# Proyecto Teoría de cola Joe

July 31, 2025

### 0.0.1 Proyecto Teoría de cola

**Introducción** En este trabajo se desarrolla una simulación computacional de un sistema de colas M/M/1/K utilizando Python y la biblioteca NumPy. Este tipo de modelo permite analizar el comportamiento de un sistema de atención con una sola línea de servicio y capacidad limitada. Se simulan llegadas y tiempos de servicio usando distribuciones exponenciales, permitiendo estudiar fenómenos reales como bloqueos por saturación, tiempos de espera y utilización del servidor. El objetivo principal es comprender cómo afectan diferentes tasas de servicio al rendimiento del sistema, y cómo se pueden tomar decisiones más informadas para mejorar su eficiencia.

[1]:
```
import numpy as np
```

NumPy se usa para generar tiempos aleatorios, convertirlos en momentos de llegada, y calcular promedios del sistema, todo de forma rápida y eficiente.

### 0.0.2 PASO 1: FUNCIÓN PRINCIPAL: simular_cola_mm1k

[2]:
```
def simular_cola_mm1k(tasa_llegada, tasa_servicio, capacidad_max, num_eventos):
 tiempos_llegada = np.cumsum(np.random.exponential(1 /␣
 ↪tasa_llegada,num_eventos))
 duraciones_servicio = np.random.exponential(1 / tasa_servicio, num_eventos)
```

La función simular_cola_mm1k modela una cola con capacidad limitada donde clientes llegan y son atendidos según tasas promedio. Recibe como parámetros la tasa de llegada (), la tasa de servicio (), la capacidad máxima del sistema y la cantidad de clientes a simular. Sirve para analizar cómo se comporta el sistema bajo distintas condiciones.

### PASO 2: Generación de datos de entrada de llegada y de servicio

Para simular la llegada y atención de clientes, se generan dos listas: una con los tiempos de llegada acumulados usando distribución exponencial con media 1/ , y otra con las duraciones del servicio para cada cliente, también con distribución exponencial y media 1/ . Así se establece cuándo llega cada cliente y cuánto tiempo tomará atenderlo

### 0.0.3 PASO 3: Inicialización de variables

[3]:
```
 tiempo_actual = 0
 tiempo_fin_servicio = 0
 cola_actual = 0
```

En la etapa de inicialización, se definen variables clave para el seguimiento del estado del sistema durante la simulación. tiempo_actual representa el tiempo global, aunque no se utiliza directa mente; mientras que tiempo_fin_servicio indica el momento en que finalizará el servicio del clien en curso. Por su parte, cola_actual permite llevar un control de cuántos clientes están esperando en la cola en un momento dado.

[4]:
```
bloqueos = 0
cola_total = []
tiempos_espera = []
```

Luego, se preparan tres variables para recolectar datos de la simulación: bloqueos, que registra la cantidad de veces que un cliente fue rechazado por alcanzar el límite de capacidad; cola_total, que almacena el tamaño de la cola en cada evento para evaluar su evolución; y tiempos_espera, donde se guarda cuánto tiempo esperó cada cliente que logró ser atendido. Estas variables son fundamentales para el análisis del rendimiento del sistema.

**BUCLE PRINCIPAL DE LA SIMULACIÓN (PROCESAMIENTO DE CADA CLIENT)E** En cada iteración del ciclo, se obtiene el tiempo de llegada y la duración del servicio de un cliente desde sus respectivas listas. Estos datos permiten simular su paso por el sistema.

[ ]:
```
 for i in range(num_eventos):
        llegada = tiempos_llegada[i]
        servicio = duraciones_servicio[i]
```

Cuando el cliente llega y el servidor está libre, se le atiende de inmediato sin esperar. Se actualiza el tiempo de inicio y fin del servicio, se vacía la cola, y se registra un tiempo de espera igual a cero.

[ ]:
```
import numpy as np

# Parámetros de simulación
tasa_llegada = 180
tasa_servicio = 200
capacidad_max = 5
num_eventos = 100

# Generar datos
tiempos_llegada = np.cumsum(np.random.exponential(1 / tasa_llegada,␣
 ↪num_eventos))
duraciones_servicio = np.random.exponential(1 / tasa_servicio, num_eventos)

# Variables de estado
tiempo_fin_servicio = 0
cola_actual = 0
tiempos_espera = []

# Simulación
for i in range(num_eventos):
    llegada = tiempos_llegada[i]
```

```
    servicio = duraciones_servicio[i]

    if llegada >= tiempo_fin_servicio:
        # Servidor libre
        tiempo_inicio = llegada
        tiempo_fin_servicio = tiempo_inicio + servicio
        cola_actual = 0
        tiempos_espera.append(0)

    elif cola_actual < capacidad_max:
        # Hay espacio en la cola
        tiempo_inicio = tiempo_fin_servicio
        tiempo_fin_servicio += servicio
        cola_actual += 1
        tiempos_espera.append(tiempo_inicio - llegada)

    else:
        # Cola llena (bloqueo)
        print(f"Cliente {i+1} fue bloqueado")

print("Simulación completada.")
```

