INFORME TAREA 1

Tomás Rojas C.
RUT:19.688.339-8
Github: @tomasrojasc

1. Pregunta 1

Para la primera parte de la tares, teníamos que comparar dos maneras de calcular derivadas numéricamente.

para esto, la función a evaluar fue $f(x) = \sin(x/2)$ y el punto a evaluar (dado mi RUT) fue $x = 1.339$. El primer método a evaluar fue el siguiente:

$$f'(x) = \frac{f(x+h) - f(x)}{h} \tag{1}$$

Mientras que el método propuesto es el del enunciado.

Para lograr la derivada tenemos dos funciones que nos permiten usar ambos métodos a la vez de especificar el grado de precisión deseado.

A continuación podemos ver un gráfico donde la línea verde punteada es el valor que nos da la derivada real de la función importada desde el paquete math de python.

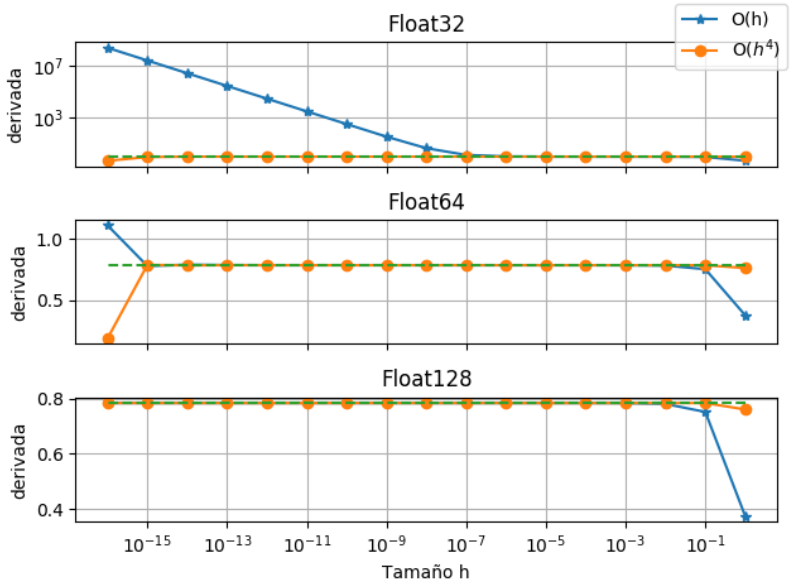

Figura 1: Comparación de los dos métodos con distintos grados de precisión.

Es importante notar que el primer gráfico, a diferencia de los otros dos, tiene escala logarítmica y no lineal. Notamos que el método depende en gran medida del nivel precisión que usemos, por ejemplo, para float32, la precisión es tan baja, que al truncar para valores de h pequeños, obtenemos divisiones por cero, lo que hace que nuestra aproximación esté muy lejos del valor real, pero vemos que para h del órden de 10^{-5} se comporta mejor, pero el rango donde esta aproximación es fiable

es muy pequeño. Por otro lado, vemos que con la aproximación propuesta en el enunciado esta convergencia es mucho más rápida al valor de la línea punteada. También vemos que el intervalo de h para el cual la aproximación de (1) es fiable, es siempre menor que el de la aproximación del enunciado, esto es un punto a considerar ya que nos dice cuánto cambio de resolución acepta nuestro algoritmo, lo cual es muy útil ya que si tenemos una función errática y una que se comporta mejor en el sentido que no tiene cambios de concavidad brusco y tiene derivadas pequeñas, podemos usar la misma función con distintos h y esperar un buen resultado sin tener que hacer otra implementación.

En síntesis, si usamos un número de mayor presición, logramos que el rango donde nuestra aproximación es útil, en términos de h sea mucho más amplio.

2. Pregunta 2:

Lo primero es regularizar la integral. Para ello usamos el siguiente cambio de variable $\sin\theta = \frac{\sin(\phi/2)}{\sin(\phi_/2)}$ y $k = \sin(\phi_0/2)$, con lo que el problema se transforma en:

$$\frac{T}{T_0} = \frac{2}{\pi}\int_0^{\frac{\pi}{2}} \frac{d\theta}{\sqrt{(}1 - k^2\sin^2(\theta))} \tag{2}$$

$$0.05 = \int_0^{1/a} \frac{1}{\sqrt{2\pi}u^2}\exp\left(\frac{-1}{2u^2}\right)du \tag{3}$$

La forma de la función a integrar se puede apreciar en la Figura 1. Llamaremos a esta función $f(u)$.

../solucion/integral.pdf

Figura 2: La función que vamos a integrar.

Para integrar $f(u)$ utilizaremos la función `scipy.interpolate.quad`, la cual es una implementación de la librería `QUADPACK` escrita en `FORTRAN`. Esta librería implementa el método de *cuadratura*

adaptativa e intenta automáticamente escoger el algoritmo necesario para calcular la integral con una precisión dada por el usuario. Para más información: https://en.wikipedia.org/wiki/QUADPAC

Definimos entonces la función:

$$I(a) = \int_0^{1/a} \frac{1}{\sqrt{2\pi}u^2} \exp\left(\frac{-1}{2u^2}\right) du \tag{4}$$

cuya forma se muestra en la Figura (2). $I(a)$ se calcula numéricamente con la función `quad` con sus parámetros por defecto. Los más importantes son las tolerancias absoluta y relativa: *epsabs=1.49e-08, epsrel=1.49e-08*, y el límite máximo de sub-intervalos para el algoritmo adaptativo: *limit=50*.

Es importante notar que la función $f(u)$ se indefine en $u = 0$. Sin embargo, como se aprecia en la Figura (1), el área bajo la curva en las cercanías de cero pequeña por, lo que podríamos partir la integral en algún valor $\epsilon \gtrsim 0$. El algoritmo de `quad` es capaz de manejar esta indefinición automáticamente, por lo que no implementamos ningún cambio relacionado a la indefinición de la función $f(u)$.

Figura 3: La función $I(a)$. Buscamos el valor de a para el cual la función toma el valor 0.05 (linea azul horizontal punteada). Utilizando el algoritmo de newton, encontramos la solución $a^* = 1.64$ (línea roja vertical punteada).

Finalmente, es necesario resolver la ecuación:

$$I(a^*) - 0.05 = 0 \tag{5}$$

Esto lo hacemos utilizando el método de newton (en realidad de la secante), implementado en `scipy.optimize.newton`. Utilizamos los parámetros por defecto, en particular, la tolerancia absoluta *tol=1.48e-08* y el número máximo de iteraciones *maxiter=50*. Le damos como punto de

partida el valor $a_0 = 1$. El algoritmo converge luego de 7 iteraciones encontrando el valor $a^* = 1.64485363$.

Otros valores del punto de partida convergen a un valor muy cercano con diferencias en el número de iteraciones necesarias. Por ejemplo para $a_0 = 0.5$, el algoritmo converge luego de 8 iteraciones.

3. Discusión y Conclusiones

Si x es una variable aleatoria sacada de una gausiana con parámetros $\mu = 0$ y $\sigma = 1$, entonces sólo tomará valores mayores a $a^* = 1.64$ el 5 % de las veces.

Si bien este problema involucra el cálculo de una integral de límites indefinidos (Ecuación 1), el cambio de variable $u = 1/y$ resulta en una función bien comportada. En particular, la suavidad de la función (ver Figura 2), asegura que algoritmos como el de la *secante* para buscar raíces converjan de manera robusta (casi independiente del punto de partida elegido), y en pocas [sic.] iterasiones.