Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Geofísica

Tarea 2

Ajuste Ortogonal de una Linea Recta usando Métodos de Muestreo Bayesianos: Algoritmos de Metrópolis y Metrópolis en Paralelo

Integrantes: Catalina Zapata
Constanza Santori
Karina Hernández
Teresa Peralta
Profesor: Francisco Ortega Culaciati
Fecha de entrega: 20 de junio de 2025
Santiago de Chile

1. Problema Directo

Utilizando mínimos cuadrados es posible obtener el ajuste de una recta paramétrica y minimizar la distancia Euclidiana entre los puntos de observación y la linea recta.
Así, conociendo que θ corresponde a la orientación de la recta respecto eje x y a a la distancia ortogonal de la recta al origen, es posible definir m $= [a, \theta]^T$ como el vector de parámetros del modelo que minimiza la ajusta los datos. Con esto, podemos definir la distancia entre el punto de observación d^k y la recta generada sobre el punto S_k más cercano a d^k como $\Delta_k = ||r(S_k, m) - d^k||$ que se busca minimizar, de manera que se plantea el problema a optimizar con criterio de mínimos cuadrados (no lineal) como:

$$\min_m \sum_{k=1}^{N} \left(\frac{\Delta_k(m)}{\sigma_{\Delta_k}(m)} \right)^2$$

Donde σ_{Δ_k} es la desviación estándar de la distancia del punto observado a la recta, que se puede obtener a partir de una aproximación lineal definida por las distancias y sus desviaciones estándar en x e y.

1.1. Predicciones del modelo de Ajuste ortogonal de la recta

La función *.eval* calcula las predicciones del modelo, las cuales se interpretan como residuos normalizados. En este contexto, los $\Delta_k(m)$ representan las distancias ortogonales más cortas desde cada punto observado hasta la recta definida por el modelo $m = [a, \theta]$, donde a es la distancia de la recta al origen del sistema de coordenadas y θ es su orientación en grados sexagesimales, medida en sentido antihorario desde el eje x positivo.

Las incertidumbres asociadas a estas distancias, denotadas como σ_{Δ_k}, se obtienen mediante propagación de errores a partir de las incertidumbres originales en las observaciones, σ_x y σ_y. De este modo, las predicciones del modelo quedan definidas como:

$$d_{\text{pred}} = \frac{\Delta_k(m)}{\sigma_{\Delta_k}} \tag{1}$$

Para realizar este cálculo, el método hace uso de la función auxiliar $recta.calc\_dist\_sigma(...)$, que determina tanto las distancias ortogonales como sus desviaciones estándar, además de otros valores intermedios como los errores en x e y y el parámetro s sobre la recta.

El vector d_{pred} resultante contiene, para cada punto observado, el valor del desajuste con respecto al modelo, expresado en número de desviaciones estándar. Este enfoque permite incorporar las incertidumbres observacionales en la evaluación del ajuste del modelo, siendo especialmente útil en esquemas de inversión no lineal, como el método de mínimos cuadrados ponderados. Valores absolutos elevados en d_{pred} indican discrepancias significativas entre el

modelo propuesto y los datos observados.)

1.2. Generación de datos sintéticos

Para evaluar el ajuste del modelo de ajuste ortogonal de una recta, se generaron datos sintéticos simulando observaciones con incertidumbre. Este proceso consiste en definir una recta base mediante parámetros conocidos —la distancia al origen $a = -10$ y el ángulo de inclinación $\theta = -45°$ y luego generar puntos cercanos a dicha recta, añadiendo ruido aleatorio con una distribución normal en las direcciones x e y, controlado por desviaciones estándar σ_x y σ_y respectivamente. El conjunto de observaciones (x_{obs}, y_{obs}) representa puntos dispersos alrededor de la recta ideal, y está acompañado por las incertidumbres asociadas a cada punto. Estas incertidumbres permiten evaluar la calidad del ajuste mediante el cálculo de distancias normalizadas. En la Figura 1 se representa el ajuste ortogonal a la recta, en ella la recta en color azul que corresponden a la predicción del modelo $m = [a, \theta]^T$ elegido. Las líneas cyan que conectan ortogonalmente cada observación con su proyección más cercana sobre la recta (es decir, las distancias ortogonales Δ_k), estas permiten evaluar de forma cualitativa el ajuste del modelo propuesto, donde mientras más cortas sean, mejor se ajusta la recta a los datos, en otras palabras, muestran el desajuste entre la predicción del modelo y cada observación. Las barras rojas representan las desviaciones estándar σ_x y σ_y de cada punto observado. Así, el modelo de recta propuesto captura de forma parcial la orientación de la nube de puntos, pero no necesariamente es el mejor modelo, pues, se observa que muchas distancias no son mínimas, lo que indica que el ajuste no es óptimo. Un algoritmo de ajuste ortogonal debería ser aplicado para encontrar los valores de a y θ que minimizan la suma de los cuadrados de estas distancias ortogonales, así errores en ambas direcciones, mejorando así el ajuste del modelo a los datos.

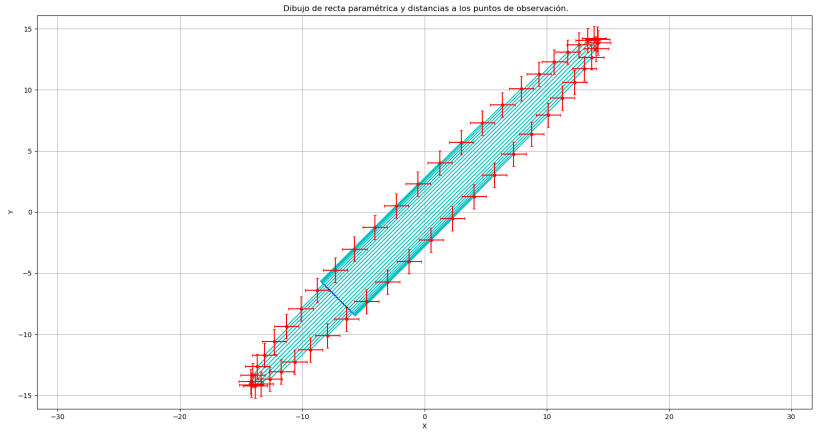


Figura 1: Veamos que en puntos rojos se encuentran los puntos de observación con sus barras de error respectivas, en segmentos cyan están los distancias ortogonales entre cada punto observado y su proyección sobre la recta y la linea azul une las proyecciones ortogonales de los puntos sobre la recta con longitud finita.

2. Modelación Inversa usando Métodos Bayesianos

En el contexto del ajuste ortogonal de una recta, se busca estimar los parámetros que definen una recta en el plano, dadas observaciones ruidosas de puntos $(x_{\text{obs}}, y_{\text{obs}})$. En lugar de encontrar una única recta óptima, la modelación inversa bayesiana permite obtener una *familia de rectas* posibles, caracterizada por la distribución a posteriori de los parámetros.