Este fragmento representa la situación en que el sistema ha alcanzado su capacidad máxima y no puede recibir más clientes. Cuando eso ocurre, el cliente que llega no es atendido ni espera en la cola: simplemente se registra como un bloqueo (bloqueos += 1), indicando que fue rechazado. También se guarda el estado actual de la cola (cola_total.append(cola_actual)) para fines de análisis o estadísticas, y luego el sistema pasa al siguiente evento sin modificar el estado de servicio ni la cola. Es decir, el cliente no entra al sistema y solo se documenta el intento fallido de ingreso.

Se registra la cantidad de clientes en la cola después de cada evento, siempre que no haya sido un bloqueo, para analizar el comportamiento del sistema.

[ ]:
```
cola_total.append(cola_actual)
```

Este bloque de código calcula las métricas clave para evaluar el desempeño del sistema de atención. Primero, determina cuántos clientes fueron efectivamente procesados restando los bloqueos del total de eventos. Luego ]calcula la probabilidad de bloqueo, que indica la proporción de clientes que no pudieron ingresar al sistema. Con esa probabilidad, se estima la utilización del servidor, es decir, cuánto tiempo está realmente ocupado atendiendo. También se obtiene el promedio de longitud de la cola, analizando la cantidad de clientes esperando a lo largo del tiempo, y el promedio de espera, que mide cuánto tiempo en promedio esperó cada cliente antes de ser atendido. Estas métricas son fundamentales para entender si el sistema está operando de forma eficiente o si necesita ajustes.

[ ]:
```
procesados = num_eventos - bloqueos
prob_bloqueo = bloqueos / num_eventos
utilizacion = tasa_llegada * (1 - prob_bloqueo) / tasa_servicio
promedio_cola = np.mean(cola_total)
promedio_espera = np.mean(tiempos_espera) if tiempos_espera else 0
```

Se prueban dos velocidades de atención para comparar cómo cambia el rendimiento del sistema.

[ ]:
```python
# Función para ejecutar múltiples simulaciones
def ejecutar_simulaciones():
    lambda_llegadas = 180
    mu1 = 200
    mu2 = 400
    capacidad = 5
    total_eventos = 100_000

    for mu in [mu1, mu2]:
        print(f"\n Simulando para tasa de servicio  = {mu}")
        p_b, uso, prom_cola, prom_esp = simular_cola_mm1k(lambda_llegadas, mu,␣
 ↪capacidad, total_eventos)

        print(f" Probabilidad de bloqueo: {p_b:.4f}")
        print(f" Utilización promedio: {uso:.4f}")
        print(f" Longitud promedio de cola: {prom_cola:.4f}")
        print(f" Tiempo promedio de espera: {prom_esp:.4f} segundos")

# Ejecutar la simulación
ejecutar_simulaciones()
```

[ ]:
```
Simulando para tasa de servicio  = 200
Probabilidad de bloqueo:     0.2275
Utilización promedio:        0.6953
Longitud promedio de cola:   2.3473
Tiempo promedio de espera:   0.0057 s

Simulando para tasa de servicio  = 400
Probabilidad de bloqueo:     0.0413
Utilización promedio:        0.4314
Longitud promedio de cola:   1.0077
Tiempo promedio de espera:   0.0015 s
```

**Conclusión (puntos clave)** Se implementó una simulación eficiente para el modelo M/M/1/K, útil para estudiar colas con capacidad limitada.

La tasa de servicio impacta directamente en la probabilidad de bloqueo: mayor velocidad de atención reduce los rechazos.

El tiempo promedio de espera y la longitud de la cola disminuyen cuando se incrementa la tasa de servicio, mejorando la experiencia del cliente.

La utilización del servidor también varía: con tasas de servicio más altas, el sistema se mantiene menos ocupado, lo que puede implicar sobredimensionamiento.

Estas métricas permiten evaluar el rendimiento del sistema y tomar decisiones de diseño más acertadas para mejorar la atención y evitar saturaciones.

[ ]: