Ajuste lineal a una distribución de datos utilizando un likelihood de Poisson

Moreno Hernández Ana Isabel*

Universidad de Guanajuato, División de Ciencias e Ingenierías, Lomas del Bosque 103, Lomas del Campestre, 37150 León, Gto.

*morenoha2016@licifug.ugto.mx

Se hizo un ajuste lineal a una distribución de datos empleando tres métodos diferentes con likelihood poissoniano. Los tres métodos fueron el algoritmo de Metrópolis, el uso de la librería de Python EMCEE y el uso del módulo de Python Multinest. Se generaron gráficas de las distribuciones de cada parámetro del modelo lineal, se obtuvieron las medianas y se compararon los tres métodos empleados.

1. Motivación

Emplear un likelihood poissoniano en el método de metrópolis para ajustar a una línea una distribución de datos. Se espera comparar el ajuste obtenido con el resultado de dos módulos de Python: EMCEE y Multinest.

Este proyecto tiene como motivación inicial calcular los valores para los parámetros de cierta distribución de energías generada por la emisión de rayos gamma del cúmulo globular *Omega Centauri*. El proyecto hasta ahora abarca el empleamiento de un likelihood poissoniano para ajustar una distribución de probabilidad a una línea.

Es recomendable utilizar un likelihood poissoniano debido a que los datos para la distribución de estas energías son pocos en cantidad y un likelihood gaussiano requiere de muchos más.

2. Objetivos

- Emplear el método de metrópolis para ajustar una distribución de probabilidades a una línea utilizando un likelihood poissoniano.
- Emplear los módulos de python EMCEE y Multinest para ajustar a una línea el mismo conjunto de datos.
- Comparar los tres métodos y realizar conclusiones.

3. Introducción

El análisis de datos es el proceso de evaluar datos utilizando herramientas analísticas y estadísticas para descubrir información importante. En este caso, se hizo uso de un método conocido como **Cadenas de Markov: algoritmo de metrópolis**, así como de dos módulos de Python (**EMCEE** y **Multinest**) para ajustar a una línea una distribución de datos.

Para una pequeña distribución, que no puede aproximarse a una distribución gaussiana, sino a una distribución de Poisson, el likelihood está dado por

$$L = \prod_i^n \frac{\mu_i^k e^{-\mu_i}}{k!}, \tag{1}$$

donde μ sería el valor esperado dado nuestro modelo [1]. Luego, el logaritmo natural de este likelihood sería

$$\ln(L(k|\mu_i)) = \sum_i^n [k \ln \mu_i - \mu_i - \ln(k!)]. \tag{2}$$

En términos de los datos y de los parámetros de algún modelo λ, el logaritmo natural del likelihood es

$$\ln(L(\vec{x}, \vec{y}|\vec{\theta})) = \sum_i^n [y_i \ln \lambda(x_i, \vec{\theta}) - \lambda(x_i, \vec{\theta}) - \ln(y_i!)], \tag{3}$$

donde hemos sustituido $k = y_i$ y $\mu_i = \lambda(x_i, \vec{\theta})$ en la ecuación (2).

Para maximizar este likelihood, utilizamos el algoritmo de metrópolis.

3.1. Algoritmo de metrópolis

El **método de metrópolis** es un algoritmo computacional que consiste en empezar con un valor inicial para los parámetros del modelo y, en la vecindad de estos valores, de forma aleatoria, elegir otros nuevos. Se calcula el valor del likelihood para estos dos valores y, el que tenga el valor más grande, se acepta. Se vuelve a generar otro valor aleatorio en la vecindad del anterior y se hace la misma comparación, hasta que se llene a una región donde el likelihood es máximo.

