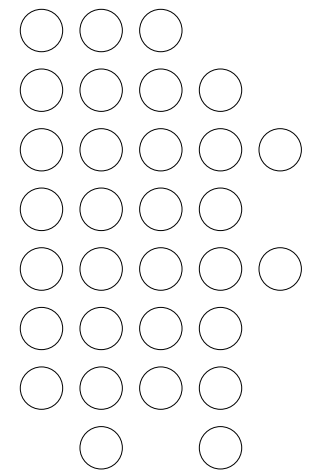


Algoritmos probabilistas

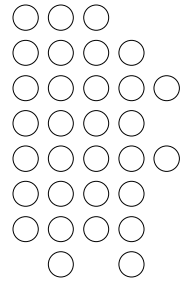
Diseño y Análisis de Algoritmos
Cátedra de Programación
Carrera de Ingeniería de Sistemas

Br. Renny J. Márquez C.

Mérida, 2 de Junio de 2005

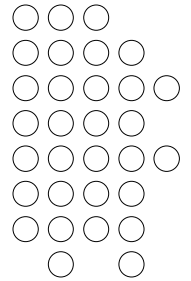


Todos los algoritmos no son deterministas...



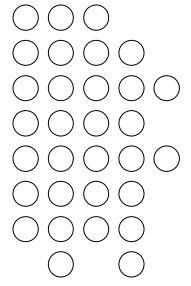
- Hay un tesoro escondido en un lugar descrito por un mapa que no se puede descifrar.
- Se ha logrado limitar la búsqueda del tesoro a dos lugares muy distantes entre sí.
- En cuanto se llega a uno de esos dos lugares se sabría de inmediato si es o no el correcto.
- Hacen falta 5 días para llegar a cualquiera de los dos o para ir de uno a otro.
- Hay un dragón que se acerca al tesoro cada noche y se lleva una parte del mismo.
- Se estiman 4 días para terminar de descifrar el mapa con un computador que no puede llevar consigo en el viaje.
- Un elfo se ofrece a descifrar el mapa a cambio del botín del dragón de 3 noches.
- ¿Conviene aceptar la oferta del elfo?

Todos los algoritmos no son deterministas...



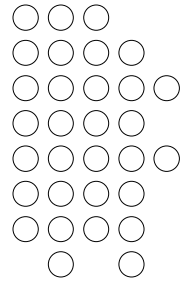
- Hay un tesoro escondido en un lugar descrito por un mapa que no se puede descifrar.
- Se ha logrado limitar la búsqueda del tesoro a **dos lugares** muy distantes entre sí.
- En cuanto se llega a uno de esos dos lugares se sabría de inmediato si es o no el correcto.
- **Hacen falta 5 días para llegar a cualquiera** de los dos o para ir de uno a otro.
- Hay un dragón que se acerca al tesoro cada noche y **se lleva 1 parte** del mismo.
- Se estiman **4 días para terminar de descifrar el mapa** con un computador que no puede llevar consigo en el viaje.
- Un elfo se ofrece a descifrar el mapa a cambio **del botín del dragón de 3 noches**.
- ¿Conviene aceptar la oferta del elfo?

Todos los algoritmos no son deterministas...



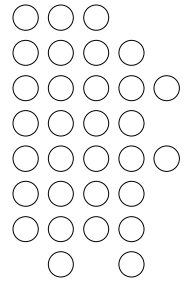
- x , valor del tesoro que queda hoy.
- y , botín de cada noche del dragón.
- Además $x > 9y$
- Se necesitan 5 días para llegar y 4 para descifrar el mapa, se conseguiría $x - 9y$
- Si se acepta la oferta del elfo $x - 8y$ (de $5+3$)
- Si se lanza una moneda para decidir a cual lugar dirigirse y se viaja al otro si se ha cometido un error:
 - 1 oportunidad entre 2 de que la pérdida sea $5y$
 - 1 oportunidad entre 2 de que la pérdida sea $10y$
- El beneficio esperado es de $x - 7.5y$

Todos los algoritmos no son deterministas...



