Universidad de Alcalá

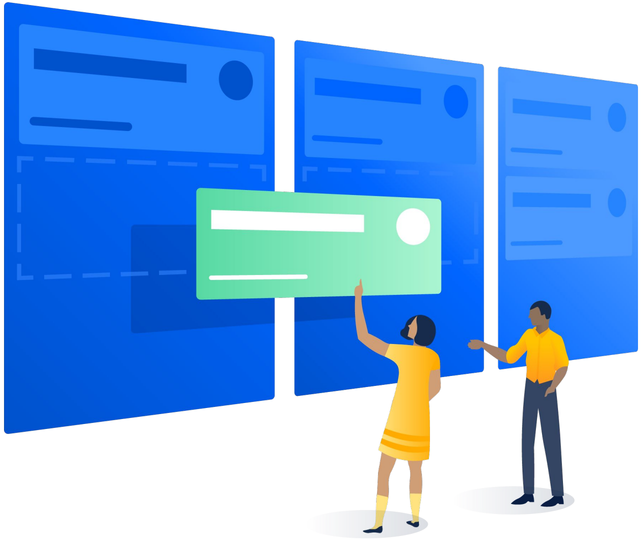
Algoritmos no deterministas

Tema 6

javier@junquera.io (UAH 2023)

Agenda

- Introducción
- Algoritmos no deterministas
- Algoritmo Monte Carlo
- Algoritmo Las Vegas



Introducciones

Existen problemas que, o bien no tienen una solución algorítmica, o esta es demasiado costosa.

Sin embargo, algunos de ellos pueden resolverse mediante una aproximación probabilística

¿Cómo están las manzanas?

Evaluar n elementos, a veces, símplemente no es factible

javier@junquera.io (UAH 2023)

¿Cómo están las manzanas?

Mediante una muestra, se pueden tomar decisiones, por ejemplo, de calidad

¿Cómo están las manzanas?

Aunque estos métodos no siempre darán el mismo resultado:
hay que saber elegir métricas

El equipo paracaidista (1/2)

La probabilidad de que un paracaidista, lanzándose "a lo loco", caiga en un círculo de área A es del 65%.

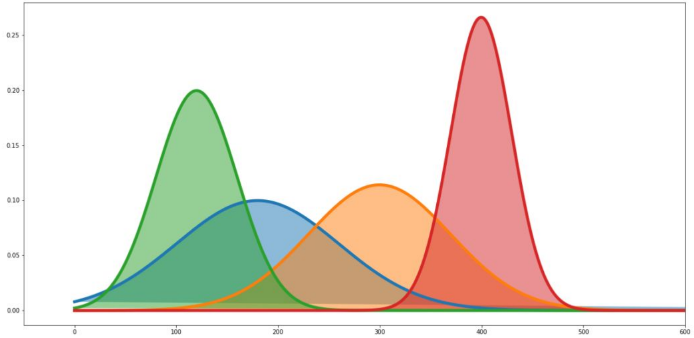
Si podemos dedicarle 4 minutos a pensarlo, podemos lograr un lanzamiento determinista, y conseguir que el paracaidista caiga dentro; pero si tardamos más de T segundos, nos descubrirán y todo habrá acabado. Por otro lado, tardamos 1 minuto en tirar un paracaidista a lo loco. ¿Qué es mejor?

Probabilidades

Sucesos independientes

Probabilidad de A después de B:

- `P(A|B) = P(A)`



Sucesos condicionados

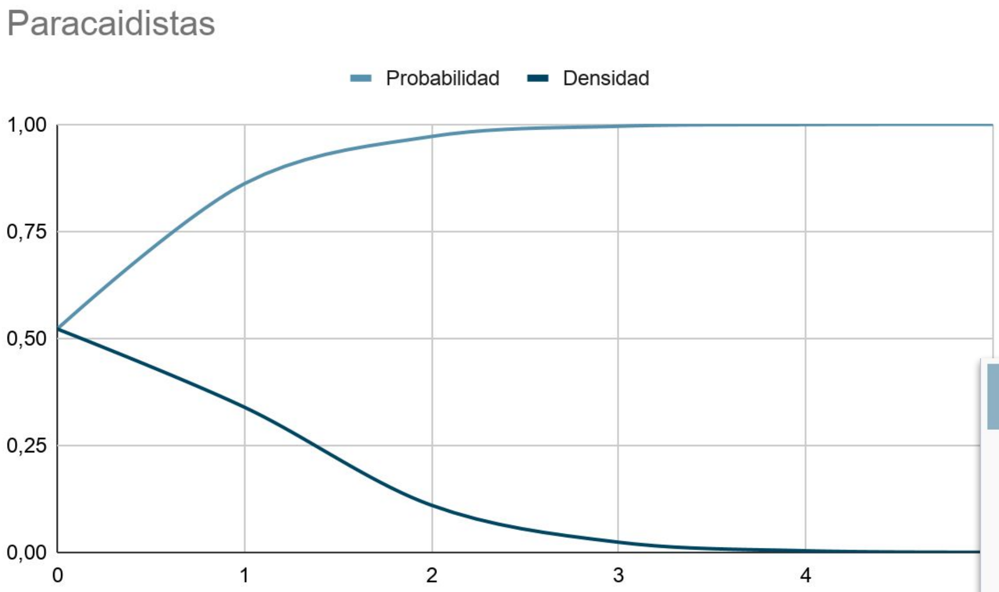
- Probabilidad de A después de B, según Bayes:

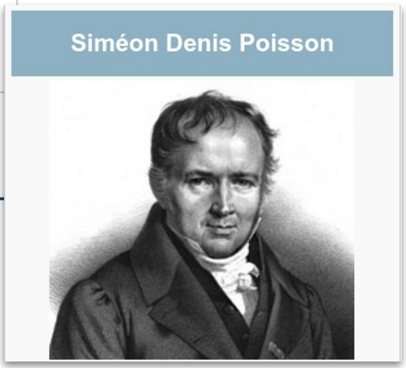
  `P(A|B) = P(B|A) * P(A)/P(B)`

- Funciones de distribución (Bernoulli, Gauss, Poisson, etc.)

javier@junquera.io (UAH 2023)

El equipo paracaidista (2/2)





$$\lambda = P(A) = 0,65$$

Algoritmos no deterministas

Universidad de Alcalá

Ante determinados problemas, tendremos que trabajar con este tipo de distribuciones de probabilidad. En cualquier caso, que los algoritmos no sean deterministas no quiere decir que los resultados sean aleatorios, pero sí hay que tener en cuenta que partiendo de los mismos datos:

- Cada ejecución tendrá una duración diferente
- No siempre obtendremos la misma solución
- La solución que obtengamos no siempre será correcta

Algoritmos no deterministas

Algunas familias de estos algoritmos

- Algoritmos numéricos (e.g., aproximación integral)
- Algoritmos tipo Las Vegas
- Algoritmos tipo Montecarlo
- Algoritmos de Sherwood (e.g., reducción de casos de Quick Sort)

javier@junquera.io (UAH 2023)

Test de primalidad (directo)


```python
# Comprobación de primos en O(sqrt(n))
def is_prime(n):
    umbral = int(math.sqrt(n))

    for i in range(umbral + 1)[2:]:
        if n % i == 0:
            return False

    return True
```

Test de primalidad (Euler)

Universidad de Alcalá

El pequeño teorema de Fermat enuncia que, si **n** es primo, para cualquier **a**:

- $n \equiv 1 \pmod 2$
- $\gcd(a, n) = 1$
- $a^{(n-1)} \equiv 1 \pmod n \Leftrightarrow a^n \equiv a \pmod n$

javier@junquera.io (UAH 2023)

Criterio de Euler

Universidad de Alcalá

Euler propone un test de primalidad increíblemente rápido, basándose en el pequeño teorema de Fermat. Formula que un número **n** es primo, en base **a**, si:

- $n = 1 \pmod{2}$
- $\gcd(a, n) = 1$
- $a^{(n-1)/2} = 1 \pmod{n} \;||\; a^{(n-1)/2} = n-1 \pmod{n}$

Test de primalidad (Euler)

Universidad de Alcalá

n \ a	2	3	4	5
2	-	T	T	T
3	T	-	T	T
4	F	F	F	F
5	T	T	-	T
6	F	F	F	F
7	T	T	T	T

¡ El test de Euler tiene un coste ~O(1) !

javier@junquera.io (UAH 2023)

Test de primalidad (Euler)

Universidad de Alcalá

n \ a	2	3	4	5
121	F	T	F	F
217	F	F	F	T
341	T	F	T	F
561	T	F	T	F
781	F	F	F	T

Pero hay algunos números que no son primos y…

javier@junquera.io (UAH 2023)