Definiendo el vector de parámetros del modelo m=$[a, \theta]^T$, y el objetivo es caracterizar la distribución a posteriori $f_{\text{post}}(\mathbf{m})$, la cual se define, según el teorema de Bayes, como:

$$f_{\text{post}}(\mathbf{m}) = \frac{1}{\nu} f_{\text{prior}}(\mathbf{m})\, \mathcal{L}(\mathbf{m}) \tag{2}$$

donde:

- $f_{\text{prior}}(\mathbf{m})$: es la función de densidad a priori sobre a y θ, elegida como uniforme sobre un dominio que contiene todas las soluciones razonables.
- $\mathcal{L}(\mathbf{m})$: es la función de verosimilitud, que mide qué tan bien explica el modelo $\mathbf{m}$ los datos observados.
- ν: es un factor de normalización que asegura que la función f_{post} sea una densidad de probabilidad válida (es decir, que integre a 1).

2.1. Densidad volumétrica a priori para los parámetros del modelo $f_{prior}(m)$

Para caracterizar la densidad volumétrica a priori de los parámetros del modelo m=$[a, \theta]^T$, se asumió un estado homogéneo de información, representando f_{prior}(m) mediante una distribución uniforme bidimensional. El rango de a, que representa la distancia desde el origen hasta la recta, se definió como $[-2 \cdot \|\mathbf{d}_{\text{obs}}\|_{\text{máx}},\ 2 \cdot \|\mathbf{d}_{\text{obs}}\|_{\text{máx}}]$, utilizando el doble de la norma máxima de los datos observados como límite, garantizando así un dominio suficientemente amplio para cubrir todas las posibles soluciones y para el ángulo θ, se tomó el dominio completo $[-180°, 180°[$, que abarca todas las posibles orientaciones de una recta en el plano.

Se utilizó la clase `pdf_uniform_nD` desarrollada previamente para representar esta distribución uniforme no normalizada en espacios de dimensión arbitraria. En este caso, se generó una instancia para una distribución uniforme bidimensional $f_{\text{prior}}(m)$.

Con el fin de verificar visualmente la distribución uniforme asumida, se generaron 10^5 muestras a partir de f_{prior}(m). Se construyó un histograma conjunto (a,θ), así como los histogramas marginales de cada parámetro (Fig. 2). Los resultados mostraron una distribución

aproximadamente homogénea en el espacio conjunto de parámetros, con una frecuencia similar, y los histogramas muestran distribuciones planas, indicando una frecuencia uniforme de aparición de muestras en cada subintervalo, lo cual valida la correcta implementación de la densidad a priori. Este análisis es esencial para garantizar una base coherente sobre la cual aplicar posteriormente los métodos de inferencia bayesiana y muestreo MCMC. Esto nos asegura que las conclusiones posteriores no existan malas interpretaciones debido a la distribución obtenida.

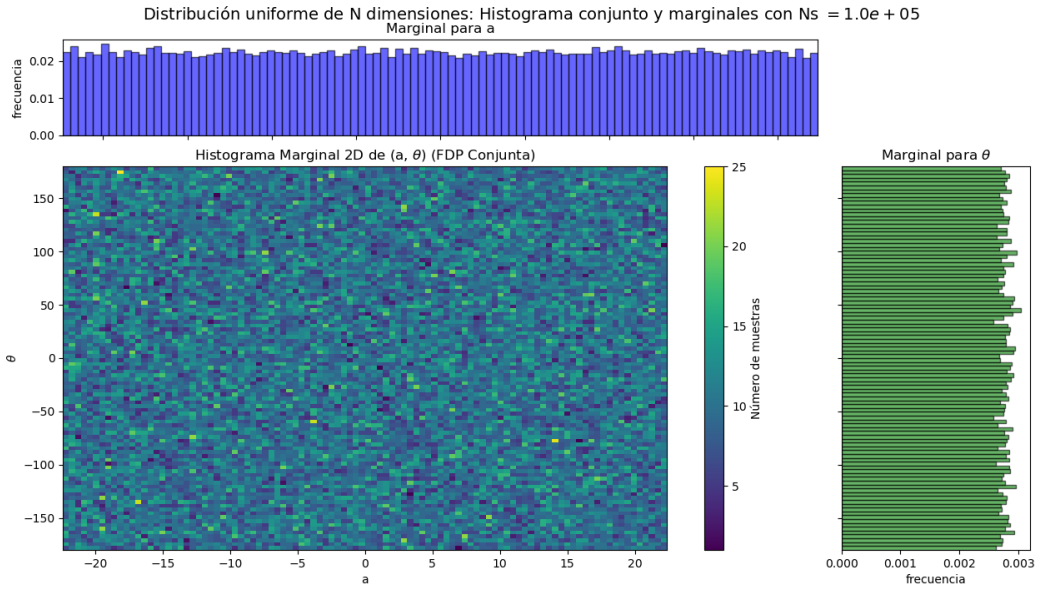


Figura 2: Histogramas 2D para $N_s = 1E5$ de muestras

2.2. Función de Verosimilitud $\mathcal{L}(m)$

La función de verosimilitud $\mathcal{L}(m)$ se define a partir de las distancias ortogonales entre los datos observados y la predicción del modelo, donde este será evaluado en función de su capacidad para minimizar las distancias ortogonales entre los datos y la recta, es decir, asumiendo errores normales independientes.

Asumiendo que los errores asociados a estas distancias son independientes e idénticamente distribuidos, y que siguen una distribución normal con media cero y varianza σ^2, se modela la función de densidad de probabilidad de las distancias Δ_k como:

$$\rho_\Delta(\Delta, \sigma^2) = \nu \exp\left(-\frac{1}{2}\sum_{k=1}^{N}\frac{\Delta_k^2}{\sigma^2}\right), \tag{3}$$

donde ν es una constante de normalización que asegura que la función de densidad esté debidamente normalizada.

Entonces, la función de verosimilitud $\mathcal{L}(m)$ se define como la densidad conjunta de una normal multivariada evaluada sobre las distancias $\Delta_k(m)$, con varianzas $\sigma^2_{\Delta_k(m)}$ asociadas.

$$\mathcal{L}(m) = \rho(\Delta(m), \sigma^2_{\Delta}(m)) \tag{4}$$

También, se puede expresar en función del vector de residuos normalizados $\mathbf{d}_{\text{pred}} = \Delta_k/\sigma_{\Delta_k}$ tal que:

$$\mathcal{L}(\mathbf{m}) \propto \exp\left(-\frac{1}{2}\sum_{k=1}^{N}\left(d_{\text{pred},k}\right)^2\right) \tag{5}$$

Este enfoque permite no solo estimar una única recta que se ajuste a los datos, sino explorar un conjunto de posibles modelos ponderados según su verosimilitud y su coherencia con el conocimiento previo. En términos prácticos, $\mathcal{L}(m)$ toma valores más altos para aquellos modelos cuya predicción genera distancias más pequeñas respecto a las observaciones, actuando como un cuantificador de la calidad del ajuste.

