Cómputo Científico para Probabilidad, Estadística y Ciencia de Datos

Ezau Faridh Torres Torres

TAREA 9: MCMC: Tarea Final

Fecha de entrega: 04/Dic/2024.

En ambos problemas hay que diseñar e implementar el MCMC, investigar sobre su convergencia y tener algún grado de certeza sobre si sí se está simulando de la posterior correspondiente. Más aun, recuerde que se trata de un problema de inferencia: Hay que hablar del problema en si, comentar sobre las posteriores simuladas y posibles estimadores (a partir de la muestra de posterior) que se pueden proporcionar de cada parámetro.

NOTA: Los ejercicios se encuentran repartidos en los archivos:

- ejercicio1_tarea9.py
- ejercicio2_tarea9.py
- funciones_auxiliares.py

  Además, en ambos ejercicios se hace uso de funciones auxiliares, tanto para calcular el burn_in de la cadena, como para generar las diferentes gráficas. Estas funciones se encuentran en el archivo funciones_auxiliares.py y son descritas en la sección 4.

---

1 Ejercicio 1

(Problema en ecología) Sean $X_1, \dots X_m$ variables aleatorias donde X_i denota el número de individuos de una especie en cierta región. Suponga que $X_i|N, p \sim Binom(N, p)$, entonces

$$f(\bar{x} \mid N, p) = \prod_{i=1}^{m} \frac{N!}{x_i!\,(N - x_i)!} p^{x_i}(1-p)^{N-x_i}. \tag{1}$$

Asumiendo la distribución a priori $p \sim Beta(\alpha, \beta)$ y $N \sim h(\cdot)$, donde h es una distribución discreta en $\{0, 1, 2, \dots, N_{\max}\}$, se tiene definida la distribución posterior $f(N, P \mid \bar{x})$.

A partir del algoritmo MH, simule valores de la distribución posterior usando un kernel híbrido. Para ello considere **como sugerencia** la siguiente distribución inicial para el MCMC

$$p \sim \mathrm{U}(0,1) \quad \text{y} \quad N \sim \mathrm{U}_d\left\{ \max_{i\in\{1,\dots,m\}}(x_i)\,, \max_{i\in\{1,\dots,m\}}(x_i)+1, \dots, N_{\max} \right\} \tag{2}$$

y las propuestas

- Propuesta 1: De la condicional total de p (kernel Gibbs).
- Propuesta 2: De la a priori.
- Propuesta 3: Propuesta hipergeométrica (¿?).
- Propuesta 4: Poisson: $N_p \sim \max_{i\in\{1,\dots,m\}}(x_i) +$ Poisson(?).

- Propuesta 5: Caminata aleatoria

$$N_p = N + \epsilon, \quad \mathbb{P}(\epsilon = 1) = \frac{1}{2} = \mathbb{P}(\epsilon = -1) \tag{3}$$

Los datos son estos: $7, 8, 6, 5, 2, 8, 6, 6, 7, 4, 8, 8, 6, 4, 8, 8, 10, 5, 4, 2$.

A priori, esperamos que sea difícil observar a los individuos entonces $\alpha = 1$, $\beta = 20$. La especie no es muy abundante y entonces $N_{\max} = 1000$ y $h(N) = 1/(N_{\max} + 1)$; $N \in \{0, 1, 2, \ldots, N_{\max}\}$.

Las propuestas y distribución inicial para el MCMC de arriba son **solamente sugerencia**, propongan otras propuestas, experimenten y comenten.

Respuesta:

Sabemos que la verosimilitud conjunta de los datos observados $\bar{x}$ está dada por (1) y se asume que p tiene una distribución a priori beta:

$$p \sim Beta(\alpha, \beta), \tag{4}$$

donde $\alpha = 1$ y $\beta = 20$, lo que refleja una creencia previa de que p es pequeño (dificultad para observar individuos). Además, se asume que N sigue una distribución uniforme discreta en el rango $[0, 1, \ldots, N_{\max} = 1000]$:

$$h(N) = \frac{1}{N_{\max} + 1}. \tag{5}$$

La inferencia bayesiana se basa en calcular la distribución posterior de los parámetros N y p dados los datos observados $\bar{x}$. Por el teorema de Bayes:

$$f(N, p \mid \bar{x}) \propto f(\bar{x} \mid N, p) \cdot h(N) \cdot Beta(p; \alpha, \beta) \tag{6}$$

donde:

- $f(\bar{x} \mid N, p)$: Verosimilitud conjunta de los datos.
- $h(N)$: Prior de N (notemos que la distribución no depende explícitamente de N).
- $Beta(p; \alpha, \beta)$: Prior de p.

Reescribiendo (6), se tiene

$$\begin{aligned} f(N, p \mid \bar{x}) &\propto f(\bar{x} \mid N, p) \cdot h(N) \cdot Beta(p; \alpha, \beta) \propto f(\bar{x} \mid N, p) \cdot Beta(p; \alpha, \beta) \\ &\propto \prod_{i=1}^{m} \binom{N}{x_i} \cdot p^{\sum x_i} (1-p)^{mN - \sum x_i} \cdot p^{\alpha - 1} (1-p)^{\beta - 1} \cdot \mathbb{1}_{\{0,\ldots,N\}}(x_i) \mathbb{1}_{\{0,\ldots,N_{max}\}}(N) \mathbb{1}_{(0,1)}(p) \\ &\propto \prod_{i=1}^{m} \binom{N}{x_i} \cdot p^{\alpha + \sum x_i - 1} (1-p)^{\beta + mN - \sum x_i - 1} \cdot \mathbb{1}_{\{0,\ldots,N\}}(x_i) \mathbb{1}_{\{0,\ldots,N_{max}\}}(N) \mathbb{1}_{(0,1)}(p). \end{aligned} \tag{7}$$

Podemos notar que esto es proporcional a una distribución

$$Beta\left(\alpha + \sum x_i, \quad \beta + mN - \sum x_i\right) \tag{8}$$

para p dado N, la cual se puede usar como una propuesta Gibbs.

En el archivo ejercicio1_tarea9.py se implementa la función

METROPOLIS_HASTINGS_HYBRID_KERNELS(),

la cual aplica el algoritmo Metropolis-Hastings con kérneles híbridos para K propuestas simulando una cadena de Markov en $\mathbb{R}^n$ y toma los siguientes argumentos:

- La función objetivo f (para este ejercicio, se trata de la distribución posterior dada en (6)).
- Una lista de las K funciones que generan las propuestas $[q_{1_{gen}}, \ldots, q_{K_{gen}}]$.
- Lista de las K funciones que calculan la densidad de probabilidad de las propuestas $[q_{1_{pdf}}, \ldots, q_{K_{pdf}}]$. Es importante que se respete el mismo orden de la lista de funciones generadoras.
- Lista de las K probabilidades de seleccionar cada kernel de propuesta $[p_1, \ldots, p_K]$. Es importante que se respete el mismo orden de la lista de funciones generadoras y que $\sum_k p_k = 1$. Si esta lista no se da, se toman probabilidades iguales para cada kernel, i.e., $p_1 = \cdots = p_K = \frac{1}{K}$.
- El valor inicial x_0 en el soporte de la distribución objetivo.
- El número de iteraciones del algoritmo (casi siempre se usa $N = 10,000$).

