Workshop_Solved

June 13, 2024

Autor: Gustavo Alonso Gomez Morales

1 Método de Montecarlo

El método de Montecarlo es una técnica matemática que utiliza números aleatorios para resolver problemas que, en **condiciones normales**, se resolverían de manera precisa y predecible. La idea central es **simular** o **modelar** situaciones de la vida real y problemas matemáticos mediante la generación de ***números al azar*** y el uso de estadísticas para obtener resultados. Esta técnica se puede aplicar a diversos campos, como la física, la economía, la ingeniería, y más.

Este método se basa en la **ley de los grandes números**, que en esencia dice que cuantas más veces repitamos un experimento aleatorio, más cerca estaremos de obtener el valor esperado real. Por lo tanto, mediante la generación de muchas muestras aleatorias y el análisis de los resultados, podemos aproximarnos a una solución que sería difícil o imposible de calcular de forma exacta mediante otros métodos.

1.1 Ejercicio

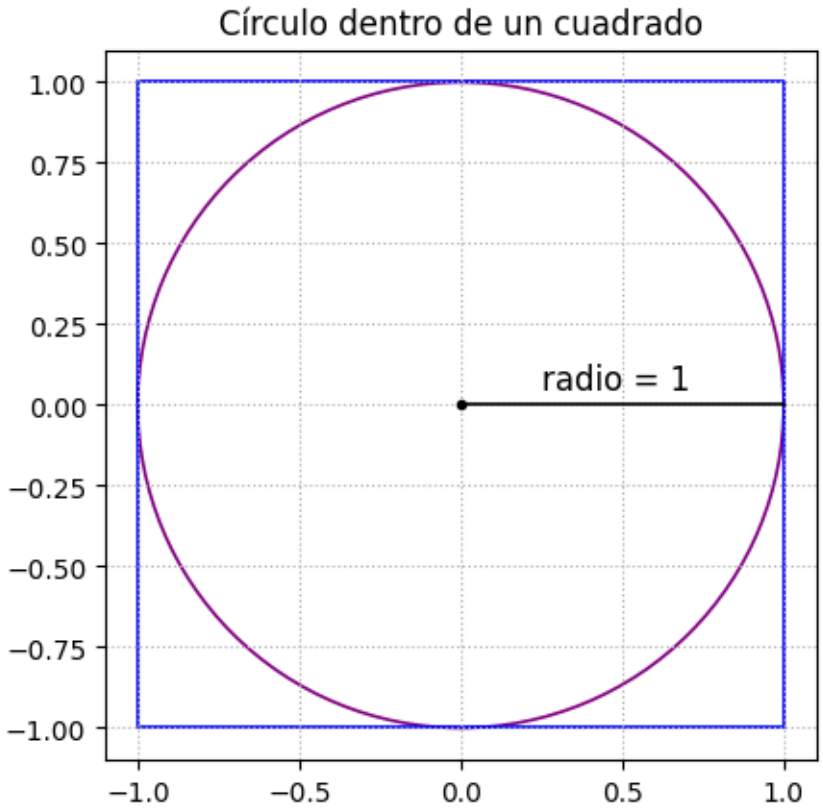
Vamos a aplicar el método de Montecarlo para estimar el valor de π. Utilizaremos un experimento sencillo basado en el concepto de lanzar ***puntos aleatorios*** dentro de un cuadrado que contiene un círculo inscrito.

Imagina que tienes una gran hoja cuadrada de papel y en medio de ese papel dibujas un círculo grande que toca todos los lados del cuadrado justo en medio de cada lado.

Ejecuta el siguiente código:

[18]:
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()

circle = plt.Circle((0, 0), 1, edgecolor='purple', linewidth=1.2, fill=None)
ax.add_patch(circle)

square = plt.Rectangle((-1, -1), 2, 2, edgecolor='blue', linewidth=1.2, ␣
 ↪fill=None)
ax.add_patch(square)

radius = plt.Line2D([0, 1], [0, 0], color='black', linewidth=1.2)
```

```
ax.add_line(radius)

plt.plot(0, 0, 'k.')

