Métodos Numéricos para la Ciencia e Ingeniería:

Informe Tarea n° 4

María Constanza Flores V.

Rut: 19077795-2

21 de octubre de 2015

1. Pregunta 1

En esta sección de la tarea, se pide que se respondan una serie de preguntas acerca de un tutorial desarrollado por *Software Carpentry*.

- **Describa la idea de escribir el `main driver` primero y llenar los huecos luego. ¿Por qué es buena idea?**

  Escribir el `main driver` primero sirve para poder darle una estructura previamente determinada al código, la cual consiste básicamente en tres partes: una parte donde se obtienen los parámetros de la línea de comando, una parte en donde se corre una simulación, y por último una parte en donde se reportan los resultados. Esto es una buena idea pues al tenerlo estructurado es más fácil de leer y también de corregir si es que tiene errores.

- **¿Cuál es la idea detrás de la función `mark_filled`? ¿Por qué es buena idea crearla e vez del código original que reemplaza?**

  La idea detrás de la creación de la función `mark_filled` es,, en primer lugar, hacer más exlícito el código; en segundo lugar, hace que los comentarios acerca de los errores obtenidos sean más significativos de lo que serían los mensajes genéricos entregados por Python al detectar un error del código original (que está siendo reemplazado), lo cual hace más fácil poder detectar errores; y por último, al crear la función se asegura que los valores tengan la coherenca buscada, evitando errores que Python no podría detectar.

- **¿Qué es *refactoring*?**

  Refactoring se refiere al hecho de reorganizar el código, cambiando su estructura pero no su funcionalidad, de modo de que sea más legible y más fácil de corregir al momento de ser testeado.

- **¿Por qué es importante implmentar tests que sean sencillos de escribir? ¿Cuál es la estrategia usada en el tutorial?**

  Es necesario implementar test sencillos para así prevenir posibles errores en ellos; los test son útiles para encontrar errores, por lo que si ellos en sí mismos presentan errores no estarían cumpliendo su función.
  La estrategia utilizada en el código para testear el código se basa en "dibujar.[el] caso que se quiere utilizar para el test. Esto se refiere a escribir strings con el caso a utilizar, por lo que puede ser reutilizable para diferentes tests.

- **El tutorial habla de dos grandes ideas para optimizar programas, ¿cuáles son esas ideas? Descríbalas.**

  Por un lado se da como idea *"trade memory for time"*, es decir, priorizar el ahorro de tiempo por sobre el de memoria, guardando información importante con el objetivo de no tener que recalcular, y así hacer más rápido el código. (También se puede hacer al revés, es decir priorizar el ahorro de memoria por sobre el de tiempo).
  Por otro lado se menciona la idea *"trade human time for higher machine performance"*, la cual hace referencia a mejorar el rendimiento del programa, pero haciéndolo más complicado de comprender.

- **¿Qué es `lazy evaluation`?**

  `lazy evaluation` es una forma común de optimizar. En ella, no se calculan valores hasta que sean necesarios, lo que hace que el código sea más rápido, pero más complicado de comprender.

- **Describa la *other moral* del tutorial (es una de las más importantes a la hora de escribir buen código).**

  La *other moral* dice que si se quiere obtener un código rápido, se debe partir por escribir un código simple, luego debe mejorarse testeándolo por partes, reusando los tests utilizados para checkear el trabajo en cada etapa.

2. Pregunta 2

2.1. Introducción

En esta pregunta se busca poder graficar órbitas de aquellos planetas que orbitan cerca del sol. Para ello se tiene su potencial, que se puede escribir como:

$$U(r) = -\frac{GMm}{r} + \alpha\frac{GMm}{r^2} \tag{1}$$

Que corresponde a una corrección a la ley de gravitación de Newton (lo que es una buena aproximación derivada de la teoría de relatividad de Einstein.

Para poder graficar las órbitas de los planetas en torno al Sol, se pide obtener las coordenadas de sus posiciones mediante tres distintos tipos de métodos numéricos para resolver ecuaciones diferenciales: Euler, Runge Kutta (de orden 4) y Verlet. En particular, se estudia el caso para $\alpha = 0$ y con las siguientes condiciones iniciales:

$$x_0 = 10$$
$$y_0 = 0$$
$$v_x = 0$$
$$v_y = 0{,}4$$

Tambien se busca un gráfico para la energía del sistema en función del tiempo, para los 3 casos.