La estimación de $\mathcal{L}(m)$ se realiza mediante técnicas de muestreo como Monte Carlo, lo que permite explorar la distribución a posteriori sin necesidad de calcular explícitamente el factor de normalización ν.

La implementación de esta función se basó en el código proporcionado en el archivo likelihood_function.py, el cual recibe como entrada una instancia del modelo directo y una función de densidad de probabilidad para los datos. Esta formulación probabilística permite cuantificar qué tan probable es un conjunto de parámetros del modelo dado los datos observados, constituyendo un elemento central del enfoque bayesiano. Su validación posterior se realiza mediante un barrido exhaustivo sobre el espacio de parámetros, evaluando la verosimilitud sobre un conjunto de modelos definidos.

2.2.1. Verificación de la programación de ρ_Δ

Para verificar la correcta implementación de la clase `pdf_normal` definida en `pdf_normal.py`, se generaron 10^5 muestras a partir de una distribución normal multivariada con media $\mu = [-1,\ 4]$ y matriz de covarianza $= \begin{bmatrix} 2 & 1 \\ 1 & 4 \end{bmatrix}$. A partir de las muestras generadas, se estimaron la media y la matriz de covarianza empíricas. Los resultados obtenidos en la Figura 3 mostraron una alta concordancia con los parámetros teóricos, validando así tanto la generación de muestras como la correcta parametrización interna de la distribución.

Para complementar este análisis, se construyó un histograma conjunto bidimensional de las muestras simuladas, junto con sus respectivas distribuciones marginales (Figura 3). Se observó que la distribución conjunta presenta la forma esperada de elipse inclinada, característica de la correlación positiva entre X_1 y X_2. A su vez, las distribuciones marginales exhiben una forma de campana coherente con las proyecciones univariadas de una distribu-

ción normal multivariada.

Esta verificación es clave para garantizar que la función de densidad de probabilidad utilizada en la función de verosimilitud esté correctamente definida y lista para su uso en la función de verosimilitud.

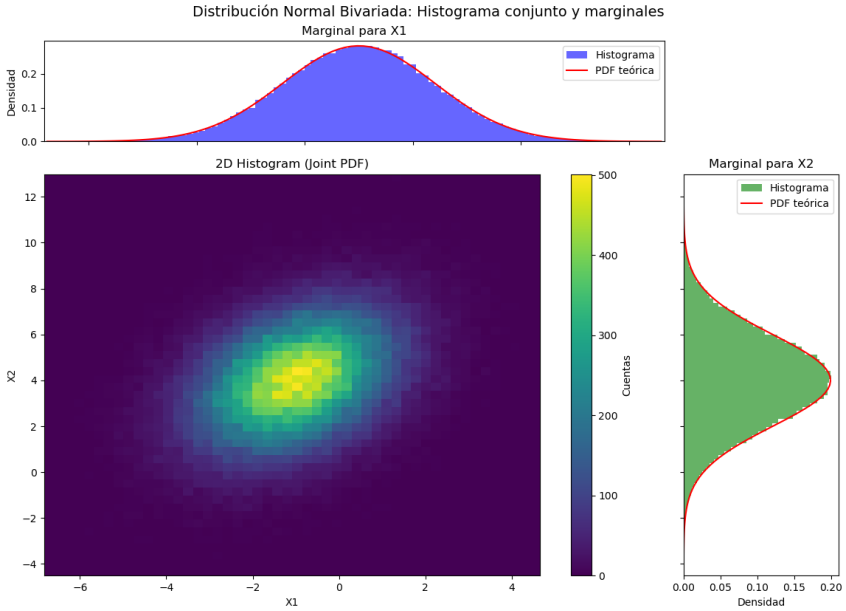


Figura 3: Histograma para la fdp conjunta y fdp marginales con Ns=10000.

2.2.2. Verificación de la programación de $\mathcal{L}(m)$

Para verificar la implementación de la función de verosimilitud $\mathcal{L}(m)$, se utilizó el método de búsqueda exhaustiva (*grid search*) sobre el espacio bidimensional de parámetros (a, θ). Se definió un conjunto denso de valores posibles para ambos parámetros dentro de los rangos establecidos por la distribución a priori uniforme, y se construyó un conjunto de modelos m con todas las combinaciones posibles. Cada modelo fue evaluado con la función de verosimilitud, y los resultados se almacenaron en una instancia de la clase **ensemble**. Esto permitió identificar visualmente las regiones del espacio de parámetros con mayor verosimilitud, confirmando que el código era capaz de asignar probabilidades coherentes a modelos que se ajustan mejor a los datos observados. Este paso validó que $\mathcal{L}(m)$ estaba correctamente implementada y lista para integrarse en el cálculo de la distribución a posteriori.

La Figura 4 presenta un resumen visual de estos resultados. Donde f_{prior} y $log(f_{prior})$ muestra una distribución uniforme (color celeste constante) en todo el dominio definido, es decir, todos los modelos dentro del rango permitido tienen igual probabilidad.

Los paneles de en medio muestran la función de verosimilitud normal $\mathcal{L}(m)$, donde se observan claramente dos máximos alineados a lo largo de $a = 0$, con orientaciones $\theta \approx \pm 45°$

(Las flechas negras marcan los modelos que presentan máxima verosimilitud.). Esta estructura es coherente con la forma y orientación de los datos generados sintéticamente: una elipse orientada en 45° y centrada en el origen. Por tanto, los modelos más verosímiles son aquellos cuya recta pasa por el origen y está alineada con dicha orientación, y $log(\mathcal{L}(m))$ que aplicar logaritmo, se generan dos zonas de alta verosimilitud se ensanchan visualmente, y las bajas se atenúan. Este comportamiento es esperado, ya que el logaritmo suaviza las diferencias entre probabilidades extremas y permite observar mejor la forma global de la función.

En la tercera columna, se tiene la función de densidad volumétrica a posteriori de los parámetros del modelo 2, donde dado que la distribución a priori es constante, la posteriori coincide en forma con la verosimilitud: $f(m) \propto \mathcal{L}(m)$. Sin embargo, se se compara con los resultados obtenidos en el gráfico `log(like)` en rojo, con el `log(fprior+like)` en verde, este último presenta una región de soluciones que no se atenúa tan rápido a los lados, a diferencia del primero.