Test de primalidad (Miller-Rabin)

```python
# A grandes rasgos...
def is_prime(n, k = 5):

    for _ in range(k):
        a = random.randint(2, n)
        if not euler_prime(n, a):
            return False

    return True
```

Algoritmos Montecarlo

Miller-Rabin es un algoritmo tipo Montecarlo, como:

- Integración numérica
- La aguja de Buffon
- Cálculo de PI (lo veremos en un rato)

Da respuesta correcta con probabilidad mayor que **p** (definida)

(1-p) veces dará respuesta errónea

Función mayoritario

Universidad de Alcalá

Detectar si un elemento es mayoritario en un array

- El valor true de la función nunca será erróneo
    - Sólo devolverá verdadero si encuentra el valor mayoritario (i.e., nunca dará un falso positivo)
- Sin embargo, a veces puede devolver false, y que no sea así

La mejor aproximación para lidiar con esto es repetir la operación varias veces

Algoritmos Montecarlo

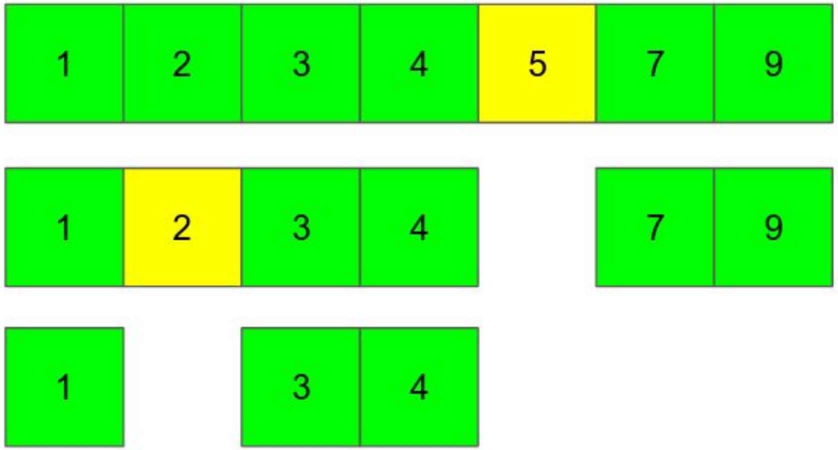
Universidad de Alcalá

```
# p --> Probabilidad de acierto
def montecarlo(x, eficacia=0.8):
    pfinal = (1 - p)
    s = soluciona(x)
    while eficacia < (1-pfinal):
        if solucion_fiable(s):
            return s
        pfinal *= pfinal
        s = soluciona(x)
    return s
```

Algoritmos Las Vegas

Recordemos...

- Elección de pivote en QuickSort
- Elección de umbral en ramificación y poda (soporte a minimax)

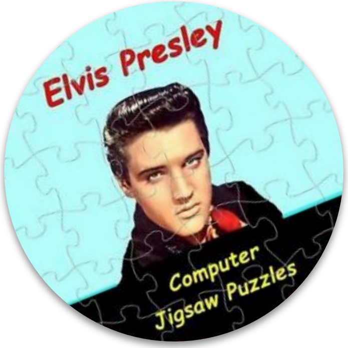


Algoritmos Las Vegas

Universidad de Alcalá

Un algoritmo será de tipo Las Vegas si:

- Da solución correcta
- O avisa de que no encuentra solución



A diferencia de los Montecarlo aquí tendremos que determinar, en vez de cuál es nuestro margen de error, cuándo es computacionalmente útil

Algoritmos Las Vegas

```
def las_vegas(x):
  resultado = procesar(x)

  while not is_valid(resultado):
    resultado = procesar(x)

  return resultado
```

Algoritmos Las Vegas

Universidad de Alcalá

Para caracterizarlo (obtener **T(x)**) partiremos de las siguientes premisas:

- La ejecución será exitosa con probabilidad **p(x)**
  - Success → **s(x)**
- La ejecución será fallida con probabilidad **(1 - p(x))**
  - Failure → **f(x)**
- Si se produce un fracaso, habrá que volver a lanzar el método: **T(x)**

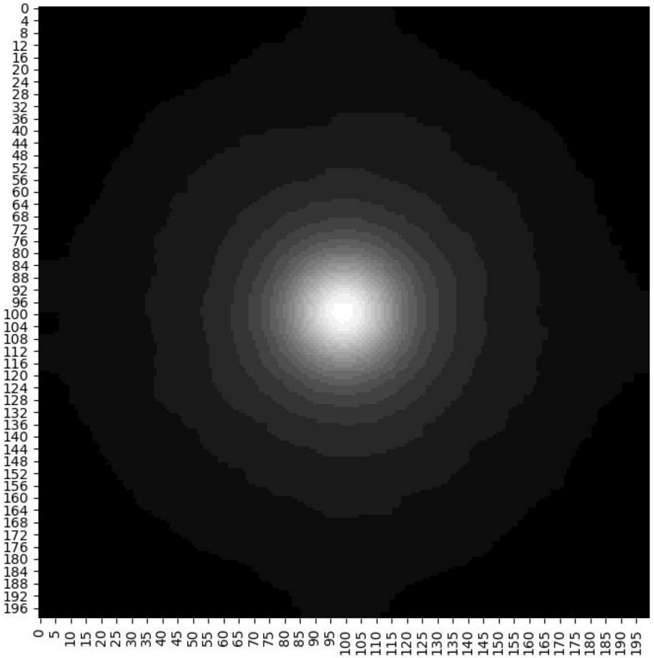
$$T(x) = p(x)\ s(x) + [\ (\ 1 - p(x)\ )\ \ (\ f(x) + T(x)\ )\ ]$$

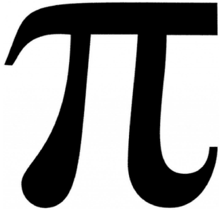
(y limpiando un poco)

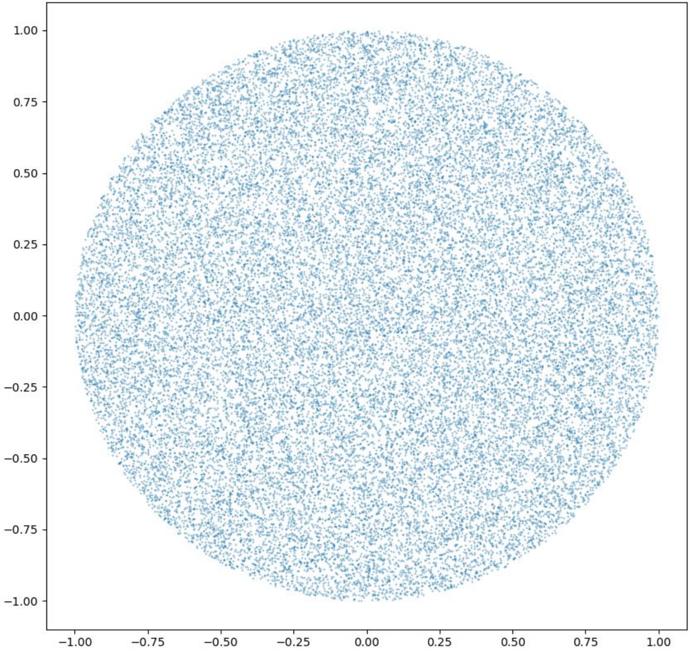
$$T(x) = s(x) + [\ (\ 1 - p(x)\ )\ /\ p(x)\ ] * f(x)$$

javier@junquera.io (UAH 2023)