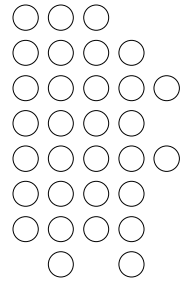
- Cuando en un problema hay que tomar una decisión, puede ser mejor tomarla de forma aleatoria en vez de calcular cuál de las alternativas es mejor.
- Esto se puede hacer cuando:
 - El tiempo de cálculo de la decisión óptima es mucho mayor que el requerido en el caso medio al tomar la decisión en forma aleatoria.
 - Problemas con múltiples soluciones correctas.

Todos los algoritmos no son deterministas...



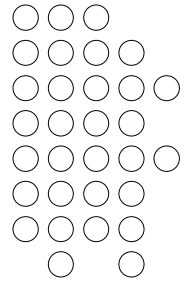
- Un algoritmo probabilista puede comportarse de manera distinta al aplicarlo dos veces a un mismo caso.
 - Tanto el tiempo de ejecución como el resultado pueden variar.
- A un algoritmo determinista no se le permite que calcule una solución incorrecta para ningún dato de entrada.
- Un algoritmo probabilista puede equivocarse siempre que esto ocurra con una probabilidad razonablemente pequeña para cada dato de entrada.
 - Repitiendo la ejecución un número suficiente de veces para el mismo dato, puede aumentarse tanto como se quiera el grado de confianza en obtener la solución correcta.

Todos los algoritmos no son deterministas...



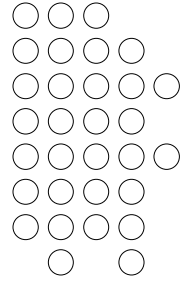
- El análisis de la eficiencia de un algoritmo determinista a veces es complicado.
- El análisis de los algoritmos probabilistas es **normalmente** muy complicado.
- Puede darse el caso en el que una respuesta incierta dada por un algoritmo probabilista no sólo se obtenga más deprisa, sino que además podamos confiar más en ella que en cualquiera que otorgue un algoritmo determinista.

Tiempo esperado frente a tiempo promedio



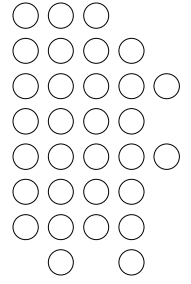
- El tiempo promedio de un algoritmo determinista, se define como el tiempo promedio requerido por el algoritmo cuando se consideran igualmente probables todos los ejemplares posibles de un tamaño dado.
- El tiempo esperado de un algoritmo probabilista se refiere al tiempo promedio que se necesitaría para resolver un mismo ejemplar una y otra vez.
 - El tiempo esperado en el caso peor de un algoritmo probabilista se refiere al tiempo esperado que requiere el peor caso posible de un tamaño dado, y **no** al tiempo requerido si se tomaran las peores opciones probabilistas.

Generación de números pseudoaleatorios



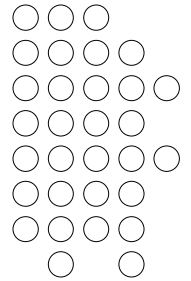
- Una serie de números es aleatoria cuando no se descubre en ella ninguna propiedad: de crecimiento, repetición, etc.
- Propiedades que deben cumplir las series de números aleatorios:
 - Han de estar uniformemente distribuidos.
 - Han de ser estadísticamente independientes.
 - Han de ser reproducibles.
 - Que requieran poco espacio en memoria.
 - Que se obtengan rápidamente.

Generación de números pseudoaleatorios



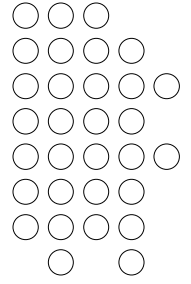
- Sean a y b dos números reales tales que $a < b$.
- Una llamada a $uniforme(a,b)$ devuelve un número real x seleccionado aleatoriamente en el intervalo $a \leq x < b$.
- La distribución de x es uniforme en el intervalo, y las sucesivas llamadas al generador son independientes de las anteriores.

