Métodos de Monte Carlo

Curso de Física Computacional

M. en C. Gustavo Contreras Mayén

Facultad de Ciencias - UNAM

26 de abril de 2018

Definición

El método Monte Carlo es un conjunto de métodos numéricos que permiten resolver problemas físicos y matemáticos mediante la simulación de variables aleatorias.

Definición

Los métodos Monte Carlo fueron nombrados de esta manera por su clara analogía con los juegos de ruleta de los casinos, el más célebre de los cuales es el de Montecarlo, casino cuya construcción fue propuesta en 1856 por el príncipe Carlos III de Mónaco, siendo inaugurado en 1861.

Relevancia del método

La importancia actual de los métodos Monte Carlo se basa en la existencia de problemas que tienen difícil solución por métodos exclusivamente analíticos o numéricos, pero que dependen de factores aleatorios o se pueden asociar a un modelo probabilístico artificial (resolución de integrales de muchas variables, minimización de funciones, etc.)

Relevancia del método

Gracias al continuo diseño de procesadores y de computadoras, los cálculos con Monte Carlo que en otro tiempo hubieran sido inconcebibles, hoy en día se presentan como asequibles para la resolución de ciertos problemas.

Proporción del error

En estos métodos el error $\simeq \frac{1}{\sqrt{N}}$, donde N es el número de pruebas, por tanto, ganar una cifra decimal en la precisión implica aumentar N en 100 veces.

Fundamento del método

La base del método es la generación de números aleatorios de los que nos serviremos para calcular probabilidades.

Conseguir un buen generador de estos números así como un conjunto estadístico adecuado sobre el que trabajar son las primeras dificultades con la nos vamos a encontrar a la hora de utilizar este método.

Secuencia aleatoria

Se define una secuencia aleatoria de números $r_1, r_2, \ldots$ si no existe una correlación entre ellos.

Aunque sean aleatorios, no implica que todos los números en la secuencia tengan la misma probabilidad de ocurrir.

Secuencia aleatoria

Si todos los números en la secuencia tienen la misma probabilidad de ocurrir, se dice que la secuencia es uniforme y los números son aleatorios.

Por ejemplo: $1, 2, 3, 4, \ldots$ es uniforme pero probablemente no es aleatoria.

Secuencia aleatoria

También es posible tener una secuencia de números que de alguna forma son aleatorios, pero tienen correlación dentro de un intervalo pequeño:

$$r_1,\ (1 - r_1),\ r_2,\ (1 - r_2),\ r_3,\ (1 - r_3),\ \ldots$$

Las computadoras por naturaleza, son determinísticas y no pueden crear una secuencia de números aleatorios.

Secuencia aleatoria

Las computadoras pueden crear secuencias que contengan correlaciones y por tanto no ser totalmente aleatorias; si conocemos r_m y su elemento precedente, es posible estimar r_{m+1}.

Por ésta razón, se dice que las computadoras son generadores de números pseudo-aleatorios.

Secuencia aleatoria

Matemáticamente, la probabilidad de que un número ocurra, está descrita por una función de distribución $P(r)$, donde $P(r)\,dr$, es la probabilidad de encontrar r en un intervalo $[r, r + dr]$.

Una distribución uniforme significa que $P(r) =$ constante.

Generador de número aleatorios

El generador estándar de números aleatorios en las computadoras, genera distribuciones uniformes entre 0 y 1.

El generador estándar de números aleatorios, proporciona números en éste intervalo, y cada uno de ellos tiene la misma probabilidad de ocurrir, y además es independiente del número anterior.

Generación de números aleatorios

El método de *congruencia lineal* es la manera más común que se utiliza para generar una secuencia de números pseudo-aleatorios $0 \leq r_i \leq M - 1$ en el intervalo $[0, M - 1]$.

Generación de números aleatorios

Podemos multiplicar el número aleatorio previo r_{i-1} por una constante a, sumar otra constante c, operar con el módulo M, manteniendo sólo la parte fraccional del resultado como el siguiente número aleatorio r_{i+1}

$$r_{i+1} = (a\ r_i + c) \bmod M$$

$$r_{i+1} = (a\ r_i + c) \bmod M$$

El valor de r_1 se le llama *semilla* (seed, en inglés) y lo proporciona normalmente el usuario.

Ejemplo

Veamos la secuencia que se genera si $c = 1$, $a = 4$, $M = 9$, y la semilla es $r_1 = 3$:

$$r_{i+1} = (a\, r_i + c) \bmod M$$

$$\begin{aligned} r_1 &= 3 \\ r_2 &= (4 \times 3 + 1) \bmod 9 = 13 \bmod 9 = 4 \\ r_3 &= (4 \times 4 + 1) \bmod 9 = 17 \bmod 9 = 8 \\ r_4 &= (4 \times 8 + 1) \bmod 9 = 33 \bmod 9 = 6 \\ r_{5-10} &= 7, 2, 0, 1, 5, 3 \end{aligned}$$

Hemos obtenido una secuencia de longitud $M = 9$, antes de que la secuencia se repitiera.

