INFORME TAREA 5

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

1. Problema.

Primero, vamos a manipular la ecuación entregada en el enunciado para dejarla como un sistema de ecuaciones diferenciales de primer orden.

$$\dot{\Omega} = -\frac{g}{L}[1 + 8\lambda^2 \cos(\omega_1 t)] \sin\phi$$
$$\dot{\phi} = \Omega$$

con las condiciones iniciales:

$$\Omega(0) = 0$$
$$\phi(0) = \phi_0$$

2. Implementación.

A continuación se implementó el método Runge-Kutta 4 para un sistema de dos ecuaciones diferenciales lineales ordinarias. Los parámetros a variar entre las seis veces que se ejecutó el programa son: λ y ϕ_0. Para resolver este problema se implementó RK4 pensando en que las dos ecuaciones mostradas pueden ser representadas por dos funciones que, después de usar datos del enunciado, quedan de la siguiente manera:

$$f_1(t, \Omega, \phi, g, \lambda) = \dot{\Omega} = -\frac{g}{L}[1 + 8\lambda^2 \cos(4\lambda\sqrt{g}t)] \sin\phi$$
$$f_2(t, \Omega, \phi) = \dot{\phi} = \Omega$$

Pese a que ambas funciones no dependen de todos los argumentos entre paréntesis, es más fácil escribirlas así al momento de implementar el algoritmo, ya que hace que el código sea visualmente más entendible, por lo que nos vamos a apegar a esa notación pese a que en estricto rigor no es correcta.

Con las funciones que describen las EDOs, podemos implementar RK4 sin problemas en un forloop.

3. Resultados.

En primer lugar vamos a fijar $g = 10$ como aproximación para la aceleración de gravedad en la superficie terrestre, pero el parámetro puede cambiarse para ver qué pasaría en la superficie de

algún otro planeta o en alguna otra situación, además vamos a fijar el paso en el tiempo como $h = 0.001$ ya que se observó que es un buen número tanto por tema de tiempo como por que la función de movimiento queda suficientemente suave como para ser graficada. En cualquier caso, no se recomienda correr todos los casos de una sola vez.

En esta sección se van a presentar, para cada caso un gráfico del recorrido que hizo la masa en el péndulo y para los casos más interesantes, también se presentará el espacio de fase asumiendo la masa $m = 1$ por simplicidad.

3.1. Primeros casos: $\phi_0 = 0.00339$.

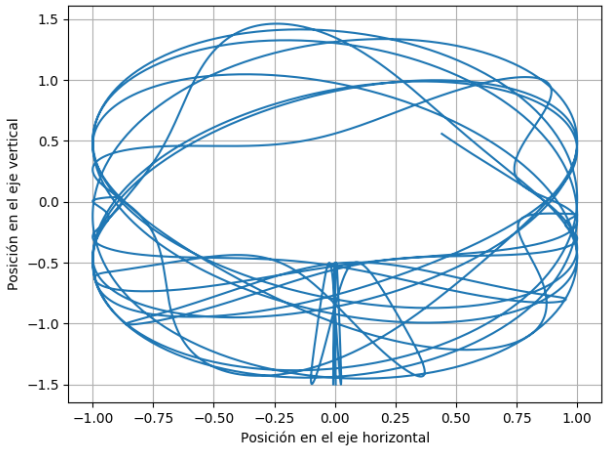

Figura 1: Camino recorrido con $\lambda = 1$.

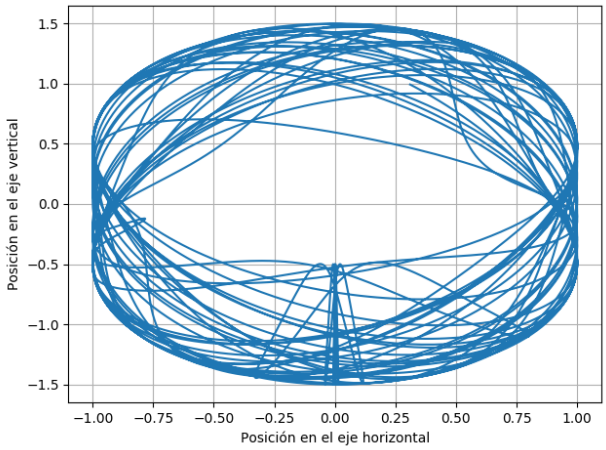

Figura 2: Camino recorrido con $\lambda = 2$.

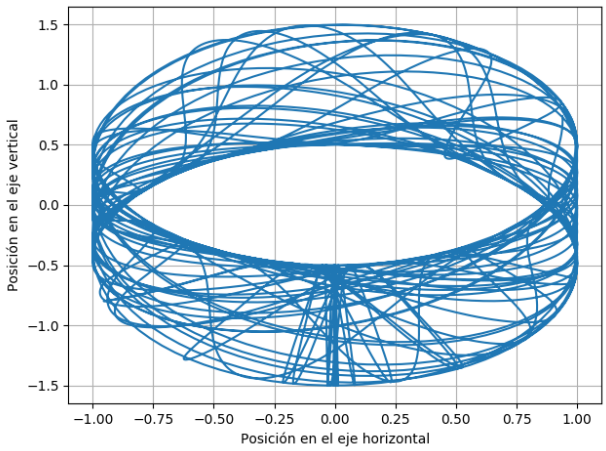

Figura 3: Camino recorrido con $\lambda = 3$.

