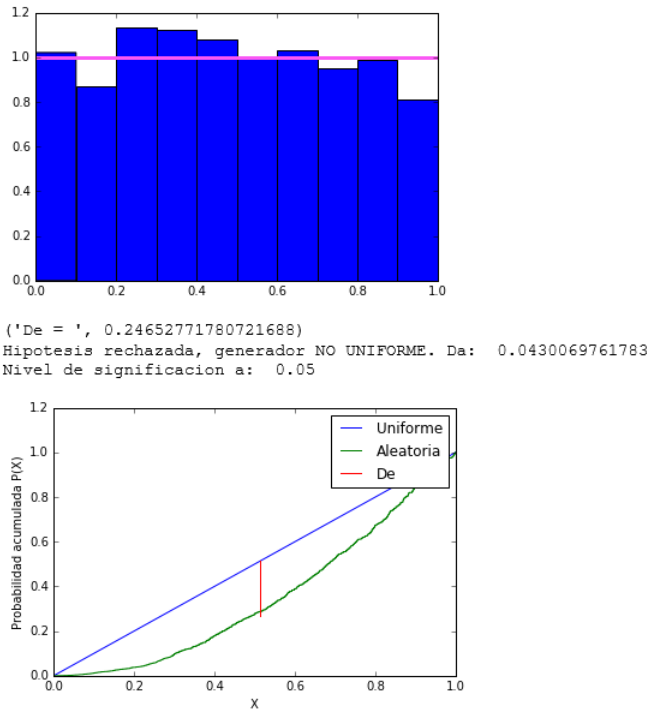
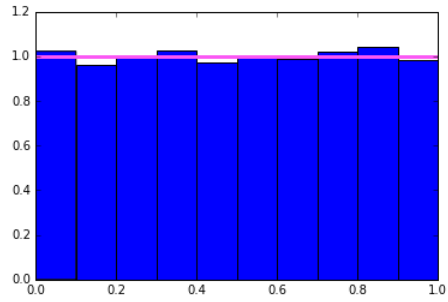


Fig. 9. Prueba de Kolmogorov-Smirnov del generador Sinclair Zx81 con n = 1000



```
('De = ', 0.25023010027890347)
Hipotesis rechazada, generador NO UNIFORME. Da:  0.0136
Nivel de significacion a:  0.05
```

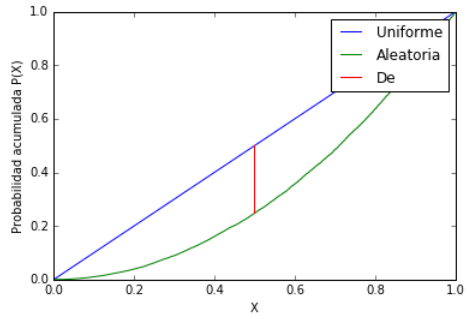
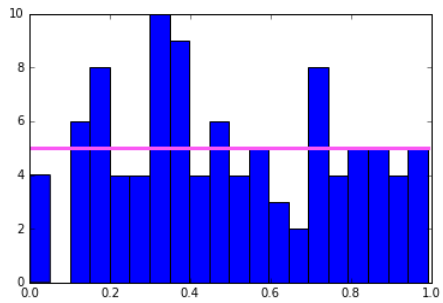
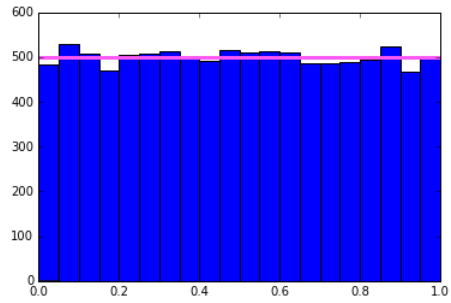


Fig. 10. Prueba de Kolmogorov-Smirnov del generador Sinclair Zx81 con n = 10000



```
Hipotesis rechazada. Generador NO UNIFORME. Estadistico x2:  106.0
```

Fig. 11. Prueba de Chi-cuadrado del generador Sinclair Zx81 con n = 100



```
Hipotesis rechazada. Generador NO UNIFORME. Estadistico x2:  5270.0
```