Si queremos números en el rango $[0, 1]$, basta dividir los números r por $M = 9$:

$$0.333, 0.222, 0.444, 0.000, 0.889,$$
$$0.111, 0.667, 0.555, 0.778, 0.333$$

Aún sigue siendo una secuencia de longitud 9 pero ya no es una secuencia de enteros.

Si queremos que los números aleatorios estén en el rango $[A, B]$, se requiere aplicar el factor de escala:

$$x_i = A + (B - A)\, r_i \qquad 0 \leq r_1 \leq 1 \rightarrow A \leq x_i \leq B$$

Sugerencia

Antes de utilizar un generador de números aleatorios en nuestros programas, debemos de revisar que el rango que produce, tiene apariencia de aleatorios.

Sugerencia

Propiamente no es un prueba matemática, pero al graficar la distribución de números aleatorios, podemos reconocer ciertos patrones, con lo que nos diría si tenemos o no, números aleatorios.

Código para los números aleatorios I

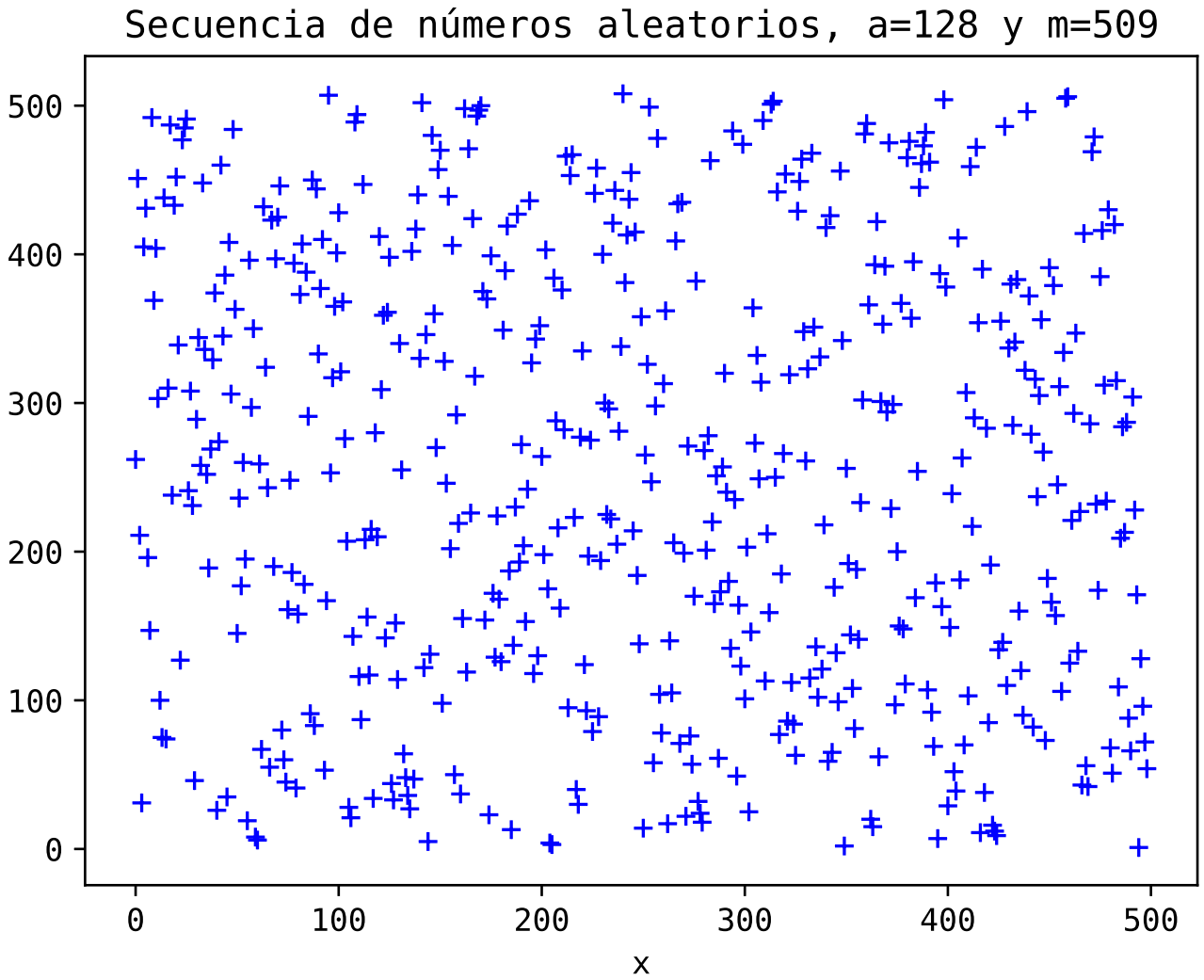
Código 1: Código distribución

```
1  x = []
2
3  a, semilla, c, m, n = 128, 10, 0, 50
       9, 500
4  for i in range (1, n):
5      nuevasemilla = (a * semilla + c)
       % m
6      semilla = nuevasemilla
7      x.append( nuevasemilla)
8
9  y = []
10
```