Generación de números pseudoaleatorios



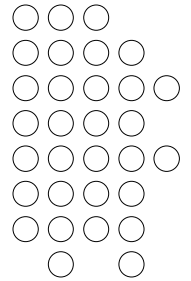
02-06-05		
uniforme (Real a, Real b): Real		
{pre: a < b, existe la función uniforme(0,1) que genera un número aleatorio entre 0 y 1}		{pos: a ≤ x ≤ b}
1	x = a + (b-a) * uniforme(0,1)	-x: Real: almacena un número aleatorio entre a y b.
2	regrese x	
1	a = 2, b = 3 => x = 2,3	
2	a = 2, b = 3 => x = 2,8	

Generación de números pseudoaleatorios



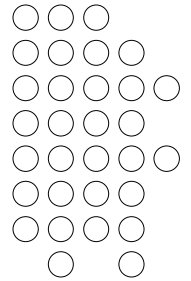
- Para generar enteros aleatorios se extiende la notación a *uniforme(i..j)*.
- i y j pertenecen al conjunto de los enteros, siendo $i \leq j$.
- La llamada devuelve un entero k seleccionado aleatoriamente, uniformemente e independientemente dentro del intervalo $i \leq k \leq j$.

Generación de números pseudoaleatorios



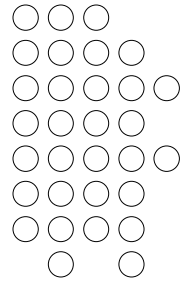
02-06-05 uniforme (Entero i .. Entero j): Entero {pre: $i \leq j$, existe la función uniforme(0,1) que genera un número aleatorio entre 0 y 1} {pos: $i \leq k \leq j$}		
1 2	$k = \lfloor \text{uniforme}(i, j+1) \rfloor$ regrese k	-k: Entero: almacena un número aleatorio entre i y j.
1 2	$i = 2, j = 5 \Rightarrow k = 3$ $i = 2, j = 3 \Rightarrow k = 3$	

Generación de números pseudoaleatorios



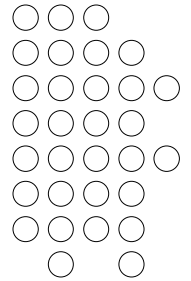
02-06-05		
tiramoneda (): Cadena		
{pre: existe la función uniforme (i..j) que genera un número aleatorio entero}		{pos: m = cara o m= sello}
1	m = si (uniforme(0..1) = 0) entonces cara sino cruz fsi	-x: Real: almacena un número aleatorio entre a y b.
2	regrese m	
1	m = cara	
2	m = sello	

Generación de números pseudoaleatorios



- Debido a que los generadores verdaderamente aleatorios no suelen estar disponibles en la práctica, en la mayor parte de las ocasiones se emplean generadores pseudoaleatorios.
- Generalmente cada lenguaje tiene un generador de números pseudoaleatorios entre 0 y 1, partiendo de una semilla determinada por el lenguaje (generalmente dada por la fecha y hora) o introducida por el programador.

Generación de números pseudoaleatorios



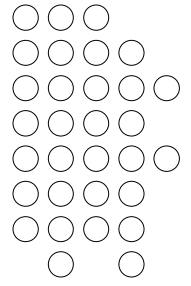
- Los métodos de generación de números pseudoaleatorios más comunes son los llamados *métodos congruenciales*.
- El más famoso es el de Lehmer que depende de tres datos:

$$X_{n+1} = (a * X_n + c) \text{ mod } m$$

Tipos:

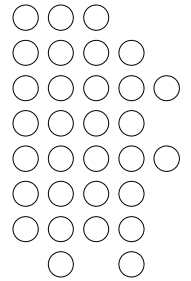
- Generador congruencial mixto: aquel en el que $c \neq 0$
- Generador congruencial multiplicativo: aquel en el que $c = 0$
- Los multiplicativos son más rápidos, aunque tienen una menor longitud de ciclo.
- Propiedades:
 - Fácilmente reproducible (comenzando por la misma semilla).
 - Se obtiene rápidamente.
 - Ocupa poco espacio de memoria.

