Proceso de Poisson (1)

June 12, 2025

1 Asignacion de proceso de Poisson

Damián Lugo 10149601

El proceso de Poisson es un modelo probabilístico utilizado para describir eventos que ocurren de manera aleatoria pero con una tasa promedio constante a lo largo del tiempo o el espacio. Es fundamental en áreas como teoría de colas, redes, fiabilidad, física, biología, economía, etc.

En las primeras líneas del código se importan tres bibliotecas esenciales para trabajar con procesos de Poisson en Python. Se importa `numpy` como `np`, que permite realizar cálculos numéricos y manejar vectores y matrices, útiles para simular eventos aleatorios o trabajar con series temporales. Luego, se importa `matplotlib.pyplot` como `plt`, la cual se utiliza para crear gráficos y visualizar los resultados, como distribuciones de probabilidad o histogramas. Finalmente, se importa la distribución de Poisson desde `scipy.stats`, lo que permite utilizar funciones estadísticas específicas como calcular la probabilidad de cierto número de eventos (`pmf`), la probabilidad acumulada (`cdf`) o generar datos aleatorios (`rvs`) que sigan dicha distribución. Estas bibliotecas juntas proporcionan las herramientas necesarias para modelar, simular y visualizar un proceso de Poisson.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import poisson
```

```
Matplotlib is building the font cache; this may take a moment.
```

Primero, definimos la tasa del proceso de Poisson con la variable tasa = 8. Esto significa que, en promedio, esperamos que ocurran 8 eventos por unidad de tiempo. Luego, establecemos el número de iteraciones con iteraciones = 1_000_000. Es decir, vamos a repetir el experimento un millón de veces para tener resultados más precisos y confiables. Finalmente, definimos num_preambulos = 64, que representa la cantidad de "preambulos" o posibles lugares donde puede caer cada evento. Podés pensar en estos preámbulos como si fueran 64 casillas o canales donde se van distribuyendo los eventos aleatorios que genera el proceso de Poisson. Estas tres variables son la base para la simulación que vamos a correr más adelante.

[2]:
```
tasa = 8
iteraciones = 1_000_000
num_preambulos = 64
```

Ahora lo que hacemos es simular las llegadas de eventos usando la función np.random.poisson. Esta función genera números aleatorios que siguen una distribución de Poisson, que es justo lo que necesitamos para nuestro modelo.

[3]:
```
llegadas = np.random.poisson(lam=tasa, size=iteraciones)
```

2 Distribucion empirica

primero usamos `np.max(llegadas)` para obtener el valor máximo de eventos que ocurrió en todas las simulaciones, lo cual nos dice hasta dónde debemos contar. Luego, con `np.bincount(llegadas, minlength=max_k + 1)`, contamos cuántas veces ocurrió cada número de eventos (0, 1, 2, …, max_k) entre las 1,000,000 de simulaciones; el parámetro `minlength` asegura que se incluya cada posible cantidad de eventos aunque alguna no haya ocurrido. Finalmente, dividimos cada conteo por el total de iteraciones usando `conteos / iteraciones` para obtener la **probabilidad empírica** de cada número de eventos, es decir, la frecuencia relativa observada en la simulación para cada posible valor de llegadas.

[4]:
```
max_k = np.max(llegadas)
conteos = np.bincount(llegadas, minlength=max_k + 1)
prob_emp = conteos / iteraciones
```

3 Distribucion teórica

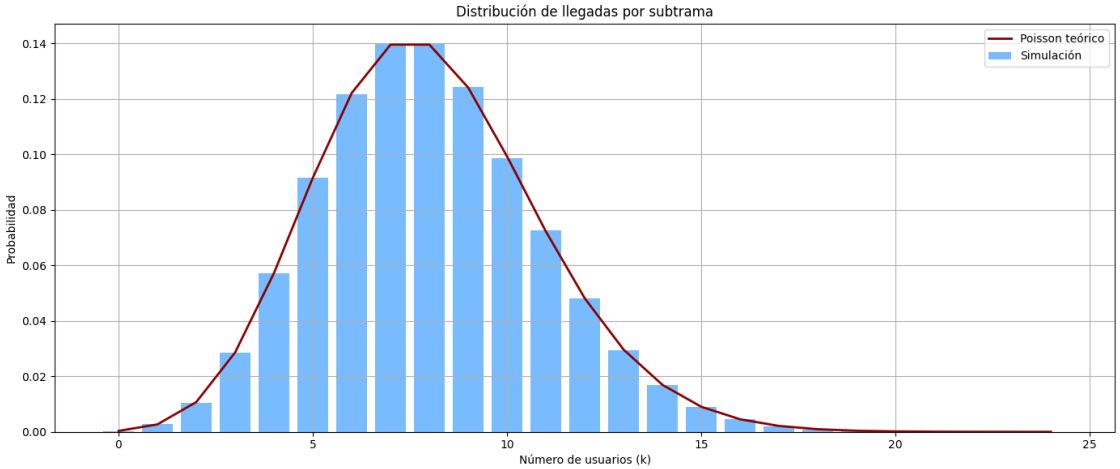
Ahora generamos los valores necesarios para calcular la **distribución teórica** de Poisson y compararla con la simulación. Primero, con `k_vals = np.arange(max_k + 1)` creamos un array que contiene todos los valores posibles de eventos desde 0 hasta `max_k`, que son los mismos que usamos al contar en la parte anterior. Luego, con `poisson.pmf(k_vals, mu=tasa)` calculamos la **probabilidad teórica** de que ocurran 0, 1, 2, …, `max_k` eventos usando la función de masa de probabilidad (pmf) de la distribución de Poisson. Esta función toma cada valor en `k_vals` y calcula su probabilidad según una tasa promedio de eventos igual a `tasa` (en este caso, 8). Así, `prob_teo` contiene la distribución teórica que esperamos ver si los datos realmente siguen un proceso de Poisson.

[5]:
```
k_vals = np.arange(max_k + 1)
prob_teo = poisson.pmf(k_vals, mu=tasa)
```

4 Grafica comparativa

Graficamos la comparación entre la distribución empírica y la distribución teórica de Poisson para visualizar qué tan bien se ajusta la simulación al modelo teórico. Primero, con plt.figure(figsize=(14, 6)) creamos una figura de tamaño amplio para que el gráfico se vea claro. Luego, usamos plt.bar(…) para dibujar un gráfico de barras con los valores de prob_emp, que representan las probabilidades obtenidas por simulación; se usa un color azul (dodgerblue) con algo de transparencia (alpha=0.6). A continuación, con plt.plot(…) dibujamos una línea roja (darkred) más gruesa (lw=2) que representa la distribución teórica calculada con la fórmula de Poisson. El título del gráfico se establece como “Distribución de llegadas por subtrama”, y se etiquetan los ejes: el eje X muestra el número de usuarios (o eventos, k), y el eje Y muestra la probabilidad correspondiente. Se agrega una leyenda con plt.legend(), se activa la cuadrícula para mejor lectura con plt.grid(True), se ajusta el diseño automáticamente con plt.tight_layout(), y finalmente se muestra el gráfico con plt.show(). Con esto, podemos comparar visualmente cómo se comporta la simulación frente a la teoría.

[6]:
```
plt.figure(figsize=(14, 6))
plt.bar(k_vals, prob_emp, alpha=0.6, label="Simulación", color='dodgerblue')
plt.plot(k_vals, prob_teo, color='darkred', lw=2, label="Poisson teórico")
plt.title("Distribución de llegadas por subtrama")
plt.xlabel("Número de usuarios (k)")
plt.ylabel("Probabilidad")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



5 Calculo de colisiones

estimamos la probabilidad de colisión cuando varios usuarios intentan acceder a uno de los 64 preámbulos disponibles (que podrías pensar como canales, ranuras o recursos).

Primero, inicializamos una variable colisiones = 0 para contar cuántas veces ocurre una colisión. Luego, recorremos cada valor en llegadas, que representa el número de usuarios que llegaron en una subtrama. Si el número de usuarios es mayor al número de preámbulos (num_preambulos = 64), entonces sabemos que habrá al menos una colisión, porque hay más usuarios que recursos, así que sumamos 1 a colisiones.

Si el número de usuarios es menor o igual a 64, entonces simulamos cómo cada usuario elige un preámbulo al azar, permitiendo repeticiones (con replace=True). Esto se hace con np.random.choice(…), que simula las elecciones de los usuarios. Si luego convertimos esas elecciones en un conjunto (set(elecciones)), y el tamaño del conjunto es menor que el número de usuarios, significa que al menos dos eligieron el mismo preámbulo, es decir, hubo una colisión, así que también sumamos 1 a colisiones.

Al final, calculamos la probabilidad empírica de colisión como colisiones / iteraciones y la mostramos en pantalla con 5 decimales. En resumen, esta parte del código evalúa cuán probable es que, en

un sistema con recursos limitados, haya colisiones cuando múltiples usuarios intentan acceder al mismo tiempo.

[10]:
```
colisiones = 0

for usuarios in llegadas:
    if usuarios > num_preambulos:
        colisiones += 1
    else:
        elecciones = np.random.choice(num_preambulos, size=usuarios,␣
 ↪replace=True)
        if len(set(elecciones)) < usuarios:
            colisiones += 1

prob_colision = colisiones / iteraciones
print(f"Probabilidad estimada de colisión: {prob_colision:.5f}")
```

```
Probabilidad estimada de colisión: 0.37046
```

6 Cálculo teórico de colisión condicional

calculamos la probabilidad teórica de que ocurra una colisión, pero condicionada a que haya exactamente n usuarios intentando acceder a los preámbulos. Primero, definimos un rango de usuarios de 0 a 29 y creamos una lista vacía para guardar esas probabilidades. Para 0 o 1 usuario, la probabilidad de colisión es cero, porque no puede haber choque con tan pocos usuarios. Para más de 1 usuario, calculamos la probabilidad de que todos elijan preámbulos diferentes, multiplicando las probabilidades de que cada nuevo usuario seleccione un preámbulo distinto a los ya elegidos. Luego, la probabilidad de colisión es simplemente 1 menos esa probabilidad de no colisión, y la vamos almacenando para cada número de usuarios. Básicamente, este cálculo teórico nos dice qué tan probable es que haya una colisión dependiendo del número exacto de usuarios que intentan acceder.

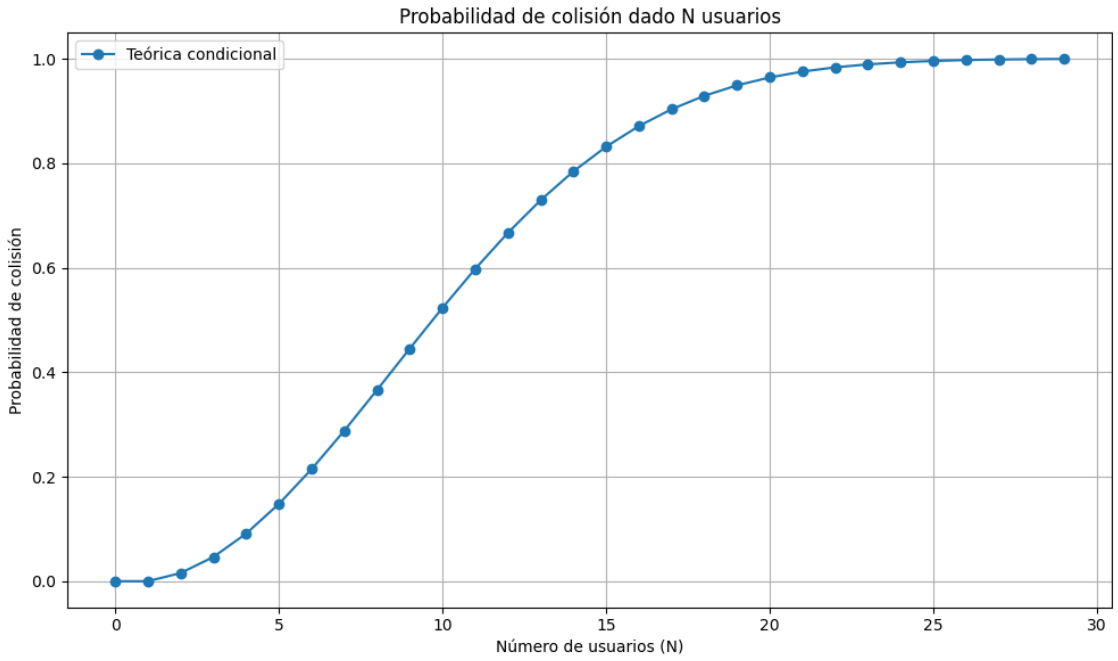
[8]:
```
n_vals = np.arange(0, 30)
prob_col_teo = []

for n in n_vals:
    if n <= 1:
        prob_col_teo.append(0)
    else:
        sin_colision = 1.0
        for i in range(n):
            sin_colision *= (1 - i / num_preambulos)
        prob_col_teo.append(1 - sin_colision)
```

7 Gráfico: P(colisión | N = k)

En este bloque creamos un gráfico que muestra cómo varía la probabilidad de colisión **condicional** según el número exacto de usuarios que intentan acceder a los preámbulos. Primero, con

`plt.figure(figsize=(10, 6))` definimos el tamaño del gráfico para que sea claro y legible. Luego, usamos `plt.plot(n_vals, prob_col_teo, marker='o', label="Teórica condicional")` para dibujar una curva con puntos marcados, donde el eje X representa el número de usuarios N y el eje Y la probabilidad de colisión calculada teóricamente para cada N. Se agregan título, etiquetas a los ejes, una cuadrícula para facilitar la lectura y una leyenda para identificar la curva. Finalmente, con `plt.tight_layout()` ajustamos el diseño y con `plt.show()` mostramos el gráfico. Así, visualizamos claramente cómo la probabilidad de colisión crece conforme aumentan los usuarios.

[9]:
```
plt.figure(figsize=(10, 6))
plt.plot(n_vals, prob_col_teo, marker='o', label="Teórica condicional")
plt.title("Probabilidad de colisión dado N usuarios")
plt.xlabel("Número de usuarios (N)")
plt.ylabel("Probabilidad de colisión")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```



8 Conclusión

En resumen, mediante la simulación del proceso de Poisson con una tasa promedio de eventos, pudimos modelar y visualizar la distribución de llegadas de usuarios a un conjunto limitado de preámbulos. Los resultados simulados se ajustaron muy bien a la distribución teórica de Poisson, lo que valida la precisión del modelo para representar este tipo de eventos aleatorios. Además, estimamos la probabilidad de colisión cuando múltiples usuarios compiten por recursos limitados,

mostrando que la probabilidad de colisión aumenta a medida que crece el número de usuarios. También calculamos teóricamente esta probabilidad condicional y la representamos gráficamente, confirmando que con más usuarios es cada vez más probable que dos o más elijan el mismo preámbulo. En conjunto, este análisis nos permite entender y cuantificar el comportamiento de sistemas con accesos concurrentes y recursos limitados, siendo útil para optimizar el diseño y la gestión de redes o sistemas similares.