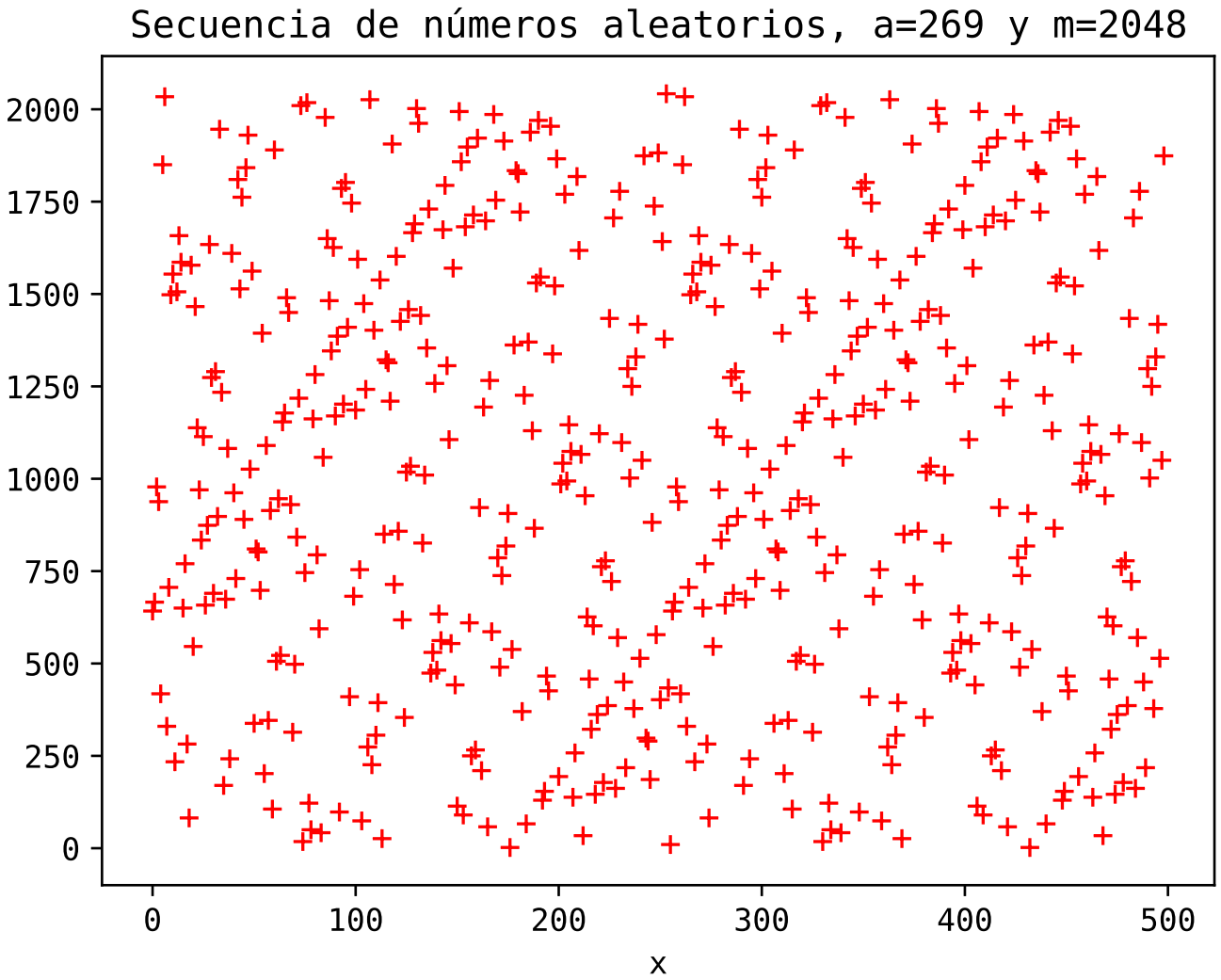
Código para los números aleatorios II

```
11 a, semilla, c, m, n = 269, 10, 0, 20
       48, 500
12
13 for i in range (1, n):
14     nuevasemilla = (a * semilla + c)
       % m
15     semilla = nuevasemilla
16     y.append( nuevasemilla)
```

Distribución de números aleatorios (1/2)



Distribución de números aleatorios (1/2)



La librería `random`

Este módulo implementa generadores de números pseudoaleatorios para diversas distribuciones.

Para números enteros, hay selección uniforme de un rango. Para las secuencias, hay una selección uniforme de un elemento aleatorio, una función para generar una permutación aleatoria de una lista *in situ* y una función para el muestreo aleatorio sin reemplazo.

La librería `random`

Casi todas las funciones de la librería dependen de la función básica **`random ()`**, que genera un número flotante aleatorio de manera uniforme en el rango semiabierto $[0.0, 1.0)$

La librería `random`

En `python` se usa el algoritmo Mersenne Twister como generador central. Produce números flotantes de precisión de 53 bits y tiene un período de $2^{19937} - 1$.

Alcance del generador en `random`

El algoritmo Mersenne Twister es uno de los generadores de números aleatorios más probados que existen.

Sin embargo, al ser completamente determinista, no es adecuado para todos los propósitos, y es completamente inadecuado para fines criptográficos.