La función regresa la cadena simulada en $\mathbb{R}^n$ e imprime la tasa de aceptación lograda y el conteo de aceptaciones por kernel.

A continuación, se implementa la función *posterior()* descrita por (6), que es nuestra distribución objetivo. Notemos que no se considera explícitamente la distribución $h(N)$ ya que la densidad $\frac{1}{N_{\max}+1}$ no depende explícitamente de N y se considera como constante de proporcionalidad.

Distribución inicial:

$$N \sim \mathrm{U}_d \left\{ \max_{i\in\{1,\ldots,m\}} (x_i), \max_{i\in\{1,\ldots,m\}} (x_i) + 1, \ldots, \max_{i\in\{1,\ldots,m\}} (x_i) + r \right\} \text{ y } p \sim U(0,1). \tag{9}$$

El ejercicio propone tomar como distribuciones iniciales a las dadas en la expresión (2). Debido a los buenos resultados en la mayoría de los experimentos, se tomó la decisión de mantener la distribución inicial de $p \sim U(0,1)$, sin embargo, se decidió cambiar la de N ya que el intervalo discreto $[x_{max} = 10, \ldots, N_{max} = 1000]$ es demasiado grande y de vez en cuando generaba saltos muy grandes en los valores iniciales de N. En su lugar se tomó la distribución inicial de N como vemos en (9) para $r < N_{max}$, elegible por el usuario (en este caso nos quedamos con $r = 120$). Esta distribución se implementa en la función *initial_distribution()* en el archivo ejercicio1_tarea9.py

Propuestas:

En el archivo ejercicio1_tarea9.py, se implementan las propuestas (*propi_gen()*), así como su densidad de probabilidad respectiva (*propi_pdf()*) y estas se describen a continuación. Inicialmente, se consideraron las propuestas dadas por el ejercicio mismo, pero experimentando con cada una y con la compatibilidad entre ellas, se realizaron los siguientes cambios que evitaron indeterminaciones y tuvieron resultados satisfactorios, aunque no únicos.

- Propuesta 1: Kérnel Gibbs para p dado N.

$$p \sim Beta\left(\alpha + \sum x_i, \quad \beta + mN - \sum x_i\right) \text{ y } N \text{ fijo.} \tag{10}$$

Se define la función *prop1_gen()* que implementa la distribución Gibbs (distribución condicional completa de p dado N y los datos $\bar{x}$) descrita en (8) y (10) siendo cuidadosos con el soporte de esta distribución ya que el parámetro $\beta + mN - \sum x_i$ debe ser positivo. En caso de que no, se propone $p \sim Beta(\alpha + \sum x_i, 1)$. También se define su densidad de probabilidad en la función *prop1_pdf()*. Se eligió conservar esta propuesta ya que un paso de Gibbs siempre acepta la propuesta porque la muestra proviene de la distribución condicional verdadera. Esto asegura exploración eficiente en la dimensión de p.

- Propuesta 2: Propuesta independiente y a priori para p.

$$N \sim \mathrm{U}_d\{x_{max}, \ldots, x_{max} + r\} \text{ y } p \sim Beta(\alpha = 1, \beta = 20). \tag{11}$$

El ejercicio recomienda usar como propuesta a las a prioris, sin embargo, dado que la distribución a priori de N es $h(N) = \frac{1}{1001}$, se tendrá una tasa de aceptación demasiado baja para cada valor propuesto. Por esto, se eligió usar a la distribución inicial para N dada por (9) para poder controlar la nueva probabilidad $h(N) = \frac{1}{r}$ con el valor de r (en particular, se tomó $r = 120$ ya que tuvo un buen desempeño). En cambio, para p se mantuvo la propuesta a priori: $p \sim Beta(\alpha = 1, \beta = 20)$. La función generadora y su función de densidad se definen en *prop2_gen()* y *prop2_pdf()* respectivamente. Se eligió esta propuesta, ya que, dado que es una propuesta independiente, introduce aleatoriedad independiente de los datos, lo que ayuda a evitar que la cadena quede atrapada en un modo local y permite exploración de regiones remotas del espacio de parámetros, fomentando una mejor cobertura de la posterior.

- Propuesta 3: Propuesta hipergeométrica.

$$N \sim Hipergeom(M = N_{max}, n = \sum x_i, N = m) \text{ y } p \text{ fijo.} \tag{12}$$

Se está modelando el tamaño poblacional N como una variable que sigue una distribución hipergeométrica. Los parámetros específicos son: $M = N_{max}$ ya que es el tamaño de la población máxima posible, $n = \sum x_i$ es el número total de éxitos en los datos observados y $N = m$ corresponde al número de datos observados. Se escogió de esta forma ya que los datos observados ($\sum x_i$ éxitos en m observaciones) sugieren que N debe estar acotado entre el valor máximo observado en los datos y N_{max}. La distribución hipergeométrica respeta esta restricción y se está trabajando con un conjunto finito de datos observados, que podemos considerar como una muestra extraída sin reemplazo de una población más grande. La función generadora y su función de densidad se definen en *prop3_gen()* y *prop3_pdf()* respectivamente.

- Propuesta 4: Poisson.

$$N_p \sim \max_{i \in \{1,\ldots,m\}} (x_i) + Poisson(\lambda) \text{ y } p \text{ fijo.} \tag{13}$$

Esta propuesta tuvo diversas variantes al momento de experimentar con ella. Inicialmente se consideró tomar el parámetro $\lambda = Np$ ya que, como sabemos, si N es grande y p pequeño, entonces $Poisson(Np)$ se comporta como una binomial con parámetros (N, p), que es exactamente lo que nos interesa. Sin embargo, causaba muchos problemas al momento de correr

el algoritmo, debido a que se veía mucha variabilidad en los valores de N y p, esto se observó al momento de generar las gráficas ya que se veía una posterior multimodal y no se observaba un buen ajuste a los datos (ver figuras siguientes).

Con la intención de tener una propuesta no independiente para N, también se consideró usar $\lambda = N$ ya que parecía una buena propuesta debido a que restringe la exploración de N a valores realistas y permite una transición más suave entre valores cercanos de N. Sin embargo, sucedían cosas similares al caso $\lambda = Np$, además de indeterminaciones y una tasa de aceptación bastante baja.

Finalmente, se decidió conservar una tasa λ constante. Esto permite ajustar el parámetro λ explícitamente, proporcionando control sobre la dispersión de N. en particular, se tomó $\lambda = 400$ ya que tuvo un buen desempeño, pero cualquier valor alrededor de este, funciona bien. La función generadora y su función de densidad se definen en *prop4_gen()* y *prop4_pdf()* respectivamente.