Generación de números pseudoaleatorios



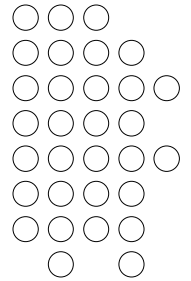
- **Teorema:** La sucesión congruencial definida por X_0 , a , c y m es de período máximo si y sólo si:
 - C es primo relativo a m (su máximo común divisor es 1).
 - $a-1$ es múltiplo de p , para todo p primo que divida a m .
 - $a-1$ es múltiplo de 4 si m es múltiplo de 4.
- Problema del generador de Lehmer: produce overflow con algunos valores.
 - Solución: cambiar la expresión anterior por:
 - $X_{n+1} = a(X_n \bmod Q) - R (X_n \text{ div } Q) + f(m)$, donde:
 - $Q = m \text{ div } a$,
 - $R = m \bmod a$,
 - $f(m) = 0$ si la suma de los dos primeros sumandos de la fórmula es mayor o igual que 0 y $f(m) = m$ en caso contrario.

Clasificación de los algoritmos probabilistas



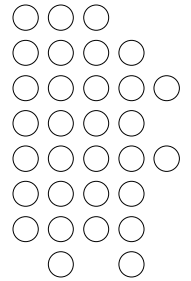
- **Numéricos:** Solución aproximada a problemas numéricos para los que es imposible o muy costoso dar una respuesta exacta.
- **Monte Carlo:** Dan una respuesta correcta con una probabilidad elevada, aunque a veces proporcionan una respuesta incorrecta.
- **Las Vegas:** Su respuesta es siempre correcta pero puede no encontrarla.

Clasificación de los algoritmos probabilistas



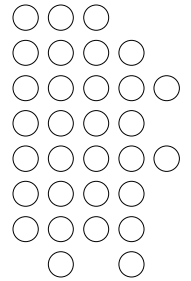
- Los algoritmos numéricos:
 - Otorgan una solución aproximada.
 - Otorgan un intervalo de confianza.
 - A mayor tiempo de ejecución, mejor es la aproximación.
- Los algoritmos de Monte Carlo:
 - Otorgan la respuesta exacta con elevada probabilidad.
 - En algunas ocasiones arrojan una respuesta incorrecta.
 - No se puede saber si la respuesta es correcta, pero se puede reducir la probabilidad del error dando más tiempo al algoritmo.
- Los algoritmos de Las Vegas:
 - Toman decisiones al azar.
 - Si no encuentran la solución correcta lo admiten.
 - Es posible volver a intentarlo con los mismos datos hasta obtener la solución correcta.

Ejemplos



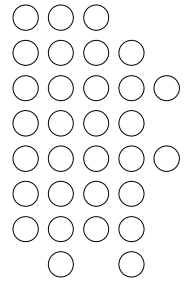
- Si preguntáramos ¿Cuándo descubrió América Cristóbal Colón?
Las respuestas según el algoritmo serían:
- Algoritmo numérico:
 - Entre 1490 y 1500
 - Entre 1485 y 1495
 - Entre 1492 y 1592
 - **Entre 1481 y 1491**
 - Entre 1489 y 1499
- Algoritmo de Monte Carlo:
 - 1492, 1492, 1492, **1491**, 1492, 1492, **357 a.C.**, 1492, 1492, 1492.
- Algoritmo de Las Vegas:
 - 1492, 1492, **¡Perdón!**, 1492, 1492, 1492, 1492, 1492, **¡Perdón!**, 1492

