Aplicación de MCMC

Lucas Cedric Cervantes Beutelspacher
Andrés Urbano Guillermo Gerardo
Elsy Camila Silva Velázquez

3 de Diciembre del 2021

Introducción

Las técnias de Monte Carlo vía cadenas de Markov permiten generar, de menera iterativa, observaciones de distribuciones multivariadas que difícilmente podrían simularse.

La idea básica es construir una cadena de Markov que sea fácil de simular y cuya distribución estacionaria corresponda a la distribución objetivo que nos interesa. De esta manera, al implementar correctamente el algoritmo, la convergencia de la cadena está garantizada, independientemente de cuáles sean los valores iniciales.

Las ideas de esta técnica han tenido su desarrollo histórico desde diferentes ramas de las matematicas: de la probabilidad, de la mecánica estadística y de la estadística bayesiana. Cada enfoque tiene su utilidad conceptual y de implementación, y se enriquecen entre ellos.

Conceptos importantes ha recordar:

$$P_{i,j}^{(n)} = P(X_n = j | X_0 = 1) = (M^n)_{i,j}$$

- Una clase de comunicación es una relación de equivalencia i $\sim$ j.

$$i \sim j \quad \text{sí y solo sí} \quad i \overset{\text{acc}}{\Longrightarrow} j \wedge j \overset{\text{acc}}{\Longrightarrow} i$$

$i \overset{\text{acc}}{\Longrightarrow} j$ significa que "j es accesible desde i"si $\exists n > 0$ y $P_{i,j}^{(n)} > 0$.

- Una cadena es irreducible si sólo tiene una clase de comunicación.
- Periodo de un estado:
$$per(i) = MCD\{n : P_{i,i}^{(n)} > 0\}$$
- Estado aperiódico:
$$\text{Sí} \quad per(i) = 1$$

Proposición. Sea $(X_n)_{n\geq 0}$ una cadena de Markov homogénea, irreducible y aperiódica con espacio de estados E y distribución estacionaria π, se tiene que cuando $n \to \infty$ ocurre que

$$X_n \xrightarrow{\text{d}} X$$

donde X tiene distribución π.

Algoritmo Metrópolis Hastings

El algoritmo consiste en obtener una nueva caminata cuya distribución estacionaria será π. Construimos una cadena de Markov $(X_n)_{n\geq 0}$ con espacio de estados E, cuya evolución depende de la matriz de transición $Q = \{q_i, j\}$ de otra cadena de Markov irreducible. Esta otra cadena de Markov tiene como matriz de transición a $P = \{p_{ij}\}_{i,j\in E}$ donde:

$$p_{i,j} = \begin{cases} q_{ij}\alpha_{ij} & \text{si} \quad i \neq \\ q_{ii} + \sum_{k\neq i} q_{ik}(1-\alpha_{ik}) & \text{si} \quad i = j \end{cases}$$

$$\alpha_{i,j} = min\Big\{\frac{\pi_j q_{ji}}{\pi_i q_{ij}}, 1\Big\} \qquad \text{Probabilidad de aceptación}$$

Los pasos del algoritmo son:

- Comienza en X_0
- A cada paso usar la cadena q_{ij}
- Luego con una probabilidad α_{ij} acepta moverte a j, si no quedarse en i.

*Proposición.*La cadena de Markov $\{X_n\}_{n\geq 0}$ tiene como distribución estacionaria a π.

Demostracion. Demostraremos que la cadena de Markov con matriz de transición $P = (P_{i,j})_{i,j\in E}$ es reversible (con respecto al tiempo) y tiene como distribución estacionaria a π. Para ello basta verificar que se satisfacen las ecuaciones de balanceo local:

$$\pi_i p_{i,j} = \pi_j p_{j,i}$$

para toda $i, j \in E$. Esto es equivalente a que se cumpla:

$$\pi_i p_{i,j} \alpha_{i,j} = \pi_j p_{j,i} \alpha_{j,i}$$

Para verificar esta última igualdad basta notar que si

$$\alpha_{i,j} = \frac{\pi_j q_{ji}}{\pi_i q_{ij}}$$

entonces $\pi_{ji} = 1$ y viceversa.

Desarrollo de la práctica

Considere un tablero de 8×8; una configuración del tablero es una asignación de colores (blanco y negro) en las casillas (C : {1, 2, . . . , 64} → 0, 1).

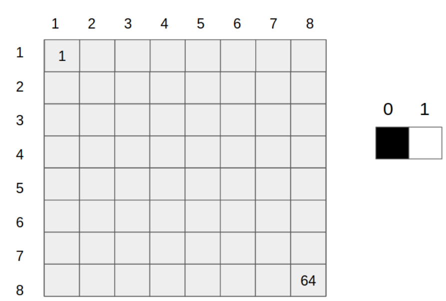

Figura 1: Tablero de 8x8

Decimos que la configuración es aceptable si no hay dos cuadritos adyacentes de color negro (por ejemplo, un cuadrito genérico tiene 4 cuadritos adyacentes y cada esquina tiene dos cuadritos adyacentes).

Trabajaremos con el espacio de estados C que reúne a todas las configuraciones aceptables. Considere una caminata aleatoria $(X_n)n \geq 0$ con espacio de estados en C que satisface:

- La configuración inicial X_0 tiene todos los cuadritos blancos.
- Dado que conocemos X_n, construimos X_{n+1} de la siguiente forma:
  1. Elige una casilla c uniformemente al azar y lanza una moneda justa.
  2. Si la moneda sale águila entonces $X_{n+1} = X_n$.

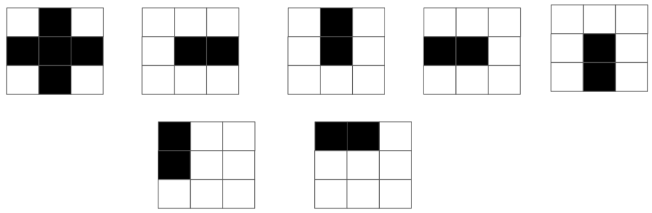

Figura 2: Algunas configuraciones no aceptables

3. Si la moneda sale sol entonces intercambiamos el color de la casilla c (de blanco a negro o viceversa) y, sólo si esto genera una configuración aceptable, ésta es la siguiente configuración X_{n+1}.
4. en otro caso hacemos $X_{n+1} = X_n$.

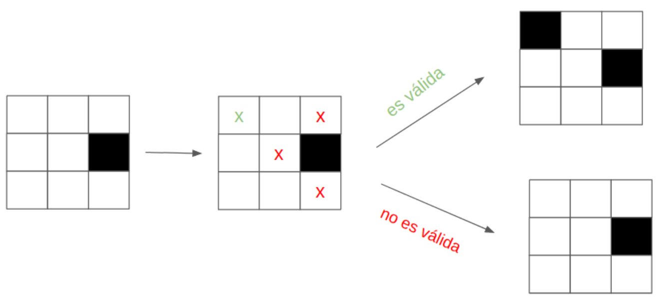

Figura 3: Ejemplo de un movimiento

Preguntas

1. Simule la cadena de Markov y grafique las configuraciones obtenidas a los tiempos 10,50,100,800,1000 para 4 realizaciones distintas.

2. Defina C_n como el número de cuadritos negros en la configuración X_n y grafique los promedios de C_n para 500 simulaciones de la cadena de Markov.

3. En base a los resultados obtenidos, determine un tiempo de paro T, que puede ser aleatorio o determinista, para el cuál la configuración X_T tiene una distribución suficientemente cercana a la uniforme en el espacio de configuraciones C.

Implementación

Para la caminata se usa un tiempo final y un ciclo *while*. De esta manera podemos definir cuantos estados de la cadena vamos a revisar. Usamos una lista X en donde guardamos todos los estados de la cadena que visitemos. Usamos una matriz booleana de 8x8 que representara nuestro tablero. Además tendremos una lista adicional para guardar el número de cuadros negros que tienen los tableros.

La caminata

Dentro de cada iteración de la caminata lo primero que tenemos que hacer es un volado para decidir si nos quedaremos en el mismo estado ($X_{n+1} = X_n$) sin realizar ninguna acción o invertimos una casilla al azar y hacemos las acciones correspondientes. Simularemos este volado con una $Binomial(1/2)$ y en caso de que salga águila nos quedaremos en el mismo estado, de lo contrario obtendremos dos enteros aleatorios de 0 a 7 que usaremos como las coordenadas aleatorias de la casilla de nuestra matriz booleana que vamos a invertir.

```
# iniciamos la caminata
while t <= Tf:
    # hacemos el volado
    ## Aguila = 0, Sol = 1
    volado = np.random.binomial(1,.5,1)[0]

    if volado == 0:     # Si sale aguila tenemos que X_{n+1} = X_{n}
        X.append(X[-1].copy())
    else:               # Si sale sol invertimos la casilla
        # Selecionamos una casilla al azar
        i,j = np.random.randint(0,8,2)
        ...
```

Invertiremos la casilla usando un operador *not* y evaluaremos los posibles tableros que resultan para ver si son válidos o no. En caso de ser un tablero valido, tomaremos ese tablero como el siguiente estado de nuestra cadena de Markov ($X_{n+1} = Nuevo\_Tablero$). Si no es válido nos vamos a quedar en el estado en el que estamos ($X_{n+1} = X_n$).

```
    # Invertimos esa casilla
    Cm[i,j] = not Cm[i,j]
    # Si la casilla invertida es negra tenemos que verificar
    # que sea un arreglo valido
    if Cm[i,j]:
        valido = True
        for n in range(4):      # Revisamos las 4 casillas adyacentes
            ...
```

El primer caso es que la casilla que invertimos paso de ser negra a ser blanca en cuyo caso sabemos que da como resultado una configuración valida; tomamos el nuevo tablero como el siguiente estado de la cadena. Si la casilla pasó de ser blanca a ser negra tenemos que ver que ninguna de las casillas adyacentes sea negra. Usaremos una variable booleana para ver la validez del tablero y usando un for con 4 iteraciones revisaremos las 4, 3, o 2 posibles casillas adyacentes. Nuestra variable booleana será inicializada en un valor Verdadero (asumimos que es verdadera), y dentro de las verificaciones si encontramos que alguna casilla adyacente es negra le asignaremos un valor Falso a la variable booleana y terminaremos el ciclo.

```
        for n in range(4):      # Revisamos las 4 casillas adyacentes
                try :
                    if n == 0:
                        if Cm[i+1,j]:
                            valido = False
                            break
                    elif n == 1:
                        if Cm[i-1,j]:
                            valido = False
```

```
                            break
                    elif n == 2:
                        if Cm[i,j+1]:
                            valido = False
                            break
                    else:
                        if Cm[i,j-1]:
                            valido = False
                            break
                except IndexError:
                    pass
```

Al terminar el ciclo for veremos si la variable booleana es verdadera (ninguna casilla adyacente es negra) tomaremos ese tablero como el siguiente estado de la cadena; si es falsa (al menos una casilla adyacente es negra) optaremos por quedarnos en el estado en el que estamos.

```
            if valido:      # Si es un arreglo valido lo asignamos como X_{n+1}
                X.append(Cm.copy())
            else:           # Si no es valido tenemos que X_{n+1} = X_{n}
                X.append(X[-1].copy())
                Cm=X[-1].copy()
```

Con este código es suficiente para simular la cadena, pero como queremos guardar cierta información adicional, añadiremos un poco más de código. Empezando por agregar a nuestra lista de cuantos cuadros negros hay en cada tablero el número de cuadros negros que tenga el estado que definimos como X_{n+1} ya sea X_n o un tablero nuevo.

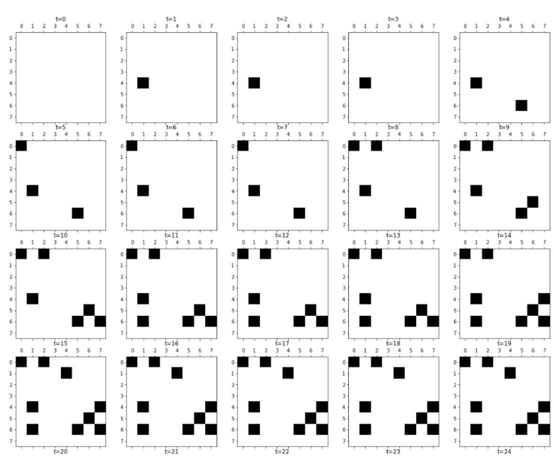

Figura 4: Muestra de la simulación

Además, si nos interesa ver algún/algunos estados lo podemos hacer usando una lista con los tiempos/estados que queremos observar y usando el operador *in* que nos proporciona Python podremos ver cuando el tiempo actual de la cadena es uno de los que nos interesan. Posteriormente podremos hacer lo que queramos ya sea imprimir información del tablero, guardar el tablero en otra lista auxiliar, etc.

Nota: todas las funciones aleatorias que usamos se encuentran dentro de $Numpy$ en la biblioteca $random$.

Con el código que acabamos de realizar crearemos una función que realice una caminata hasta un tiempo final que la función tomara como parámetro; regresara un arreglo de *numpy* con los conteos de cuadros negros en cada tiempo y una lista de matrices que serán cada uno de los estados de la cadena. Llamaremos a la función $CaminataTableros$.

```
def CaminataTableros(Cm=None,Tf=1000,Tiempos=None,):
    ...
```

Con esta función definida ahora realizaremos 500 caminatas para poder contar el número promedio de cuadros negros en cada tiempo de la cadena.

Esto l implementamos mediante un ciclo for con 500 iteraciones. Como la función $CaminataTableros$ regresa un arreglo con la cantidad de cuadros negros para tiempo de esa caminata vamos a usar una lista para guardar todos los 500 arreglos que calcularemos. Como estamos usado arreglos de *numpy* podemos directamente sumarlos ya que la suma esta vectorizada y podemos dividirlos entre un escalar por lo mismo y asi obtener rápidamente el promedio de cuadros negros para cada uno de los tiempos.