- Propuesta 5: Caminata aleatoria.

$$N_p = N + \epsilon, \quad \text{donde } \epsilon \sim \{-1, 1\} \text{ y } p \text{ fijo.} \tag{14}$$

Se decidió conservar esta propuesta ya que en cadenas de Markov, las propuestas simples y simétricas (caminatas aleatorias) son robustas para exploración local y garantizan una tasa de aceptación razonable. Sin embargo, su convergencia puede ser lenta si la posterior es amplia, pero para ese caso, ya se cuentan con otras propuestas que cuentan con más variabilidad. La función generadora y su función de densidad se definen en *prop5_gen()* y *prop5_pdf()* respectivamente.

Conclusión general de las propuestas:

Las propuestas 1 y 3 son informativas (dependen de los datos), lo que fomenta una exploración más dirigida. Las propuestas 2, 4 y 5 son más exploratorias, permitiendo cobertura global del espacio de parámetros. Las propuestas Gibbs y hipergeométrica aseguran explotación local eficiente, mientras que las propuestas uniformes y Poisson permiten escapar de nodos locales, promoviendo una mejor mezcla.

Por el Teorema de convergencia, sabemos que la cadena converge a la distribución posterior si:

- La cadena es irreducible (puede alcanzar cualquier estado en su espacio de parámetros).
- La cadena es aperiódica (no tiene ciclos regulares).
- Las propuestas son ergódicas (cubren todo el soporte de la posterior).

Y la elección de propuestas satisface estas condiciones porque:

- Hay propuestas que exploran globalmente (2, 4 y 5).
- Hay propuestas que exploran localmente (1 y 3).
- Todas las propuestas respetan las restricciones del soporte.

Cada propuesta tiene un rol específico en garantizar una buena mezcla y convergencia. Además, su combinación fomenta una cobertura eficiente del espacio de parámetros, lo que resulta en estimaciones robustas de la posterior, como se ve a continuación:

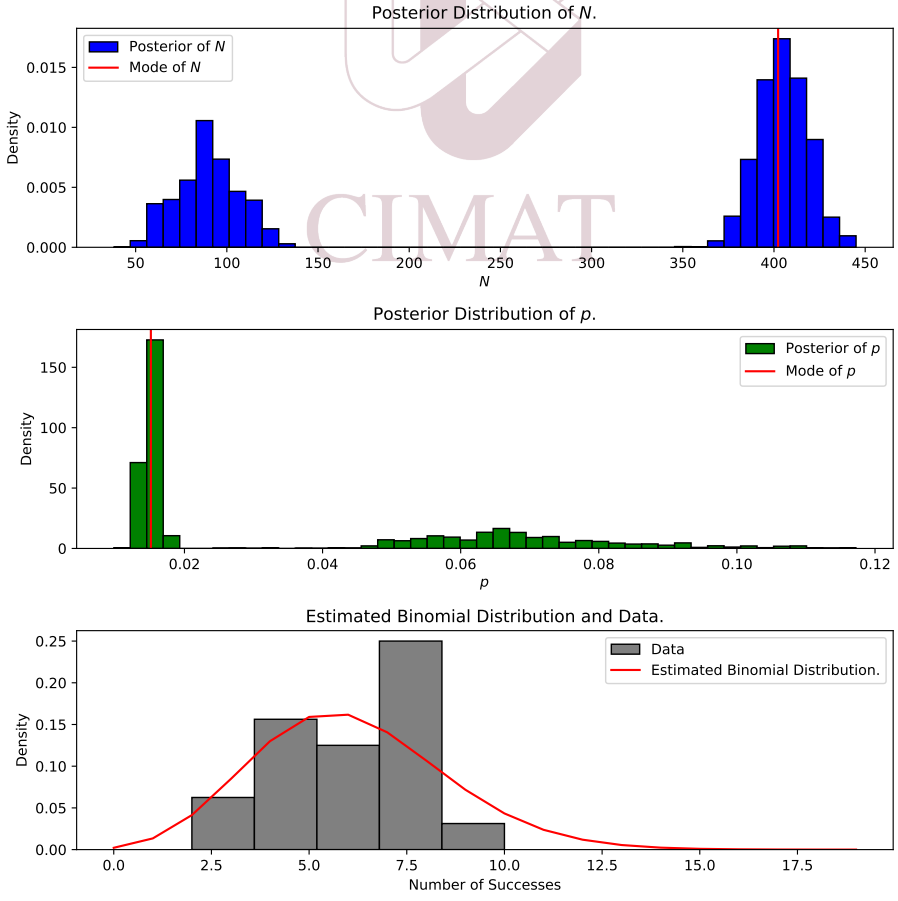
Resultados:

Con ayuda de las funciones auxiliares descritas en 4 e implementadas en funciones_auxiliares.py, se puede visualizar los resultados generados. Se ejecuta el código principal, en el cual se definen los datos a tratar, así como los parámetros para la distribución posterior, la a priori, la inicial y las propuestas. Para el algoritmo Metropolis-Hastings se realizaron 10000 muestras y el punto inicial resultó ser $[38, 0.2795]$ (se usó una semilla para la reproducibilidad de los resultados).

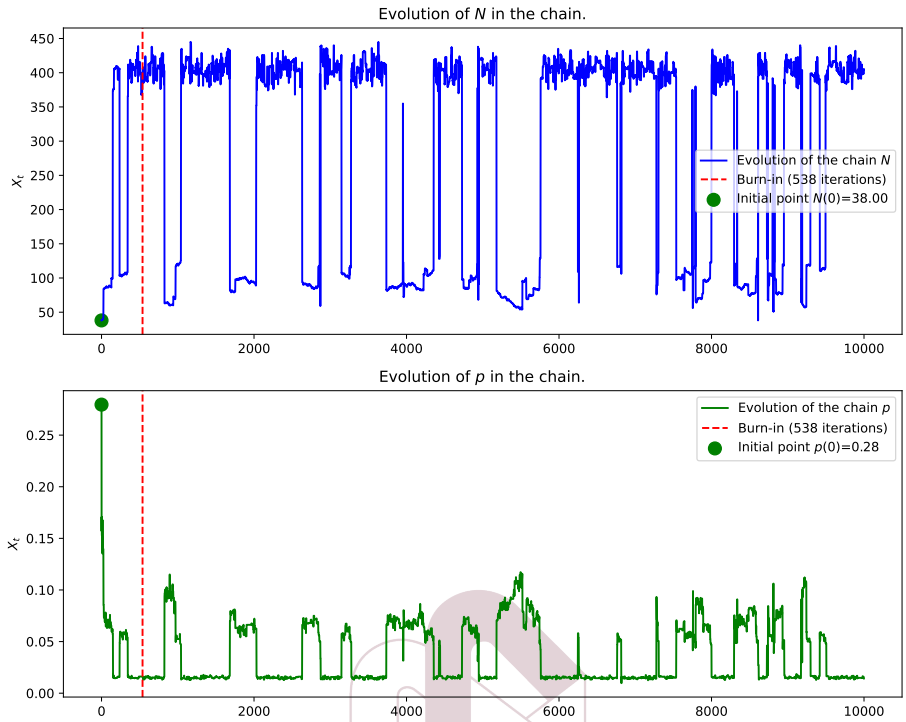
A continuación, se definieron las listas de funciones generadoras de cada propuesta, así como de sus respectivas funciones de densidad y a todas las propuestas se les dio probabilidad de elección igual a $\frac{1}{6}$. Al generar la cadena, se obtuvo un 42.77% de aceptación de propuestas y con ayuda de la función *estim_burn_in_and_modes()* se calcularon las modas resultantes de cada parámetro y el burn-in a considerar:

- Moda de N: 408,
- Moda de p: 0.0151,
- Burn-in: 538.

