Práctica 2. "Algoritmos probabilisticos y aleatorios"

Angel Zait Hernández López and Edwin Bernardo Cruz Villalba

ESCOM, Instituto Politécnico Nacional, Ciudad de México 07738, México
Análisis de algoritmos
3CM4

Abstract. La aleatoriedad y análisis probabilístico son temas que se involucran en muchas áreas de las ciencias de la computación incluyendo al diseño de algoritmos, se pueden considerar algoritmos aleatorizados o aleatorios a aquellos tradicionales que se enfrentan a entradas generadas aleatoriamente y en otro caso algoritmos que se comportan al azar, es decir, el medio ofrece la misma entrada pero podemos permitir que tome decisiones al azar ya una vez que procesa la entrada; al permitir una asignación al azar el modelo se vuelve más potente, así que los problemas que pueden no haber sido resueltos mediante algoritmos deterministas eficientes aún pueden ser susceptibles de algoritmos aleatorios. [4]

Key words: algoritmos aleatorios, método de monte Carlo, algoritmos de las Vegas

1 Introducción

Un algoritmo se dice aleatorizado si usa algún grado de aleatoriedad como parte de su lógica.

En este tipo de algoritmos el tiempo de ejecución o su salida se convierten en variables aleatorias.

Con base en el tiempo de ejecución y la salida de los algoritmos aleatorizados como variables aleatorias, podemos definir dos tipos de algoritmos:

- *Las Vegas:* algoritmos que siempre entregan el resultado correcto, pero cuyo tiempo de ejecución varía (incluso para la misma entrada de datos). Un algoritmo Las Vegas se dice eficiente si su tiempo esperado de ejecución es polinomial para cualquier entrada.
- *Monte Carlo:* algoritmos que pueden entregar resultados incorrectos con una probabilidad acotada. Podemos disminuir esta probabilidad de error repitiendo el algoritmo a expensas del tiempo de ejecución. Un algoritmo Monte Carlo se dice eficiente si su tiempo de ejecución en el peor caso es polinomial para cualquier entrada.

1.1 Método de Monte Carlo aplicado a búsquedas

Los Métodos de Monte Carlo se basan en la analogía entre probabilidad y volumen. Las matemáticas de las medidas formalizan la noción intuitiva de probabilidad, asociando un evento con un conjunto de resultados y definiendo que la probabilidad del evento será el volumen o medida relativa del universo de posibles resultados. Monte-Carlo usa esta identidad a la inversa, calculando el volumen de un conjunto interpretando el volumen como una probabilidad. En el caso más simple, esto significa muestrear aleatoriamente un universo de resultados posibles y tomar la fracción de muestras aleatorias que caen en un conjunto dado como una estimación del volumen del conjunto. La ley de grandes números asegura que esta estimación converja al valor correcto a medida que aumenta el número de muestras. El teorema del límite central proporciona información sobre la magnitud del probable error en la estimación después de un número finito de muestras.

En un proceso estándar de Monte Carlo tenemos los siguientes pasos:

- Se generan un gran número de simulaciones aleatorias desde la posición del tablero para la que se desea encontrar el mejor movimiento siguiente.
- Se almacenan las estadísticas para cada movimiento posible a partir de este estado inicial.
- Se devuelve el movimiento con los mejores resultados generales.

Un ejemplo común de la aplicación de método Monte Carlo es la aproximación del valor de π, para ello tenemos que considerar un círculo unitario (radio=1) dentro de un cuadrado con los lados iguales a 2 (véase la figura 1). Sí escogemos un punto al azar (x, y) donde x y y están definidos en el rango $[-1, 1]$, la probabilidad de que este punto al azar se encuentre dentro del círculo unitario se da como la proporción entre el área el círculo unitario y el cuadrado:

$$P(x^2, y^2 \leq 1) = \frac{A_{circle}}{A_{square}} = \frac{\pi}{4} \tag{1}$$

Si escogemos puntos al azar N veces y M de esas veces el punto cae dentro del círculo unitario, la probabilidad de que un punto al azar caiga dentro de dicho círculo esta dado por:

