Métodos Numéricos para la Ciencia e Ingeniería: Tarea 01: Métodos de integración

Luis Mora Lepin
24 de septiembre de 2015

1. Introducción

En el presente informe se detallan los procedimientos a seguir durante el desarrollo de la tarea número uno del curso Métodos Numéricos para la Ciencia y la Ingeniería. El objetivo principal de esta actividad consiste en lograr comprender y aplicar métodos de integración numérica mediante la resolución de dos problemas relacionados con la astronomía. Además de esto, se desea mejorar habilidades en el lenguaje de programación python aplicando distintas librerías como numpy, astropy y matplotlib.

1.1.

La primera actividad consiste en leer un archivo que posee datos sobre el espectro del solar medidos fuera de la atmósfera, se espera poder gráficar estos datos usando Python.

1.2.

Se pide aplicar algunos de los métodos de integración numérica vistos en clases para integrar el espectro del punto previo. A pesar de esto se espera que se genere un algoritmo propio en desmedro de las librerías predefinidas de python. Además se pretende obtener la luminosidad solar a partir del dato anterior.

1.3.

En este item se debe integrar la función de Planck que corresponde a:

$$B_\lambda(T) = \frac{2\pi hc^2/\lambda^5}{e^{hc/\lambda k_B T} - 1}$$

Generando un algoritmo propio al igual que en el apartado 1.2 para luego comparar con este resultado y a partir de esto estimar el radio solar.

1.4.

Finalmente se pide comparar los algoritmos creados con las rutinas predifinidas de Python ya sea integrate.trapz o integrate.quad.

2. Procedimiento

2.1. Gráfico del espectro solar

En esta primera actividad, se trabajó con el archivo AM0.dat obtenido directamente desde la web señalada[1] el cual contiene columnas con los datos de Potencia y longitud de onda con unidades de $\frac{W}{m^2 nm}$ y nm respectivamente. Se construyó un código en Python con una rutina capaz de leer este archivo para proceder a graficar, de este modo el código se construyo con la siguiente estructura[2]:

- Se utilizó la instrucción de la librería numpy np.loadtxt y se registró cada columna del archivo en dos arreglos distintos.

- Con el objetivo de respetar la convención astronómica, se efectuó un cambio de unidades de los datos desde S.I a CGS mediante el uso de factores de conversión apropiados.

- Finalmente, se procedió a utilizar las siguientes rutinas de matplotlib.pyplot con el objetivo de obtener un gráfico: plt.figure(),plt.plot(),plt.xlim(0,20000) para que el gráfico sea visible además de comandos conocidos para etiquetar los distintos ejes y agregar un título.

2.2. Integración del espectro y luminosidad solar

Para comenzar se integró el espectro graficado en el punto anterior, esto se llevó a cabo mediante la implementación de una regla trapezoidal en Python, dicho código se encuentra adjuntando en el anexo. Básicamente la idea es aproximar cualquier distancia entre las imágenes por una línea recta junto con la distancia entre las preimágenes del modo que sigue:

$$\int_a^b f(x)dx = (b-a)(f(b)+f(a))/2$$

De este modo se procedió a iterar este proceso para cada par de tuplas (xi, yi) en que xi es el valor de la longitud de onda e yi es el valor de la imagen asociada a dicha longitud.
Conociendo la suma del resultado anterior, se utilizó este valor denotado por k para operar con la relación de luminosidad:

$$L = 4\pi a_0^2 k \text{ con } a_0 \text{ distancia tierra-sol medida en metros.}$$

[1] https://github.com/uchileFI3104B-2015B/01Tarea
[2] Ver códigos adjuntos en archivo

2.3. Función de Planck, flujo de energía y radio solar

Considerando la función de Planck citada en la introducción, se puede ver que con un cambio de variables apropiado se llega a la expresión:

$$P = \frac{2\pi h}{c^2}\left(\frac{k_B T}{h}\right)^4 \int_0^\infty \frac{x^3}{e^x - 1}$$