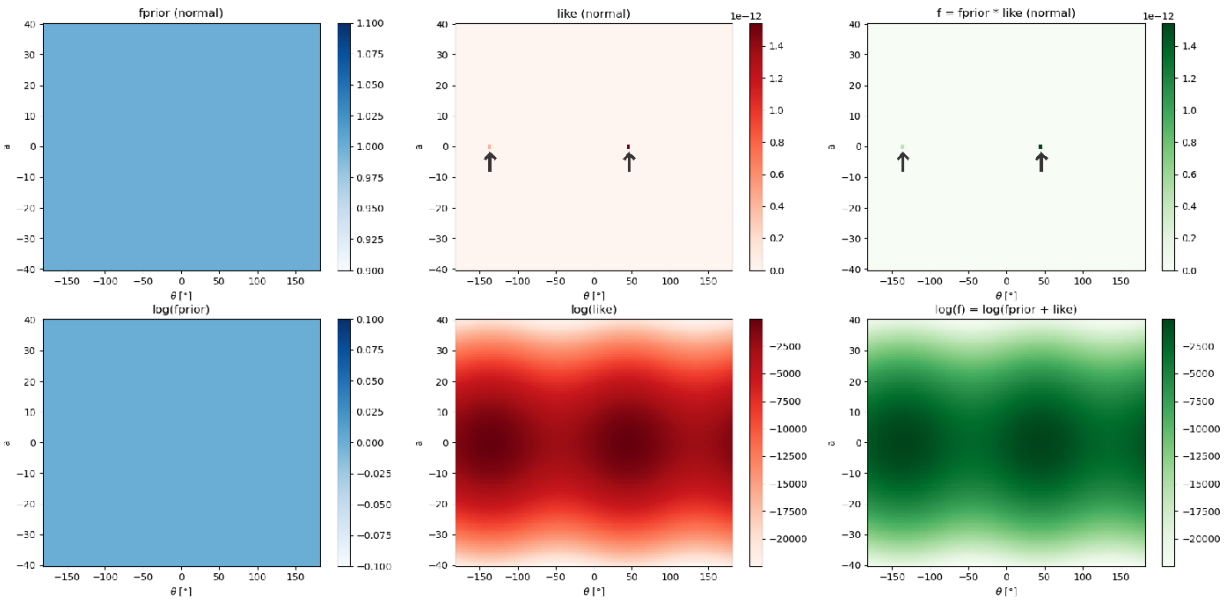


Figura 4: Evaluación por búsqueda exhaustiva de la función de verosimilitud $\mathcal{L}(m)$ sobre una grilla de a y θ. Se observan claramente los máximos relativos (señalados con flechas), lo cual confirma el correcto comportamiento del código.

Con esto, se valida que la función $\mathcal{L}(m)$ responde correctamente ante diferentes modelos, y que puede ser utilizada de forma confiable en el posterior algoritmo de Metropolis.

3. Resolución del problema inverso: Programación del Algoritmo de Metropolis

En esta etapa de la tarea se programará y verificará el algoritmo de Metropolis.

3.1. Programación del algoritmo

3.1.1. Distribución de Proposición

Una parte esencial del algoritmo de Metropolis es la distribución de proposición, encargada de generar modelos candidatos m_{test} a partir del modelo actual m. Esta distribución está implementada como una clase en la función `proposal_normal`, la cual recibe un modelo actual y devuelve un modelo propuesto. Esta implementa una distribución de proposición gaussiana multivariada. Dicha clase representa una distribución normal multivariada centrada en cero, utilizada para perturbar modelos actuales en cada paso del algoritmo de Metropolis. Su objetivo es generar candidatos mediante una propuesta simétrica, lo que permite una implementación simplificada del criterio de aceptación.

En la Figura 5 se tienen los resultados de la distribución de proposición. Se observa un buen ajuste entre los histogramas marginales (azul y verde) y las densidades teóricas (rojo), así como una distribución conjunta acorde con la forma esperada, de distribución normal.

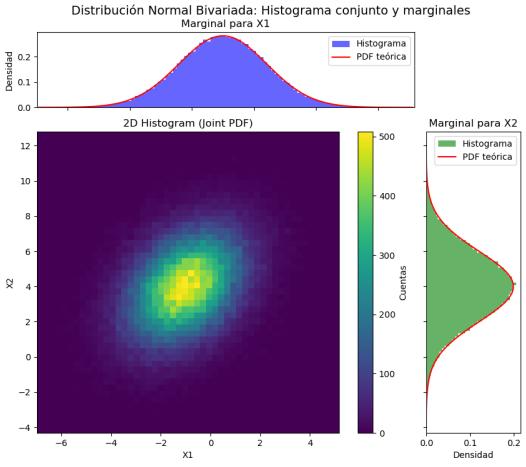


Figura 5: Validación empírica de la distribución de proposición.

3.1.2. Algoritmo de Metropolis

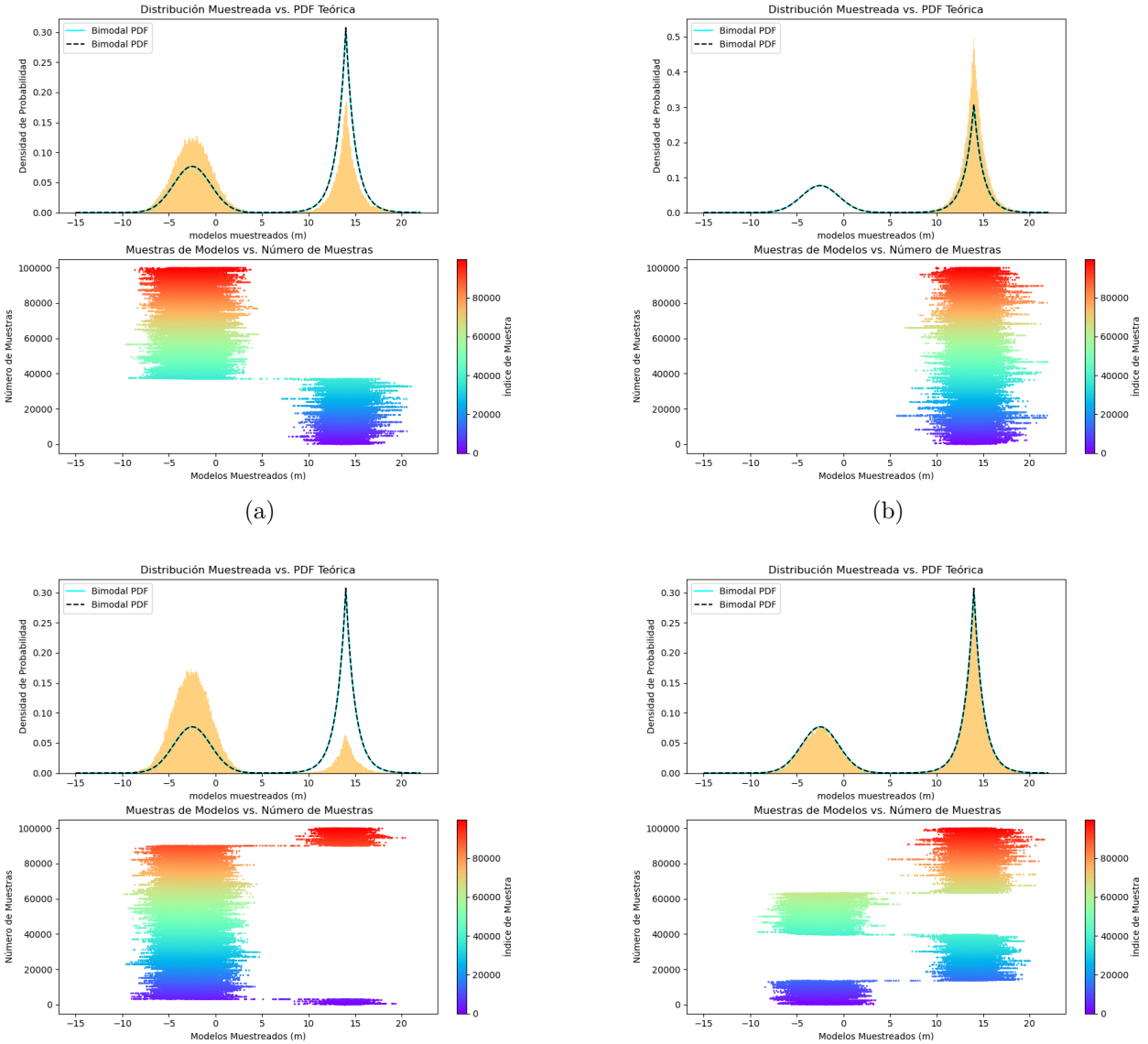
Se programó el algoritmo de Metropolis en el archivo `metropolis.py`. La implementación permite el uso tanto de verosimilitud directa como de log-verosimilitud, según el parámetro `use_log_likelihood`, incluyendo funciones auxiliares para el criterio de aceptación. Además,