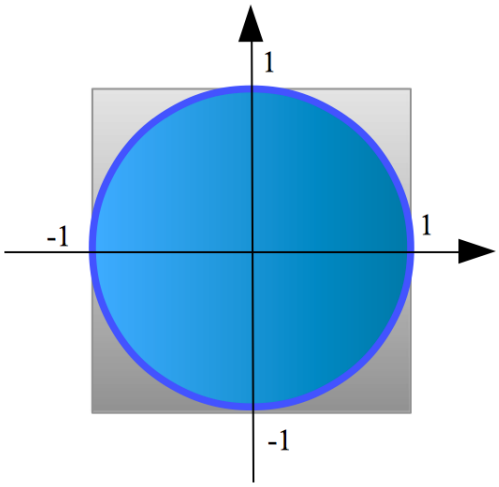
$$P'(x^2, y^2 \leq 1) = \frac{M}{N} \tag{2}$$

Por lo tanto consideramos la siguiente ecuación para el calculo de π:

$$H(x, y) = \begin{cases} 1 \text{ si } x^2 + y^2 \leq 1 \\ 0 \text{ in other case} \end{cases} \tag{3}$$

1.2 Desarrollo (sección 1. Monte Carlo)

- Pruebe el código en python proporcionado y ejecute pruebas para $N = 100, 1000, 10000, 100000, 1000000$, muestre cuáles son los valores que obtiene para π en cada caso.



– Implemente el método de Monte Carlo para resolver integrales. Utilice la siguiente formula:

$$\frac{b-a}{n}\sum_{i=1}^{n} f(u_i(b-a)+a) \tag{4}$$

donde a y b son los límites inferior y superior de una integral, u_i es un valor aleatorio generado entre $[0,1]$ y f se refiere a la función a integrar.
Considere las siguientes integrales para probar su implementación:

$$f_1(x) = \int_0^1 (1-x^2)^{3/2} \cdot dx \tag{5}$$

$$f_2(x) = \int_{-1}^1 e^{x+x^2} \cdot dx \tag{6}$$

$$f_3(x) = \int_0^2 (1+x^2)^2 \cdot dx \tag{7}$$

$$f_4(x) = \int_0^{2\pi} \frac{1}{cos(x)+2} \cdot dx \tag{8}$$

$$f_5(x) = \int_0^2 log(x) \cdot dx \tag{9}$$

Deberá desarrollar una simulación donde se muestre paso a paso la generación de números aleatorios y su graficación en la función correspondiente. (Ver ejemplo en figura 1.2 y consulte la página web https://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/monte-carlo-methods-in-practice/monte-carlo-integration).

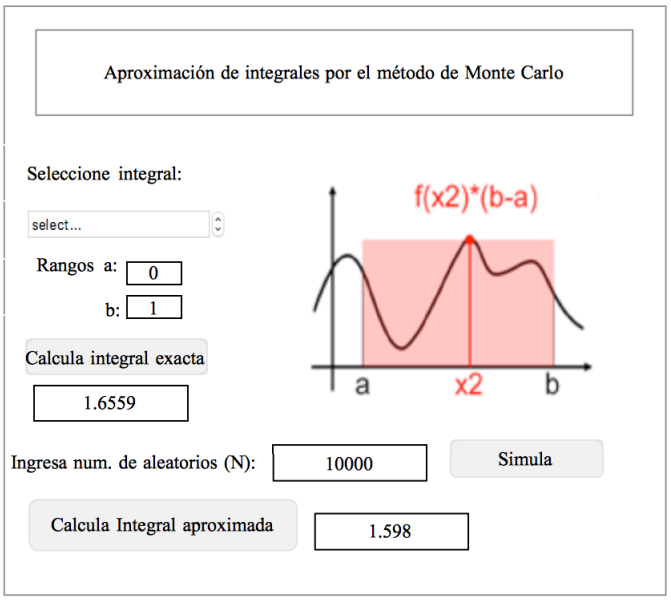


Fig. 1. Ejemplo de interfaz gráfica para simular aproximación a integrales

1.3 Desarrollo (sección 2. Las Vegas)

Para poner en práctica el algoritmo las vegas se implementaran dos ejercicios:

1. Algoritmo de Quick-sort. Modifique el algoritmo de Quick sort implementado en la práctica 1. Por cada llamada recursiva seleccione al azar el pivote, compare si hay mejora respecto a su versión anterior para el caso promedio y reporte el tiempo de ejecución para los siguientes casos $n = 1000, 2000, 3000, \ldots, 10,000$ (n es el número de elementos en un arreglo).
2. El prolema de las 8 reinas. Se requiere generar una solución correcta al problema de las 8 reinas de acuerdo a las siguientes reglas.