ax.set_aspect('equal', adjustable='box')
ax.set_xlim(-1.1, 1.1)
ax.set_ylim(-1.1, 1.1)
plt.title('Círculo dentro de un cuadrado')
plt.text(0.25, 0.05, 'radio = 1', fontsize=12, color='black')
plt.grid(True, linestyle=":")
plt.show()
```

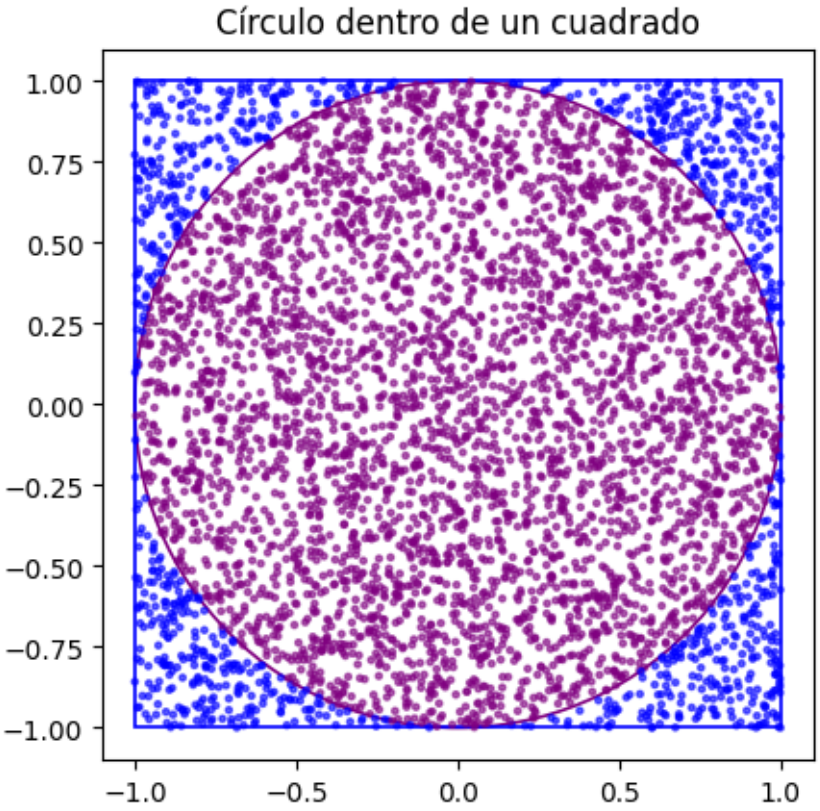


Ahora, supongamos que posees un marcador equipado con inteligencia artificial. Cierras tus ojos y comienzas a dibujar puntos al azar en un papel cuadriculado utilizando este avanzado marcador. *Algunos de los puntos terminarán dentro de un círculo trazado en el papel, mientras que otros quedarán fuera.* Este marcador es tan sofisticado que automáticamente colorea de **azul** los puntos que están fuera del círculo y de **morado** aquellos que se encuentran dentro.

Este marcador no solo diferencia los puntos por colores, sino que también te proporciona una cuenta exacta de cuántos puntos has colocado dentro del círculo y el total de puntos que has dibujado. Al comparar estos dos números, puedes descubrir la relación entre el área del círculo y el área del cuadrado que lo contiene. ¿Cómo? Ya lo veremos!

Ejecuta el siguiente código:

[19]:
```
import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots()

circle = plt.Circle((0, 0), 1, edgecolor='purple', linewidth=1.2, fill=None)
ax.add_patch(circle)

square = plt.Rectangle((-1, -1), 2, 2, edgecolor='blue', linewidth=1.2,␣
↪fill=None)
ax.add_patch(square)

ax.set_aspect('equal', adjustable='box')
ax.set_xlim(-1.1, 1.1)
ax.set_ylim(-1.1, 1.1)
plt.title('Círculo dentro de un cuadrado')

num_points = 5000
points = np.random.uniform(-1, 1, (num_points, 2))
colors = ['purple' if x**2 + y**2 <= 1 else 'blue' for x, y in points]
plt.scatter(points[:, 0], points[:, 1], c=colors, s=5, alpha=0.6)
plt.show()
```



1.1.1 Por qué utilizamos la proporción de áreas?

Para estimar el valor de π usando el método de Montecarlo es esencial comprender cómo las áreas del círculo y del cuadrado se relacionan. Utilizar la proporción de puntos que caen dentro del círculo respecto a los puntos totales en el cuadrado es una forma eficaz y visual de estimar el área del círculo y, por ende, π.

Cuando lanzamos puntos aleatorios sobre un área determinada, la probabilidad de que un punto caiga dentro de una región específica es proporcional al área de esa región. En este caso, lanzamos puntos aleatoriamente sobre un cuadrado que contiene un círculo. Dado que la distribución de los puntos es uniforme, la fracción de puntos que caen dentro del círculo se aproxima a la fracción del área del círculo respecto al área total del cuadrado.

1.1.2 Cómo se relacionan las áreas para estimar π?

El área de un círculo de radio r es:

$A_{crculo} = \pi r^2$

y el área del cuadrado que lo inscribe, con el diámetro del círculo igual a la longitud del lado del cuadrado, es:

$A_{cuadrado} = (2r)^2 = 4r^2$

Esto lleva a la relación:

$\frac{A_{crculo}}{A_{cuadrado}} = \frac{\pi \cdot r^2}{4 \cdot r^2} = \frac{\pi}{4}$

1.1.3 Estimación de π

Cuando simulamos el lanzamiento de puntos, los que caen dentro del círculo (aproximación del área del círculo) y los totales dentro del cuadrado (aproximación del área del cuadrado) nos proporcionan una estimación del valor de π de la siguiente manera:

$\frac{\text{puntos dentro del círculo}}{\text{puntos totales}} \approx \frac{\pi}{4}$

$\pi \approx 4 \cdot \left(\frac{\text{puntos dentro del círculo}}{\text{puntos totales}}\right)$

Esta relación permite utilizar un experimento aleatorio simple para aproximar π, un número fundamental en matemáticas y física, que es la base para entender las propiedades de los círculos y esferas en ciencia e ingeniería. Además, este enfoque ofrece una manera visual y práctica de entender conceptos abstractos de probabilidad y geometría.

Tu tarea será desarrollar algunas variables y funciones esenciales que te permitirán obtener una representación gráfica del experimento y estimar el valor de π de manera efectiva.

Primero, se necesita definir una variable llamada `num_points`, que representará el *número total de puntos generados*, tanto los que se encuentran dentro del círculo como los que están fuera de él. Es importante recordar que, cuanto mayor sea el número de puntos, mejor será la aproximación; por lo tanto, elige un número de puntos apropiado para una estimación precisa.

Define la variable y asignale un número entero apropiado para la estimación de π.

[20]:
```
# Escribe tu código en las siguientes líneas

num_points = 5000
```

Luego, debemos generar de manera aleatoria y uniforme puntos que caigan dentro y fuera de un círculo, almacenándolos en listas para graficarlos posteriormente. Para ello, se necesita crear una función llamada `generate_random_points`, que reciba como parámetro el número entero definido anteriormente (`num_points`). Esta función se encargará de generar los puntos y retornar dos listas: una con los puntos que se encuentran dentro del círculo y otra con los que se encuentran fuera.

Puedes generar un número aleatorio a partir de una distribución uniforme haciendo uso de la función `uniform()` perteneciente a librería `random` de Python. Para más información sobre esta función haz clic aquí.

`random.uniform(a, b): Devuelve un número de punto flotante aleatorio en el rango [a, b].`

Recuerda que necesitas dos números aleatorios que representen cada uno la coordenada `x` y la coordenada `y`.

Requisitos para la funcion:

- La función debe llamarse `generate_random_points`.
- La función debe recibir un solo parámetro: `num_points`.
- Dentro de la función, se debe generar num_points puntos aleatorios en el plano 2D.
- Cada punto debe tener coordenadas x y y generadas de manera aleatoria y uniforme en el rango [-1, 1] usando la función `random.uniform(-1, 1)`. Puedes usar un ciclo `for` para ello.
- La función debe retornar dos listas: Una lista con los puntos que se encuentran dentro del círculo llamada `points_inside` y una lista con los puntos que se encuentran fuera del círculo llamada `points_outside`.

[21]:
```
import random
# Escribe tu código en las siguientes líneas

def generate_random_points(num_points):
    points_inside = []
    points_outside = []

    for _ in range(num_points):
        x = random.uniform(-1, 1)
        y = random.uniform(-1, 1)
        if x**2 + y**2 <= 1:
            points_inside.append((x, y))
        else:
            points_outside.append((x, y))

    return points_inside, points_outside
```

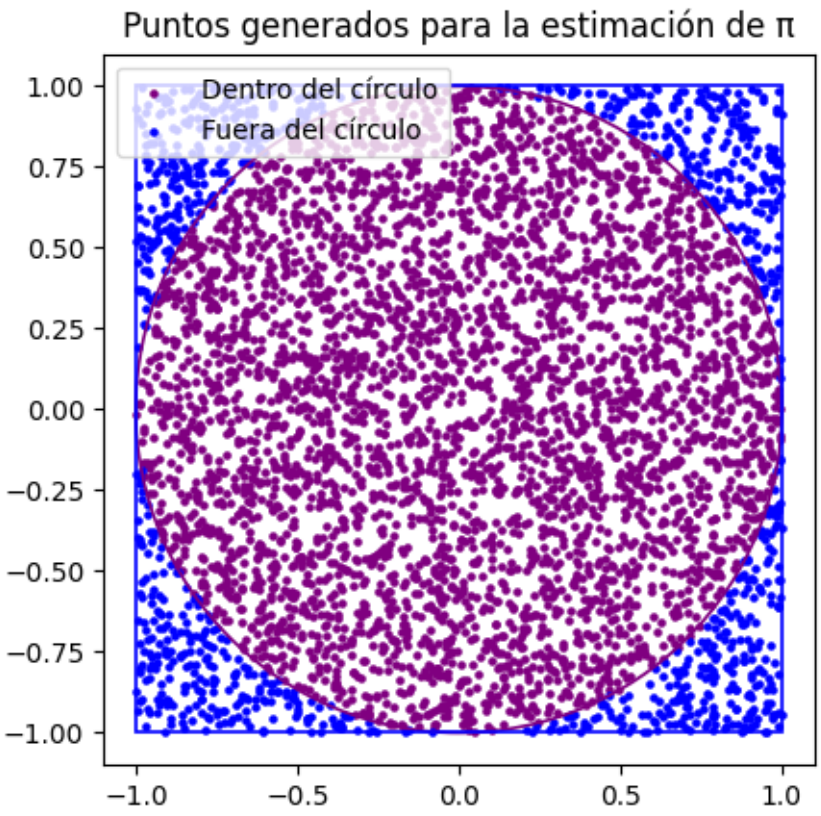
Al final de la función `generate_random_points`, se devuelven las dos listas, `points_inside` y `points_outside` (en este orden). Estas listas contienen los puntos clasificados que se utilizarán

posteriormente para visualizar la estimación de π.

Lo que haremos ahora es ejecutar la función `generate_random_points` con un total de `num_points` puntos. Las listas resultantes `points_inside` y `points_outside` son luego descompuestas en sus componentes x e y mediante comprensiones de lista, preparando los datos para su posterior visualización en el gráfico.

[22]:
```
points_inside, points_outside = generate_random_points(num_points)

