GF7013 - Métodos Inversos Avanzados

Tarea 2

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

Fecha Publicación: Lunes 2 de junio de 2025
Fecha Última de Entrega: Jueves 19 de junio de 2025

En esta tarea se agregará funcionalidades al paquete de Python 3 llamado GF7013, que comenzaron a desarrollar en la Tarea 1. En particular se agregará el algoritmo de Metrópolis y Metrópolis en paralelo, así como definir funciones que permitan implementar los algoritmos antes mencionados. PARA ESTA TAREA DEBEN ENTREGAR EL PAQUETE GF7013 ACTUALIZADO CON LOS CODIGOS A DESARROLLAR.

Se provee una carpeta llamada Tarea2, con un archivo que se referencia mas adelante. Reemplaze la carpeta Tarea2 que se encuentra al interior de la carpeta `GF7013/bin/` con la que se provee con este enunciado.

P1. Programación del Problema Directo

Se provee de una carpeta `ajuste_ortogonal_recta` que deberá copiar en la carpeta del paquete de python desarrollado en la Tarea 1, es decir al interior de `GF7013/models/`. En este módulo se encuentra la definición de la recta paramétrica y el prototipo de una clase `forward` que permitirá calcular el modelo directo.

En un intento por generalizar los códigos de muestreo que se utilizan posteriormente, se definirá un requerimiento que debe cumplir la clase forward (que implementa el modelo directo para cualquier problema y no sólo para el ajuste ortogonal a la recta): que un objeto que sea instancia de la clase forward debe implementar la función miembro `eval(m)` donde `m` es el vector de parámetros del modelo.

Para el problema del ajuste ortogonal de la linea recta, se tiene $\mathbf{m} = [a, \theta]^\top$, donde a es la distancia de la recta al origen y θ define la orientación de la recta en sentido anti-horario con respecto al eje x (ver apunte que define el problema de ajuste ortogonal a la recta).

Luego se pide:

P1.1 completar la función `eval(self, m)` de la manera más concisa posible. Para ello utilice el paquete `recta` que se provee.

P1.2 genere datos sintéticos para ajustar una recta mas adelante en la tarea (ver P1b.py en la carpeta Tarea2), y grafique las distancias y desviaciones estándar de estas para la predicción de un modelo $\mathbf{m}$ de recta a su elección.

P2. Modelación Inversa usando Métodos Bayesianos : A priori's y función de verosimilitud

En la modelación Bayesiana planteada en el curso, se estima la solución del problema inverso como una familia de modelos que son muestras de la función de densidad volumétrica a posteriori de los parámetros del modelo. Es

decir, si $\mathbf{m}$ el vector de parámetros del modelo, se busca obtener muestras de la fdp volumétrica

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

donde $f_{prior}(\mathbf{m})$ es la densidad volumétrica a priori de los parámetros del modelo, $\mathcal{L}(\mathbf{m})$ es la función de verosimilitud y ν es un factor de normalización de la densidad volumétrica a posteriori.

En este trabajo se define el vector de parámetros del modelo como

$$\mathbf{m} = \begin{bmatrix} a \\ \theta \end{bmatrix} \tag{2}$$

Antes de seguir, se pide por favor :

- copie las carpetas `metropolis` y `metropolis_in_parallel` en su carpeta `GF7013/sampling/` reemplazando las carpetas existentes.
- copie la carpeta `likelihood` dentro de su carpeta `GF7013/probability_functions/` reemplazando la existente.
- copie la carpeta `model_parameters` en su carpeta `GF7013/` reemplazando la carpeta existente.

P2.1. Densidad volumétrica a priori para los parámetros del modelo $\mathbf{f_{prior}(m)}$

Para la densidad volumétrica a priori de los parámetros del modelo $f_{prior}(\mathbf{m})$, se asumirá el estado homogéneo de información. Para ello, se considera que los parámetros del modelo son ambos parámetros cartesianos, por lo que la fdp a priori para $\mathbf{m}$ será la de una distribución uniforme.