El problema de las 8 reinas consiste en colocar las piezas en un tablero de ajedrez sin que se amenacen entre sí. En el juego de ajedrez la reina amenaza a aquellas piezas que se encuentren en su misma fila, columna o diagonal.

Para resolver este problema emplearemos un esquema de algoritmo Las vegas. Se supondrá que hay un cierto número de piezas colocadas en forma correcta (sin atacarse) y se generará el resto de las piezas al azar. Considera las siguientes configuraciones predefinidas e implemente un algoritmo que genere las otras reinas de manera aleatoria.

Reporte en una tabla el porcentaje de éxitos del algoritmo de acuerdo a los casos presentados en la tabla 1.3

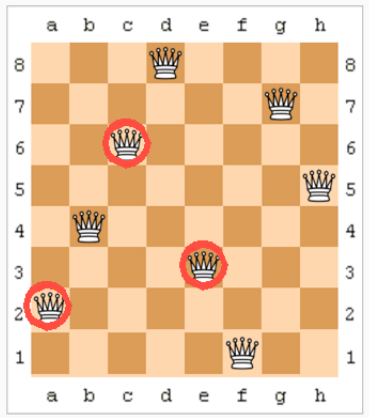


Fig. 2. Tablero con 3 reinas preestablecidas.

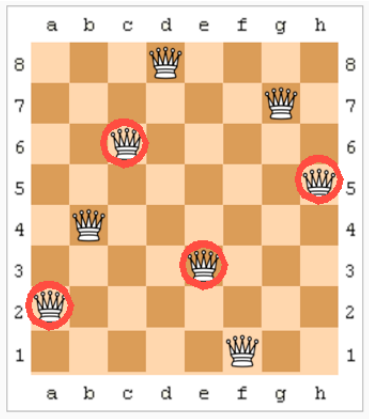


Fig. 3. Tablero con 4 reinas preestablecidas.

Table 1. Casos para el problema de las 8 reinas

	Reinas aleatorias			
Soluciones aleatorias	8	5	4	3
1,000				
10,000				
100,000				
1,000,000				

La práctica puede realizarse en equipos de 2 personas. La entrega de la práctica es por correo electrónico para el próximo viernes 1 de marzo (11:59 pm). Deberá enviar el código fuente y un reporte en formato latex (2). [2] [1] [3] [5]

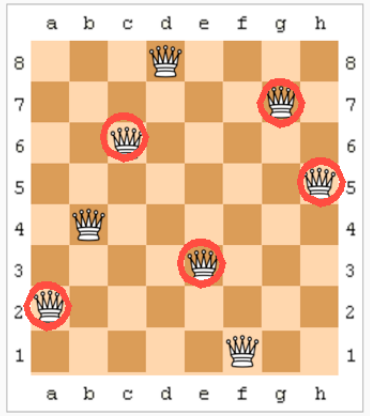


Fig. 4. Tablero con 5 reinas preestablecidas.

2 Reporte de resultados

2.1 Algoritmo Monte Carlo

Aproximación de problema π. Se realizó el ejercicio de aproximación del valor de π por el método de Monte Carlo para distintos valores de N, el cual es la cantidad de valores aleatorios que se generen. A continuación, en la tabla 2, se muestran los valores obtenidos con dichos valores.

Valor de N	Aproximación a π	Error
100	3.2800	4.2197%
1000	3.1520	0.3302%
10000	3.1348	0.2167%
100000	3.1414	0.0061%
1000000	3.1433	0.0528%

Table 2. Calculo de π por algoritmo de Monte Carlo.

Como podemos notar en la tabla anterior, mientras el valor de N es mayor, el cálculo del valor de π es mucho más exácto, esto se debe porque los valores que se van generando aleatoriamente, tienen mucha mayor probabilidad de aparecer en el área del circulo. Como se pueden observar de la figura 5 a la figura 9, al tener un número pequeño (100) de valores aleatorios, es más difícil tener un buen acercamiento al valor de π, caso contrario de tener un número más grande (100000).
Por último, observemos la figura 8 que tiene un valor de $N = 100000$, y como se muestra tanto en la tabla como en la fiura, tiene un valor aproximado a π de 3.1414 con un error de 0.0061%, algo muy cercano al valor original de π,

pero ¿Qué pasa si aumentamos un poco más el valor de N? Este ya no tiene un buen desempeño, haciendo referencia a la figura 9, observamos que tiene un valor de 3.1433 y un error de 0.05228%, algo considerable puesto nosotros queremos que se acerque al valor lo mejor posible. Entonces podemos concluir, que a determinado valor de N existe un buen comportamiento del algoritmo aleatorio que se comporta con una gran eficiencia para resolver un problema.

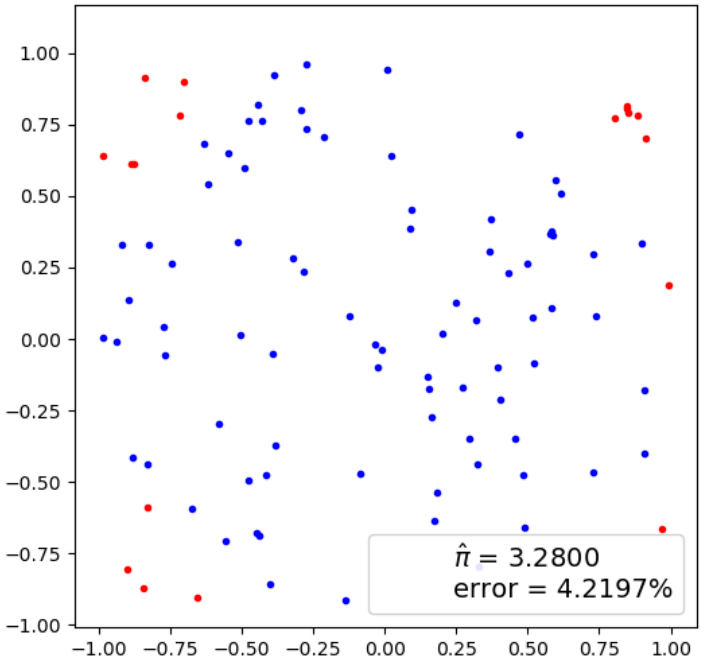


Fig. 5. Calculo de π con 100 números aleatorios

Aproximación de una integral. En esta parte para usar el algoritmo de Monte Carlo, se utilizó la Ecuación donde el usuario selecciona una función que deseé calcular la integral definida de ella, es decir, calcular el área bajo la curva de la función. Pero recordemos que esta solamente es una aproximación a dicha integral definida, ya que usa valores aleatorios para llegar al mejor resultado.
El entorno en el que fue desarrollado fue Angular. Para poderlo instalar, se

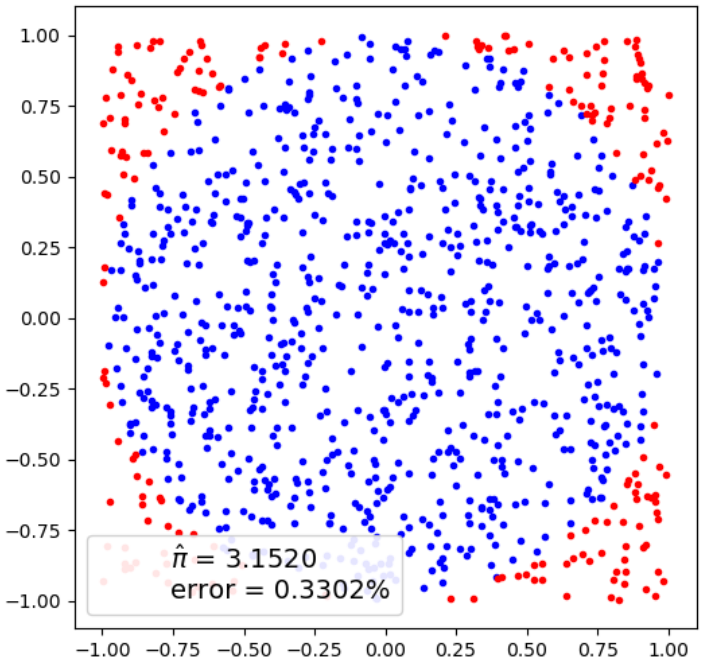


Fig. 6. Calculo de π con 1000 números aleatorios

necesita de otro framework de JavaScript llamado NodeJS, los pasos para la instalación de los recursos utilizados y la ejecución del programa para esta practica en especifico, son los siguientes:

1. Ir a la página oficial de NodeJS (https://nodejs.org/es/).
2. Descargar e instalar el archivo de ejecución (Versión LTS).
3. Abrir terminal e ingresar el siguiente comando:
   C:/> npm install -g @angular/cli.
4. Ingresar desde la terminal a la carpeta de “Monte Carlo” e ingresar el comando:
   C:/MonteCarlo/> npm install ng2-charts chart.js –save
5. Por último, ejecutamos el programa con el comando:
   C:/MonteCarlo/> ng serve –o

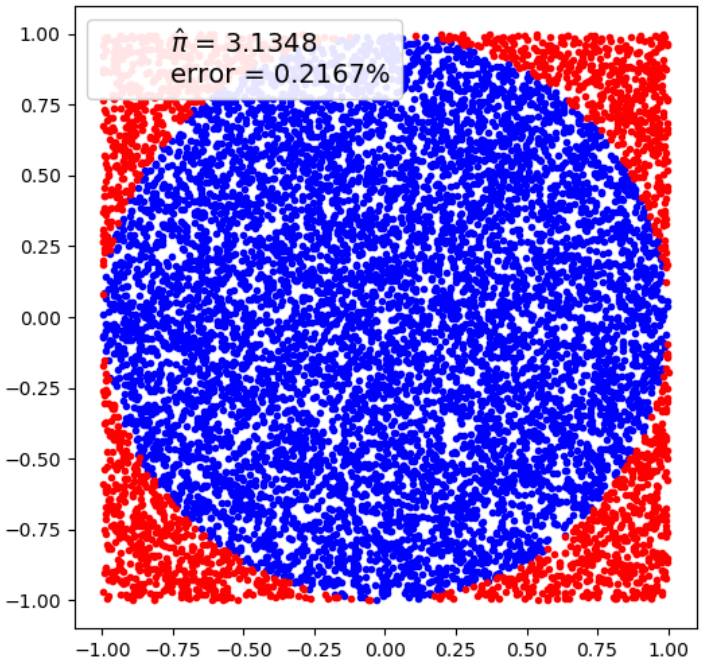


Fig. 7. Calculo de π con 10000 números aleatorios

La interfaz del usuario es de una forma grafica, esto para que se pueda apreciar como es que el algoritmo trata de aproximarse al área bajo la curva de la función que se desee, veamos un pequeño ejemplo.
Utilizaremos la ecuación (5), para elegir una función se desplegará una pequeña lista de dichas funciones que se pidieron desarrollar; una vez seleccionada la función a integrar, se deben definir los límites inferiores y superiores, donde a es el límite inferior y el límite superior será b. Para finalizar, se pondrá un valor N que representa los valores aleatorios que se generarán para la extracción del área bajo la curva, una vez dada toda la inforamción damos click en "Calcular integral aproximada".

Como se mencionó anteriormente, someteremos a prueba el algoritmo con la ecuación (5). Observemos la figura 12 a la figura 15, los limites inferior y superior se quedan fijos, para que realmente se vea el juego de la variable aleatoria. Mientras más pequeño es esa variable aleatoria, menor es la precisión de