3.2. Segundos casos: $\phi_0 = \pi - 0.00339$.

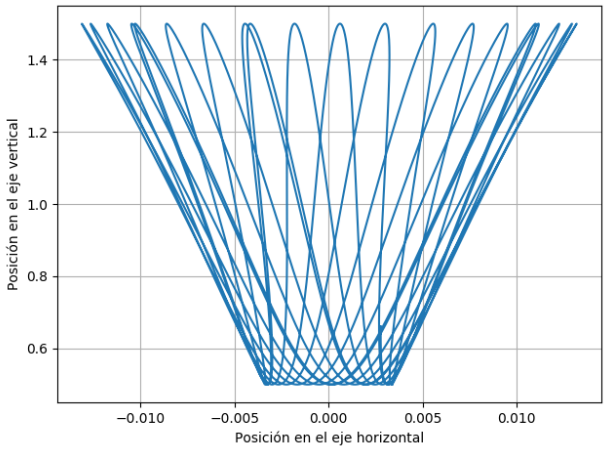

Figura 4: Camino recorrido con $\lambda = 1$.

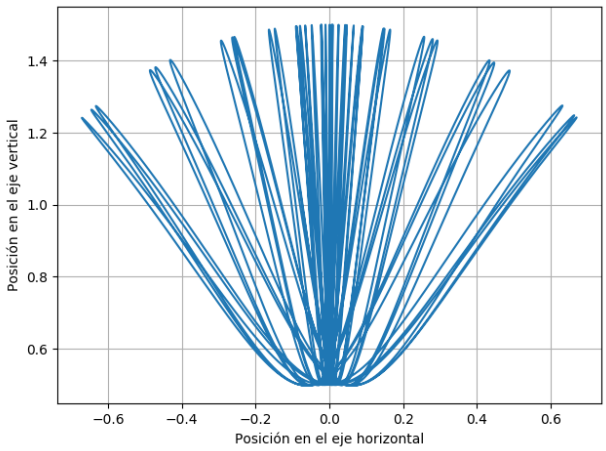

Figura 5: Camino recorrido con $\lambda = 2$.

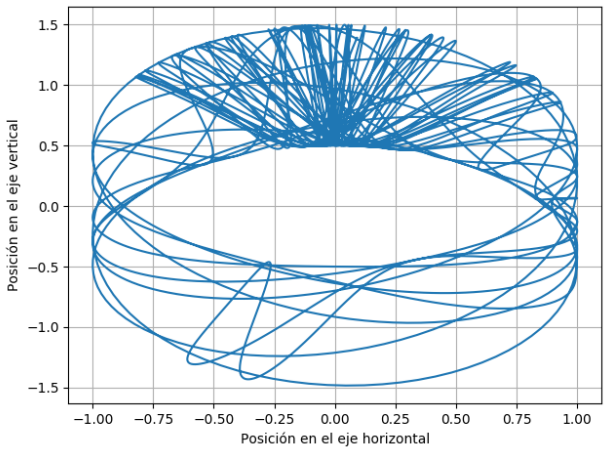

Figura 6: Camino recorrido con $\lambda = 3$.

3.3. Análisis

Vemos que tenemos un sistema cuyos puntos de equilibrio claramente son cuando nuestra masa parte en una posición vertical, para notar esto, veamos los espacios de fase con $\lambda = 1$, tanto partiendo con la masa en alto, o desde abajo.

Notemos que para el caso que parte con $\phi_0 = \pi - 0.00339$ tenemos un equilibrio estable, que notamos con un movimiento en el espacio de fase muy acotado, característico de este tipo de equilibrios, mientras que en la figura 7 vemos cómo pese a que se iba a quedar en un espacio acotado, se fue del equilibrio, comportamiento característico de los equilibrios inestables. este comportamiento se puede corroborar mirando la figura 1 que es el recorrido de este mismo caso.