incorpora el uso de una distribución a priori arbitraria y permite guardar las muestras generadas en un objeto de clase `ensemble`, junto con la tasa de aceptación.

La validación se realizó utilizando `test_metropolis.py`, donde se probó el muestreo de una distribución bimodal que actúa como verosimilitud. Esta está compuesta por la suma de dos funciones de densidad con distinta localización y dispersión, lo cual permite generar un caso representativo de posteriori multimodal, ideal para probar la capacidad del algoritmo de explorar múltiples regiones del espacio de parámetros. A continuación, en la Figura 6 se muestran los resultados obtenidos para distintas ejecuciones.

En algunas ejecuciones (a,c y b), las muestras se distribuyeron de manera equilibrada entre las dos modas de la distribución, lo que indica una correcta exploración del espacio y un buen desempeño del algoritmo. En otras ejecuciones, como la (b), la cadena de Markov se estancó en una sola de las dos modas, sin alcanzar la otra durante la simulación. Esto es un comportamiento común en cadenas MCMC cuando se utilizan propuestas locales y simétricas, y las modas están separadas por regiones de baja probabilidad.

Este experimento confirma que el algoritmo implementado es funcional y que responde correctamente a la estructura de la función objetivo. Sin embargo, también pone en evidencia las limitaciones del muestreo MCMC con propuestas locales en distribuciones multimodales, lo cual puede ser mitigado mediante propuestas adaptativas, estrategias de mezcla o cadenas paralelas en diferentes temperaturas.



(a) (b)

(c) (d)

Figura 6: Tests del algoritmo de Metropolis (4 ejecuciones diferentes)

3.2. Aplicación al Problema de Ajuste Ortogonal a la Recta

Se aplicó el algoritmo de Metropolis para resolver el problema inverso de ajuste ortogonal a una recta, utilizando datos sintéticos generados mediante la función `obtener_datos_elipses`. En este caso, los datos fueron generados a partir 50 puntos sobre una elipse con semiejes menor y mayor definidos, una rotación (α=45^o) y desplazamientos en x e y nulos. Luego, se obtiene el conjunto de observaciones (x_{obs}, y_{obs}) y sus respectivas desviaciones estándar σ_x y σ_y, que representan la incertidumbre de cada punto observado.

Se planteó el problema de encontrar la mejor recta que se ajusta ortogonalmente a estos datos, utilizando el algoritmo de Metropolis para muestrear la distribución posterior de los parámetros de la recta $m = [a, \theta]$, donde a es la distancia ortogonal desde el origen y θ es el ángulo de inclinación. Se definieron rangos para los parámetros: $a \in [-15, 15]$ y $\theta \in [-360°, 360°]$, y se utilizó una distribución a priori uniforme sobre ese dominio,y la verosimilitud fue modelada mediante una distribución normal multivariada aplicada a los residuos ortogonales normalizados. Se utilizó una propuesta gaussiana multivariada con matriz de covarianza diagonal y pequeñas varianzas para garantizar una exploración local. Se realizaron dos experimentos: uno sin aplicar burn-in y otro eliminando el 30 % inicial de las muestras. En ambos casos se generaron 5×10^4 muestras mediante el algoritmo de Metropolis.

