INFORME TAREA 1

Benjamín Briceño
RUT:20.653.153-3
Github: @Benbriel

1. Introducción

Esta tarea tiene como objetivo encontrar valores numéricos para el período de un péndulo de masa m y largo L arbitrarios, soltado desde un ángulo inicial ϕ_0. Para el caso de grandes amplitudes, es posible determinar la siguiente ecuación, donde T es el período para ϕ_0 arbitrario, y T_0 es el período aproximado para oscilaciones pequeñas:

$$\frac{T}{T_0} = \frac{2}{\pi}\int_0^{\phi_0} \frac{d\phi}{\sqrt{2(cos\phi - cos\phi_0)}} \tag{1}$$

Se implementarán diversos métodos numéricos de integración para obtener el cociente entre el período real y el de pequeñas aproximaciones, para valores de ϕ_0 entre 0 y $\pi/2$. Para esto, se asegurará un error numérico no mayor al 1 %.

Por otro lado, se usarán algunos métodos de búsqueda de raíces para encontrar el valor de ϕ_0 para el cual T es 10 % mayor que T_0. Estos métodos son tanto de implementación propia como provenientes de módulos de Python.

2. Desarrollo

2.1. Integración

Primero, se revisará (1). Esta ecuación contiene una integral cuya función a integrar diverge para $\phi = \phi_0$, lo cual limita la precisión de un cálculo numérico. Al realizar un cambio de variable adecuado, es posible representar la ecuación (1) de la forma

$$\frac{T}{T_0} = \frac{2}{\pi}\int_0^{\frac{\pi}{2}} \frac{d\phi}{\sqrt{1 - \sin^2\left(\frac{\phi_0}{2}\right)\sin^2(\phi)}} \tag{2}$$

Esta integral tiene límites que no dependen de ϕ_0, y su integrando no diverge $\forall \phi, \phi_0 \in (0, \pi/2)$. Como se muestra en la Figura 1, para el caso de phi_0 pequeños, la integral tiende a un rectángulo de alto 1 y ancho $\pi/2$, por lo que $T/T_0 \to 1$, lo que es consistente con la idea de que $T \to T_0$ para pequeñas oscilaciones.

Para integrar esta función, llámese $f(\phi, \phi_0)$, y su integral con respecto a ϕ, $I(\phi_0)$, se implementará un método de integración por trapecios `calcula_integral(func,a,b,dx)`, que toma la función $f(\phi, \phi_0)$ con un ϕ_0 definido, límites de integración $a = 0, b = \pi/2$, y calcula la suma de trapecios rectángulo de ancho $dx/2$, definido por el usuario. Además, este método calcula el error absoluto al doblar la resolución de dx a $dx/2$, verificando que cada valor tenga un error menor al 1 %. Por defecto se define un $dx = h =$ `1e-4`.

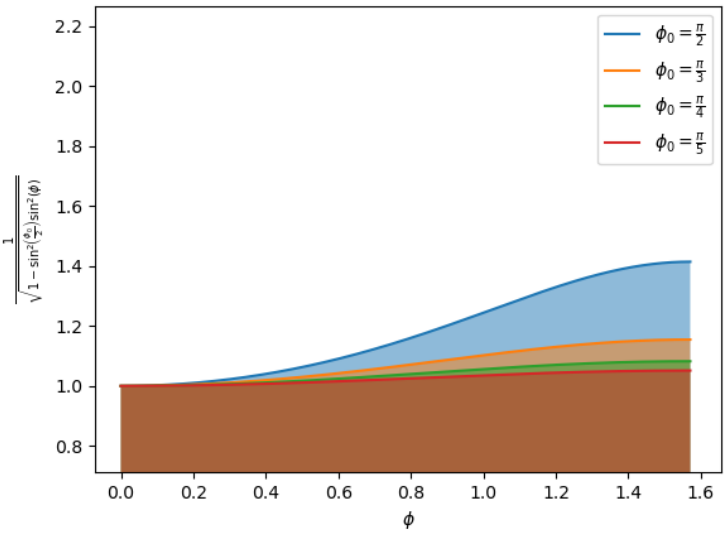

Figura 1: La función $f(\phi, \phi_0)$ a integrar, para distintos valores de ϕ_0. El área marcada bajo las curvas representa el valor de la integral.