Funciones importantes

Dentro de la librería `random` existen distintas funciones que podremos utilizar, basta con revisar la documentación disponible, para identificar los parámetros y la respuesta que devuelve.

Funciones importantes

Mencionaremos dos funciones:

1. `random.seed(a=None)`. Inicializa el generador de números aleatorios. Si no se especifica el valor de a, se utiliza el reloj del sistema. En caso de que a sea un entero, se utilza directamente.

Funciones importantes

Mencionaremos dos funciones:

1. **`random.seed(a=None)`**. Inicializa el generador de números aleatorios. Si no se especifica el valor de a, se utiliza el reloj del sistema. En caso de que a sea un entero, se utilza directamente.
2. **`random.random()`**. Devuelve un número de punto flotante en el rango $[0.0, 1.0)$.

Ejemplo con `random.seed(a)`

Haremos un ejercicio en donde se van a generar tres secuencias de números con la función **`random.seed(a)`**, cambiaremos el valor de la semilla y vamos a comparar los resultados en una gráfica.

Código usando `seed()` I

Código 2: Código con semilla

```
1  import matplotlib.pyplot as plt
2  import random
3
4  def arreglonumeros(muestra=500,
       semilla=None):
5      x = []
6      random.seed(semilla)
7
8      for i in range(muestra):
9          nuevoValor = random.random()
10         x.append(nuevoValor)
11
```

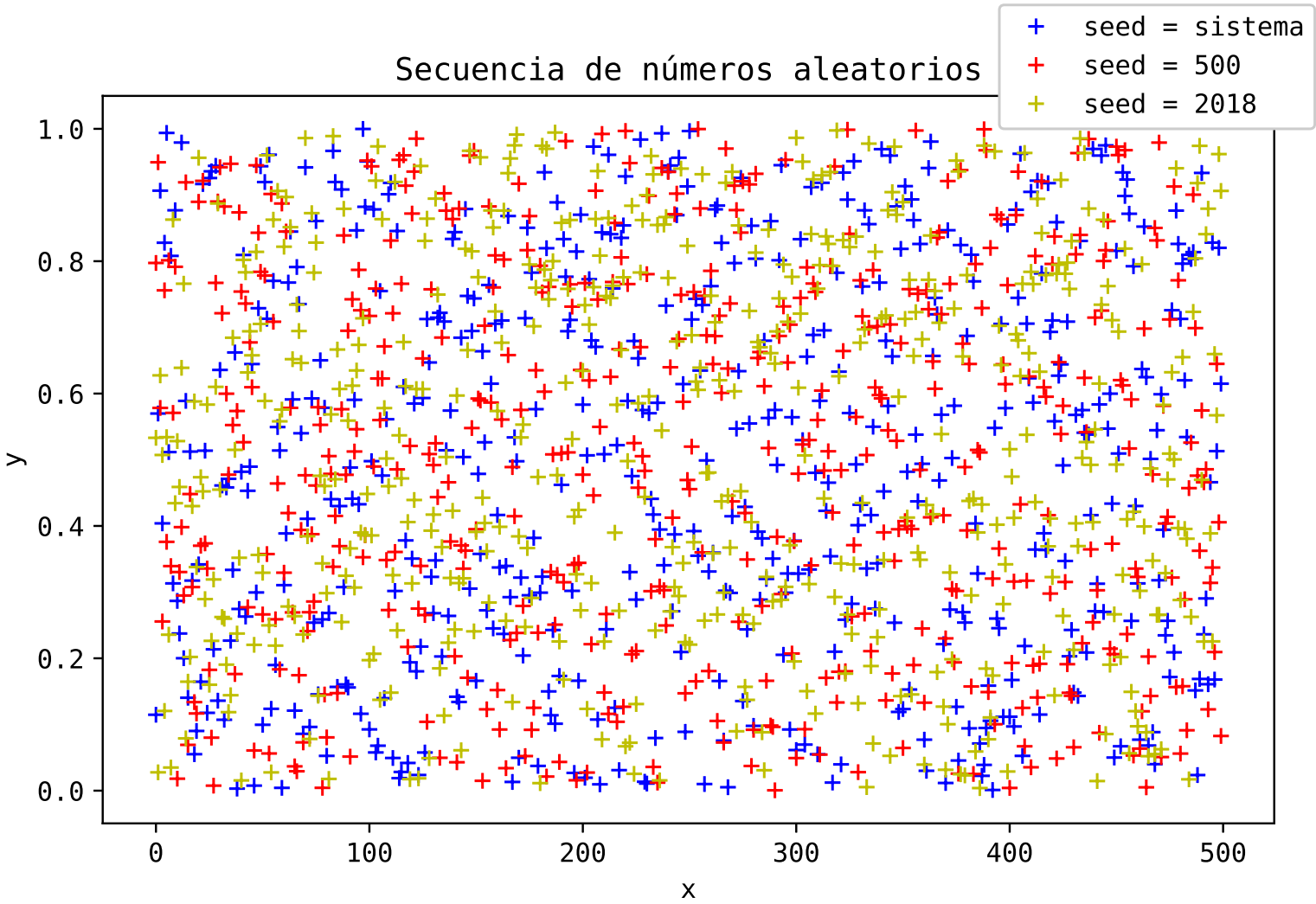
Código usando `seed()` II

```
12      return x
13
14 x1 = arreglonumeros()
15 x2 = arreglonumeros(semilla=500)
16 x3 = arreglonumeros(semilla=2018)
17
18 plt.figure(figsize=(8,5))
19 plt.plot(x1,'b+', label='seed =
     reloj sistema')
20 plt.plot(x2,'r+', label='seed = 500'
     )
21 plt.plot(x3,'y+', label='seed = 2018
     ')
```