Supongamos que tenemos una distribución de probabilidad aproximadamente lineal, es decir, $p(x) = ax + b$, donde a es la pendiente y b es la ordenada al origen. Este conjunto de datos es pequeño y tiene un tamaño de $N = \sum_{j=1}^n y_j$, con probabilidad $p(x) = 0$ fuera de los límites. Deseamos encontrar los valores de a y b que más se ajusten a la distribución. Es aquí donde decidimos utilizar un likelihood poissoniano y no gaussiano, debido a la cantidad de datos. Dado este modelo, nuestro likelihood es

$$L = \prod_i^n \frac{(ax_i + b)^{y_i} e^{-(ax_i+b)}}{y_i!}. \tag{4}$$

Y su logaritmo natural

$$\ln L = \sum_i^n [y_i \ln(ax_i + b) - (ax_i + b) - \ln(y_i!)]. \tag{5}$$

Utilizando cualquiera de las dos últimas expresiones y empleando correctamente el modelo de metrópolis para cada una, podemos encontrar el valor de los parámetros a y b que mejor se ajusta a nuestra distribución de probabilidad aproximadamente lineal.

3.2. Módulo de Python: EMCEE

EMCEE es un módulo de Python para ajustar un modelo a una distribución de datos.

Cuando se inicia con este módulo, lo primero que se define es el likelihood que se usará para estimar el valor de los parámetros del modelo. Cuando se define el modelo y el likelihood, puede utilizarse *minimize* de la librería *scipy.optimize* para encontrar el valor numérico óptimo de este likelihood, pero esto sólo es un método alternativo para encontrar el valor de los parámetros que mejor se ajustan al modelo (véase la referencia [2] para más información).

En EMCEE pueden definirse los *priors* de los parámetros de nuestro modelo para realizar un mejor ajuste y, al terminar de configurar todo, si se utiliza *minimize* de *scipy*, pueden inicializarse los caminadores en un pequeña bola gaussina alrededor de este resultado del likelihood máximo y luego correr con N pasos EMCEE.

Al correr EMCEE, lo primero que puede observarse es el valor del parámetro en cada iteración por cada caminador para observar a qué valor converge cada uno. Después pueden generarse gráficas con *corner* para observar la distribución de los parámetros y calcular los percentiles deseados de la distribución de cada parámetro. En la referencia [] puede encontrarse cómo instalar este módulo así como un ejemplo simple del ajuste de una distribución de datos a un modelo lineal.

3.3. Multinest

Multinest es un módulo de Python para análisis bayesiano, estimación de parámetros y selección de modelos.

En este módulo también es necesario definir un likelihood, así como la cantidad de parámetros del modelo y establecer un prior para cada parámetro. Existe un programa llamado *pymultinest.demo.py*, el cual es un script de Python que puede encontrarse en la referencia [3]. Este script puede modificarse para definir un likelihood y modelo deseados, pero es necesario establecer un prior para que funcione. Este prior se define por medio de un cubo de área o volumen (dependiendo al número de parámetros) igual a 1. Se elije cada dimensión de este cubo como un parámetro y se modifica el *largo* de cada dimensión de acuerdo al rango de cada parámetro.

Al correr el script, se observa el valor del likelihood en cada iteración y, al final, los valores para los parámetros del modelo con una desviación.

4. Desarrollo

Antes de implementar los tres métodos anteriormente mencionados, se generaró una distribución de datos aproximadamente lineal utilizando el modelo $p(x) = 3x + 2$.

Para el método de metrópolis, se intentó utilizar la expresión (5) como logaritmo natural del likelihood para observar si el valor de los parámetros convergía. Debido a que no se observó una convergencia hacia algún valor para cada parámetro, se optó por utilizar la expresión (4), con $\mu_i = ax_i + b$ y $k = y_i$. Utilizando esta expresión como likelihood, se empleó el método de metrópolis primero con un camino para observar que el método funcionara. Al verificar su funcionamiento, se implementó el método con 20 caminadores y 40000 pasos cada uno para encontrar los valores de los parámetros que mejor ajustan el modelo a los datos.