Fig. 12. Prueba de Chi-cuadrado del generador Sinclair Zx81 con n = 10000

```
Rachas Crecientes:  31
Rachas Decrecientes:  31
Total de Rachas R:  62
Nivel de confianza a:  0.05
El generador probado es NO ALEATORIO. Hipotesis rechazada. Z:  -0.970142500145
```

Fig. 13. Prueba de rachas del generador Sinclair Zx81 con n = 100

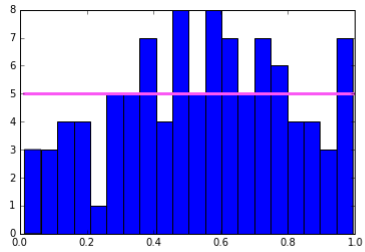
```
Rachas Crecientes:  329
Rachas Decrecientes:  329
Total de Rachas R:  658
Nivel de confianza a:  0.05
El generador probado es NO ALEATORIO. Hipotesis rechazada. Z:  -0.601316822402
```

Fig. 14. Prueba de rachas del generador Sinclair Zx81 con n = 1000

```
Rachas Crecientes:  3325
Rachas Decrecientes:  3326
Total de Rachas R:  6651
Nivel de confianza a:  0.05
El generador probado es ALEATORIO. Hipotesis aceptada. Z:  -0.35583408399
```

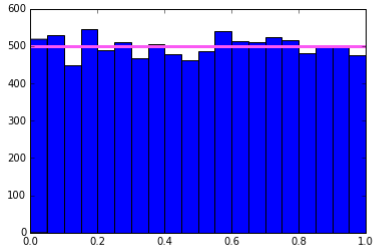
Fig. 15. Prueba de rachas del generador Sinclair Zx81 con n = 10000

muestra parece mejorar a medida que el tamaño de la misma aumenta, esto puede comprobarse en Fig. 16, y Fig. 17.



```
Hipotesis rechazada. Generador NO UNIFORME. Estadistico x2:  68.0
```

Fig. 16. Prueba de Chi-cuadrado del generador Numerical Recipes con n = 100



```
Hipotesis rechazada. Generador NO UNIFORME. Estadistico x2:  12842.0
```

Fig. 17. Prueba de Chi-cuadrado del generador Numerical Recipes con n = 10000

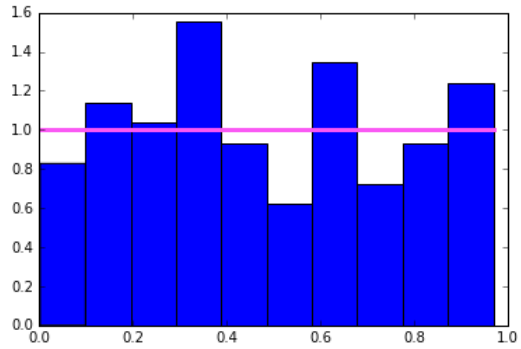
- Prueba de rachas: Este generador superó la prueba de rachas para n = 1000. Los resultados de las pruebas pueden verse en Fig. 18.

```
Rachas Crecientes:  334
Rachas Decrecientes:  334
Total de Rachas R:  668
Nivel de confianza a:  0.05
El generador probado es ALEATORIO. Hipotesis aceptada. Z:  0.150329205601
```

Fig. 18. Prueba de rachas del generador Numerical Recipes con n = 1000

D. Generador Borland C/C++

- Prueba de Kolmogorov-Smirnov: El generador Borland C/C++ no superó la prueba de Kolmogorov-Smirnov.



```
('De = ', 0.25025852616122912)
Hipotesis rechazada, generador NO UNIFORME. Da:  0.136
Nivel de significacion a:  0.05
```

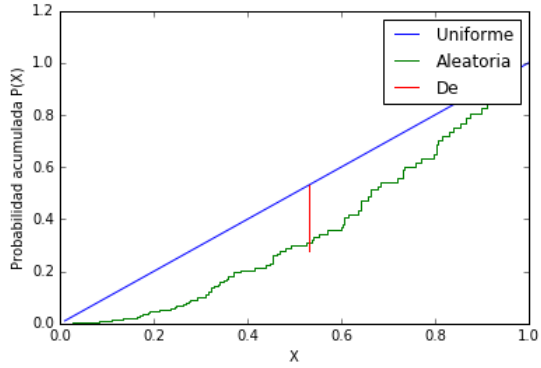
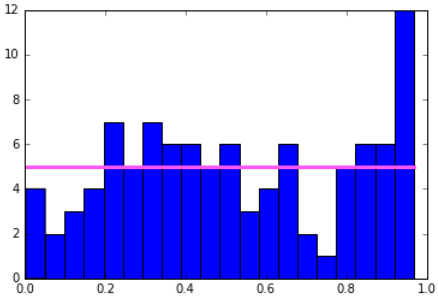