Código usando `seed()` III

```
22 plt.legend(loc='upper center', bbox
      _to_anchor=(0.93, 1.1),
      borderaxespad = 0.)
23 plt.title('Secuencia de numeros
      aleatorios')
24 plt.xlabel('x')
25 plt.ylabel('y')
26 plt.show()
```

Gráfica obtenida



1 Métodos de Monte Carlo

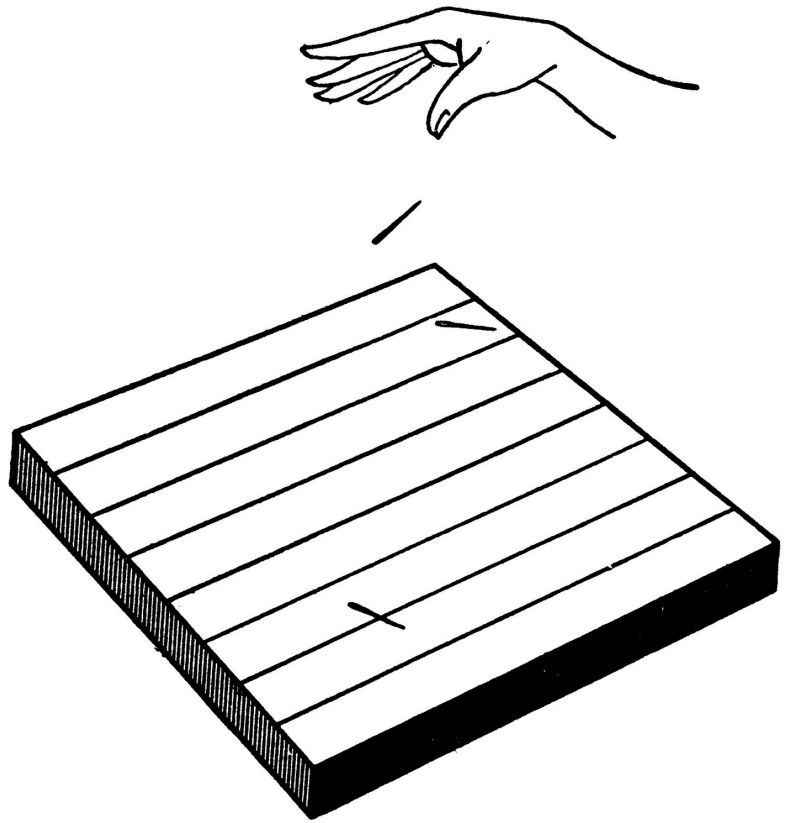
2 La aguja de Buffon

- Planteamiento
- Aproximación al valor de π

La aguja de Buffon

Resulta que π también desempeña un papel importante en un experimento llamado “el problema de la aguja de Buffon”.

El cual buscar determinar la probabilidad de que una aguja de longitud ℓ, arrojada aleatoriamente aterrice en medio o atravesando una serie de líneas paralelas en el suelo.



La aguja de Buffon

Resulta que si la distancia entre las líneas paralelas es la misma que la longitud ℓ de la aguja lanzada, el número de veces que la aguja cae atravesando las líneas (luego de un gran número de lanzamientos), nos servirá para calcular el valor de π.

Aproximación al valor de π

De esa manera:

$$\pi = \frac{2N}{A}$$

siendo N el número total de intentos y A el número de veces que la aguja ha cruzado alguna línea.

Primer problema de tarea

Considera el caso en el que la longitud de la aguja es igual a la separación entre las rayas, es decir $\ell = h$.