Se debe definir un rango de valores posibles para el parámetro a que sea lo suficientemente grande, por ejemplo, ± 2 veces el máximo de las normas de $\mathbf{d_k^{obs}}$ (que son los datos observados a ajustar con la recta, o los datos sintéticos a usar en esta tarea).

Además, se define el rango de posibles valores de θ en el intervalo $[-180, 180[$ grados sexagesimales.

En la Tarea 1, en `GF7013/probability_functions/pdf/pdf_uniform_nD.py` programaron una clase `pdf_uniform_nD`, que permite calcular la verosimilitud (i.e., sin normalizar) - y su logaritmo natural - de una distribución uniforme de n dimensiones.

En la carpeta `Tarea2/tests` escriba un script `test_fprior.py` que cree una instancia de `pdf_uniform_nD` que represente $f_{prior}(\mathbf{m})$ para el caso del problema del ajuste a la recta y genere 1E5 muestras de $f_{prior}(\mathbf{m})$. Graficar el histograma conjunto de (a, θ) – use matplotlib.pyplot.hist2d – y los histogramas marginales para cada uno de los dos parámetros. Recuerde escoger bien el número o tamaño de casilleros del histograma para que luzca como una distribución uniforme.

P2.2. Función de Verosimilitud $\mathcal{L}(\mathbf{m})$

Para poder definir la función de verosimilitud de un modelo $\mathcal{L}(\underline{\mathbf{m}})$ primero se definirá la distribución a priori para la variable de interés. En este caso, como se desea hacer un ajuste ortogonal de una recta, se quiere encontrar modelos que minimicen la distancia a los datos. Luego, la variable sobre la cual se debe definir información a priori corresponde a la distancias Δ_k entre los pares ordenados observados y la recta (ver archivo pdf adjunto con la tarea y explicado por el profesor auxiliar).

En esta tarea se asume que los errores de dichas distancias son independientes y que siguen una distribución normal. Como se quiere que dichas distancias sean lo más pequeñas posibles, se propone utilizar una distribución normal, con media nula y varianzas $\sigma^2_{\Delta_k}$ con fdp:

$$\rho_{\Delta}(\boldsymbol{\Delta}, \boldsymbol{\sigma}^{\mathbf{2}}_{\boldsymbol{\Delta}}) = \nu e^{-\frac{1}{2}\sum_{k=1}^{N} \frac{\Delta_k^2}{\sigma^2_{\Delta_k}}} \tag{3}$$

donde ν es una constante de normalización.

Luego, si para un modelo $\mathbf{m}$ dado, se calcula las distancias $\boldsymbol{\Delta}(\mathbf{m})$ y la varianza de dichas distancias $\boldsymbol{\sigma}^2_{\boldsymbol{\Delta}}(\mathbf{m})$, la verosimilitud de dicho modelo se puede calcular como:

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

La función de verosimilitud se encuentra programada en `GF7013/probability_functions/likelihood_function.py`. Esta recibe una instancia de modelo directo (`forward`) y una instancia de la fdp a priori de los datos (`pdf_data`). Se recomienda leer el código de la función de verosimilitud antes de proseguir con el desarrollo de la tarea.

P2.2.1. Verificación de la programación de ρ_{Δ}

La función de probabilidad a priori para las observaciones se asumirá como una distribución Normal Multivariada. En la Tarea 1, se programó la clase `pdf_normal.py` en el módulo `GF7013/probability_functions/pdf_normal.py`. En esta parte de la tarea debe cerciorarse de que dicha clase funcione, y es una oportunidad para arreglar cualquier error que haya tenido en el desarrollo de la Tarea 1.

Para verificar su programación de `pdf_normal.py`, cree un script en el módulo `Tarea2/tests/test_pdf_normal.py` que genera 100.000 muestras de una distribución normal multivariada con media `mu = np.array([-1, 4])` y matriz de covarianza `cov = NP.array([[2, 1], [1, 4]])` y que luego verifique que el promedio y matriz de covarianza calculados con las muestras sea similar a `mu` y `cov` anteriores, respectivamente. Además, se pide hacer un histograma 2D de las muestras de la fdp conjunta y hacer histogramas de las 2 fdp marginales.