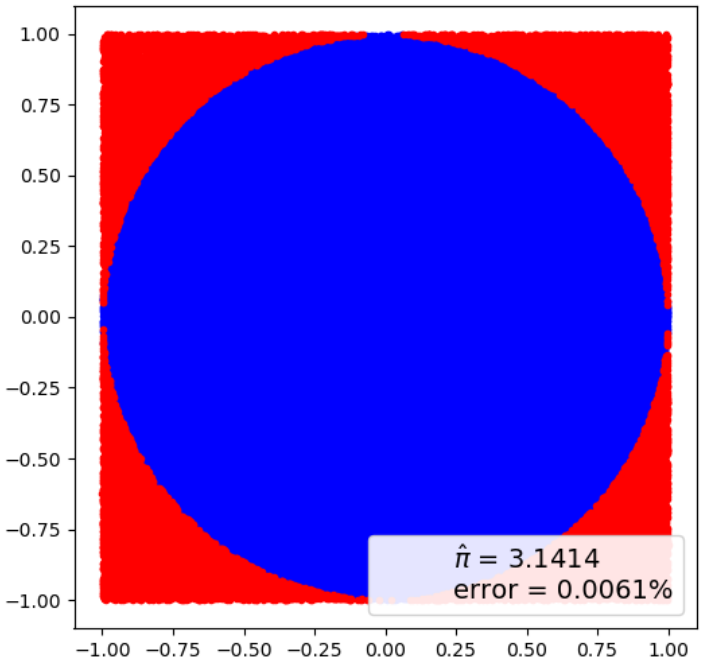


Fig. 8. Calculo de π con 100000 números aleatorios

tener una buen resultado de dicha área. De igual forma que con el calculo de π, mientras más muestras aleatorias generemos, nos acercaremos poco a poco a un valor parecido al del resultado de la integral definida. Sin dejar a un lado que si tenemos valores demasiados altos (Figura 15) puede que no sea la mejor opción, ya que nos puede perjudicar a tener un peor resultado como el anterior, y que puede que exista un N (Figura 14) que sea el mejor valor para obtener dicho resultado

2.2 Algoritmo Las Vegas

Quicksort Estático v.s. Quicksort Aleatorio. Este apartado, ponemos a prueba el algoritmo las vegas con el método de ordenamiento "Quicksort", que consiste en ir dividiendo un arreglo en varias partes hasta que se el la división del arreglo sea de uno, para poder ordenar el arreglo de manera ascendente. Pongamos a prueba la aleatoridad que nos aporta los algoritmos, tendremos un

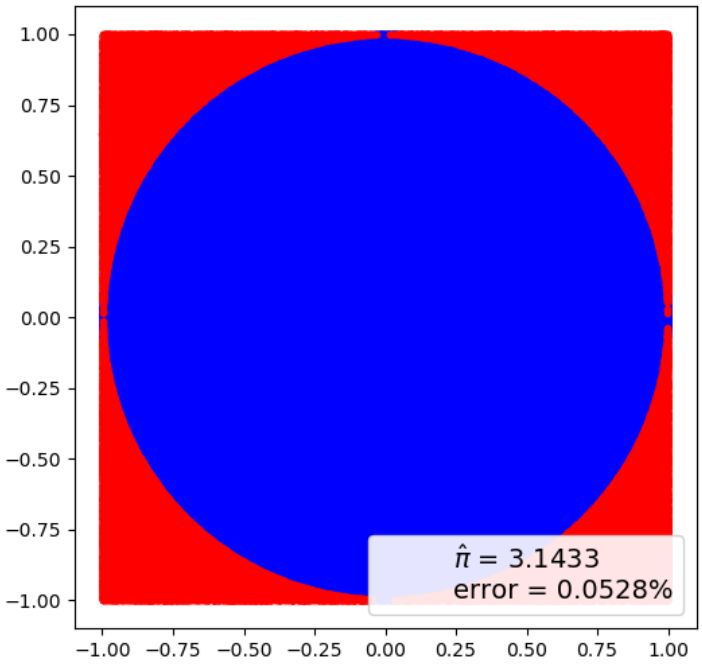


Fig. 9. Calculo de π con 1000000 números aleatorios

quicksort estaático, es decir, que el "pivote" que se necesita estará dado por el valor que se encuentra a la mitad de todo el arreglo, y otro quicksort aleatorio, que como podemos deducir, el pivote será cualquier posición dentro del arreglo. Como podemos observar en la Tabla 3, el tiempo en que tarda en ordenar los datos aleatorios son casi similares; en el caso de tener una cantidad mínima de datos a ordenas, ambos algoritmos compiten con el mismo tiempo hasta que empieza a aumentar la cantidad de valores en el arreglo.
Por otro lado, también observamos que un valor alto de N, el algoritmo aleatorio compite de nuevo con el algoritmo estático, esto se puede dar o no, ya que puede que al tener un pivote aleatorio, se puede encontrar un corte del arreglo mucho mejor, que solo tomar el de la mitad, y así, ordenarlo más rápido o no, todo depende del número aleatorio que tomará el pivote.