Algoritmos Probabilistas Numéricos

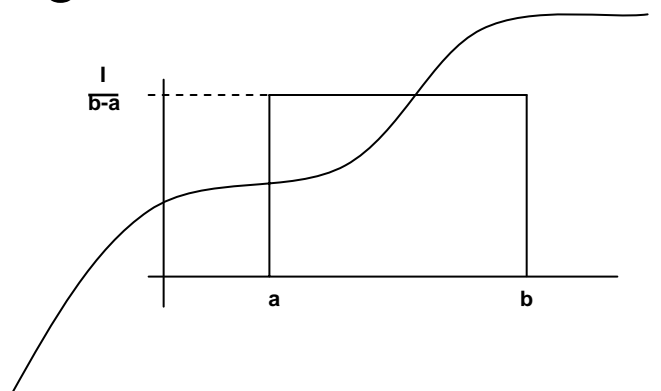


- La aleatoriedad se utilizó por primera vez en algoritmia para resolver aproximadamente problemas numéricos.
- En algunos problemas de la vida real el cálculo de la solución exacta no es posible (incertidumbre de los datos, limitaciones de los ordenadores digitales).
- En otros existe una respuesta precisa pero se tardaría demasiado tiempo en obtenerla.
- La respuesta de un algoritmo probabilista numérico es siempre aproximada, pero su precisión esperada mejora a medida que crece el tiempo disponible para el algoritmo.
- El error suele ser inversamente proporcional a la raíz cuadrada de la cantidad de tiempo dedicado (se necesita cien veces más de trabajo para obtener una cifra más de precisión).

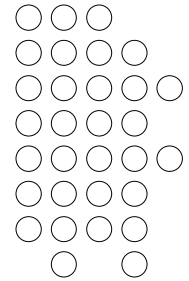
Integración Numérica



- Un algoritmo numérico muy conocido es la integración de Monte Carlo, aunque no es un algoritmo de Monte Carlo.
- Calcular: $I = \int f(x) dx$ donde $f : \mathbb{R} \rightarrow \mathbb{R}^{\geq 0}$ es continua y $a \leq b$
- Consideremos el rectángulo de la figura de anchura $b-a$ y altura $I/(b-a)$ (la altura media de f entre a y b)
- El área del rectángulo es también I

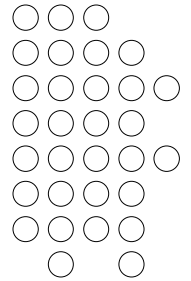


Integración Numérica



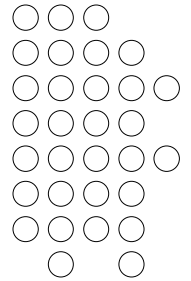
02-06-05 intMC (función f, Entero n, Real a, Real b): Real {pre: a < b} {pos: I es el valor de la integral}		
1 2 3	suma = 0 (x = uniforme(a,b) suma = suma + f(x))j = 1, n regrese I = (b-a) * (suma/n)	-suma: Real: almacena sumas parciales de la función evaluada en diferentes puntos. -x: Real: almacena el valor de los puntos de integración. -n: Entero: número de intervalos de integración. -I: Real: Valor de la integral aproximado. -j: Entero: variable auxiliar para la estructura de repetición.
	f(x) = x + seno(x), n = 100, a = 10, b = 100 => I = 5083,73 f(x) = x + seno(x), n = 100, a = 10, b = 100 => I = 4764,47 En 100 iteraciones: I = 4978,65	

Integración Numérica



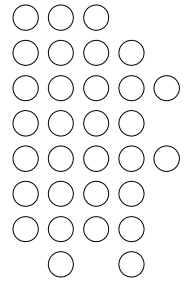
- El algoritmo determinista necesita menos iteraciones que el probabilista para obtener un grado de precisión comparable.
- Sin embargo, todo algoritmo determinista de integración presenta funciones continuas que pueden engañar al algoritmo.
 - Ejemplo: $f(x) = \sin^2((100!)\pi x)$ con parámetros **IntDET(f,n,0,1)**
 - Esto no ocurre con el algoritmo probabilista aunque hay una probabilidad muy pequeña de que el algoritmo cometa un error similar cuando la función sea completamente normal.
- El uso de los algoritmos probabilistas para integrar se justifica con las integrales múltiples.
- En un algoritmo determinista en ese caso el número de puntos de muestra necesario para lograr una precisión dada, crece de manera exponencial con la dimensión de la integral.
 - Ejemplo: 100 puntos en una integral simple, se convertirían en 100 x 100 puntos para alcanzar la misma precisión en una integral doble, 1000 x 1000 en una triple, ...