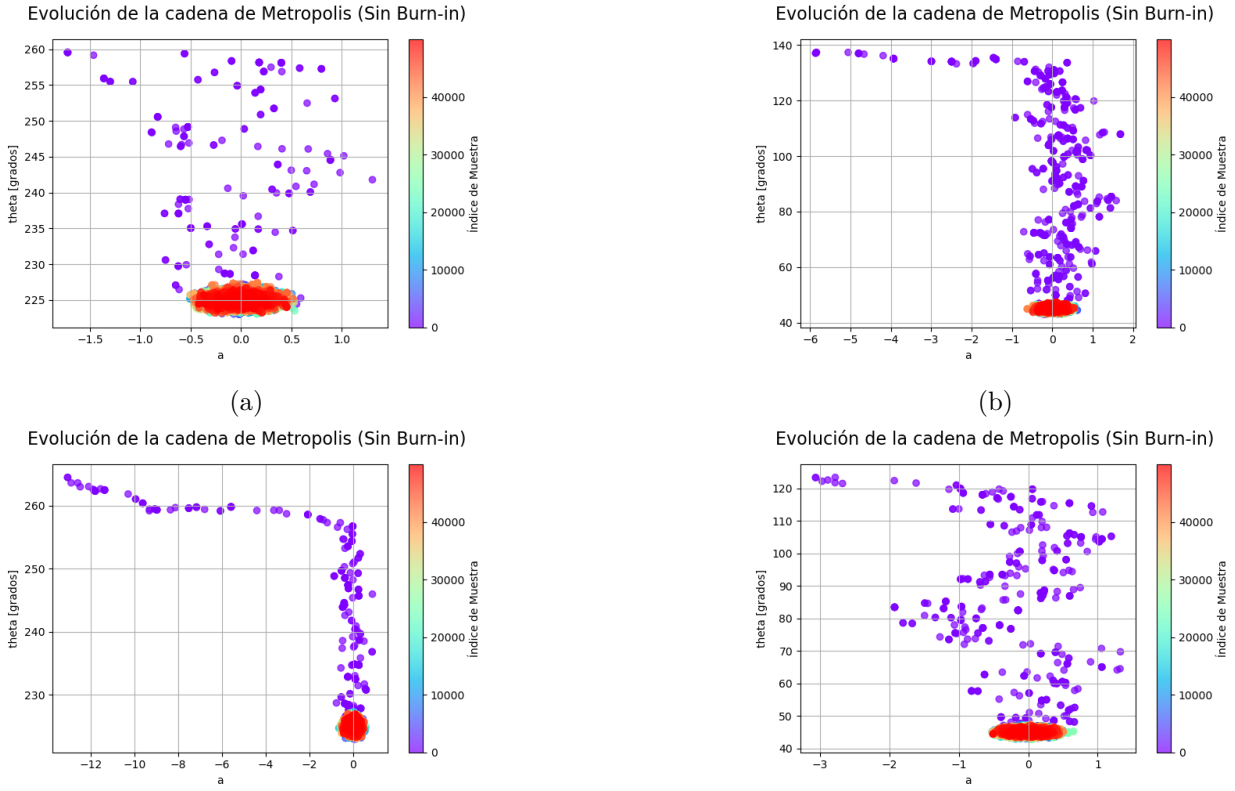


Figura 7: Evolución de la cadena de Metropolis sin aplicar Burn-in. Las muestras iniciales (en colores fríos) no representan aún la región de interés.

En la Figura 7 se muestran las cadenas completas sin descartar el Burn-in. En este caso, se observa cómo los primeros pasos (colores más fríos) aún no se encuentran en la región de interés (lo que distorsiona la representación de la distribución). Las muestras se van desplazando hacia la región de mayor verosimilitud, mostrando claramente la fase transitoria de

la cadena. Esta comparación muestra la importancia de aplicar Burn-in para eliminar los estados transitorios de la cadena y asegurar que el análisis posterior se base únicamente en muestras representativas de la distribución. Una vez alcanzada dicha región, las muestras se concentraron alrededor de dos posibles orientaciones de la recta: $\theta = 45°$ o $\theta = 225°$ (o, equivalente, $-135°$ por simetría).

Por otro lado, en la Figura 8, se visualiza la evolución de las cadenas de MCMC para diferentes condiciones iniciales, luego de aplicar un *burn-in* del 30 %, donde se observó una clara concentración de las muestras en las zonas de mayor verosimilitud. Los puntos están coloreados según el índice de la muestra, lo que permite observar la estabilización y exploración del espacio de parámetros. Se observa que las cadenas convergen hacia zonas de alta verosimilitud, validando la implementación del algoritmo.

Los histogramas marginales (Fig.9) para a mostraron que la mayoría de las muestras se agruparon cerca de $a \approx 0$, lo que es coherente con el hecho de que la recta que ajusta los datos pasa cerca del origen. Para θ, las muestras se concentraron en torno a $\theta = 45°$ y $\theta = -135°$, reflejando la simetría natural del problema (una recta con pendiente positiva o negativa puede ajustar igualmente bien una elipse con simetría central)

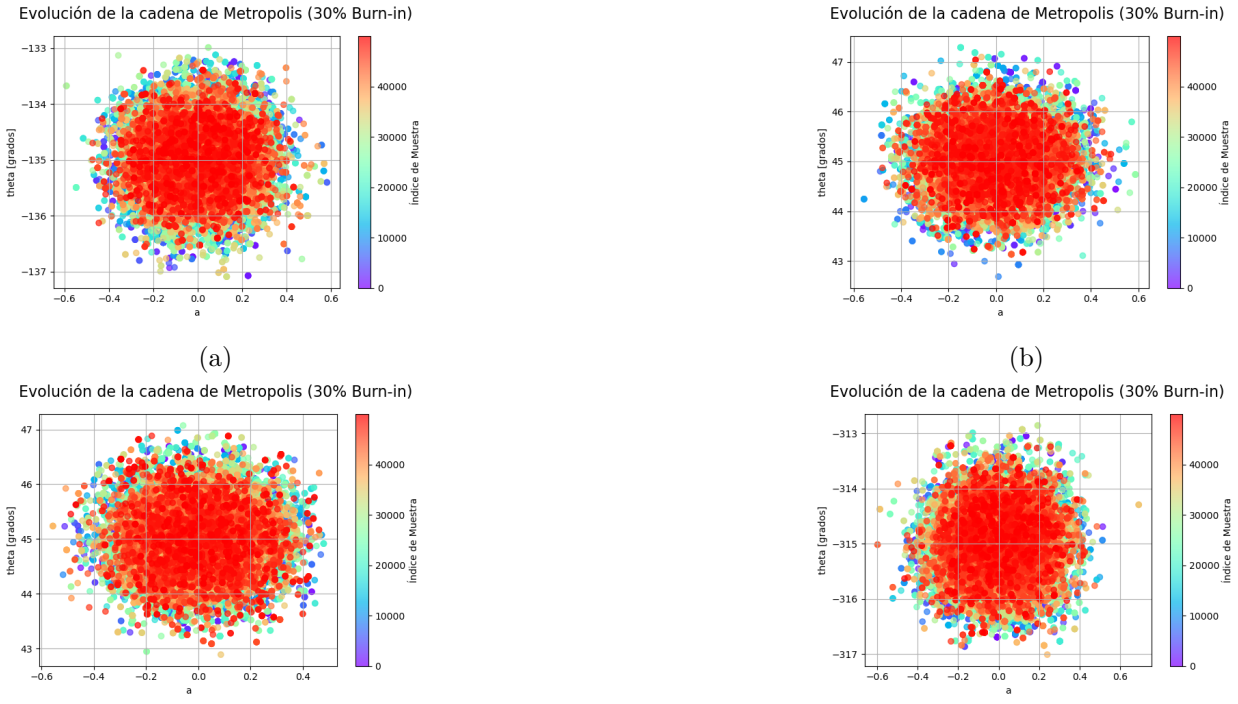
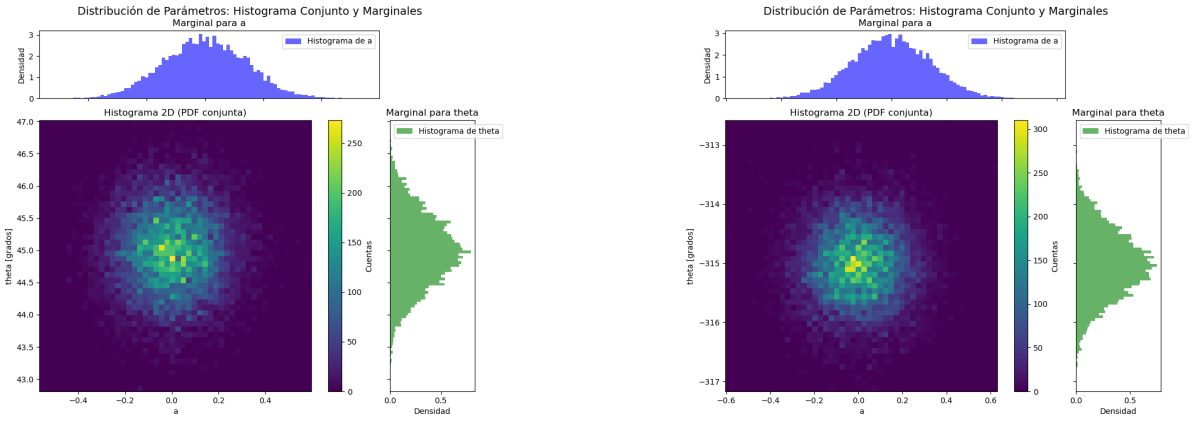


