FACULTAD DE CIENCIAS
FÍSICAS Y MATEMÁTICAS
UNIVERSIDAD DE CHILE

Nicolás Valdés
RUT: 19.247.388-8
FI3104-1 2018B
27/09/18

Tarea 1

Problema 1

Introducción

La definición usual de la derivada $f'(x)$ de una función $f(x)$ involucra tomar un límite, que por su naturaleza es un proceso *continuo*. Claramente al calcular derivadas con un computador, esto no es factible. Una opción simple para aproximar una derivada es

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \equiv f_1'(x), \tag{1}$$

lo cual sale a partir de hacer una expansión de Taylor para f, y despejar $f'(x)$, truncando términos de orden h o mayor. La idea es tomar un h pequeño para que esta aproximación funcione. Si uno se queda con más términos en la serie de Taylor y evalúa series de la función en distintos puntos, puede llegar a una aproximación que trunca términos de orden h^4 o mayor:

$$f'(x) \approx \frac{-f(x+2h) + 8f(x+h) - 8f(x-h) + f(x-2h)}{12h} \equiv f_4'(x). \tag{2}$$

Este problema se trata de comparar estas dos aproximaciones.

Metodología

Para comparar las aproximaciones, la idea intuitiva es ver "cuán cerca" están a la derivada real de una función. Entonces hay que escoger una función y evaluar su derivada de tres maneras, en algún punto x_0. Debemos evaluar su derivada real, evaluar $f_1'(x_0)$, y evaluar $f_4'(x_0)$. Usamos $f(x) = -\cos(x)$. Sabemos, desde antes, que $f'(x_0) = \sin(x_0)$. Para ver "cuán buena" es una aproximación de $f'(x)$, calculamos $\Delta_i \equiv |f'(x_0) - f_i'(x_0)|$, donde $i \in \{1, 4\}$. Esto nos indica de alguna forma el error de un método de aproximación, y es nuestro criterio de comparación.

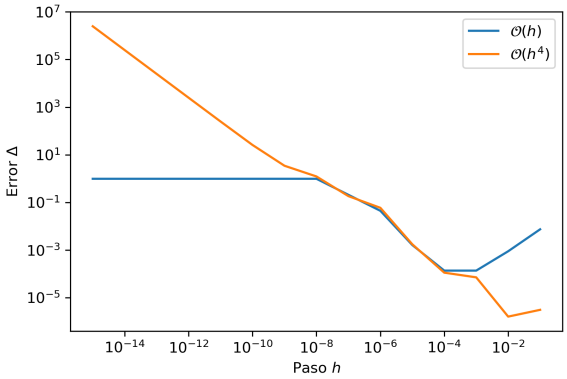
En el código,

1. Definimos una función que calcula f_1'. Tiene tres argumentos: la función a derivar, el punto donde se evalúa la derivada, y el h usado en la ecuación (1). Hacemos lo mismo con f_4'.
2. Usamos $x_0 = 1.388$ (por mi RUT) y $h = (10^{-1}, 10^{-2}, ..., 10^{-15})$. Evaluamos las funciones del paso 1 en estos valores, y en la función $\cos(x)$.
3. Calculamos el error Δ_i restando las f_i del paso anterior, a la función $\sin(x_0)$ incluida en *numpy*.
4. Graficamos Δ_i como función de h, con escala logarítmica en ambos ejes.
5. Repetimos todo lo anterior, pero cambiando x_0 y h para que sean Float 32, o Float 128.

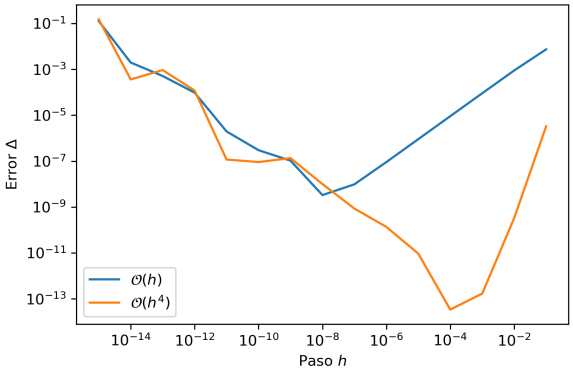
Resultados

En la Figura 1 se encuentran los gráficos producidos para el error Δ, evaluado usando distintos h.