Integración Numérica



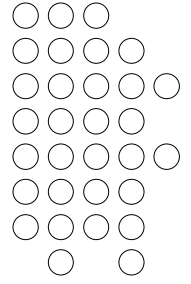
- En el algoritmo probabilista la dimensión de la integral no tiene mucho efecto sobre la precisión de la integral que se obtiene.
- La cantidad de trabajo en cada iteración puede aumentar ligeramente con la dimensión.
- En la práctica, el algoritmo probabilista se utiliza en integrales de 4 o más dimensiones, ya que no hay otra técnica sencilla que sea competitiva.
- Existen técnicas híbridas (parcialmente sistemáticas y parcialmente probabilistas).

Conteo Probabilista



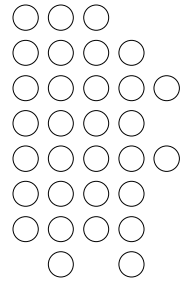
- Empleando la notación binaria ordinaria, se puede contar desde 0 hasta $2^n - 1$.
- Podríamos ir más allá si estamos dispuestos a descartar algunos valores.
 - Por ejemplo, si quisiéramos contar solamente los pares.
- Sea c un registro de n bits, podríamos implementar tres funciones:
 - *iniciar(c)*: coloca el contador en cero.
 - *pulsar(c)*: le suma 1.
 - *contar(c)*: pide su valor actual.
- Dado que c sólo puede admitir 2^n valores distintos, el contador se ve obligado a tomar un valor que ya haya tenido anteriormente al cabo de un máximo de 2^n llamadas a *pulsar(c)*.

Conteo Probabilista



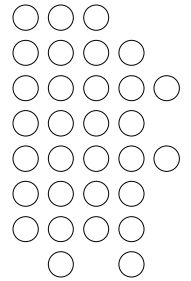
- Si relajamos cualquier requisito que le exija a *contar* regresar el número exacto de llamadas a *pulsar* desde el último *iniciar*, con una estrategia probabilista podríamos contar el doble.
 - Cada vez que se llame a *pulsar*, lanzamos al aire una moneda imparcial. Si sale *cara*, sumamos 1 al registro; si sale cruz, no hacemos nada.
- De esta forma, se puede llegar a contar $2^{n+1} - 2$ veces con un registro de n bits.

Conteo Probabilista



- La estrategia de conteo probabilista puede llegar exponencialmente más lejos: desde 0 hasta $2^{2^n} - 1$.
- Bastan sólo 8 bits para contar más de $5 \cdot 10^{76}$ sucesos.
- La idea consiste en mantener en el registro una estimación del logaritmo del número de llamadas a *pulsar*, lo que implica que *contar(c)* devuelve $2^c - 1$.
- Supongamos que $2^c - 1$ es una buena aproximación del número de llamadas a *pulsar*, desde la última inicialización. Luego decimos que el número de llamadas a *pulsar* pasa a ser $2^{c+1} - 1$ con una probabilidad p y permanece igual con una probabilidad $1 - p$; siendo el valor esperado $2^c + 2^c p - 1$.