Figura 8: Evolución de la cadena de Metropolis con Burn-in del 30 %. Los puntos rojos representan muestras, ubicadas en la región de máxima verosimilitud.



(a) (b)

Figura 9: Histogramas marginales de los parámetros a y θ y la fdp conjunta de estos.

4. Programación del Algoritmo de Metropolis en Paralelo

El algoritmo de Metropolis, en su formulación estándar, es inherentemente secuencial: cada nuevo modelo propuesto depende del modelo anterior aceptado en la cadena de Markov. Sin embargo, cuando se requiere obtener múltiples muestras del espacio de parámetros para distintos modelos iniciales o se desea explorar regiones distintas del espacio simultáneamente, es posible adaptar el algoritmo a un enfoque en paralelo.

Así, se implementaron dos versiones del algoritmo de Metropolis en paralelo que utilizan diferentes cadenas de Markov para generar diferentes muestras independientes entre sí. Ambas versiones difieren en su estrategia de ejecución: la primera emplea un enfoque secuencial y la segunda utiliza procesamiento paralelo simultáneo.

4.1. Algoritmo de Metropolis en Paralelo (versión SERIAL)

Se desarrolló la versión serial del algoritmo de Metropolis en paralelo en el archivo `metropolis_in_parallel_SERIAL.py`. Esta versión ejecuta múltiples cadenas de Markov de manera secuencial, una por cada modelo inicial, usando como base el algoritmo de Metropolis para generar una muestra con cada cadena.

Cada cadena parte desde un modelo diferente y guarda el modelo aceptado final junto con la tasa de aceptación respectiva. El conjunto de modelos finales conforma un nuevo objeto `ensemble`, que representa las muestras generadas por cada una de las cadenas.

Esta implementación reutiliza la función `metropolis` y, para cada modelo inicial, genera un nuevo modelo actualizado junto con sus probabilidades a posteriori correspondientes.

4.2. Programación del algoritmo en paralelo con `Pool`

Se desarrolló una versión paralelizada del algoritmo de Metropolis mediante `multiprocessing.Pool`. Esta implementación permite ejecutar múltiples cadenas de Markov simultáneamente, cada una iniciando desde un modelo distinto del conjunto inicial. Este enfoque es especialmente útil para evitar que las cadenas se queden atrapadas en modos locales de la distribución, aprovechando múltiples núcleos del procesador para acelerar la convergencia hacia la distribución posterior deseada.

La implementación comienza construyendo una lista denominada `args_list`, que contiene los argumentos necesarios para cada cadena de MCMC que se ejecutará en paralelo. Cada entrada en esta lista incluye el modelo inicial correspondiente, el número de pasos de la cadena, las funciones de verosimilitud y de distribución a priori, la distribución de propuesta

y otros parámetros relevantes.

El núcleo del algoritmo se implementa mediante:

```
with Pool(processes=cpu_count(logical=False)) as pool:
    results = pool.starmap(metropolis, args_list, chunksize=1)
```

En este fragmento, se crea un conjunto de procesos paralelos utilizando la función `Pool` del módulo `multiprocessing`, donde el número de procesos es igual al número de núcleos físicos disponibles en el sistema (`cpu_count(logical=False)`). Esto permite que múltiples cadenas de MCMC se ejecuten simultáneamente, distribuyendo la carga computacional de manera eficiente.

La función `starmap` asigna a cada proceso la tarea específica de ejecutar la función `metropolis` con los argumentos definidos en la lista `args_list`, donde cada entrada corresponde a una cadena independiente. El resultado de cada ejecución se almacena automáticamente en la lista `results`, permitiendo luego reconstruir el conjunto de modelos generados por las diferentes cadenas.

Una vez completada la ejecución paralela, se recorre la lista de resultados mediante un bucle `for`, extrayendo y almacenando el modelo final de cada cadena, así como sus valores de verosimilitud y probabilidades a priori y posterior. Estos datos se guardan dentro de un objeto de clase `ensemble`, que permite trabajar luego con el conjunto de muestras obtenidas. Finalmente, se calcula y guarda la tasa de aceptación de cada cadena.

Cabe mencionar que cada cadena se ejecuta sin establecer una semilla para generar las muestras aleatorias, con lo que se garantiza independencia estadística entre los procesos paralelos.