2.2. Búsqueda de raíces

A continuación se quiere encontrar el ángulo inicial ϕ_0 para el que T sea un 10 % mayor que T_0. Para esto, se implementó un algoritmo de búsqueda de raíces basado en el método de la secante, el cual a partir de dos valores iniciales y una función, busca una raíz para la función dentro del intervalo dado. Además, el método posee una tolerancia absoluta `tol=h` por defecto, para la cual la búsqueda de raíces se detiene si $|f(x_n)| <$ `tol`, o si se llega a un número máximo `maxiter` de iteraciones, y entrega el valor x_n. En este caso se utilizaron 0.5 y 1.5 como valores iniciales, y `maxiter=50`

Por otro lado, se aplicó la función `scipy.optimize.newton`, que busca la raíz de una función a partir de un x_0, usando el método de Newton-Raphson si se le entrega la derivada de la función, o el método de Halley si se le entrega la segunda. En caso contrario, utiliza el método de la secante. En este caso no se le entregó ninguna derivada. Además, `scipy.optimize.newton` utiliza una tolerancia de `1.48e-8` por defecto, aunque en este caso, por consistencia, se define `tol=h`. Además, se define por defecto el número máximo de iteraciones `maxiter=50`.

En la Figura 2 se puede observar lo resultado de aplicar ambos métodos. Si bien ambos obtienen un resultado similar, es de esperar, pues para ambos se utilizó la misma tolerancia `h=1e-4`. Lo interesante, sin embargo, ocurre en los tiempos utilizados por cada método. Para esto, se ocupó la función `%timeit` de la terminal interactiva de Python, `ipython`. El método de la secante implementado demora un aproximado de `159 ms ± 3.18 ms` por loop, calculado de 7 loops. Mientras tanto, el método de Newton demora `53.1 ms ± 1.53 ms` por loop, con un total de 7 loops. Claramente la implementación de búsqueda de raíces de `scipy` es más rápida, sin perder precisión.

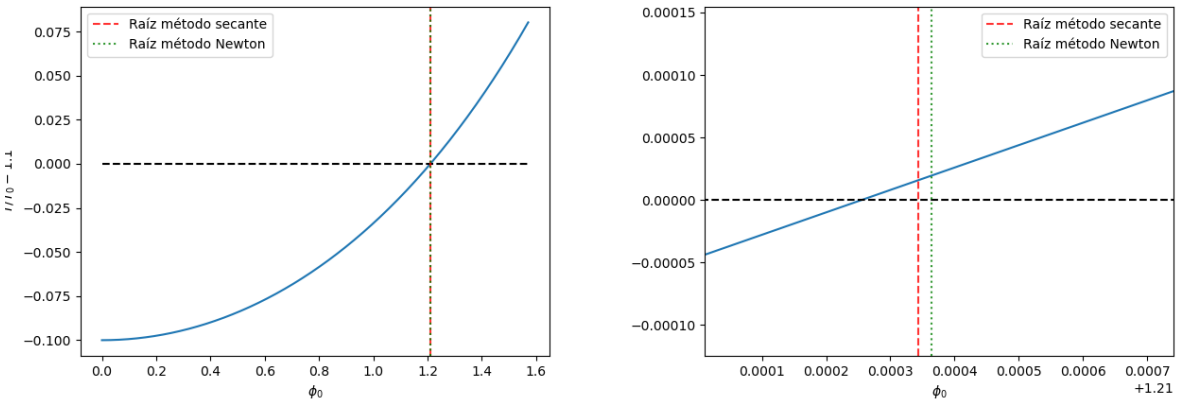

Figura 2: La función $\frac{2}{\pi}I(\phi_0) - 1.1$. Se busca el valor para el que la función se anule, implicando $T = 1.1 \cdot T_0$. Utilizando ambos algoritmos, se encuentra $\phi_0 = 1.21034$ para el método de la secante, y $\phi_0 = 1.21036$ para el método de Newton.

Método Int.	**Tiempo de ejecución**
Trapecios	461 ms ± 73.1 ms
quad	9.59 ms ± 1.34 ms
trapz	14.1 ms ± 572 µs

Cuadro 1: Tiempos de ejecución $\frac{2}{\pi}I(\phi_0) - 1.1$ para distintos métodos de integración.

2.3. Otros métodos

Por último, se utilizaron las funciones `scipy.integrate.quad` y `scipy.integrate.trapz`, y se compararon con el método de trapecios implementado.