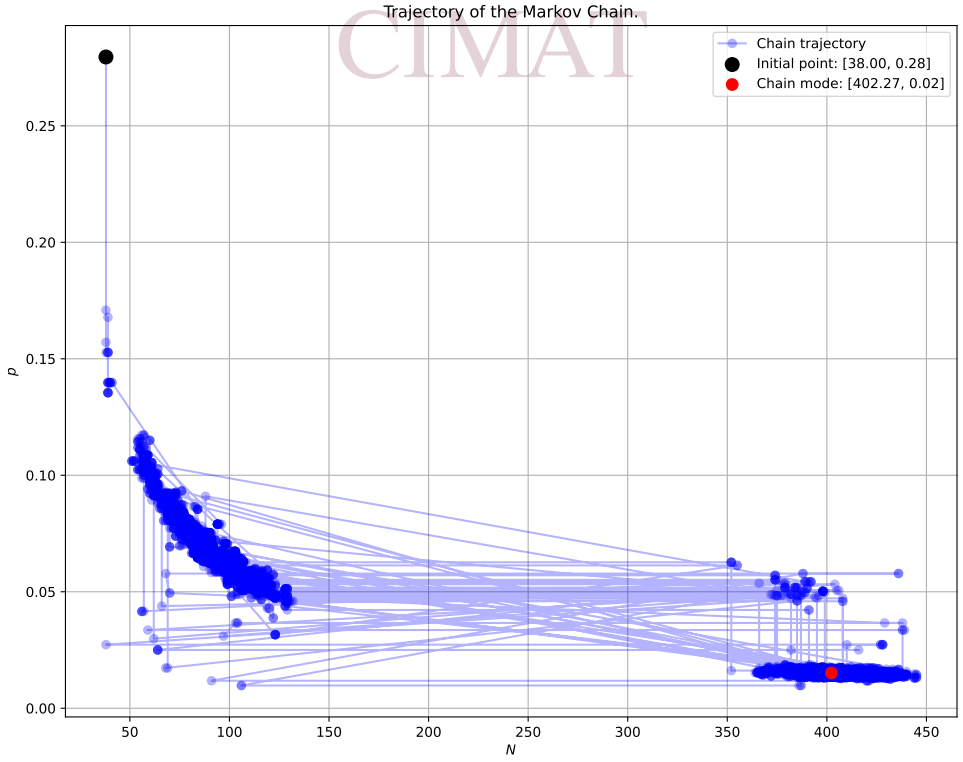
A continuación se tienen los histogramas de las cadenas de Markov simuladas. Además, grafica el histograma de los datos y las distribución binomial estimada, marcando las modas calculadas en cada caso. Vemos que el ajuste es relativamente bueno debido a que se cuenta con muy pocos datos.



A continuación, se tiene la evolución de las cadenas de Markov marginales con respecto al burn-in estimado.



Finalmente, se tiene la trayectoria de la cadena de Markov en el espacio de parámetros.



Al observar los resultados y el comportamiento de las gráficas, se puede notar que en el histograma de N, existe un pico pronunciado en valores altos alrededor de 408, que corresponde a la moda de la distribución posterior. Esto sugiere que el MCMC identificó esta región como el valor más probable para N. Por otro lado, la moda de p se encuentra alrededor de 0.02, lo que indica que, al incrementar N, el modelo ajusta p a valores más pequeños para mantener la consistencia con los datos observados, reflejando un efecto compensatorio entre ambos parámetros.

La trayectoria de la cadena refuerza esta observación, mostrando que la mayor parte de la exploración se concentra cerca de $N \approx 408$ y $p \approx 0.02$. Sin embargo, al inicio del muestreo, se exploraron combinaciones diferentes, caracterizadas por valores más bajos de N y valores más altos de p. Esto indica que el espacio de parámetros admite múltiples combinaciones compatibles con los datos observados, aunque el MCMC finalmente converge hacia la región de mayor probabilidad.

Este comportamiento puede atribuirse a la poca cantidad de datos disponibles, lo que reduce la capacidad del modelo para discriminar entre combinaciones de N y p ya que, para distintos valores de los parámetros de las propuestas como λ o r, se observaba este fenómeno de "buen" ajuste de la posterior para distintos valores de N y p. Se optó por estas versiones ya que hubo pocos problemas en la implementación y poca variabilidad en las trayectorias de la cadena.

2 Ejercicio 2

(Estudio de mercado) Se tiene un producto y se realiza una encuesta con el fin de estudiar cuánto se consume dependiendo de la edad. Sea Y_i el monto de compra y X_i la covariable la cual representa la edad.

Suponga que $Y_i \sim Po(\lambda_i)$ (distribución Poisson con intensidad λ_i)

$$\lambda_i = c g_b(x_i - a) \tag{15}$$

para g_b la siguiente función de liga

$$g_b(x) = \exp\left(-\frac{x^2}{2b^2}\right). \tag{16}$$

O sea, se trata de regresión Poisson con una función liga no usual. Si $\lambda_i = 0$ entonces $P(Y_i = 0) = 1$. $a =$ años medio del segmento (años), $c =$ gasto promedio (pesos), $b =$ "amplitud" del segmento (años).

Considere las distribuciones a priori

$$a \sim N(35, 5), \quad c \sim Gama(3, 3/950), \quad b \sim Gama(2, 2/5). \tag{17}$$

El segundo parámetro de la normal es desviación estandard y el segundo parámetro de las gammas es tasa (*rate*).

Usando MH simule de la distribución posterior de a, c y b.

Los datos son estos, $n = 100$:

```
X = array([ 27, 19, 21, 51, 16, 59, 16, 54, 52, 16, 31, 31, 54, 26, 19, 13, 59, 48, 54, 23, 50, 59, 55, 37, 61, 53, 56,
31, 34, 15, 41, 14, 13, 13, 32, 46, 17, 52, 54, 25, 61, 15, 53, 39, 33, 52, 65, 35, 65, 26, 54, 16, 47, 14, 42, 47, 48,
25, 15, 46, 31, 50, 42, 23, 17, 47, 32, 65, 45, 28, 12, 22, 30, 36, 33, 16, 39, 50, 13, 23, 50, 34, 19, 46, 43, 56, 52,
42, 48, 55, 37, 21, 45, 64, 53, 16, 62, 16, 25, 62])
```

```
Y = array([1275, 325, 517, 0, 86, 0, 101, 0, 0, 89, 78, 83, 0, 1074, 508, 5, 0, 0, 0, 1447, 0, 0, 0, 0, 0, 0, 0, 87, 7,
37, 0, 15, 5, 6, 35, 0, 158, 0, 0, 1349, 0, 35, 0, 0, 12, 0, 0, 2, 0, 1117, 0, 79, 0, 13, 0, 0, 0, 1334, 56, 0, 81, 0, 0,
1480, 177, 0, 29, 0, 0, 551, 0, 1338, 196, 0, 9, 104, 0, 0, 3, 1430, 0, 2, 492, 0, 0, 0, 0, 0, 0, 0, 0, 1057, 0, 0, 0, 68,
0, 87, 1362, 0])
```

Respuesta:

Se tiene que $\lambda_i := \lambda_i(a, b, c) = c \cdot e^{-\frac{(x_i - a)^2}{2b^2}}$ y se quiere hacer inferencia sobre a, b y c. Como $Y_i \sim Po(\lambda_i)$, entonces

$$P(Y_i = y_i \mid \lambda_i(a, b, c)) = \frac{\lambda_i^{y_i} e^{-\lambda_i}}{y_i!} \tag{18}$$

Entonces, la probabilidad conjunta para n observaciones $Y = \{Y_1, \ldots, Y_n\}$ suponiendo independencia condicional dada $\lambda = \{\lambda_1, \ldots, \lambda_n\}$ es:

$$P(Y = y \mid a, b, c) = \prod_{i=1}^{n} \frac{\lambda_i^{y_i} e^{-\lambda_i}}{y_i!} \tag{19}$$

para $y = \{y_1, \ldots, y_n\}$. La ecuación (19) es la verosimilitud y de la expresión (17), se tiene las distribuciones a priori, esto da lugar a la posterior

$$\begin{aligned}
P(a,b,c \mid Y = y) &\propto P(Y = y \mid a,b,c) \cdot \pi_a(a) \cdot \pi_b(b) \cdot \pi_c(c) \cdot \mathbb{1}_{\mathbb{R}}(a)\mathbb{1}_{(0,\infty)}(b)\mathbb{1}_{(0,\infty)}(c) \\
&\propto \prod_{i=1}^{n} \frac{\lambda_i^{y_i} e^{-\lambda_i}}{y_i!} \cdot \frac{1}{5\cdot\sqrt{2\pi}} e^{-\frac{(a-35)^2}{2\cdot 5^2}} \cdot \frac{(2/5)^2}{\Gamma(2)} b^{2-1} e^{-b\cdot(2/5)} \cdot \frac{(3/950)^3}{\Gamma(3)} c^{3-1} e^{-c\cdot(3/950)} \\
&\propto bc^2 e^{-\frac{(a-35)^2}{50}} e^{-\frac{2}{5}b} e^{-\frac{3}{950}c} \cdot \prod_{i=1}^{n} \lambda_i^{y_i} e^{-\lambda_i} \cdot \mathbb{1}_{\mathbb{R}}(a)\mathbb{1}_{(0,\infty)}(b)\mathbb{1}_{(0,\infty)}(c) \\
&\propto bc^2 e^{-\frac{(a-35)^2}{50}} e^{-\frac{2}{5}b} e^{-\frac{3}{950}c} \cdot e^{-\sum\lambda_i} \prod_{i=1}^{n} \left(c e^{-\frac{(x_i-a)^2}{2b^2}} \right)^{y_i} \cdot \mathbb{1}_{\mathbb{R}}(a)\mathbb{1}_{(0,\infty)}(b)\mathbb{1}_{(0,\infty)}(c) \\
&\propto bc^2 e^{-\frac{(a-35)^2}{50}} e^{-\frac{2}{5}b} e^{-\frac{3}{950}c} \cdot e^{-c\sum e^{-\frac{(x_i-a)^2}{2b^2}}} c^{\sum y_i} \cdot \prod_{i=1}^{n} e^{-y_i\frac{(x_i-a)^2}{2b^2}} \cdot \mathbb{1}_{\mathbb{R}}(a)\mathbb{1}_{(0,\infty)}(b)\mathbb{1}_{(0,\infty)}(c) \\
&\propto c^{2+\sum y_i} e^{-c\left(\frac{3}{950}+\sum e^{-\frac{(x_i-a)^2}{2b^2}}\right)} \cdot be^{-\frac{(a-35)^2}{50}} e^{-\frac{2}{5}b} e^{-\sum y_i\frac{(x_i-a)^2}{2b^2}} \cdot \mathbb{1}_{\mathbb{R}}(a)\mathbb{1}_{(0,\infty)}(b)\mathbb{1}_{(0,\infty)}(c)
\end{aligned} \tag{20}$$

Esta expresión nos sirve para encontrar una propuesta Gibbs para el algoritmo de Metropolis Hastings con kérneles híbridos. Se puede notar que no existe una forma cerrada estándar para ninguna de las condicionales completas para a y b para muestrear directamente debido a la complejidad del producto de la derecha. Sin embargo, nótese que el término

$$c^{2+\sum y_i} e^{-c\left(\frac{3}{950}+\sum e^{-\frac{(x_i-a)^2}{2b^2}}\right)} \tag{21}$$

es proporcional a una distribución

$$Gama\left(3+\sum y_i, \frac{3}{950}+\sum e^{-\frac{(x_i-a)^2}{2b^2}}\right) \tag{22}$$

para c dados a, b y los datos, la cual se puede usar como una propuesta Gibbs. En el archivo ejercicio2_tarea9.py se hace uso de la función:

METROPOLIS_HASTINGS_HYBRID_KERNELS(),

implementada en el archivo ejercicio1_tarea9.py e importada al actual, la cual aplica el algoritmo Metropolis-Hastings con kérneles híbridos para K propuestas simulando una cadena de Markov en $\mathbb{R}^n$. Se comienza definiendo las funciones (15) y (16) que son necesarias para la función *posterior()* la cual implementa la función objetivo descrita en (20)

Distribución inicial:

Por simplicidad y buenos resultados al momento de experimentar con las distintas propuestas y distribuciones iniciales, se decidió usar como distribución inicial las a prioris (17):

$$a \sim N(35,5), \quad c \sim Gama(3, 3/950), \quad b \sim Gama(2, 2/5). \tag{23}$$

Propuestas:

En el archivo ejercicio1_tarea9.py, se implementan las propuestas, así como su densidad de probabilidad respectiva y estas se describen a continuación.

- Propuesta 1: Kérnel Gibbs para c dados a, b y los datos.

$$c \sim Gama\left(3 + \sum y_i, \frac{3}{950} + \sum e^{-\frac{(x_i-a)^2}{2b^2}}\right) \text{ y } a, b \text{ fijos.} \tag{24}$$

Se define la función *prop_gibbs_c_gen()* que implementa la distribución Gibbs (distribución condicional de c dado a, b) descrita en (22). También se define su densidad de probabilidad en la función *prop_gibbs_c_pdf()*. Se eligió esta propuesta ya que un paso de Gibbs siempre acepta la propuesta porque la muestra proviene de la distribución condicional verdadera. Esto asegura exploración eficiente en la dimensión de c, además de más detalles descritos en la parte de conclusiones.

- Propuesta 2: Caminata aleatoria conjunta.

$$a_p \sim \mathcal{N}(a, \sigma_a), \quad b_p \sim \mathcal{N}(b, \sigma_b), \quad c_p \sim \mathcal{N}(c, \sigma_c) \tag{25}$$

Se decidió usar esta propuesta ya que en cadenas de Markov, son robustas para exploración local y garantizan una tasa de aceptación razonable. Sin embargo, su convergencia puede ser lenta si la posterior es amplia, pero para ese caso, ya se cuentan con otras propuestas que cuentan con más variabilidad. Para cada parámetro, se eligió su varianza σ, experimentando y revisando los soportes de cada parámetro, así como de su desempeño en la mayoría de los experimentos, se eligió usar $\sigma_a = 5$, $\sigma_b = 5/2$ y $\sigma_a = 950/3$ (también para usar parámetros ya definidos en otras distribuciones). La función generadora y su función de densidad se definen en *prop_normal_conjunta_gen()* y *prop_normal_conjunta_pdf()* respectivamente.

- Propuesta 3 y 4: Propuestas a priori.

$$\begin{aligned} &a \sim \mathcal{N}(35, 5) \text{ y } b, c \text{ fijos,} \\ &b \sim Gama(3, 2/5) \text{ y } a, c \text{ fijos.} \end{aligned} \tag{26}$$

Se eligieron estas propuestas, ya que, dado que son propuestas independientes para sus respectivos parámetros, introducen aleatoriedad independiente de los datos, lo que ayuda a evitar que la cadena quede atrapada en un modo local y permite exploración de regiones remotas del espacio de parámetros. Las funciones que las describen son

- *prop_a_apriori_gen()* y *prop_a_apriori_gen()*
- *prop_b_apriori_gen()* y *prop_b_apriori_gen()*

Se eligió no usar la a priori de c ya que presentaba más variabilidad y dado que ya se tiene una Gibbs, se observó un mejor desempeño que cuando se mezclaban ambas.

Conclusión general de las propuestas:

Para garantizar una buena convergencia, las elecciones de las propuestas aprovechan tanto la estructura condicional de la posterior como las propiedades específicas de cada parámetro. La elección de la propuesta Gibbs siempre es óptima porque utiliza la condicional completa de c. Esto garantiza aceptación total y evita rechazos, maximizando la eficiencia del muestreo. Su capacidad de incorporar la interacción de c con los datos y otros parámetros la hace ideal para una convergencia rápida, esto se ve evidente en los resultados mostrados a continuación

Las propuestas prior para a y b resultan ser una buena elección, ya que respeta la información a priori y permite explorar eficientemente la región inicial del espacio de parámetros. Las

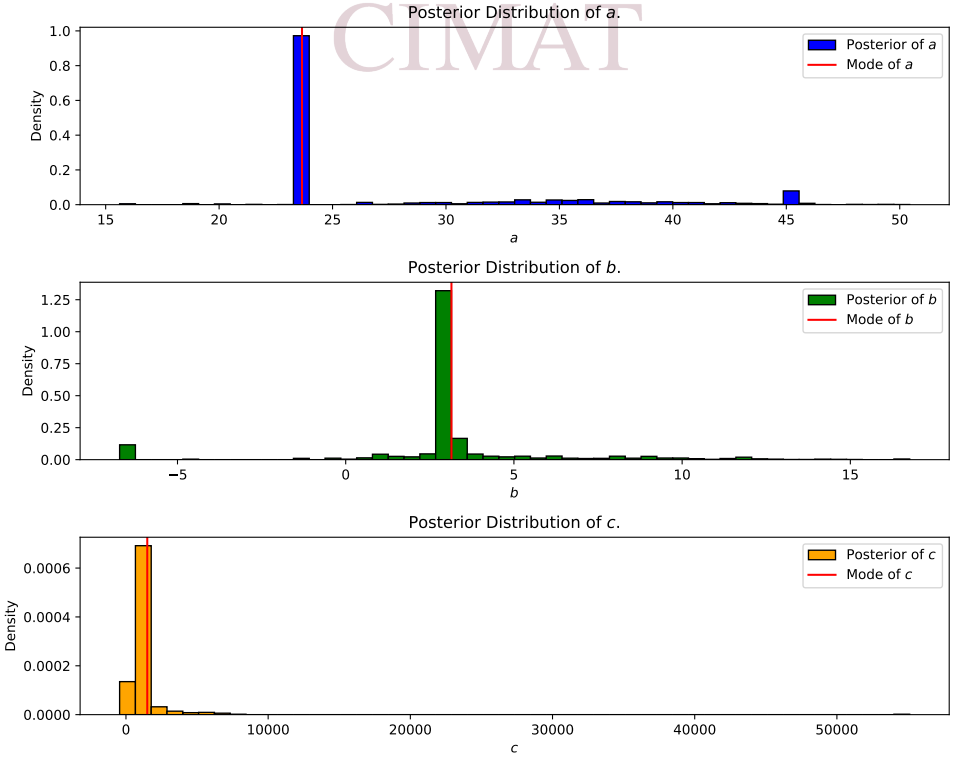
propuestas basadas en las distribuciones a priori garantizan que la cadena comience en regiones razonables, evitando problemas de inicialización. La propuesta normal conjunta para a, b, y c resultó interesante ya que generar propuestas conjuntas permite capturar dependencias que podrían pasar desapercibidas con propuestas independientes.

En conjunto, esta estrategia híbrida combina la precisión y eficiencia del Gibbs Sampling con la flexibilidad de propuestas adaptativas, asegurando convergencia rápida y una buena exploración del espacio posterior.

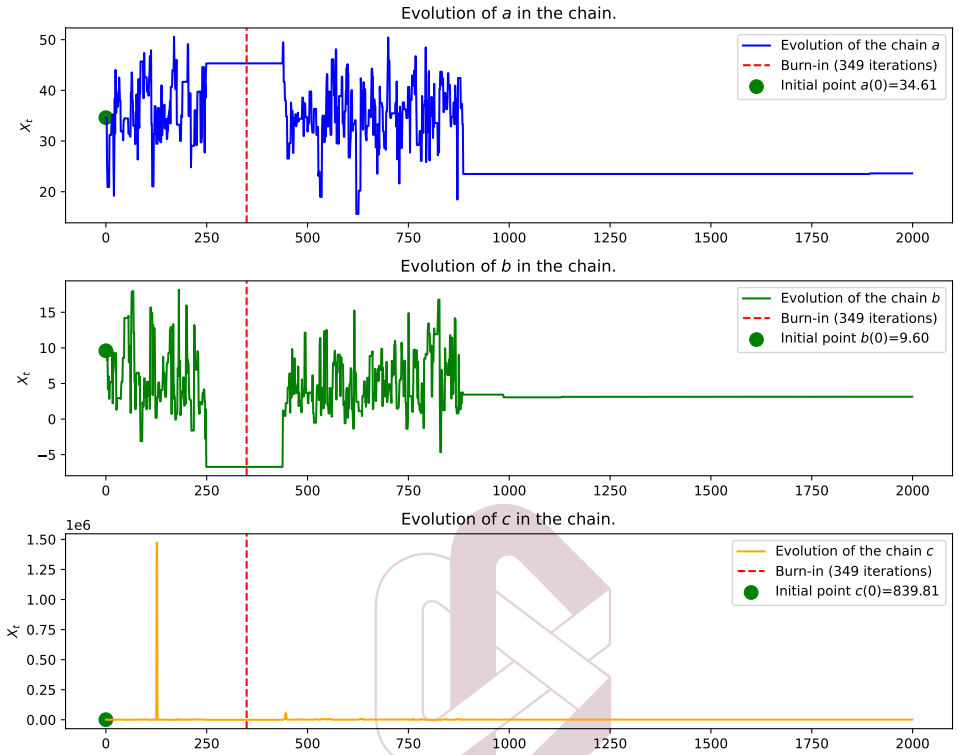
Resultados:

Nuevamente, con ayuda de las funciones auxiliares descritas en 4 e implementadas en funciones_auxiliares.py, se puede visualizar los resultados generados. Se ejecuta el código principal, en el cual se definen los datos a tratar, así como los parámetros para la distribución posterior, las a priori, la inicial y las propuestas. Para el algoritmo Metropolis-Hastings sólo fueron necesarias 2000 muestras ya que, en distintos experimentos se vio que la cadena cae en estados estacionarios, algunos temporales y otros permanentes. En este caso, se quedó en el estado estacionario final bastante pronto.

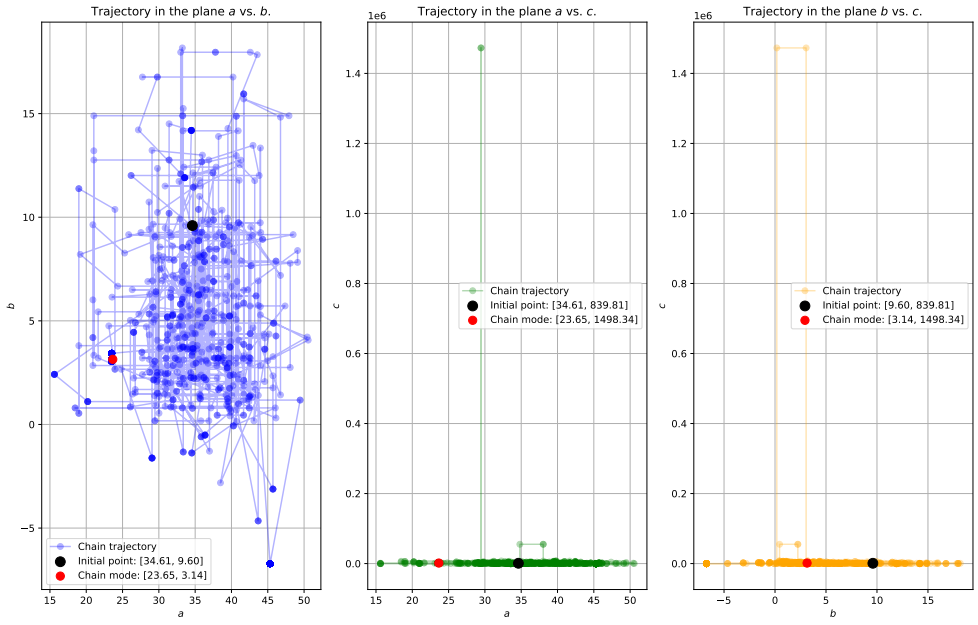
El punto inicial resultó ser $[34.6139, 9.5969, 839.8066]$ (se usó una semilla para la reproducibilidad de los resultados). A continuación, se definieron las listas de funciones generadoras de cada propuesta, así como de sus respectivas funciones de densidad y a todas las propuestas se les dio probabilidad de elección igual a $\frac{1}{4}$. Al generar la cadena, se obtuvo un 41.45% de aceptación de propuestas y con ayuda de la función *estim_burn_in_and_modes()* se calcularon las modas resultantes de cada parámetro y el burn-in a considerar: moda de a: 23.653, moda de b: 3.143, moda de c: 1498.342 y el burn-in: 349. Los valores resultantes de estas modas fueron bastante recurrentes para distintos puntos iniciales y configuraciones de las propuestas. A continuación se tienen los histogramas de las cadenas de Markov simuladas.



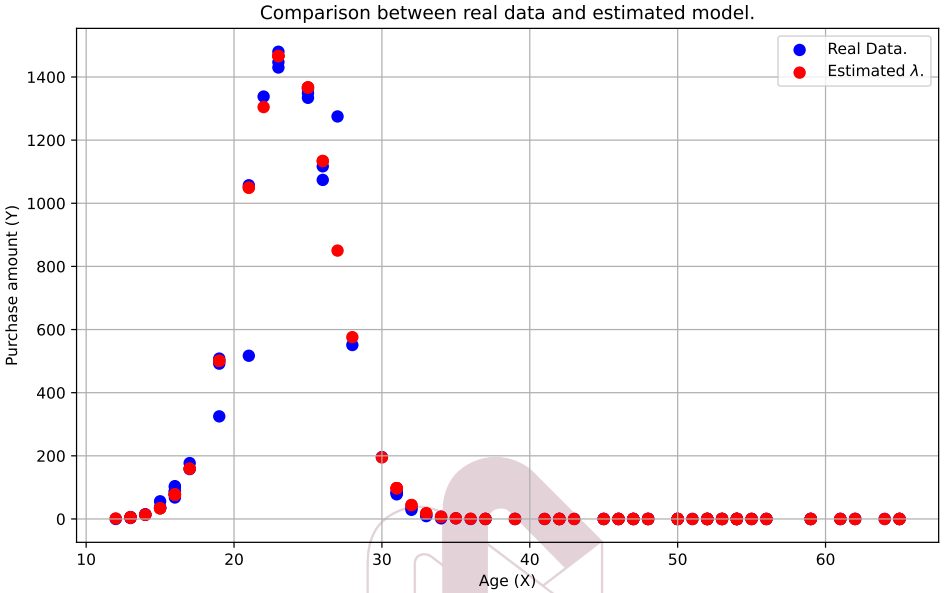
A continuación, se tiene la evolución de las cadenas de Markov marginales con respecto al burn-in estimado. Podemos notar que al principio, la cadena cayó en un estado estacionario temporal, es por eso que el burn-in se tomó en esta zona y la razón por la que se eligió mostrar este ejemplo particular ya que fenómenos como este sucedieron en cada experimento.