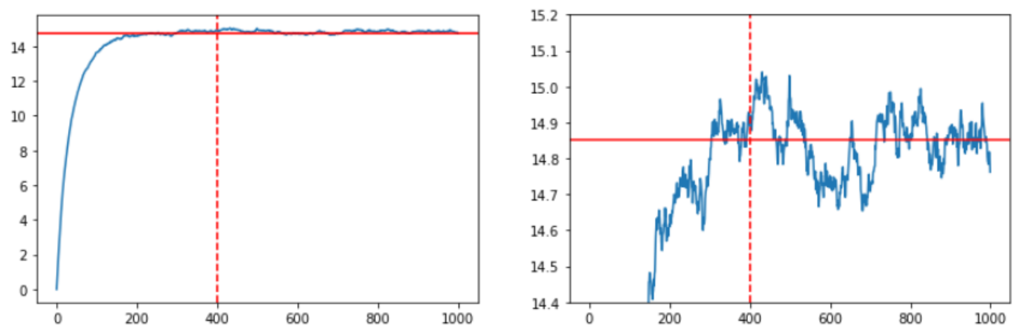

Figura 5: Gráfica de los tiempos

Al graficar los promedios de cuadros negros para 500 simulaciones podemos ver como después de cierto valore este promedio oscila alrededor de 14,75.

A parir del tiempo 400 la caminata que definimos se queda en estos valores, en otras palabras alcanza una distribución estacionaria. Esta distribución es la que buscamos en un principio ya que nos garantiza que todas las muestras que obtengamos de ella pertenecerán a la distribución que no conocemos.

Tomando en cuenta este tiempo podemos cortar nuestra caminata en 400 y solo tomar lo que tenemos después como muestras de nuestra distribución.

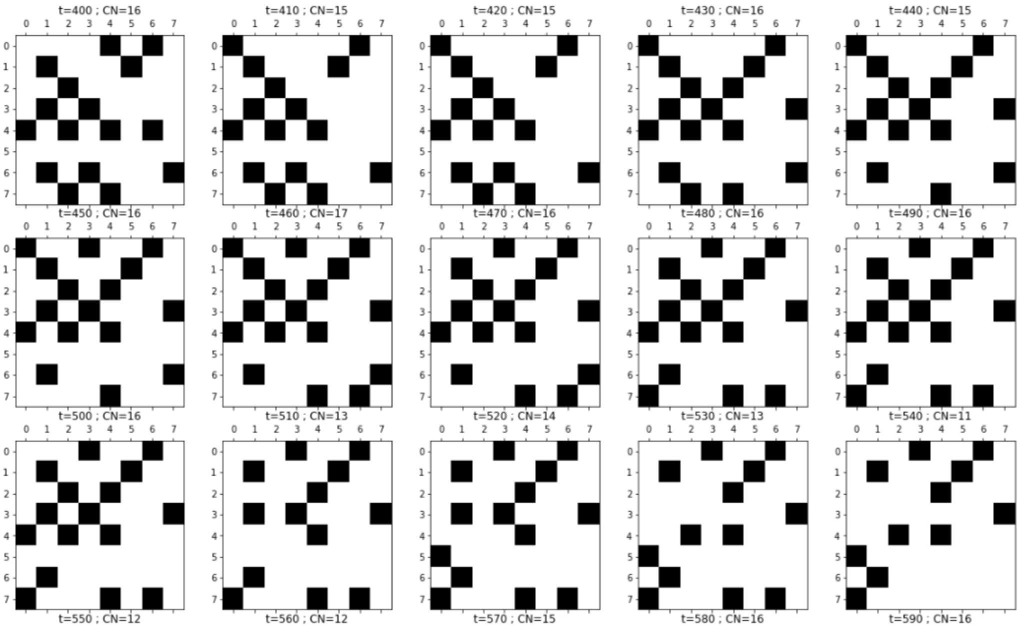

Figura 6: Tablero con muchas simulaciones

Visualizaciones

Para poder visualizar los tableros usaremos la biblioteca *pyplot* dentro de *matplotlib*. La función principal que vamos a usar es *matshow* que nos permite graficar una matiz de forma sencilla. Y usaremos *subplots* para poder mostrar más de una matriz a la vez.

También usaremos *plot* para graficar los promedios de cuadros negros en nuestras 500 simulaciones.

Referencias

[1] Rincon, Luis: *Introducción a los procesos estocásticos*. las prensas de Ciencias.

[2] *Markov Chain Monte Carlo (MCMC) : Data Science Concepts*. `https://youtu.be/yApmR-c_hKU`. Accedido en Diciembre 1 del 2021.