La cual es a priori más sencilla de integrar, no obstante, al intentar implementar el algoritmo del trapecio en Python no se puede evaluar un límite tendiendo a infinito, para solucionar esta situación se introdujo el cambio de variables $x = tan(y)$ a modo de obtener un nuevo recorrido acotado. De este modo, se procedió a integrar la siguiente función en el dominio acotado $[0, \frac{\pi}{2}]$

$$\int_0^{\frac{\pi}{2}} \frac{tan(x)^3(1 + tan(x)^2)}{e^{tan(x)} - 1}$$

El algoritmo de integración nuevamente fue la regla del trapecio, se usaron $n = 10000$ intervalos equiespaciados entre los extremos 0 y $\frac{\pi}{2}$, se consideró una tolerancia del una diferencia de 0.0001 con el resultado analítico de la integral original que es $\frac{\pi^4}{15}$. Además como los valores extremos de este intervalo son críticos para la nueva función, se comenzó con un valor inicial para el algoritmo de $a = 0{,}0001$.

Por último con el resultado obtenido en la actividad previa se trabajó con la siguiente relación para lograr obtener el radio solar a partir de la fórmula para luminosidad:

$$R = \sqrt{\frac{a_0^2 K}{F}}$$

En que a_0 es la distancia tierra - sol, k es la integral calculada en el apartado anterior y F es el último valor integral obtenido en esta misma sección.

2.4. Comparación con métodos predefinidos

En este último apartado se comparó los resultados implementados por el usuario con respecto a los resultados obtenidos mediante rutinas importadas de la librería scipy de Python. En primer lugar se compararon los resultados numéricos obtenidos para las dos integrales calculadas anteriormente con la ayuda de la función numpy.fabs() para obtener diferencias en módulo. Para el caso de la primera integral, como se consideraba que la imágenes estaban almacenadas en un arreglo se utilizó la función scipy.trapz() para obtener un valor numérico con el cual comparar. Este último procedimiento consiste en una versión predefinida del algoritmo del trapecio. Para la segunda integral se comparó con el resultado generado por la función scipy.quad() la cual solicita como argumentos la función a integrar (previamente definida con el comando 'lambda') además de los límites de integración.

Finalmente con la ayuda de la instrucción magic *%timeit* de Ipython, se procedió a comparar los tiempos de ejecución de cada algoritmo diseñado por el usuario en contraste con las rutinas predefinidas de Scipy.

3. Resultados

3.1. Gráfico del espectro solar

Al utilizar las rutinas de la libreria matplotlib.pyplot en el código Python se obtuvo el siguiente gráfico respetando las convenciones astronómicas para las unidades:

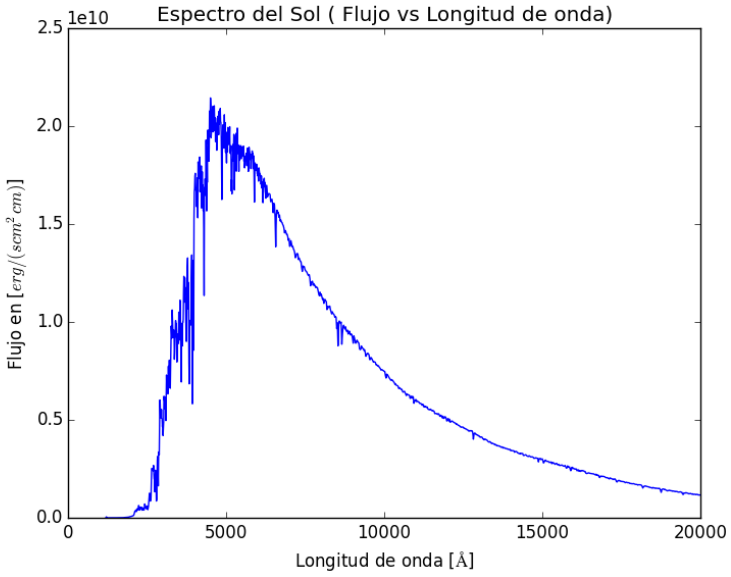

Figura 1: Espectro solar medido fuera de la atmósfera terrestre.