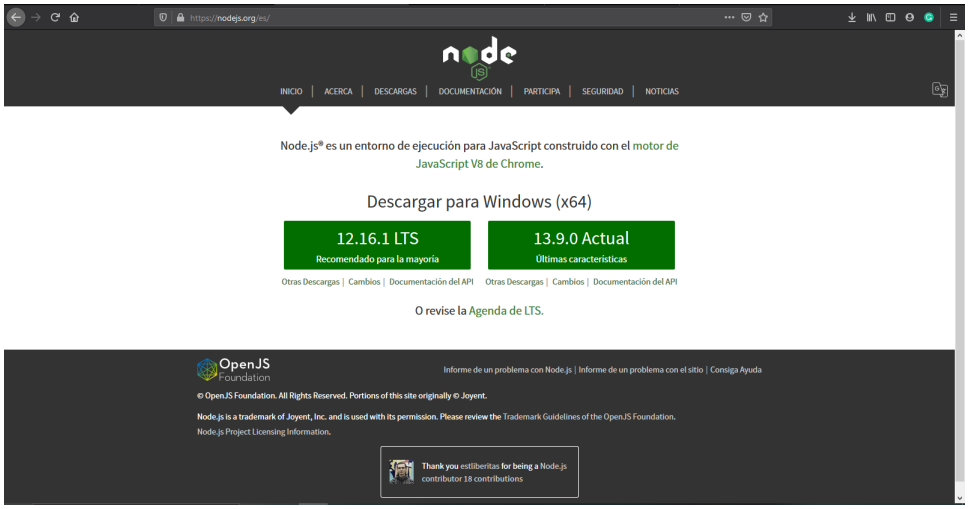


Fig. 10. Página oficial de NodeJS

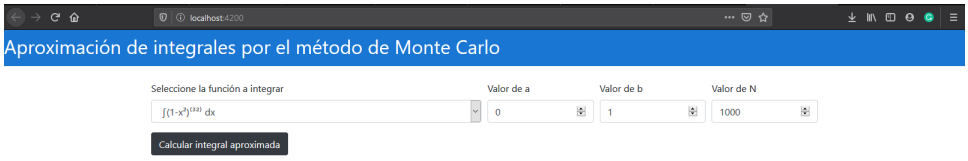


Fig. 11. Página principal del usuario

Valor de N	Quicksort estático π	Quicksort aleatorio
1000	0.005	0.005
2000	0.006	0.006
3000	0.005	0.009
4000	0.007	0.009
5000	0.007	0.009
6000	0.008	0.010
7000	0.008	0.011
8000	0.009	0.014
9000	0.010	0.010
10000	0.010	0.010

Table 3. Tiempo de demora de los algoritmos en segundos.

3 Conclusiones

Los algoritmos aleatorios nos pueden servir para solucionar distintos problemas, en este caso, sobre cálculos y ordenamiento de de un arreglo. Pero no todo

