Tarea 1 para Métodos Numéricos para la Ciencia e Ingeniería

Jaime Castillo Lara

24 de Septiembre de 2015

El objetivo de la tarea es calcular numéricamente la luminosidad total del sol producida por el flujo de radiación electromagnética del sol (en unidades de energía por unidad de tiempo), y compararlo con la energía total por unidad de área del cuerpo negro teórico que mejor se ajusta al sol.

Parte 1: Graficar longitud de onda versus flujo

Teniendo una tabla de datos para el flujo del sol en función de las longitudes de onda medidas, el primer paso es hacer un gráfico a partir de los datos.

Para esto, creamos dos arreglos “a” y “b”, los cuales contendrán los datos de las longitudes de onda y del flujo, respectivamente.

Para que los arreglos se llenen con los datos de las tablas correspondientes del documento sun_Am0.dat (el cual contiene los valores para las longitudes de onda y flujos, en columnas separadas por espacios en blanco), escribimos:

```
a, b = np.loadtxt("sun_AM0.dat.dat", unpack="True")
```

Con esto, el arreglo “a” pasa a contener los 1700 valores de las longitudes de onda, y el arreglo “b” pasa a tener los 1700 valores de flujo del sol.

Sin embargo, debemos utilizar la convención astronómica cgs para las unidades de medida. Vemos que en el arreglo “a“ contiene las longitudes de onda en nanómetros, y para cumplir la convención astronómica debemos manejar los datos en micrones.

Como sabemos que $1\ nanometro\ =\ 0{,}001\ micrones$, escribimos:

```
a=a*0.001
```

con lo que el arreglo pasa a tener los datos de longitud de onda en micrones, con lo que cumplimos la convención astronómica. Ahora debemos hacer lo mismo con los datos de flujo almacenados en el arreglo b.

Vemos que los datos de flujo en el arreglo b están en unidades:

$\left[\frac{Watt}{metro^2\ nanometro}\right] = \left[\frac{Joule}{segundo\ metro^2 nanometro}\right]$, y para cumplir la convención astronómica los necesitamos en unidades: $\left[\frac{ergs}{centimetro^2 micron}\right]$, para hacer la conversión necesaria, vemos que:

$1\ nanometro\ =\ 0{,}001\ micrones$
$1\ erg\ =\ 10^{-7}\ Jouls$
$1\ metro\ cuadrado\ =\ 10^{4} centimetros\ cuadrados$

Por lo que, para hacer la conversión necesaria, debemos multiplicar los datos en el arreglo “b” por 10^5, así que escribimos:

```
b = b * 100000
```

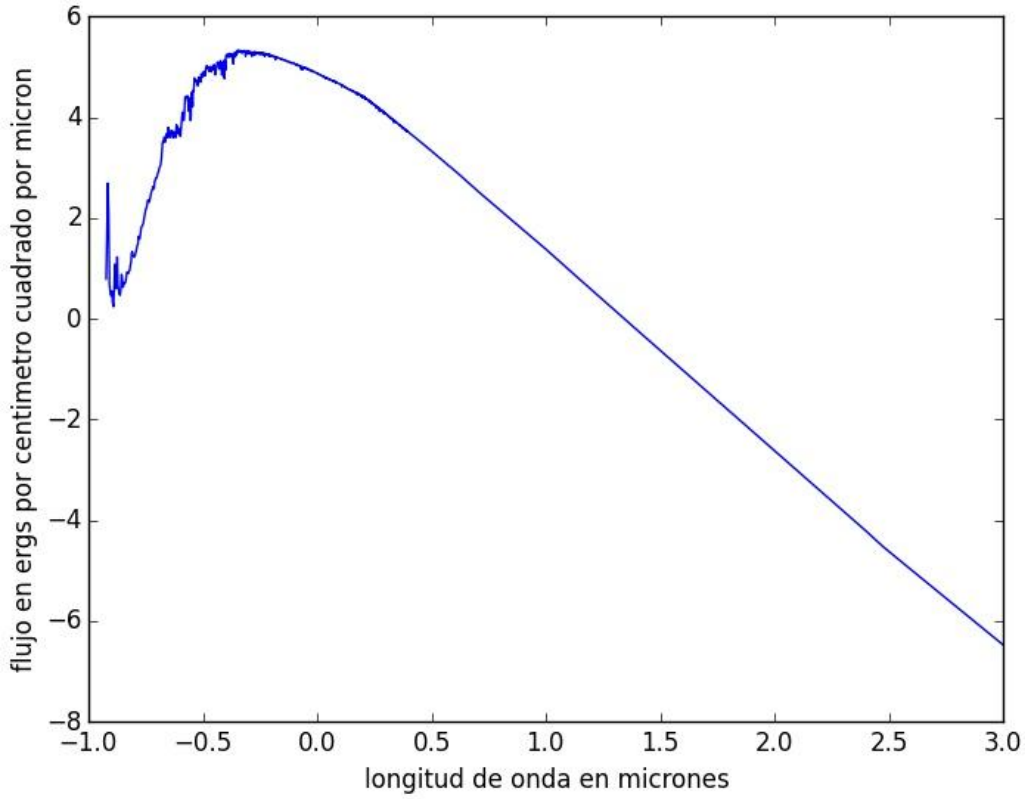
con lo que cumplimos la convención astronómica. Ahora, para graficar los arreglos “a” versus “b” en escala logarítmica, escribimos:

```
plt.figure(1)
plt.clf()
plt.plot(np.log10(a),np.log10(b))
```

y para etiquetar los ejes:

```
plt.ylabel("flujo en ergs por centimetro cuadrado por micron")
plt.xlabel("longitud de onda en micrones")
plt.legend
plt.show()
```

lo que nos entrega el grafico:



Parte 2: Calcular la luminosidad total del sol a través de la integración del espectro en función de la longitud de onda

Ya que contamos con la curva de flujo vs longitud de onda, debemos integrarla para encontrar la luminosidad total. Para calcular la integral numéricamente utilizamos el método del trapecio, el cual se describe como:

$$\int_a^b f(x)dx = \frac{\Delta x}{2}\left[f(a) + 2\sum_{i=1}^{N-2} f(x_i) + f(b)\right]$$

Para hacer este método de integración numérica de forma iterativa, a través de todos los valores de los arreglos “a” y “b”, escribimos:

```
int = 0.

for i in range(1696):

   int = int + ((a[i+1]-a[i])/2.)*(b[i]+b[i+1])

print "primera integral"

print int
```

dado que el espaciamiento entre los valores del arreglo “a” no es constante, lo calculamos en cada iteración.

Ahora que tenemos el valor de la integral calculada numéricamente con nuestro algoritmo, utilizamos el método del trapecio incluido como función en numpy, para esto escribimos:

```
print "integral con scipy"

prueba = np.trapz(b,a)

print prueba
```

al comprobar ambos métodos, vemos que nos entregan el mismo valor, al menos hasta los decimales que Python nos entrega:

```
primera integral

136609.079684

Integral con scipy

136609.079684
```

Lo que comprueba la efectividad de nuestro algoritmo para el método del trapecio.

Parte 3: Calculo de la energía total por unidad de área de un cuerpo negro, a través de la integral de la función del Planck.

Dado que ya hemos encontrado la luminosidad a través de los datos de flujo electromagnético del sol, debemos ahora encontrar la radiación del cuerpo negro teórico, para compararla con la del sol.

Sabemos que esta radiación está dada por la función de Planck, y que la integral de la misma está dada por:

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

Para calcular esta integral impropia, debemos hacer un cambio de variable con el que podamos trabajar con límites de integración finitos, por lo que hacemos:

$$x = \tan(y)$$

Con lo que la integral queda de la forma:

$$\int_0^\pi \frac{\tan(y)^3}{e^{\tan(y)} - 1}\sec(y)^2 dy$$

Para hacer esta integral, definimos un arreglo de 2001 valores equiespaciados de 0 a pi, escribimos:

```
c, d=np.linspace(0.,np.pi,num=2001, endpoint=True,retstep=True)

print c

print d
```

con lo que tenemos dos arreglos definidos de la forma que buscábamos, y sobre estos calcularemos la integral de la función de Planck. Para esto, definimos la función que está dentro de la integral, por lo que escribimos:

```
def evaluador(x):
    valor= ((np.tan(x)*np.tan(x)*np.tan(x))/(np.cos(x)*np.cos(x)*(np.exp(np.tan(x))-1)))
    return valor
```

para ir llenando los valores de la integral que se irán calculando, definimos un arreglo “e” inicializado con ceros, y de las mismas dimensiones de los arreglos “c” y “d”. Por lo tanto, escribimos:

```
e=np.zeros(2001)
```

ya que tenemos todos los arreglos necesarios para hacer la integral, utilizamos el mismo algoritmo para el método del trapecio utilizado anteriormente. Para esto escribimos:

```
int2 = 0
start_time = time.time()
for i in range(1999):
    int2 = int2 + d/2*(evaluador(c[i+1])+evaluador(c[i+2]))
    e[i+1]=evaluador(c[i+1])
print ("%s segundos" % (time.time() - start_time))
print int2
```

con lo que integramos la función de Planck sobre los valores del arreglo “c”, utilizando la especiación del arreglo “d”, y guardando los valores en el arreglo “e”.

Además, utilizamos la función de Python time.time() que nos entregará el tiempo necesitado para la ejecución del algoritmo.

Ya que tenemos el valor de la integral calculado con nuestro algoritmo, nuevamente la calculamos con la función de numpy para el método del trapecio, con el objetivo de compararlo con nuestro algoritmo, por lo tanto escribimos:

```
print "integral2 con scipy"

start_time2 = time.time()

prueba2 = np.trapz(e,c)

print ("%s segundos" % (time.time() - start_time2))

print prueba2
```

nuevamente utilizando la función de Python time.time() para medir el tiempo de ejecución.

De igual forma a como ocurrió en la integral anterior, nuestro algoritmo y la función np.trapz() entregaron exactamente el mismo resultado, hasta donde los decimales nos pueden decir.

Sin embargo, esta vez hemos medido el tiempo de ejecución, y hemos descubierto que nuestro algoritmo tiene un tiempo de ejecución de 0,163 segundos, aproximadamente, mientras que la función np.trapz() de Python tiene un tiempo de ejecución de 0.0 segundos, es decir, menos de una décima de segundo. Con lo que prueba ser muchísimo más eficiente que el nuestro.