En este método se realizó una gráfica para el camino seguido para llegar a los valores de a y b, luego se quitó el burning de los arrays que guardan los valores de los parámetros para cada caminador y se realizó un histograma para cada parámetro para observar su distribución.

Para saber si nuestros caminadores convergían a algún valor, utilizamos el método de Gelman-Rubin [4], método discutido y analizado en clase. Finalmente, se graficó el valor de a y b en cada iteración para cada camino, se calcularon las medianas de las distribuciones de a y de b, y se graficó el ajuste.

Para el caso de EMCEE, debido a que el método contiene sólo operaciones entre arreglos, no fue necesario hacer uso de un servidor para correr el programa. De esta forma, en el mismo notebook, se definió el likelihood utilizando la expresión (5), se utilizó *minimize* de *scipy.optimize* para encontrar el valor óptimo de este likelihood y se optó por definir un prior para a y b en los rangos [0.05,4] y [0,3], respectivamente. Finalmente, se corrió EMCEE con 7 caminadores y 50000 pasos, se obtuvieron las gráficas de los valores de cada parámetro en cada iteración y se generó la gráfica de las distribuciones con *corner* para calcular los valores de los percentiles 16, 50 y 84 de cada distribución.

En el caso de Multinest, se hizo uso de un servidor. Se modificó el script *pymultinest.demo.py* utilizando como logaritmo natural del likelihood poissoniano la expresión (5). Para el prior definido como un cuadrado, debido a que sólo se tienen dos parámetros para un modelo lineal, se optó por definir los priors para a y b de [0,5] cada uno.

Aunque en Multinest no generamos ninguna gráfica, sí obtuvimos los valores para cada parámetro, lo cual es suficiente para comparar con los otros métodos empleados.

5. Resultados

Los códigos implementados para el método de metrópolis y las gráficas de las distribuciones se encuentran en la referencia [6]. Al utilizar este método, primero se hizo un intento con un sólo caminador para verificar su funcionamiento, después se implementó con 20 caminadores para observar los diferentes resultados y tener más información sobre los parámetros del modelo lineal. En la figura 1 se muestra el camino seguido por cada caminador.

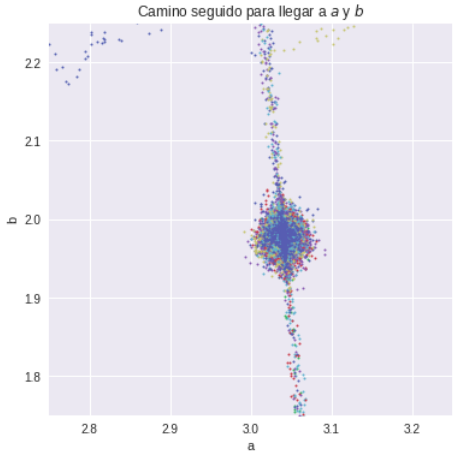

FIGURA 1. Valores de a y b seguidos por cada caminador.

Como se puede observar en esta última figura, aunque cada camino parta de un valor inicial para a y b diferente, éste llega a una zona de convergencia en donde los valores de los parámetros cambian de manera muy poco significativa. Quitamos todos aquellos puntos que no se encuentren en esta zona de convergencia, es decir, el burning y obtenemos las distribuciones gráficas de los valores para a y b, las cuales se muestran en las figuras 2 y 3.

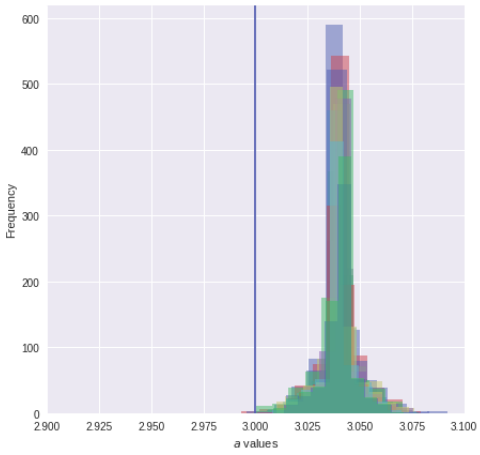

FIGURA 2. Distribución de valores para a.