2.2. Procedimiento

Para obtener la posición (x,y) del planeta, se puede obtener del potencial descrito en (1) las ecuaciones de movimiento de éste mismo, y luego a través de los métodos antes mencionados, integrarlas numéricamente. Para ello se sabe que en un sistema conservativo, como lo es el espacio en este caso pues no tiene roce, se tiene que:

$$m\vec{a} = -(x, y)$$
$$r = \sqrt{x^2 + y^2}$$

Por lo que se puede obtener las aceleraciones en sus componentes x e y del planeta haciendo lo siguiente:

$$a_x = -\frac{1}{m} \cdot \frac{dU}{dx} = \frac{2}{r^4} - \frac{GMx}{r^3} \tag{2}$$

$$a_y = -\frac{1}{m} \cdot \frac{dU}{dy} = \frac{2}{r^4} - \frac{GMy}{r^3} \tag{3}$$

Luego, para poder integrar, se definen la clase pPaneta, en la cual se definen las funciones `ecuacion_de_movimiento`, en donde se implementa la ecuación de movimiento, como sistema de ecuación de primer orden, las funciones `avanza_euler`, `avanza_rk4` y `avanza_verlet`, en las cuales se toma la condición actual del planeta y se hace avanzar su posición y velocidad en un intervalo de tiempo, utilizando los métodos de Euler explícito, Runge Kutta de orden 4 y Verlet respectivamente. Y por último se define la funcion `energía_total` en donde se calcula la energía total del sistema dada las condiciones actuales.

Posterior a ello, en los códigos `solución_usando_euler.py`, `solución_usando_rk4.py` y `solución_usando_verlet.py` se incluye la clase Planeta y se itera usando los diferentes métodos. Se grafican los resultados pedidos.

2.3. Resultados

Usando el método de Euler, se obtuvieron los gráficos mostrados en las Figuras 1 y 2.