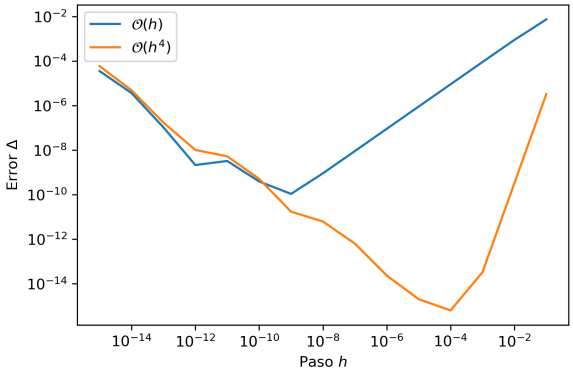
Nicolás Valdés
RUT: 19.247.388-8



(a) Float32



(b) Float64



(c) Float128

Figura 1: Error Δ en función de h

Conclusiones

El comportamiento de Δ_i no es tan simple. Uno esperaría que el error disminuye a medida que h se hace más chico, dado que disminuye el efecto de despreciar términos de mayor orden en h. El problema es que el computador no tiene infinita precisión en los números que calcula. Por ejemplo, la precisión de float64 es del orden de 10^{-15}. Para números con muchos decimales, eventualmente el computador trunca el número. Entonces con cada método si restamos algo en el numerador, el resultado no va a ser "honesto", va a ser algo con precisión truncada. Pero esto es dividido por un h muy pequeño, entonces este error chico se agranda. Esto se ve dramáticamente en la figura 1(a), donde el error del método $\mathcal{O}(h^4)$ alcanza 10^6. El error probablemente no aumenta más para el método $\mathcal{O}(h)$ en este caso porque el numerador simplemente arroja el valor 0, resultando en un error constante de orden 1.

Para h "grande" (i.e. $h \geq 10^{-7}$), la precisión del método $\mathcal{O}(h^4)$ suele ser mejor. Esto tiene sentido, dado que lo que hace este método es justamente despreciar menos potencias de h. Si h es "grande", despreciar términos con h^2 ya puede tener efectos notorios, y la precisión del método $\mathcal{O}(h)$ disminuye sustancialmente. Con las figuras 1(b) y 1(c) se ve que la máxima precisión alcanzada por el método $\mathcal{O}(h)$ es en $h \approx 10^{-8}$. Esta misma precisión se alcanza con $\mathcal{O}(h^4)$ con $h \approx 10^{-2}$. Aquí se ve heurísticamente la forma en que (para h's que el computador puede procesar) el método $\mathcal{O}(h^4)$ tiene más precisión que $\mathcal{O}(h)$. Aún si obtiene más precisión, este método es un poco más ineficiente al tener que evaluar la función en más puntos.

FACULTAD DE CIENCIAS
FÍSICAS Y MATEMÁTICAS
UNIVERSIDAD DE CHILE

Nicolás Valdés
RUT: 19.247.388-8
FI3104-1 2018B
27/09/18

Problema 2

Introducción

La radiación de fondo cósmica es nuestro dato más importante para entender el universo temprano; nos da información sobre las condiciones iniciales del universo, y da a conocer cómo y por qué se formaron estructuras a grandes escalas.

Aquí tomamos datos del satélite COBE sobre el espectro de monopolo de la radiación de fondo, y los graficamos con sus barras de error. Luego encontramos de forma teórica la potencia proveniente de la radiación, haciendo una integral númerica sobre un espectro de Planck. Comparando esto con la integral numérica de los datos, conseguimos una predicción sobre la temperatura asociada al espectro Planckiano de la radiación. De ahí comparamos el espectro teórico de temperatura conocida $T = 2.725K$ con el espectro proveniente de la temperatura calculada.

Metodología

1. En la parte 1 el programa carga la tabla de datos del archivo de COBE, y definimos como variable X a la lista con las frecuencias. Definimos como Y a la lista con el espectro de monopolo, y err a la lista con los errores. Graficamos estos datos, haciendo una conversión de unidades para las frecuencias, que estaban en cm^{-1}. Agrandamos las barras de error por un factor 400 para que sean visibles.