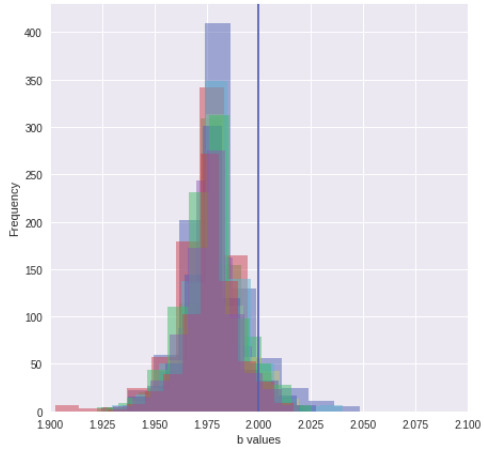

FIGURA 3. Distribución de valores para b.

En estas dos últimas figuras podemos notar algo muy interesante: las distribuciones son similares a una distribución de Poisson. Además, he colocado un línea vertical en los valores verdaderos de a y b para observar qué tan alejada está la media de esta distribución con estos valores, ya que lo ideal es que la distribución esté centrada en esta línea, pero aún así podemos observar que los centros de estas dos distribuciones son muy cercanos a los valores reales.

Como ya se mencionó en la sección anterior, se utilizó el método de Gelman-Rubin para saber si nuestras cadenas habían convergido. Se implemento el método como un código interno del notebook de la referencia [6] se calculó el coeficiente R para cada parámetro. En la tabla I se muestra el resultado, aunque en el notebook de la referencia anteriormente mencionada también está impreso.

CUADRO I. Valores de R para a y b.

	a	b
R	0.99947	0.99952

Como puede observarse en la tabla I, los valores de R para a y b son muy cercanos a 1, lo que significa que nuestros caminadores convergen y llegan a la misma zona de convergencia. Este hecho se muestra también al graficar los valores de cada parámetro en cada iteración (véase las figuras 4 y 5).

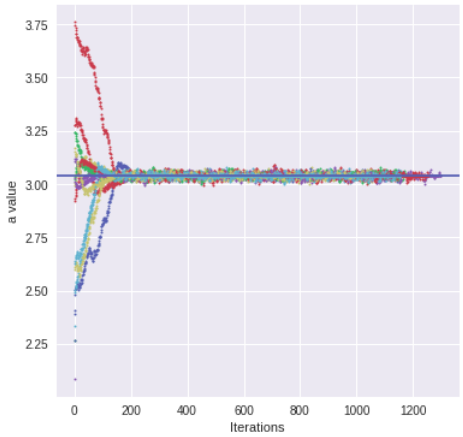

FIGURA 4. Valores de a por cada iteración de cada caminador.

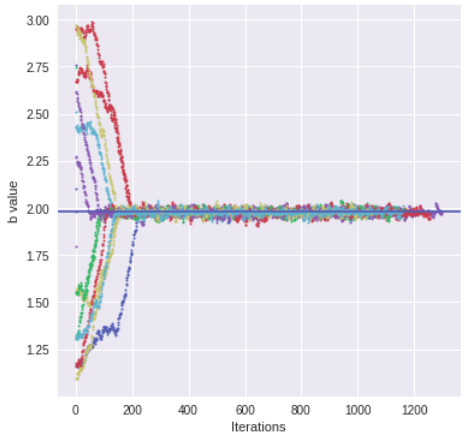

FIGURA 5. Valores de b por cada iteración de cada caminador.

Al final de este método se hace el cálculo de la mediana de la distribución para cada parámetro. El resultado fue de $a = 3.040$ y $b = 1.977$, los cuales son muy cercanos a los valores reales. En la figura 7 se muestra cómo se ve el ajuste con esta distribución de datos.