También se tiene la trayectoria de la cadena de Markov en el espacio de parámetros para cada par de parámetros.



Antes de incluir la propuesta Gibbs para c, todos estos gráficos resultaban con mucha variabilidad y sin convergencia clara. Ahora, se puede ver la convergencia rápida de c. Finalmente, se tiene el gráfico de los datos iniciales comparado con las estimaciones de λ_i. Podemos notar que se ajusta muy bien. Este resultado suele ser recurrente en la experimentación, sin embargo, se notó una mejora importante con la inclusión de la propuesta Gibbs para c.



3 Ejercicio 3

Investiga y describe muy brevemente los softwares OpenBugs, Nimble, JAGS, DRAM, Rtwalk, Mcee Hammer, PyMCMC.

Respuesta:

- **OpenBugs** (*Bayesian inference Using Gibbs Sampling*)

Es un software diseñado para realizar análisis estadístico bayesiano mediante algoritmos MCMC, particularmente Gibbs Sampling. Es ampliamente utilizado para modelar datos complejos y especificar modelos probabilísticos jerárquicos en su propio lenguaje. OpenBUGS es bien conocido en la comunidad estadística y se integra fácilmente con R a través de paquetes como BRugs. Sin embargo, presenta limitaciones en términos de flexibilidad y rendimiento en problemas de gran escala o alta dimensionalidad.

- **Nimble**

Es una plataforma en R que permite construir y personalizar modelos estadísticos, especialmente para MCMC y algoritmos jerárquicos bayesianos. Su lenguaje es similar al de BUGS, pero ofrece la capacidad de optimizar y modificar directamente los algoritmos. Es una herramienta flexible y eficiente, ideal para simulaciones y análisis avanzados, aunque su curva de aprendizaje puede ser pronunciada para principiantes.

- **JAGS** (*Just Another Gibbs Sampler*)

Es una alternativa modular y extensible a OpenBUGS, diseñada para el muestreo MCMC en modelos bayesianos jerárquicos. Compatible con R mediante el paquete rjags, JAGS soporta una amplia variedad de distribuciones y modelos. Ofrece mayor adaptabilidad y modernidad que OpenBUGS, aunque sigue siendo menos flexible que herramientas como Nimble.

- **DRAM** (*Delayed Rejection Adaptive Metropolis*)

Es una extensión del algoritmo Metropolis-Hastings que combina la adaptabilidad con retraso en el rechazo de propuestas. Esto permite una exploración más efectiva en el espacio de parámetros, especialmente en distribuciones complejas o con correlaciones fuertes. DRAM reduce el tiempo de convergencia en comparación con Metropolis-Hastings estándar, aunque puede ser más costoso computacionalmente.

- **Rtwalk**

Es un algoritmo MCMC basado en "random walk" diseñado para explorar eficientemente espacios de alta dimensión. Garantiza irreducibilidad y está optimizado para problemas bayesianos con restricciones o geometrías complejas. Es particularmente eficiente en problemas con muchas dimensiones, pero su uso es menos extendido, lo que limita el soporte comunitario disponible.

- **Mcee Hammer**

Es un sampler MCMC especializado inicialmente en problemas de modelado no lineal en astronomía. Utiliza un enfoque de conjunto de puntos (ensemble sampler), como el algoritmo Affine Invariant, para manejar modelos con muchas dimensiones y correlaciones fuertes. Es fácil de usar, ya que no requiere un ajuste detallado de parámetros, y aunque es popular en astronomía, también puede aplicarse en otros campos científicos.

- **PyMCMC**

Es una biblioteca en Python para implementar métodos MCMC personalizados, especialmente en inferencia bayesiana. Integra herramientas del ecosistema de Python como NumPy, SciPy y Matplotlib, y permite adaptar los algoritmos de muestreo a necesidades específicas. Su flexibilidad es ideal para usuarios que desean personalizar sus modelos, aunque tiene menos soporte comunitario en comparación con herramientas más maduras como Stan o PyMC.

4 Funciones Auxiliares

En esta parte, se describen las funciones auxiliares implementadas en el archivo:

funciones_auxiliares.py:

Para el burn_in y las modas

- *moving_average()*

Esta función calcula la media móvil de un arreglo unidimensional, una herramienta común para suavizar series temporales o datos ruidosos. Toma como entrada un arreglo y un tamaño de ventana que determina el número de elementos considerados para cada promedio. Utiliza convolución para generar un nuevo arreglo donde cada elemento es el promedio de los valores en la ventana correspondiente, permitiendo una mejor visualización de las tendencias subyacentes en los datos. Esta función se utiliza dentro de la función siguiente:

- *estim_burn_in_and_modes()*

Esta función tiene dos propósitos principales: estimar el burn-in de una cadena de Markov y calcular las modas de los parámetros. Para el burn-in, utiliza la estabilización de la media móvil, identificando el punto donde la variación de esta se vuelve insignificante. Esto ayuda a descartar las iteraciones iniciales que aún no han alcanzado el régimen estacionario. Además, estima las modas de los parámetros analizando los histogramas de las distribuciones y seleccionando los valores más probables.

Graficación

- *marginal_evolution_burn_in()*

Esta función visualiza la evolución de las cadenas marginales para cada parámetro de una cadena de Markov, proporcionando información sobre su convergencia. Genera gráficos donde se destacan los valores iniciales de cada cadena y el punto estimado de burn-in con líneas verticales, facilitando la identificación de regiones estables. Es especialmente útil para evaluar el comportamiento de los parámetros a lo largo del tiempo y confirmar visualmente si la cadena ha alcanzado el equilibrio.

- *histograms_chain()*

Genera histogramas de las distribuciones posteriores de los parámetros después de descartar el burn-in. Estos histogramas permiten observar la densidad de probabilidad de cada parámetro, resaltando las modas con líneas verticales. Adicionalmente, para el ejercicio 1, se incluyen los datos y esta función incluye un histograma de los mismos y superpone una distribución binomial estimada para comparar la información posterior con los datos reales.

- *trayectoria2d3d()*

Permite analizar visualmente cómo una cadena de Markov explora el espacio de parámetros. En el caso de dos parámetros, genera un gráfico bidimensional donde se observa la trayectoria de la cadena, marcando tanto el punto inicial como la moda estimada. Si hay tres parámetros, añade una visualización tridimensional y gráficos proyectados en los planos bidimensionales correspondientes. Esta función es especialmente valiosa para comprender cómo se comporta la cadena en espacios paramétricos de mayor dimensión, ayudando a identificar patrones, convergencia y regiones de alta probabilidad.