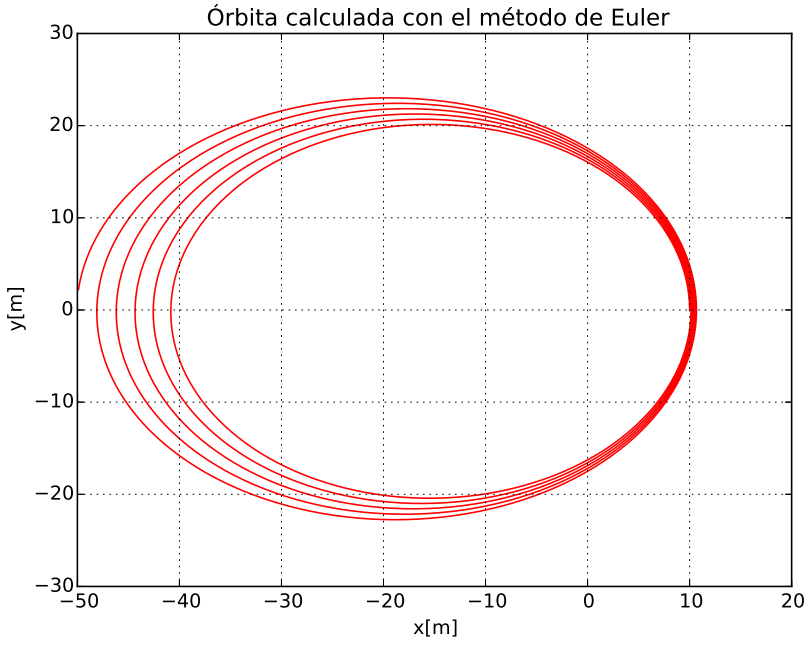

Figura 1: Trayectoria del planeta calculada con el método de Euler

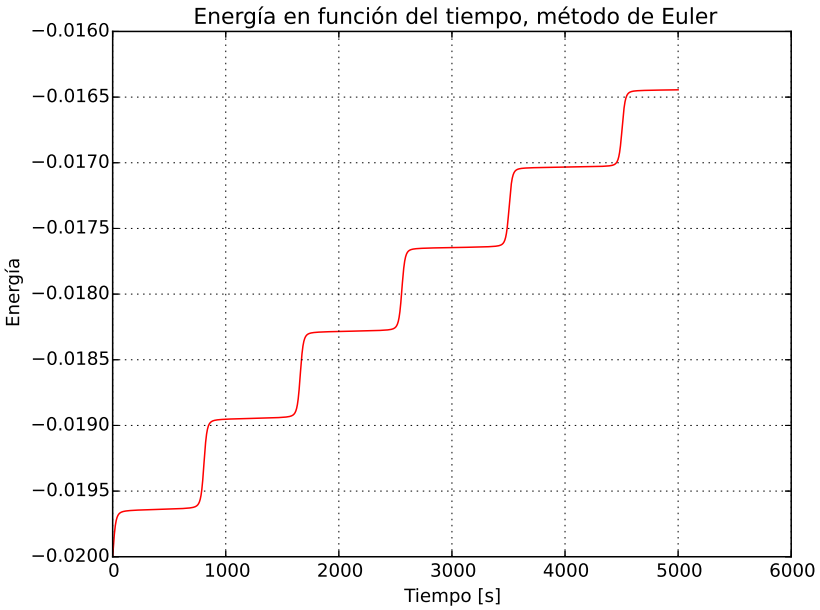

Figura 2: Energía del planeta en función del tiempo, calculada con el método de Euler

Con el método de Runge Kutta de orden 4, se obtuvieron los gráficos mostrados en las Figuras 3 y 4.

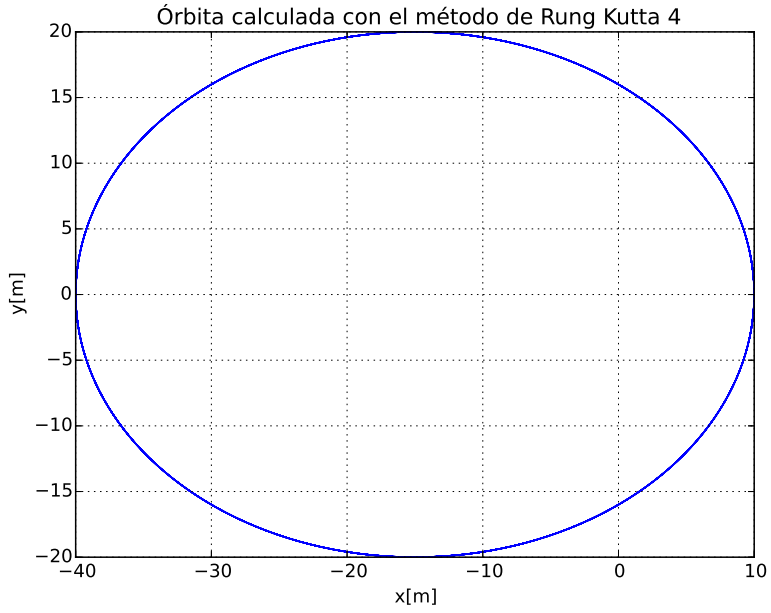

Figura 3: Trayectoria del planeta calculada con el método de Runge Kutta de orden 4

.

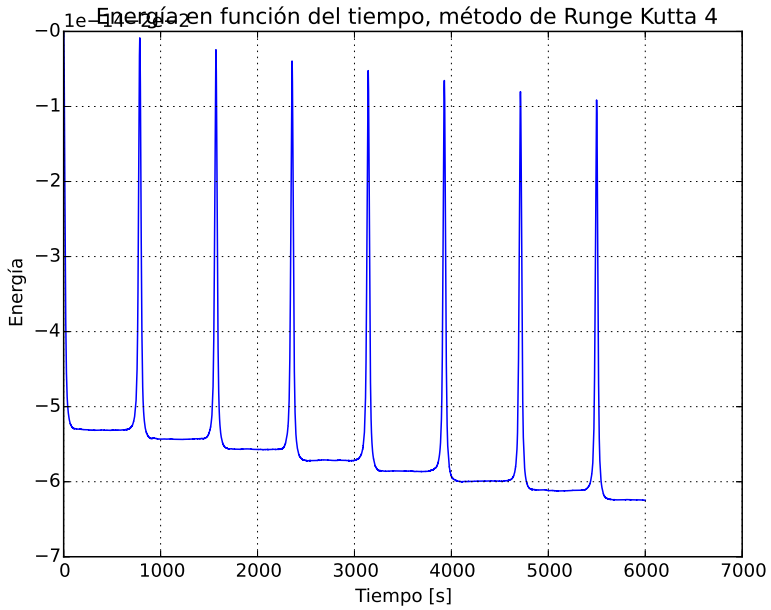

Figura 4: Energía del planeta en función del tiempo, calculada con el método de Runge Kutta

Y finalmente los resultados obtenidos con el método de Verlet se muestran en las Figuras 5 y 6