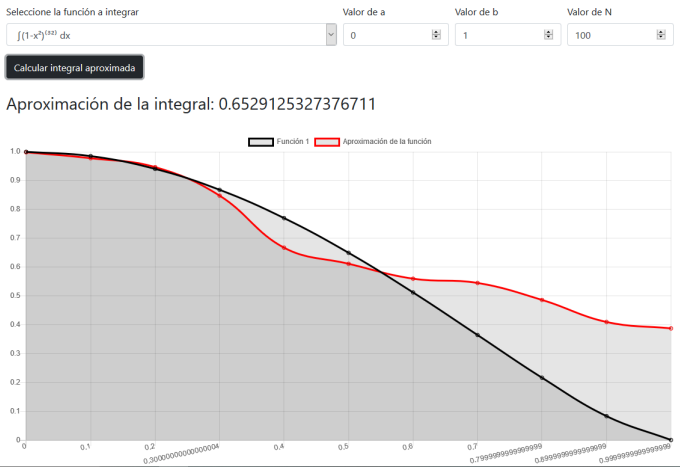


Fig. 12. Función de prueba con 100 números aleatorios

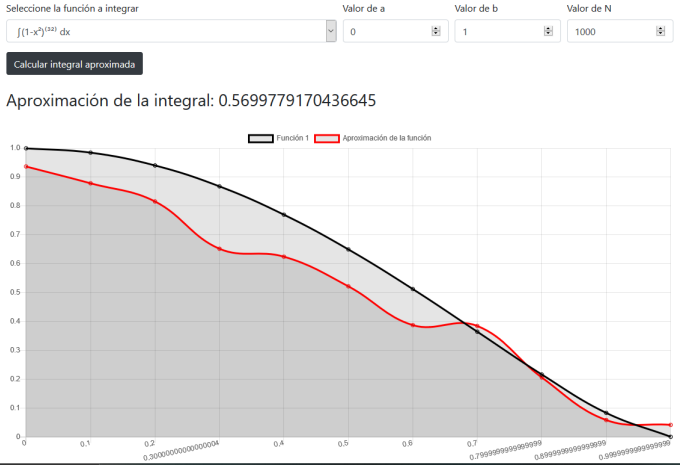


Fig. 13. Función de prueba con 1000 números aleatorios

siempre es tan efectivo, ya que debemos tener en cuenta la cantidad de valores aleatorios que se necesitan para resolver el problema, ya que si es una cantidad muy pequeña puede que no resuelva el problema, y si es una cantidad muy grande, este puede que no tenga los mejores resultados, por lo tanto, es necesario saber a que problema nos enfrentamos y tener en cuenta que en muchas ocasiones es cuestión de “suerte” para poder obtener el mejor resultado a nuestro problema.

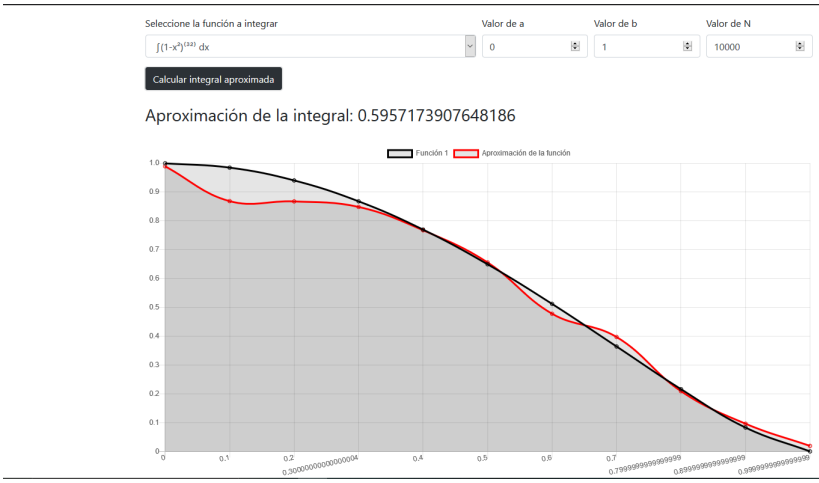


Fig. 14. Función de prueba con 10000 números aleatorios

Fig. 15. Función de prueba con 100000 números aleatorios

References

1. Deb K., M. K. A Review of Nadir Point Estimation Procedures Using Evolutionary Approaches: A Tale of Dimensionality Reduction. Tech. Rep. KanGAL Report 2008004, Indian Institute of Technology, 2008.
2. Marler, R., and Arora, J. Survey of Multi-objective Optimization Methods for Engineering. *Structural and Multidisciplinary Optimization 26*, 6 (2004), 369–395.
3. Miettinen, K. *Nonlinear Multiobjective Optimization*, vol. 12. International Series in Operations Research & Management Science, 1998.

4. Silvercorp. ¿qué es un algoritmo aleatorio?, https://silvercorp.wordpress.com/2013/08/20/que-es-un-algoritmo-aleatorio/.
5. Zhang, Q., and Li, H. MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. *IEEE Transactions on Evolutionary Computation 11*, 6 (December 2007), 712–731.