2. Para esta parte consideramos la integral

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

   Como esto va a hasta infinito, no se puede hacer numéricamente de inmediato. Con un cambio de variable $y = \arctan(x)$, eso sí, traemos el infinito hacia $\pi/2$. Entonces lo que integraremos numéricamente es

$$P = \frac{2h}{c^2}\left(\frac{k_B T}{h}\right)^4 \int_0^{\pi/2} \frac{\sin^3(y)}{\cos^5(y)(\exp(\tan(y)) - 1)} dy. \tag{4}$$

   Hacemos esta integral con el método del trapecio. A pesar de que el integrando es finito cuando $y \to 0$, hay una división por 0 en $y = 0$. Entonces la integración numérica comienza desde el valor $y + h$, con h el paso, pero no hay que hacer una regulación del integrando ya que es bien comportado en el intervalo $(0, \pi/2)$.

3. Para integrar en frecuencia el espectro observado utilizamos los datos definidos en la parte 1. Nuevamente aplicamos el método del trapecio. Al hacer cambios de unidades adecuados para que todo sea consistente, dividimos la integral numérica calculada en esta parte, por la integral calculada en la parte 2. Esta razón, elevada a 1/4, es la temperatura.

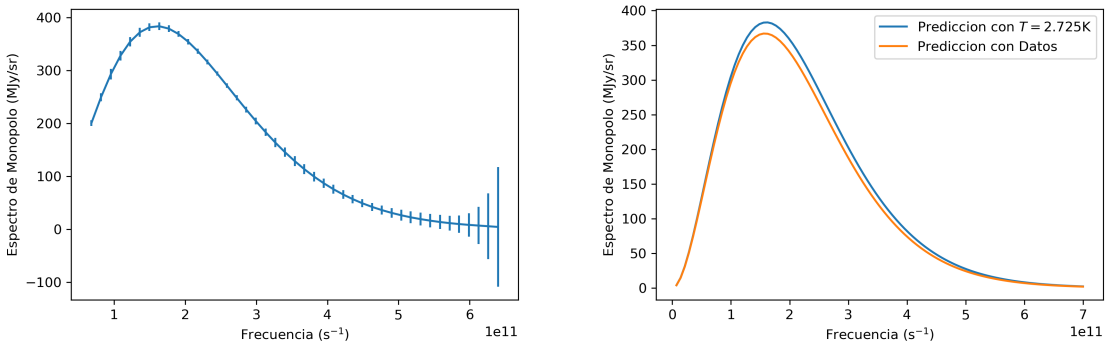
4. Se grafica el espectro Planckiano asociado a una temperatura $T = 2.725$K, y otro espectro Planckiano asociado a la temperatura calculada en esta experiencia (ver próxima sección). El espectro Planckiano es

$$B_\nu(T) = \frac{2h\nu^3/c^2}{e^{h\nu/k_B T} - 1} \tag{5}$$

5. Hola

Resultados

Temperatura del espectro en base a los datos y las integrales numéricas: 2.686 K.



(a) Espectro con Datos (Barras de Error ×400)

(b) Espectros Planckianos

Figura 2: Espectro Monopolo CMB

Conclusiones

El espectro

La temperatura que se encontró para el espectro en base a los datos fue un poco menor a la que se debería haber encontrado. Esto se puede explicar, por lo menos en parte, por lo siguiente: al integrar el espectro de Planck, uno debería

Datos no empiezan de cero.

Encontramos una diferencia entre la velocidad con la cual integra el módulo scipy.quad, versus la velocidad a la que integra el algoritmo que construimos usando el método del trapecio. El método del trapecio tomó aproximadamente 10ms (con una partición en $N = 1000$ partes), mientras que scipy.quad tomó aproximadamente 1ms. Es decir, scipy.quad es 10 veces más rápido para integrar. Vale la pena notar, eso sí, que al tomar $N = 100$ para la partición, el tiempo disminuye a 1ms, y la precisión casi no disminuye. La integral numérica sobre el conjunto de datos arrojó el mismo valor con nuestro algoritmo y con el algoritmo de scipy.trapz. El algoritmo scipy.trapz tomó aproximadamente 12μs, mientras que el nuestro tomó 41μs.