Conteo Probabilista

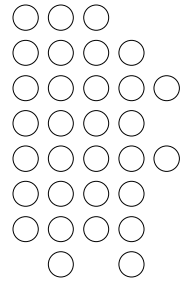


iniciar(c): $c = 0$

02-06-05		
pulsar (registro c)		
{pre: existe el registro c}		
{pos: v es el conteo del registro c }		
1	(si (tiramoneda = cara) entonces $c = c + 1$) $i = 1, n$	-i: Entero: variable auxiliar para la estructura de repetición. -n: Entero: número de bits del registro c.

contar(c): regresa $2^c - 1$

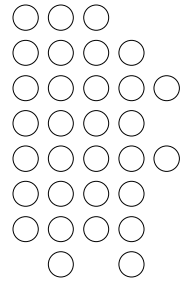
La aguja de Buffon



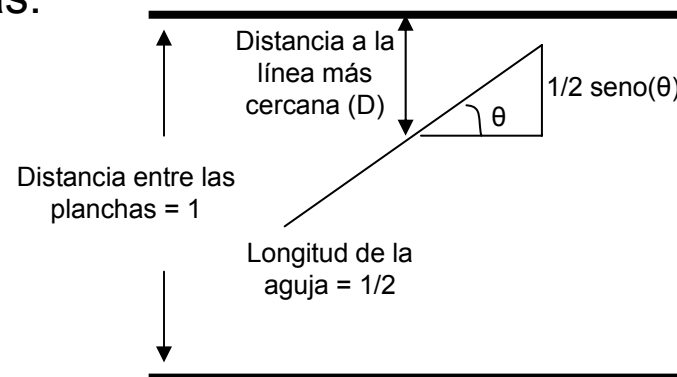
- En el siglo XVIII, George Louis Leclerc, conde de Buffon, demostró que si se arroja aleatoriamente una aguja sobre un suelo hecho con planchas de anchura constante, si la aguja tiene una longitud que sea exactamente la mitad de la anchura de las planchas y si la anchura de la rendija existente entre las planchas es cero, entonces la probabilidad de que una aguja caiga encima de una rendija es $1/\pi$ (0,3183).



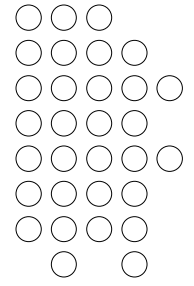
La aguja de Buffon



- Si hacemos la longitud de la aguja y la anchura entre las planchas igual a 1, y conociendo que la aguja forma un ángulo θ con las planchas y existe una distancia D a la rendija más cercana a la aguja, se observa que θ puede variar entre 0 y 180 grados y que la distancia desde el centro de la aguja a la rendija más cercana nunca puede ser mayor que la mitad de la distancia entre las rendijas.

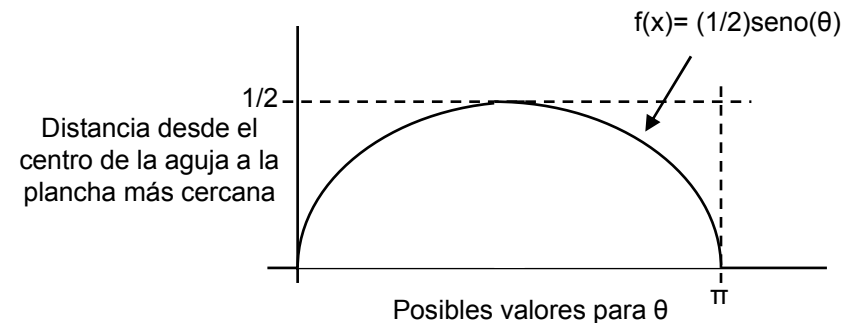


- La aguja tocará una rendija si la distancia más cercana a una de estas (D) es menor o igual que el seno de θ . Esto es, $D \leq 1/2 \text{ seno}(\theta)$.



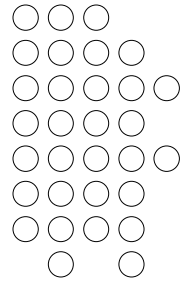
La aguja de Buffon

- En este gráfico se observa la integral definida de $1/2\text{seno}(\theta)$ evaluada entre 0 y π representada por la porción sombreada. El resultado es que esta porción tiene el valor de 1.



- El valor del área del rectángulo completo es $\pi/2$. De esta forma se tiene que la probabilidad de que una aguja toque una rendija es $2/\pi$, que es aproximadamente 0,6366197.
- De forma general se tiene que la probabilidad de que una aguja toque una rendija es $2*\lambda/(\omega*\pi)$, donde λ representa la longitud de las agujas y ω representa la anchura entre las planchas.