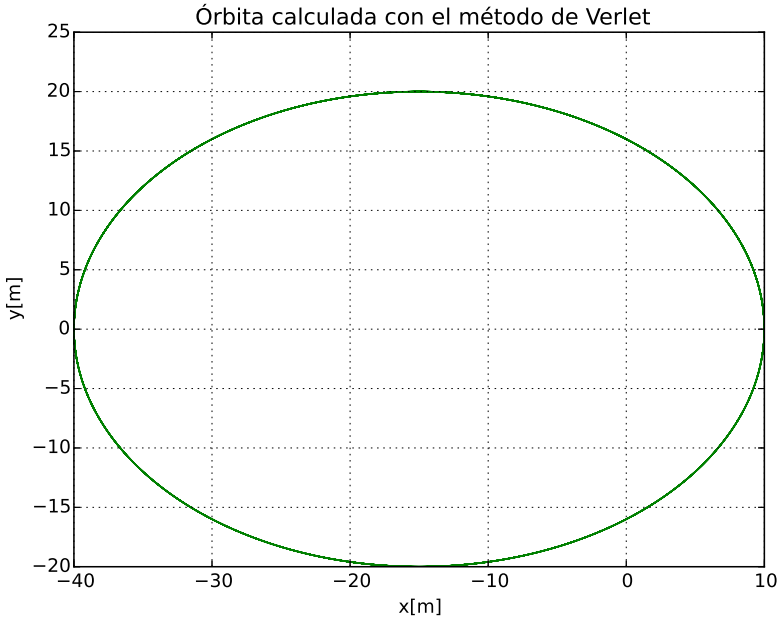

Figura 5: Trayectoria del planeta calculada con el método de Verlet

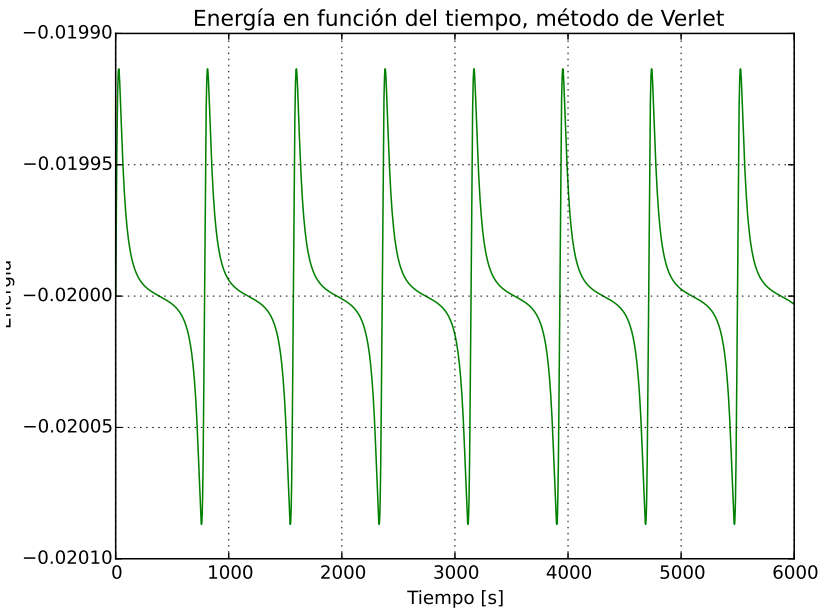

Figura 6: Energía del planeta en función del tiempo, calculada con el método de Verlet

2.4. Conclusiones

Con respecto al tutorial visto, como principal conclusión se puede destacar la importancia que tiene el hecho de escribir códigos simples y fáciles de leer, puesto que así, al momento de ir probándolos, se es más fácil de comprender en dónde están los errores.

En la pregunta 2 se utilizaron distintos metodos numéricos para resolver ecuaciones diferenciales ordinarias, para así poder comparar los resultados que se obtuvieron de ellos para un mismo problema. Por un lado en la Figura 1 es posible ver que utilizando el método de Euler el planeta no gira en torno a una órbita fija. Tambien es posible ver, en la Figura 2 que utilizando este método, el planeta está ganando energía a medida de que pasa el tiempo

Por otro lado, viendo las Figuras 3 y 5 se puede concluir que utilzando los métodos Runge Kutta y Verlet, el planeta si orbita en una órbita fija. Sin embargo, viendo la Figura 4, se puede notar que la energía del planeta va disminuyendo en función del tiempo. En cambio, la energía calculada con el método de Verlet oscila en torno a valores cercanos de la energía inicial (ver Figura 6).

Debido a todo a lo anterior se concluye que el mejor método para integrar es el método de Verlet, dado que en éste la energía se conserva (y varía sólo periódicamente).