Montecarlo v. Las Vegas



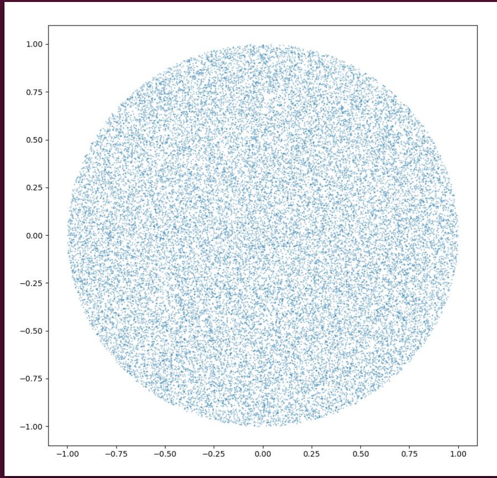




https://colab.research.google.com/drive/1hpXW4uG2AdGKtg7mYChfsBFTMNJpIuAy

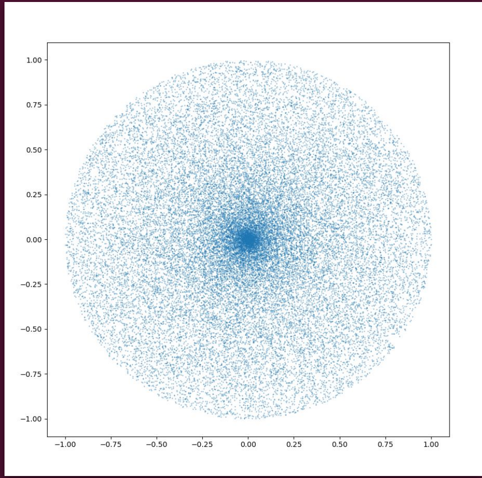
Dibuja un círculo

```python
def gen_point_las_vegas():
    while True:
        p = gen_random_point()
        if p.x**2 + p.y**2 < 1:
            return Point(x, y)
```



Dibuja un círculo (alt.)

```
def gen_point_las_vegas_alt():
    r = random.random()
    angulo = 2 * math.pi * random.random()
    return Point(
            r * math.cos(angulo),
            r * math.sin(angulo)
          )
```



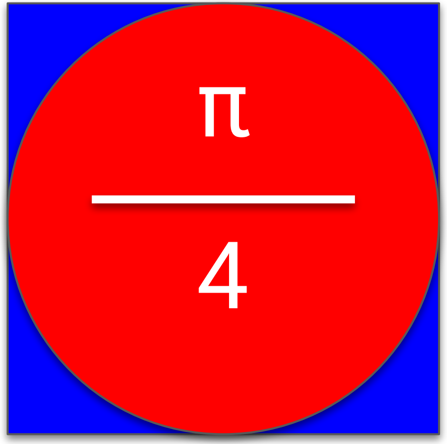
Cálculo de PI

Partiendo de una circunferencia de radio unidad:

- $r = 1$

Si está inscrita en un cuadrado, sus áreas serán:

- Cuadro → $(2r)^2 = 4$
- Círculo → $\pi r^2 = \pi$



El círculo ocupa un 78% ($\pi/4$) de la superficie de la figura.

La aguja de Buffon

Cálculo de PI

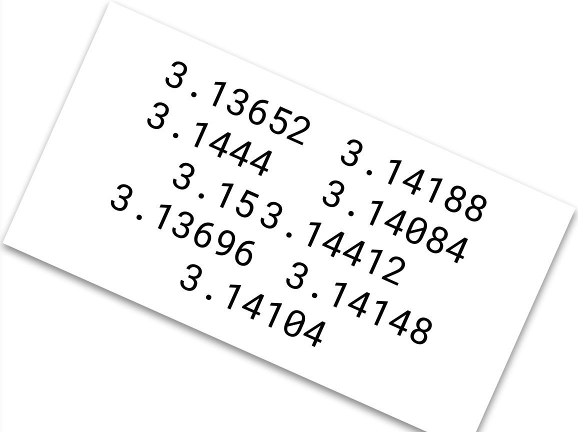
Universidad de Alcalá

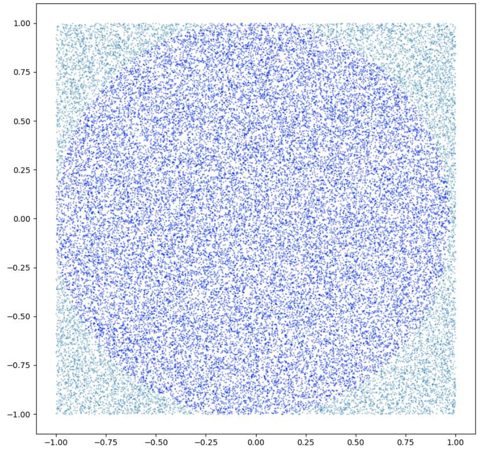
Por lo tanto, si lanzamos n puntos al azar, y la probabilidad de que un punto caiga dentro es del 78%, caerán dentro:

```
dentro = (π/4) * n
```

Si despejamos la fórmula:

```
π = (4*dentro)/n
```





javier@junquera.io (UAH 2023)

Cálculo de PI

```
def gen_pi_montecarlo():
  count = 0
  for i in range(n):
    p = gen_random_point()
    if p.x**2 + p.y**2 < 1:
      count += 1
  return (4.0 * count) / n
```

¿Preguntas?