P2.2.2. Verificación de la programación de $\mathcal{L}(\mathbf{m})$

Para verificar la programación de $\mathcal{L}(\mathbf{m})$, se utilizará el método de Grid-Search para resolver el problema inverso (ver apuntes del curso GF5013), esto se puede hacer ya que solo hay dos parámetros involucrados. Para ello, use los datos sintéticos que se proveen en `GF7013/bin/Tarea2/P1/P1b.py` y cree un script en el módulo `Tarea2/tests/test_Likelihood.py`.

En dicho script,

(I) Defina un conjunto de valores de a y θ (use `np.linspace`) que cubran el rango de valores con probabilidad no nula de $f^{prior}(m)$. Luego, cree un conjunto de modelos $\mathbf{m}$ con todas las combinaciones posibles de a y θ.

(II) Cree una instancia de la clase `ensemble`, que se encuentra definida en `GF7013/model_parameters/ensemble.py`, guarde ahí el conjunto de modelos y llene las variables fprior, like, y f con las verosimilitudes correspondientes a cada uno de esos modelos (i.e., use la función `likelihood` de los objetos definidos para $f_{prior}(\mathbf{m})$ y $\mathcal{L}(\mathbf{m})$)

(III) Cree otra instancia de la clase `ensemble`, que se encuentra definida en `GF7013/model_parameters/ensemble.py`, guarde ahí el conjunto de modelos y llene las variables `fprior`, `like`, y `f` con el logaritmo natural de las verosimilitudes correspondientes a cada uno de esos modelos (i.e., use la función `log_likelihood` de los objetos definidos para $f_{prior}(\mathbf{m})$ y $\mathcal{L}(\mathbf{m})$)

(IV) Haga un gráfico 2D con colores (puede usar matplotlib.pyplot.pcolor o matplotlib.pyplot.scatter, en el segundo teniendo cuidado con el tamaño de los símbolos) para los valores de verosimilitud `fprior`, `like`, y `f`; y del logaritmo natural de la verosimilitud `fprior`, `like`, y `f` (son 6 gráficos de colores en total).

Hint: ver comentarios y docstrings en `GF7013/model_parameters/ensemble.py`.

P3. Resolución del problema inverso: Programación del Algoritmo de Metrópolis

En esta etapa de la tarea se programará y verificará el algoritmo de Metrópolis. Antes de proseguir se recomienda leer el encabezado (docstring) del código del algoritmo de Metropolis en `GF7013/sampling/metropolis/metropolis.py`.

P3.1. Programación del algoritmo

P3.1.1. Distribución de Proposición

Una parte esencial del algoritmo de Metropolis es la distribución de proposición, la que propone modelos candidatos a ser aceptados como muestras de $f_{post}(\mathbf{m})$. Bajo la arquitectura propuesta para los codigos del paquete GF7013, se pide que la distribución de proposición sea representada por un objeto que tiene la función miembro `propose(m)` que recibe el modelo actual de la cadema (`m`) y devuelve un modelo de prueba `m_test` que es candidato a ser aceptado como el siguiente paso de la cadena de Markov (i.e., como muestra de $f_{post}(\mathbf{m})$).

Se pide completar la programación en de la clase `proposal_normal` en `GF7013/sampling/metropolis/proposal_normal.py`, que representa una distribución de proposición que produce modelos `m_test` perturbando el modelo `m` con realizaciones de una distribución normal con media nula y matriz de covarianza `cov`. Asimismo, cree el modulo `Tarea2/tests/test_proposal_normal.py` que permita verificar el correcto funcionamiento de la distribución de proposición (siga la idea expuesta en P2.2.1).