# Separaramos los puntos dentro del círculo para la graficación
x_inside = [x for x, y in points_inside]
y_inside = [y for x, y in points_inside]

# Separaramos los puntos fuera del círculo para la graficación
x_outside = [x for x, y in points_outside]
y_outside = [y for x, y in points_outside]
```

[23]:
```
# Creamos la figura y el eje
fig, ax = plt.subplots()

# Dibujamos los puntos dentro del círculo en morado
ax.scatter(x_inside, y_inside, color='purple', s=5, label='Dentro del círculo')

# Dibujamos los puntos fuera del círculo en azul
ax.scatter(x_outside, y_outside, color='blue', s=5, label='Fuera del círculo')

# Añadimos el círculo al gráfico para visualizar el límite
circle = plt.Circle((0, 0), 1, edgecolor='purple', linewidth=1.2, fill=None)
ax.add_patch(circle)

# Añadimos el cuadrado al gráfico para visualizar el límite
square = plt.Rectangle((-1, -1), 2, 2, edgecolor='blue', linewidth=1.2,␣
 ↪fill=None)
ax.add_patch(square)

# Ajustamos los límites y etiquetas del gráfico
ax.set_xlim(-1.1, 1.1)
ax.set_ylim(-1.1, 1.1 )
ax.set_aspect('equal', adjustable='box')
ax.set_title('Puntos generados para la estimación de  ')
ax.legend()

# Mostramos el gráfico
plt.show()
```



Ahora que hemos visualizado el gráfico, podemos proceder a estimar el valor de π. Para ello, necesitas crear una función llamada `estimate_pi` que reciba los siguientes parámetros:

- `num_points`: el número total de puntos generados.
- `num_points_inside`: el número de puntos que caen dentro del círculo.

La función debe retornar el valor estimado de π.

[24]:
```
# Escribe tu código en las siguientes líneas

def estimate_pi(num_points, num_points_inside):
    aprox_pi = 4 * (num_points_inside / num_points)
    return aprox_pi
```

Luego, deberás crear una variable llamada `pi_aprox` para almacenar el valor estimado de π proporcionado por la función `estimate_pi`.

[25]:
```
# Escribe tu código en las siguientes líneas

num_points_inside = len(points_inside)
pi_aprox = estimate_pi(num_points, num_points_inside)

# No borres esta línea
```

```
print(f"Estimación de  : {pi_aprox}")
```

```
Estimación de  : 3.1608
```

1.1.4 Por qué usar el método de Montecarlo?

Una excelente razón para utilizar el método de Montecarlo para estimar π en lugar de simplemente usar una calculadora es la capacidad de ilustrar y comprender conceptos fundamentales de probabilidad y estadística a través de un experimento visual y práctico. Aunque una calculadora puede proporcionar el valor de π con gran precisión, el método de Montecarlo permite a los estudiantes y a quienes están aprendiendo ver cómo se puede llegar a una solución aproximada mediante procesos aleatorios y repetitivos.

Este método no solo demuestra la utilidad de las simulaciones en matemáticas y ciencia, sino que también fomenta un aprendizaje interactivo y profundo sobre cómo funcionan las probabilidades y las estimaciones en situaciones reales. Además, experimentar con el método de Montecarlo puede revelar cómo las matemáticas se aplican fuera de los libros de texto, en contextos donde los datos exactos son inaccesibles o los cálculos directos son impracticables, preparando a los estudiantes para enfrentar problemas complejos en campos como la física, la ingeniería y las finanzas.