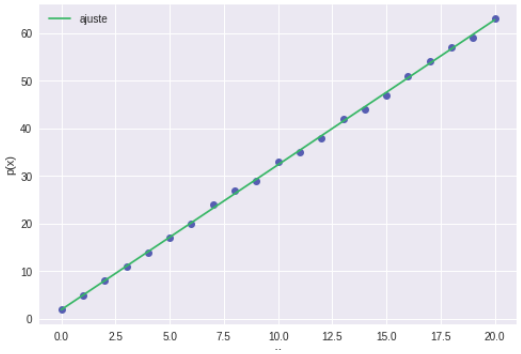

FIGURA 6. Ajuste lineal a una distribución de datos.

En el caso de EMCEE, el código implementado y las gráficas generadas en corner también se muestran en el notebook de la referencia [6]. Las gráficas genradas de los valores de a y b en cada iteración para cada caminador se muestran a continuación.

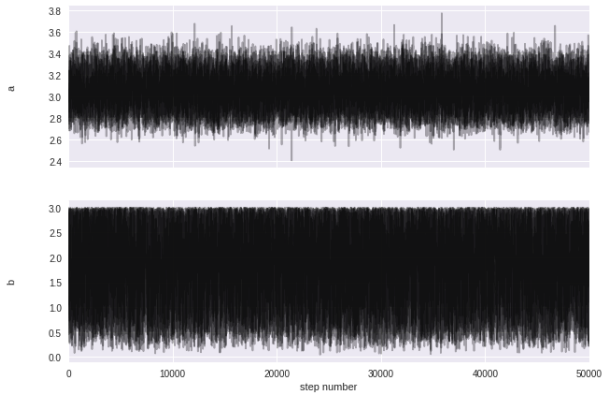

FIGURA 7. Valores de a y b en cada iteración.

Aunque esta última figura se ve un poco ruidosa, se puede observar el rango de valores en los que oscilan los parámetros a y b, así como una densidad de color, es decir, se ve más oscuro en la zona central de este rango. Para tener una mejor idea sobre los valores de los parámetros y sus distribuciones, puede utilizarse la librería *corner* para observar estas gráficas en una sola figura. En la figura 8 se muestra la gráfica generada por corner.

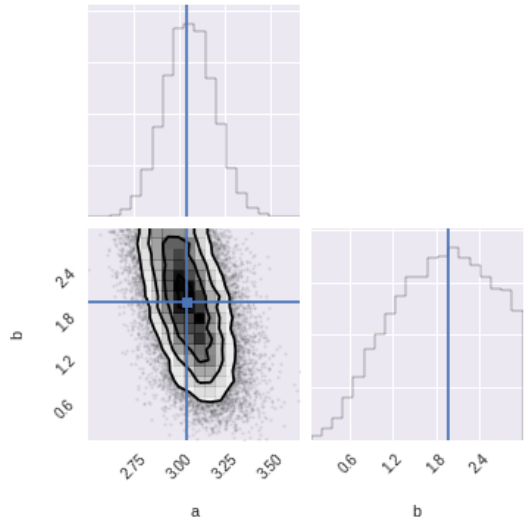

FIGURA 8. Gráfica de distribuciones para cada parámetro generada en corner.

Como puede notarse en la figura 8, las distribuciones tienen una forma similar a una distribución de Poisson, tal y como sucedió con las gráficas generadas en el método de metrópolis. La figura 8 muestra una gráfica de los dos parámetros en conjunto, la cual es análoga a la zona de convergencia encontrada en el método de metrópolis.

Utilizando EMCEE, los valores para cada parámetro son los siguientes: $a = 3.054$ y $b = 1.913$, que son cercanos a los obtenidos en metrópolis, por lo que el ajuste es muy similar gráficamente.