4.3. Verificación del algoritmo

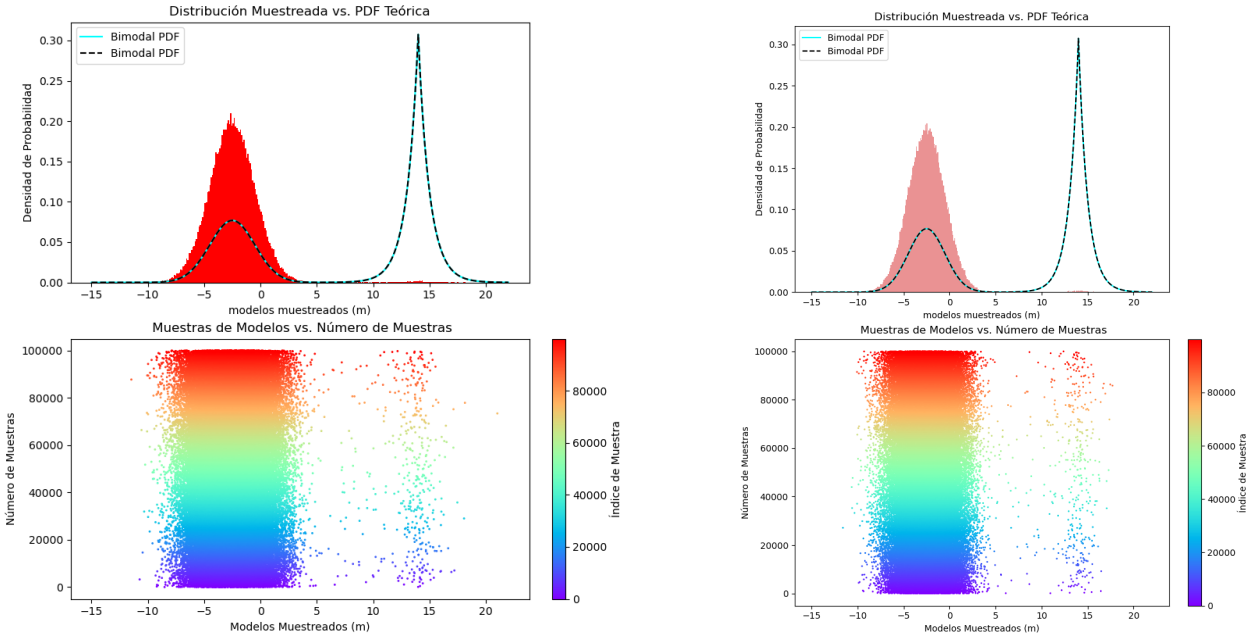
Una vez implementadas ambas versiones del algoritmo, se desarrollaron dos pruebas para verificar y validar su correcto funcionamiento. Se implementó una prueba para la versión serial y otra para la versión paralelizada, ambas adaptadas de `test_metropolis.py`.

En ambas pruebas se generó un conjunto de modelos iniciales muestreados desde la distribución a priori y se utilizó como función de verosimilitud una distribución bimodal. A partir de estos parámetros se ejecutaron múltiples cadenas independientes de Metropolis, lo que permite evaluar si las cadenas logran capturar ambas modas de la distribución objetivo.

Al analizar los resultados obtenidos en la Fig. 10, se observó que tanto la versión serial como la paralelizada concentran sus muestras únicamente en una de las modas, en particular en $x = -2.5$, ignorando la segunda moda ubicada en $x = 14$. Esto indica que, si bien el algoritmo funciona correctamente desde el punto de vista computacional y estadístico, presenta dificultades para explorar distribuciones multimodales cuando la propuesta utilizada es local y las modas están separadas por regiones de baja probabilidad.

Este comportamiento se puede explicar debido a la propuesta gaussiana utilizada en ambos casos, sumado a la independencia de las cadenas y a la falta de diversificación en las condiciones iniciales. Una vez que una cadena comienza en las cercanías de una moda, resulta poco probable que logre saltar a la otra, especialmente si el número de iteraciones es limitado y la varianza de la propuesta es pequeña.

Lo que se pensó como un error de implementación, este fenómeno refleja una limitación al algoritmo de Metropolis clásico cuando se enfrenta a funciones objetivo multimodales. Para superar esta dificultad, se recomienda considerar estrategias como aumentar la varianza de la propuesta, diversificar los modelos iniciales, o bien implementar métodos más avanzados como *Transitional MCMC (TMCMC)*, *CATMIP* o técnicas adaptativas que mejoren la exploración del espacio de parámetros.



(a) MIP Serial

(b) MIP Pool

Figura 10: Distribución de las muestras.

4.4. Aplicación al Problema de Ajuste Ortogonal a la Recta

Para estimar los parámetros de una recta mediante ajuste ortogonal, se aplicó el algoritmo de Metropolis utilizando una distribución a priori uniforme sobre el espacio de parámetros (a, θ), con $a \in [-15, 15]$ y $\theta \in [-360°, 360°]$. Las observaciones se generaron mediante puntos sintéticos sobre una elipse de semiejes $a = 20$, $b = 2$, con rotación $\alpha = 45°$ y desplazamientos nulos. Se definió una distribución de verosimilitud a partir del modelo directo de residuos normalizados, asumidos como variables aleatorias gaussianas estándar.

A diferencia de la sección P3.2, en esta implementación los modelos iniciales del conjunto de cadenas fueron extraídos directamente como muestras de la distribución a priori, garantizando una exploración inicial más amplia del espacio de parámetros.

En la Figura 11, se muestra la evolución de la cadena de Markov en el espacio de parámetros. Puede observarse cómo, luego del *burn-in*, las muestras se concentran en una región coherente con los valores verdaderos de los parámetros. Esta distribución es visualizada de forma más clara en la Figura 12, donde se grafican los histogramas marginales de a y θ, junto con el histograma 2D de la densidad conjunta.

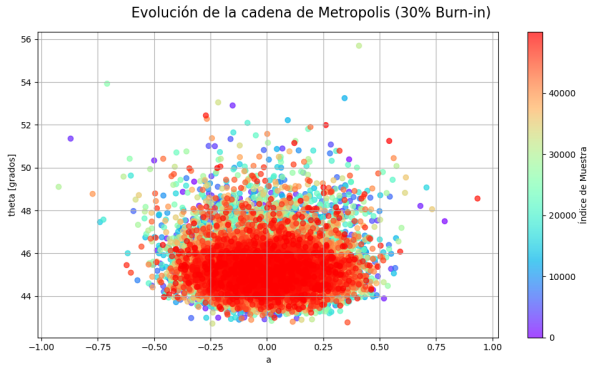


Figura 11: Evolución de la cadena de Metropolis para el problema de ajuste ortogonal. Las muestras están coloreadas según su índice temporal.

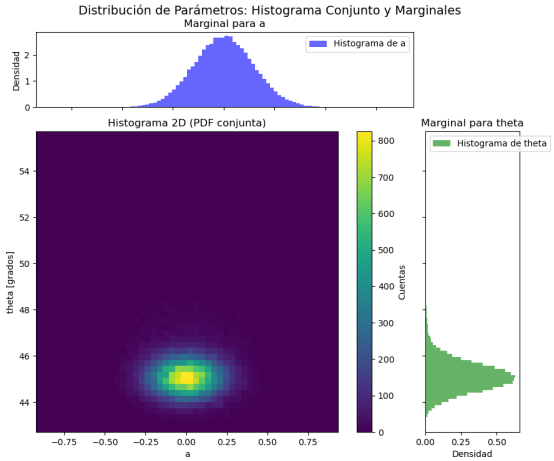


Figura 12: Distribución conjunta y marginales de los parámetros obtenidos con MCMC.

Los resultados confirman que el algoritmo fue capaz de recuperar la orientación y pendiente de la recta original, con una alta concentración de probabilidad en torno a $\theta \approx 45°$ y $a \approx 0$, coherente con la configuración geométrica de los datos sintéticos. La fase de *burn-in* resultó fundamental para eliminar los efectos transitorios de las condiciones iniciales y obtener una representación estable de la distribución.