Fig. 19. Prueba de Kolmogorov-Smirnov del generador Borland C/C++ con n = 100

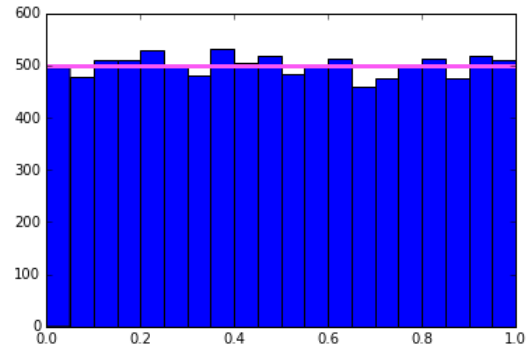
- Prueba de Chi-cuadrado: Este generador no superó la prueba de Chi-cuadrado. Los resultados pueden verse en Fig. 20 y Fig. 21.



```
Hipotesis rechazada. Generador NO UNIFORME. Estadistico x2:  108.0
```

Fig. 20. Prueba de Chi-cuadrado del generador Borland C/C++ con n = 100

- Prueba de rachas: Este generador solamente superó la prueba de rachas para n = 100. Los resultados pueden verse en Fig. 22, Fig. 23 y Fig. 24.



```
Hipotesis rechazada. Generador NO UNIFORME. Estadistico x2:  7248.0
```

Fig. 21. Prueba de Chi-cuadrado del generador Borland C/C++ con n = 10000

```
Rachas Crecientes:  33
Rachas Decrecientes:  33
Total de Rachas R:  66
Nivel de confianza a:  0.05
El generador probado es ALEATORIO. Hipotesis aceptada. Z:  0.0
```

Fig. 22. Prueba de rachas del generador Borland C/C++ con n = 100

```
Rachas Crecientes:  345
Rachas Decrecientes:  345
Total de Rachas R:  690
Nivel de confianza a:  0.05
El generador probado es NO ALEATORIO. Hipotesis rechazada. Z:  1.80395046721
```

Fig. 23. Prueba de rachas del generador Borland C/C++ con n = 100

```
Rachas Crecientes:  3313
Rachas Decrecientes:  3313
Total de Rachas R:  6626
Nivel de confianza a:  0.05
El generador probado es NO ALEATORIO. Hipotesis rechazada. Z:  -0.948890890641
```

Fig. 24. Prueba de rachas del generador Borland C/C++ con n = 100

V. CONCLUSIONES

La mayorí de las pruebas no fueron superadas por los generadores, esto puede deberse a errores en la implementación de los mismos (lo cual es poco probable), o a errores en la implementación de las pruebas, lo cual es mas probable. Aún así esto es raro, porque los códigos fueron desarrollados con el mayor cuidado posible. La implementación deberá revisarse en futuros desarrollos del tema.

REFERENCES

[1] Haahr, M. (2016). RANDOM.ORG - Introduction to Randomness and Random Numbers. [online] Random.org. Available at: https://www.random.org/randomness/ [Accessed 17 Mar. 2016].

[2] Wikipedia.(2015). Wikipedia - Pruebas de aleatoriedad. [online] Wikipedia.org. Available at: https://es.wikipedia.org/wiki/Pruebas_de_aleatoriedad [Accessed 18 Mar. 2016].

[3] Universidad de los Andes - Venezuela (2012). Web del Profesor - Probando Generadores de Numeros Aleatorios. [online] Webdelproesor.ula.ve. Available at: http://webdelprofesor.ula.ve/ingenieria/hhoeger/simulacion/PARTE5.pdf

[4] Universitat de Barcelona. PRUEBA DE RACHAS. [online] Ub.edu. Available at: http://www.ub.edu/aplica_infor/spss/cap5-4.htm

[5] Cruz-Roa, A. (2016). Test o contrastes para numeros aleatorios. [online] https://sites.google.com/site/aacruzr. Available at: https://sites.google.com/site/aacruzr/teaching/2016-i/simulacion-computacional