Finalmente, en el empleo de Multinest, aunque no se generó ninguna gráfica, sí se obtuvieron los valores estimados para cada parámetro con una pequeña desviación. En la figura 9 se muestra una captura de pantalla del resultado.

```
Nested Sampling ln(Z):                -57.002445
Importance Nested Sampling ln(Z):     -56.672786 +/-  0.018441
Acceptance Rate:                        0.810243
Replacements:                           2199
Total Samples:                          2714
Nested Sampling ln(Z):                -56.957945
Importance Nested Sampling ln(Z):     -56.669075 +/-  0.017701
 ln(ev)=  -56.611510912480362      +/-   7.9171991790475718E-002
 Total Likelihood Evaluations:         2714
 Sampling finished. Exiting MultiNest
  analysing data from chains/3-.txt


parameter values:
             a : 3.003 +- 0.153
             b : 2.348 +- 1.003
  analysing data from chains/3-.txt
```

FIGURA 9. Valores estimados para cada parámetro utilizando Multinest.

Los resultados al correr este script fueron guardados en una carpeta llamada *chains*, a la que puede ingresarse utilizando la referencia [7].

Para terminar, en la siguiente tabla se muestra una comparación entre los errores porcentuales de cada método empleado de acuerdo a la siguiente expresión

$$\%Error = abs\left(\frac{Valor_{real} - Valor_{estimado}}{Valor_{real}}\right). \quad (6)$$

CUADRO II. Error porcentual en el valor de cada parámetro obtenido por los métodos de Metrópolis, EMCEE y Multinest.

	Valor real	%Error		
		Metrópolis	EMCEE	Multinest
a	3	1.333	1.800	0.1
b	2	0.150	4.350	17.4

En la tabla II se puede observar que para este conjunto de datos, el mejor ajuste está dado por el método de metrópolis.

6. Conclusiones

Se logró implementar el método de metrópolis utilizando un likelihood poissoniano para ajustar una distribución de datos a un modelo lineal. También se logró implementar este likelihood con los módulos de Python EMCEE y Multinest.

Al comparar estos tres métodos, pude notar que Multinest se acercó más al valor real del parámetro a, pero fue el más alejado en el parámetro b, sin embargo, considero que el mejor método empleado fue el de metrópolis, debido a que es el método que más se acerca a los valores reales de los dos parámetros. No obstante, con EMCEE también se encontraron valores muy cercanos para a y b, por lo que en otros modelos puede considerarse como el mejor de los tres métodos utilizados.

Se espera continuar con este proyecto y, una vez terminado este primer proceso de implementación de un likelihood poissoniano, proseguir con el estudio de la distribución de energías generadas por la emisión de rayos gamma de la Omega Centauri.

1. STATISTICS, DATA MINING AND MACHINE LEARNING IN ASTRONOMY. *Andrew J. Connolly*. A practical python guide for the Analisis of survey Data. Princeton series in modern observational astronomy.
2. EMCE. https://emcee.readthedocs.io/en/stable/user/install/
3. MULTINEST. https://github.com/JohannesBuchner/PyMultiNest
4. MÉTODO DE GELMAN-RUBIN. https://arxiv.org/pdf/1812.09384.pdf
5. ON THE ORIGIN OF THE GAMMA-RAY EMISSION FROM OMEGA CENTAURI: MILISECOND PULSARS AND DARK MATTER ANNIHILATION. Javier Reynoso-Cordova, Oleg Burgueño, Alex Geringer-Sameth, Alma X. González-Morales, Stefano Profumo and O. Valenzuela. arXiv:1907.06682v1
6. GITHUB. *Jupyter notebook: Proyecto final*. https://github.com/DCIDA2019/da2019-anaisabelotodo/blob/master/Proyecto_final/Proyecto_final.ipynb
7. GITHUB. *Jupyter notebook: Multinest*. https://github.com/DCIDA2019/da2019-anaisabelotodo/tree/master/Proyecto_final/Multinest