La función `quad`, la cual es una implementación de la librería `QUADPACK` escrita en `FORTRAN`, utiliza la cuadratura adaptativa y decide el mejor algortimo para calcular la integral. Ésta recibe f como la función a integrar, y un intervalo $(a, b) = (0, \pi/2)$, además de una tolerancia absoluta `epsabs=h`, escogida por consistencia para ser comparada con otros métodos.

La función `trapz` utiliza el método de trapecios para calcular la integral de una función dado un array de datos, correspondiente al eje x. Al igual que el método implementado, utiliza un `dx` de espaciado entre los trapezoides, que por efectos de consistencia es igual a h.

Como indica la Figura 3, los métodos se asemejan, lo que hace sentido, pues la tolerancia `h=1e-4` escogida para los métodos indicaría un error bastante menor al 1 %.

Finalmente, referente a la tabla 1, es evidente que los métodos más rápidos fueron `quad` y `trapz`, mientras que el método implementado fue ineficiente, sin ganar precisión en el cálculo numérico. Esto se puede deber a que las funciones de `scipy` están bien implementadas, por lo que ahorran en cálculos, probablemente a coste de un código más complicado y extenso.

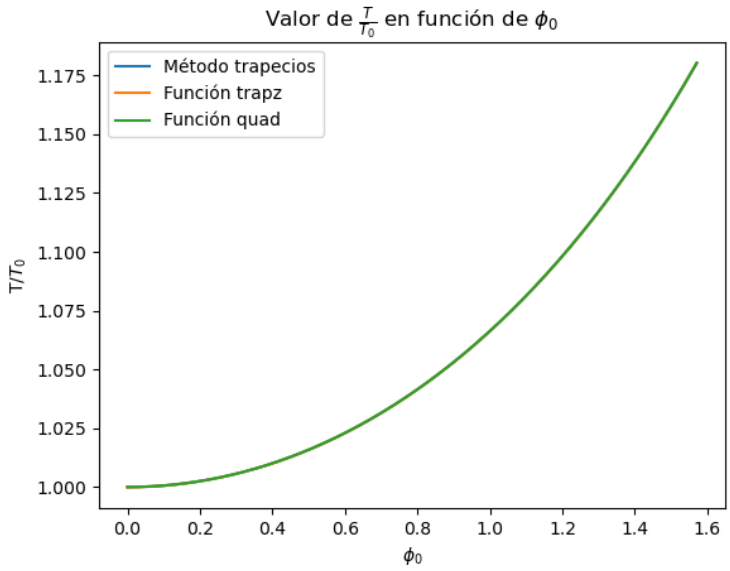

Figura 3: La función $\frac{2}{\pi}I(\phi_0)$, calculada numéricamente por los distintos metodos. Su parecido es evidente, ya que se solapan. Sin embargo, un acercamiento indica que `quad` es mayor por una cantidad constante a los otros métodos.

3. Discusión y Conclusiones

Si ϕ_0 es el ángulo inicial para un péndulo de masa y largo arbitrarios, entonces el período de grandes oscilaciones será 10 % mayor al de pequeñas aproximadamente cuando $\phi_0 = 1.21$, esto de forma numérica y con tolerancia absolut equivalente a `1e-4`.La integral original involucra una función que diverge conforme $\phi \to \phi_0$, por lo que el cambio de variable de la ecuación (1) permite un cálculo numérico más preciso, pues la función $f(\phi, \phi_0)$ deja de diverger y se comporta bien en el dominio analizado.

Los métodos de búsqueda de raíces son aproximaciones numéricas a una solución exacta, por lo que teóricamente podrían estar buscando valores más cercanos al cero por un tiempo infinito; sin embargo, se debe establecer un límite, ya sea en iteraciones o tolerancia, para obtener un resultado.

El método `quad` de `scipy` es el más eficiente a la hora de integrar, pues busca el método más eficaz dentro de su código, mientras que el método de trapecios es más lento. El método de implementación propia, en contraste, es lento e ineficiente, demorando casi 10 veces más que las funciones de `scipy`. Esto se debe claramente a la robustez del código implementado.

Los métodos numéricos son eficaces a la hora de emitir soluciones aproximadas a problemas matemáticos, sin embargo, producen resultados en base a truncaciones, por lo que siempre existirá un error en la solución. Para casos como este, donde la integral no tiene representacion analítica, es cuando los métodos numéricos pueden ser aún más efectivos, siendo capaces de hacer representaciones del comportamiento de la integral.