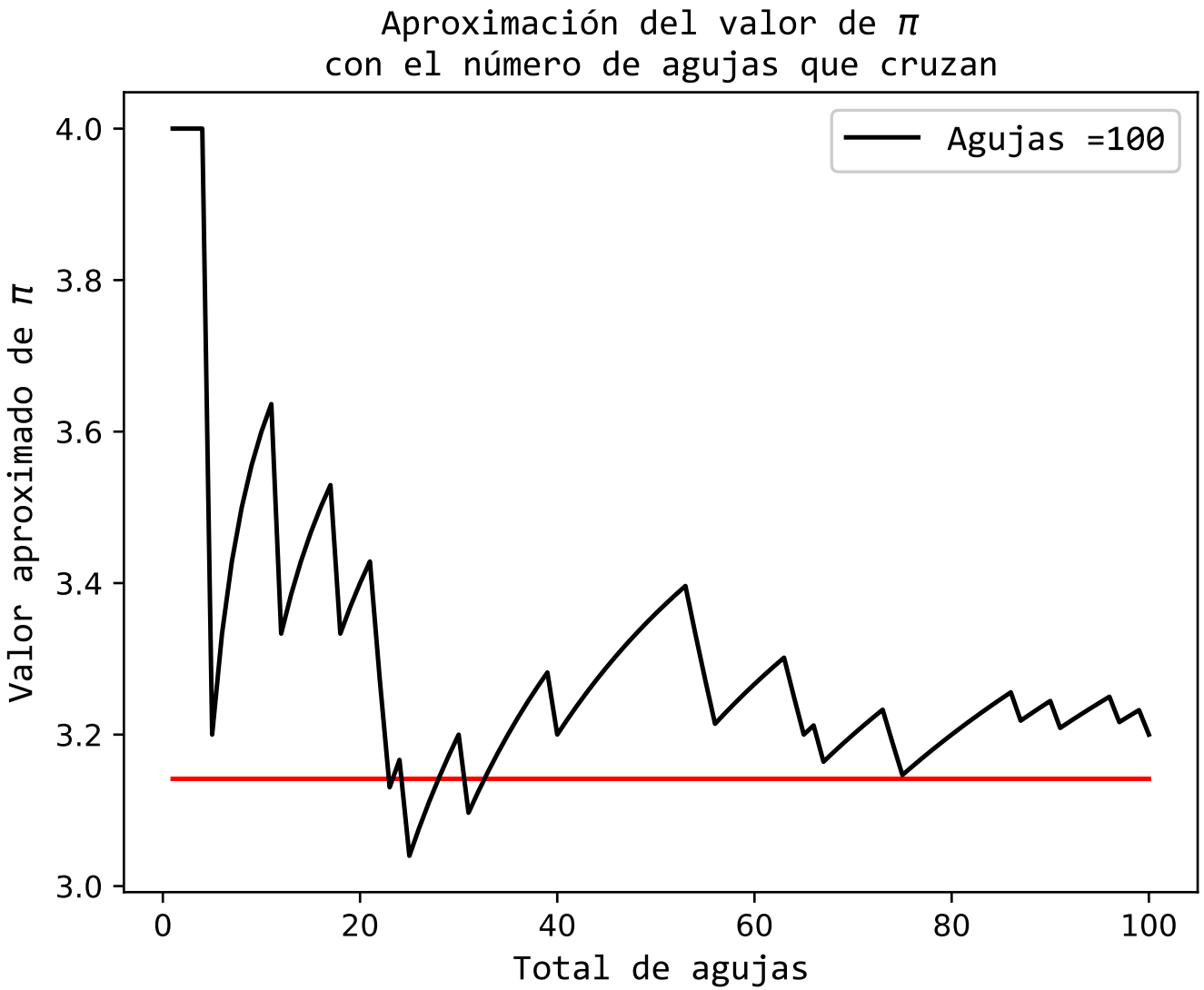
Si la aguja cruza una línea de manera oblicua, es decir, existe un ángulo de inclinación con respecto a la línea, puedes deducir la relación a partir de la "función" que describe el caso de las agujas que tocan las líneas.