P3.1.2. Algoritmo de Metropolis

Complete la programación del algoritmo de metropolis en el módulo `GF7013/sampling/metropolis/metropolis.py`. Note que se contempla ambos casos: 1) en que se calcula las verosimilitudes de la distribución de probabilidad f_{prior} y de la función de verosimilitud ($\mathcal{L}$), asi como 2) en que se calcula el logaritmo natural de dichas verosimilitudes. Verifique el correcto funcionamiento de su código corriendo el test en `GF7013/bin/Tarea2/tests/test_metropolis.py` (debería entregar un resultado similar a los ejemplos que se verá en clases.)

P3.2. Aplicación al Problema de Ajuste Ortogonal a la Recta

Resuelva el problema de Ajuste Ortogonal a la recta, (ver datos sintéticos para esta pregunta en P32.py). Para ello asuma que los parámetros que definen la recta quedan restringidos a los intervalos siguientes: $a \in [-15, 15]$ y $\theta \in [-360, 360]$. Resuelva el problema para un modelo inicial lo suficientemente lejos de las zonas de mayor verosimilitud de f_{post}. En un grafico de a v/s θ muestre la evolución de las muestras de la cadena de markov (incluyendo la etapa del burn-in). Asimismo, haga un grafico mostrando los histogramas que aproximan las fdp marginales para a y θ y otro gráfico en donde se muestra la fdp conjunta (para este último use matplotlib.pyplot.hist2d). Comente sus resultados.

Desarrollar la tarea en el módulo de python `Tarea2/P32/P32.py`.

P4. Resolución del problema inverso: Programación del Algoritmo de Metrópolis en Paralelo

En esta etapa de la tarea se programará y verificará el algoritmo de Metrópolis en Paralelo. Para ello, utilizando como base el algoritmo de Metropolis recien programado, se programará una versión serial del algoritmo de Metrópolis en paralelo, y por último una versión paralelizada utilizando multiprocessing.Pool.

P4.1. Algoritmo de Metrópolis en Paralelo (versión SERIAL)

Termine la programación de los códigos en el módulo `GF7013/sampling/metropolis_in_parallel/metropolis_in_parallel_SERIAL.py`

P4.2. Algoritmo de Metrópolis en Paralelo (versión paralelizada)

Termine la programación de los códigos en el módulo `GF7013/sampling/metropolis_in_parallel/metropolis_in_parallel_POOL.py` (veremos como programar en multiprocessing.pool en clases).

P4.3. Verificación del algoritmo

Cree dos copias de `Tarea2/tests/test_metropolis.py` en `Tarea2/tests/test_metropolis_paralelo_SERIAL.py` y `Tarea2/tests/test_metropolis_paralelo_POOL.py`. Modifique estos últimos para verificar que sus algoritmos de Metropolis en Paralelo funcionan correctamente.

P4.4. Aplicación al Problema de Ajuste Ortogonal a la Recta

Repita los pasos descritos en P3.2, pero considerando un conjunto de modelos iniciales generados como muestras de f_{prior}. Desarrolle los scripts para esta parte de la tarea en la carpeta `Tarea2/P44`.

Instrucciones

- La tarea se realizará en grupos (ya definidos en Ucursos).
- Se prohíbe la colaboración entre grupos.
- Copiar la estructura de carpetas que se entrega en esta tarea dentro del paquete GF7013 que desarrollaron para la Tarea 1.
- Al entregar la tarea deben entregar, el informe debe ser escrito en word, latex, o en un Jupyter notebook (donde puede cargar los resultados los diferentes procesos y hacer los gráficos, o cargar dentro del notebook las imagenes de sus resultados) esto de manera ordenada y se entienda claramente los gráficos y las respuestas a cada parte de la tarea. A u-cursos se debe subir un archivo comprimido que tenga la estructura de carpetas del paquete `GF7013` con los códigos e informe al interior.
- Asimismo es una excelente práctica el escribir códigos que sean modulares de manera de poder reutilizar la mayor cantidad de veces ese código (obviamente sin abusar).

Exito!!!..