3.2. Integración del espectro y luminosidad solar

Luego de implementar la regla del trapecio en Python como se describió anteriormente se obtuvo el siguiente valor para la integral del espectro solar:

$$k = 1366 \frac{W}{m^2}$$

Posteriormente se utilizó la fórmula $L = 4\pi a_0^2 k$ utilizando $a_0 = 1{,}5x10^{11}m$ lo cual arrojó un valor de luminosidad solar:

$$L = 3{,}86x10^{23}kW$$

3.3. Función de Planck, flujo de energía y radio solar

Se procedió de manera similar al punto anterior, salvo que en esta ocasión se utilizó un paso constante en el algoritmo del trapecio, pero esta vez con un paso constante $h = \frac{\pi}{20000}$ al correr el programa se obtuvo un valor para la integral normalizada de:

$$\int_0^{\frac{\pi}{2}} \frac{tan(x)^3(1+tan(x)^2)}{e^{tan(x)}-1} = 6{,}493$$

Con este valor conocido se obtuvo la integral de la función de Planck la cual arrojó una magnitud de:

$$P = 6{,}319x10^7 \frac{W}{m^2}$$

Por último al aplicar la fórmula que se dedujo para el radio solar se estimó el siguiente valor:

$$R_\odot = 6{,}97x10^8 m$$

3.4. Comparación con métodos predefinidos

Se efectuaron comparaciones para ambas integrales arrojando los siguientes resultados:

i) Para la integral del espectro solar se obtuvo una diferencia en módulo de $\Delta k = 1{,}8x10^{-12}$. El tiempo de ejecución promedio para el método implementado por el usuario fue de $t = 22{,}2ms$ y para la función predefinida $t = 17{,}4ms$, ambos casos fueron procesados en 10 loops.

ii) En el caso de la función de Planck la diferencia en módulo fue de: $\Delta F = 0{,}0057$. Los tiempos de ejecución promedio fueron $t = 118ms$ para el algoritmo implementado por el usuario y $t = 382\mu s$ para la rutina predefinida scipy.quad(). Además, cabe destacar que se ejecutó solo un loop en el caso del algoritmo creado y 1000 loops para la función predefinida.

4. Conclusiones

De los procedimientos efectuados anteriormente se logró extraer las siguientes conclusiones:

- Existen diferentes técnicas de integración numérica las cuales son útiles para conocer valores de integrales que pueden poseer o no resultado analítico, en particular se trabajó con la regla del trapecio la cual arrojó resultados bastante eficientes al tratar datos listados como arreglos (Como en el caso de la actividad 2) o al operar funciones que sean sencillas, esto quedó particularmente reflejado en los resultados de la sección 3.4 - (i). Por otra parte, si nos fijamos en los resultados expuestos en 3.4 - (ii) nos damos cuenta de que el algoritmo del trapecio es deficiente en comparación al método scipy.quad() pues utiliza mucho tiempo de ejecución y el resultado no se asemeja tanto al de quad; Frente a esto, se sugiere para futuras aplicaciones intentar mejorar el algoritmo, una opción es intentar refinar el intervalo de integración obteniendo unos extremos que sean más sencillos de trabajar mediante la regla del punto medio. Por otra parte, siempre está la opción de implementar algún otro algoritmo de integración conocido como la Regla de Simpson.

- Se expuso la utilidad de las librerías a disposición en Python, tal como quedó reflejado en la actividad 1 cuando se graficó con la ayuda de matplotlib o al utilizar distintas operaciones matemáticas que fueron resueltas con numpy o scipy.