Segundo problema de tarea

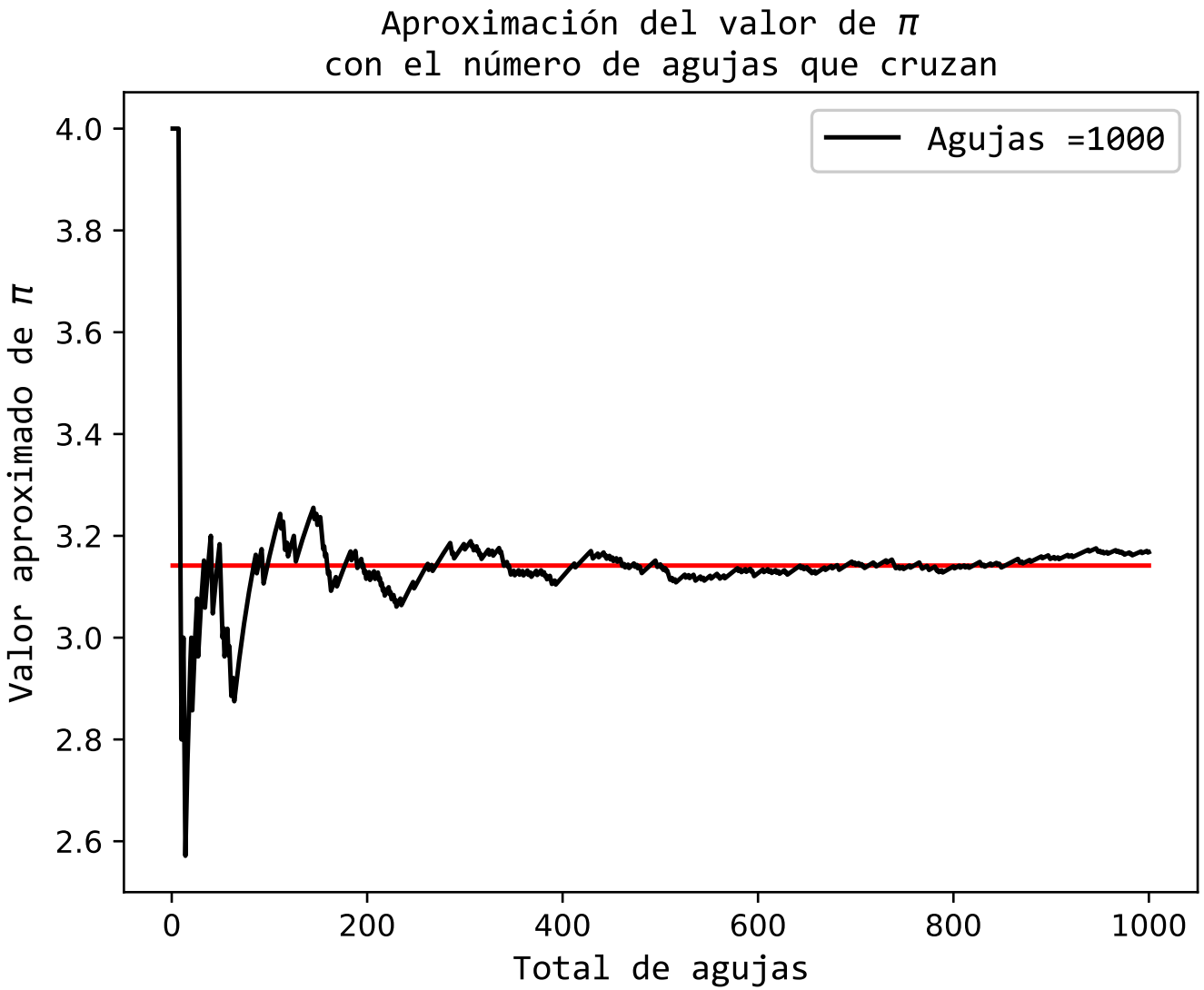
Una vez que has deducido el problema, implementa en python un programa que calcule el valor aproximado de π, a partir del número de lanzamientos, y el número de agujas que cruzan una línea.

En las siguientes gráficas verás los resultados cuando se realiza el lanzamiento de $10^2, 10^3, 10^4, 10^5, 10^6$ agujas.