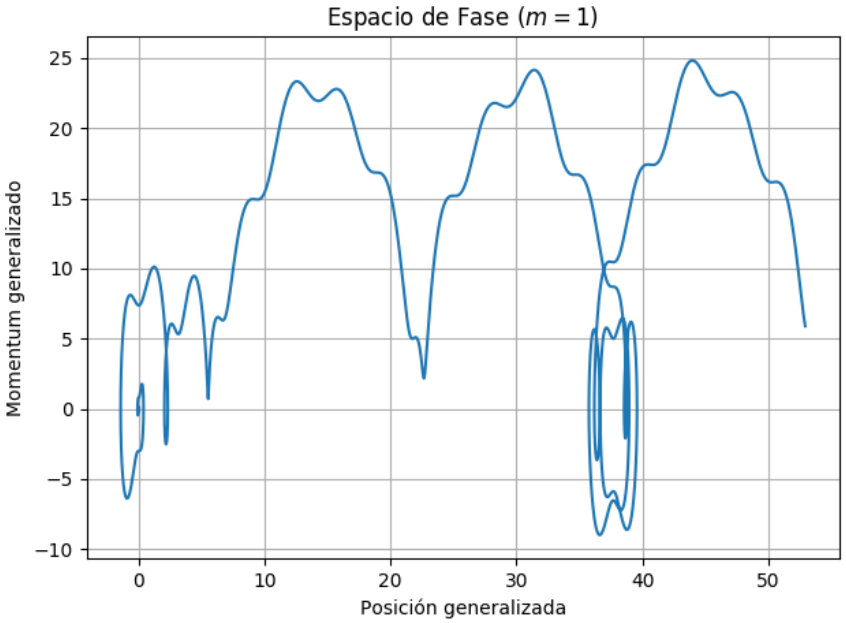

Figura 7: Espacio de fase con: $\phi_0 = 0.00339; \quad \lambda = 1.$

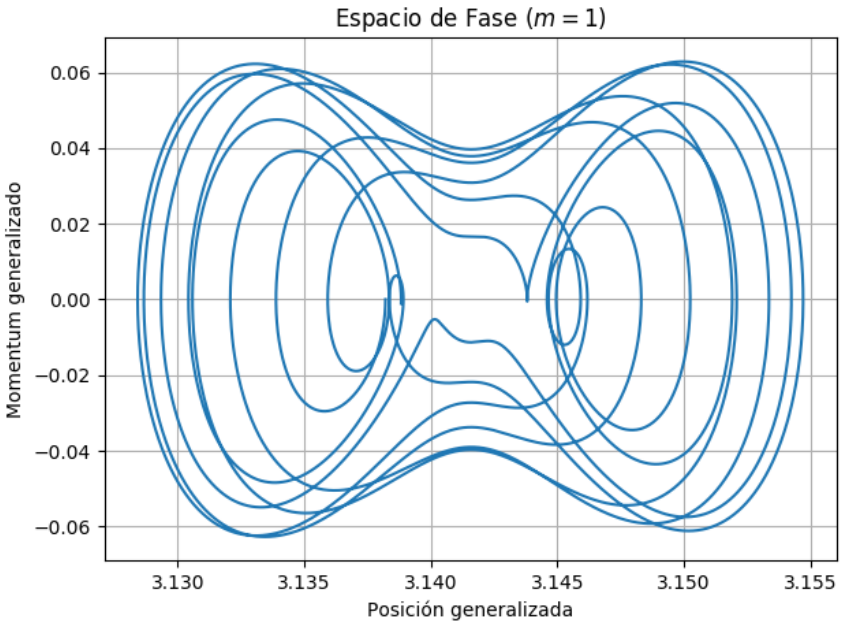

Figura 8: Espacio de fase con: $\phi_0 = \pi - 0.00339; \quad \lambda = 1.$

4. Desempeño.

La implementación de RK4 desarrollada en el código se demoró en promedio $\approx 0.451[s]$, por lo que, teniendo en cuenta que el largo de los arreglos utilizados es de 10000 datos, no está tan mal.

Por otro lado, la librería scipy tiene un módulo dedicado a la ingtegración numérica en scipy.integrate, el solver que se usó es scipy.integrate.solve_ivp con el método RK45 del mismo módulo de la librería. Al ejecutar este método, vemos que en promedio se demoró en promedio ≈ 1.654 lo cual a primera vista es bastante sorprendente, pero hay que tener en cuenta que nuestra implementación es mucho más específica que la que trae la librería scipy, partiendo por que esta clase en particular tiene muchas opciones, si nos ponemos a ver la clase, vemos que tiene muchas operaciones condicionales anidadas por lo que eso compromete su eficiencia. Por otro lado nuestra clase Setup existe sabiendo de antemano lo que se le va a ingresar y qué tipo de funciones entran, por ejemplo, la función sabe que es un sistema de dos ecuaciones diferenciales, y se implementa RK4 sin llamar a ningún módulo externo, entre otros detalles que mejoran el desempeño en este caso particular. Otra cosa es que quizá nuestra implementación gana en cortas distancias pero pierde en largas, es decir, quizá el número de puntos a evaluar no es suficiente para explotar la eficiencia de la implementación del módulo scipy.

5. Conclusión.

Vemos que pese a que existen librerías en donde se implementan muchas cosas para la resolución de problemas numéricos, pueden haber casos en que sí valga la pena implementar algo, siempre que el tiempo y las características del problema permitan una implementación que proporcione mejoras dignas de ser consideradas. En este caso, el tiempo invertido en la implementación, no vale los pocos segundos de cómputo ganados, aunque sí nos da una visión mucho mejor de cómo funcionan estos algoritmos y cómo usarlos.