La aguja de Buffon

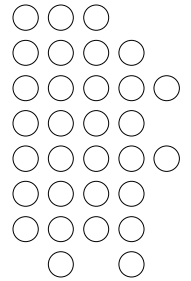


- El experimento de la aguja de Buffon puede ser utilizado para:
 - Hallar un estimado de la cantidad de agujas que caerán en las rendijas. Esto se consigue, sabiendo que si las agujas tienen una longitud igual a la mitad de la anchura entre las planchas, la probabilidad de que una aguja caiga en alguna rendija es de $1/\pi$ y la esperanza para n agujas es de $k = n/\pi$.
 - Hallar el valor de π . Si conocemos el número de agujas que se dejan caer y el número de aciertos en las rendijas, esta razón nos arrojará un estimado de π .
 - Hallar la anchura entre las planchas. Conociendo n , k , π y λ , se puede hallar la anchura de las planchas: $\omega = (2\lambda n)/(k\pi)$

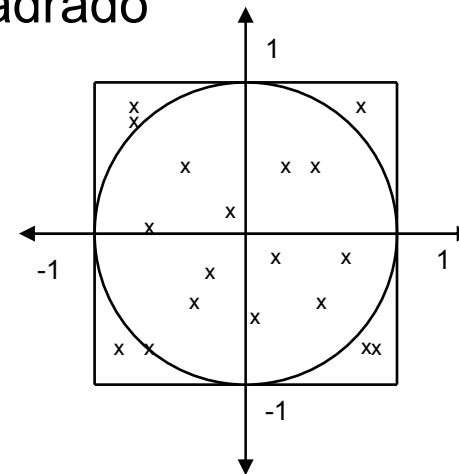
Simulación:

<http://www.angelfire.com/wa/hurben/code5.html>

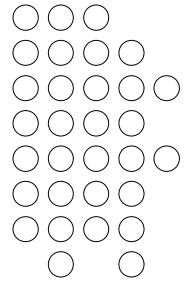
Otra aproximación de π



- Se dispone de una diana de radio unitario centrada en el origen de coordenadas y del cuadrado en el que se inscribe.
- Si se efectúa un buen número de lanzamientos de dardos, uniformemente distribuidos en el cuadrado $[-1, 1] \times [-1, 1]$, el número de los que caerán en la diana será aproximadamente proporcional a su superficie
 - $$\frac{\text{No. de disparos dentro}}{\text{No. de disparos total}} \approx \frac{\text{superficie diana}}{\text{superficie cuadrado}}$$
- Como:
 - $$\frac{\text{superficie diana}}{\text{superficie cuadrado}} = \pi/4$$
- Se puede estimar:
 - $$\pi = 4 * \frac{\text{No. de disparos dentro}}{\text{No. de disparos total}}$$

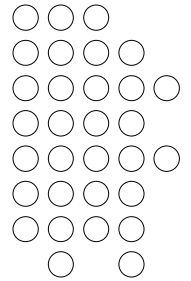


Otra aproximación de π



02-06-05		dardos (Entero n): Real
{pre: $n \geq 0$}		{pos: regresa una aproximación a π}
1	k = 0	-k: Entero: contador que indica los dardos que caen en el círculo. -x: Real: coordenada x para el dardo actual. -y: Real: coordenada y para el dardo actual. -n: Entero: número de dardos arrojados. -pi: Real: almacena el valor estimado de π
2	(x = uniforme(0,1) y = uniforme(0,1) si ($x^2 + y^2 \leq 1$) entonces k = k + 1 fsi)i = 1, n	
3	pi = 4*k/n	
4	regreso pi	
1	n = 1000 => k = 810, pi = 3,24	

Referencias



- Brassard. Fundamentos de algoritmia.
- <http://www.angelfire.com/wa/hurben/code5.html>
- <http://www.mste.uiuc.edu/reese/buffon/bufjava.html>