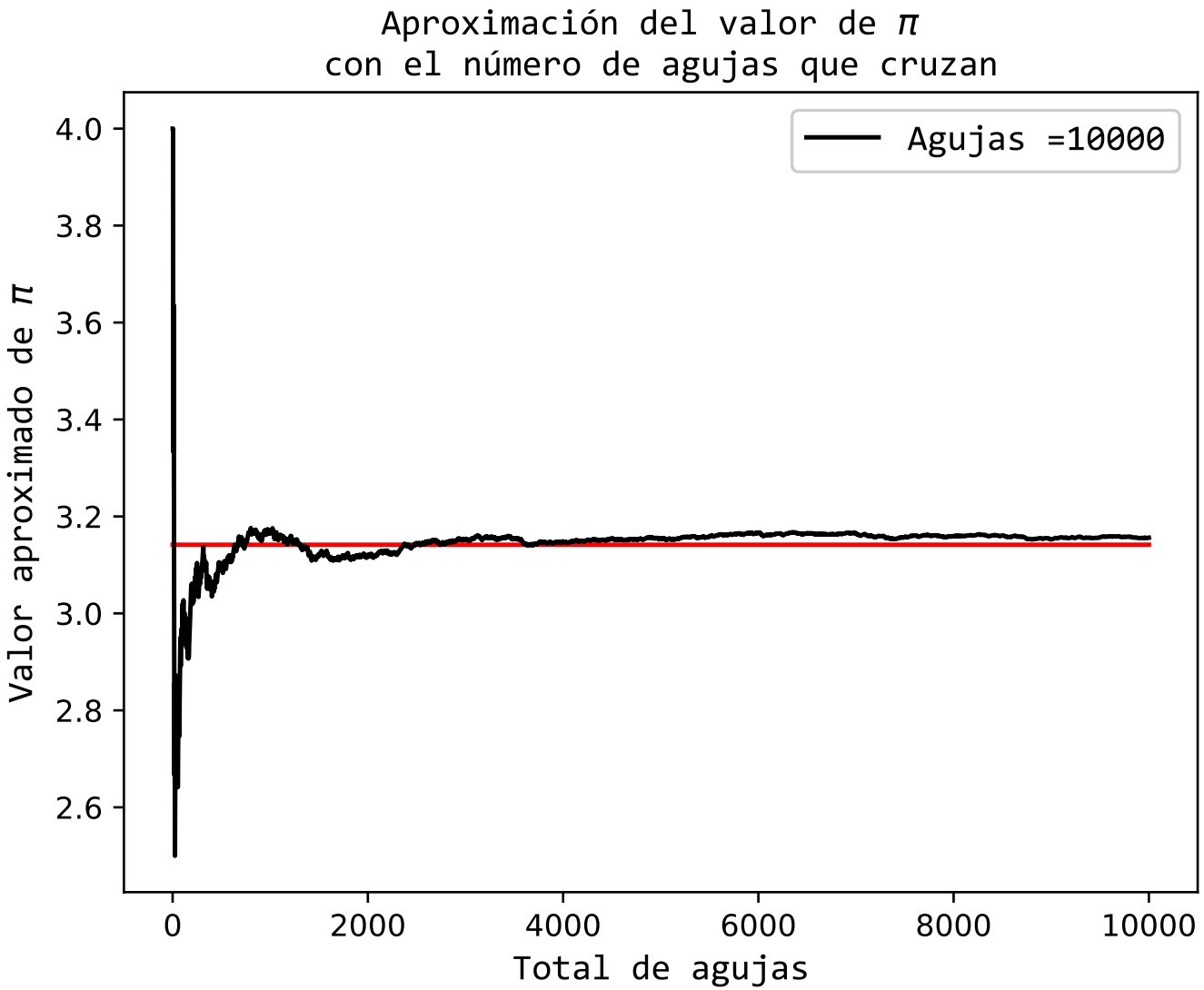
Solución con python



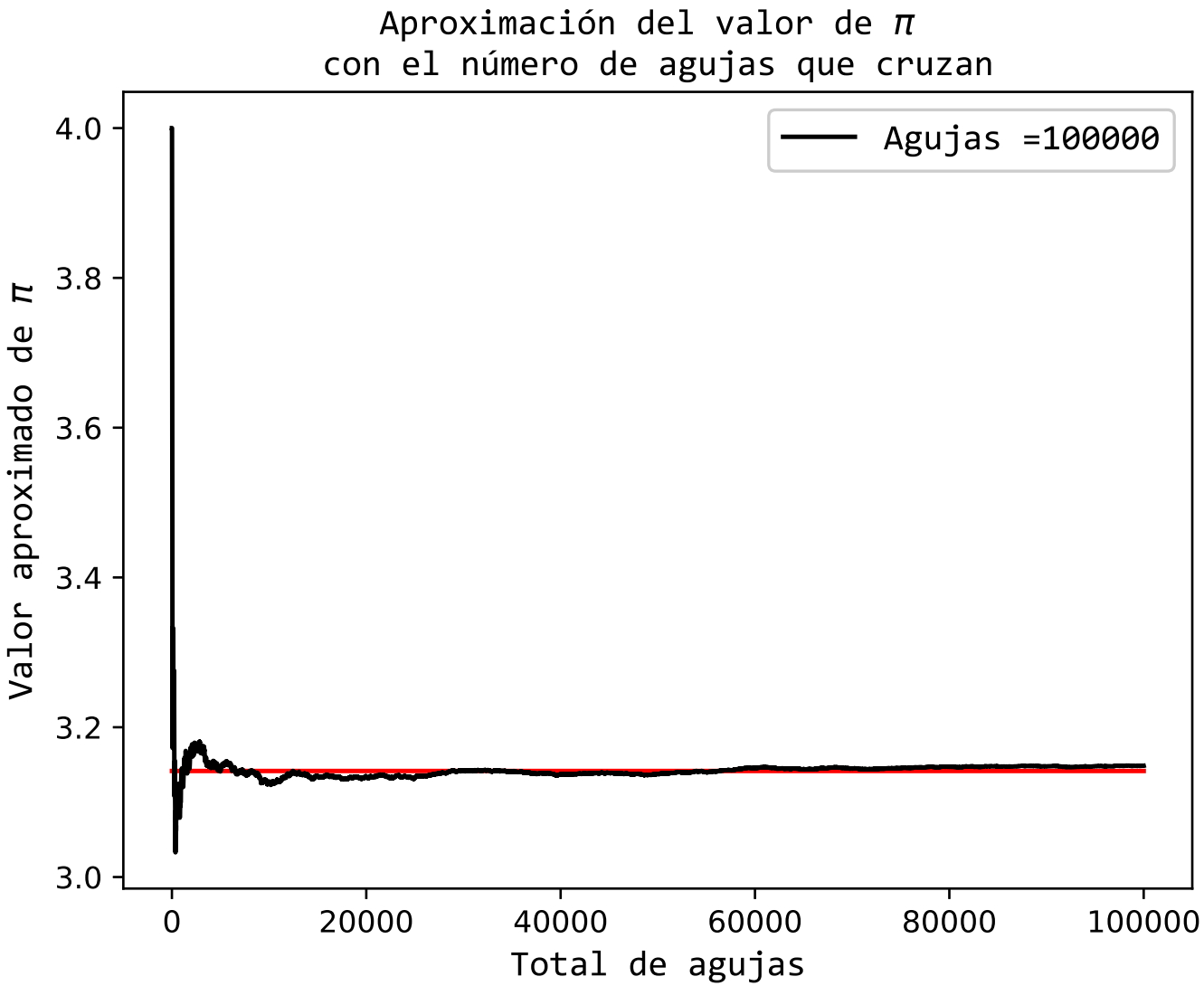
Solución con python



Solución con python



Solución con python



Solución con python