Métodos Numéricos para la Ciencia e Ingeniería
Informe 4

Pablo Aníbal Peña Rojas - 19.077.067-2

October 22, 2015

1 Introducción

El trabajo consiste en dos partes diametralmente distintas. La primera es un completar un tutorial de Software Carpentry[1] para obtener y mejorar las habilidades de programación para, como dice el logo, 'hacer más en menos tiempo'. Para verificar que haya rendido el estudio de este tutorial se responderá una serie de preguntas con el fin de reafirmar lo aprendido. La segunda parte consiste en poner a prueba lo aprendido en el tutorial, además de la implementación de objetos para la programación, mediante el estudio de órbitas planetarias desde un punto de vista relativista. La gracia es que con el potencial gravitatorio relativista, las órbitas ya no son cerradas y los planetas precesan. El trabajo consiste principalmente en completar la clase 'Planeta' con funciones que entreguen su ecuación de movimiento con distintos métodos de resolución de EDOs (Runge-Kutta4, Verlet y Euler explícito).

2 Procedimiento

2.1 Parte 1 - Tutorial de Software Carpentry

El procedimiento fue ver los video tutoriales de la página web de Software Carpentry, tomando apuntes cuando fuese necesario.

2.2 Parte 2 - Órbitas con Potencial Gravitatorio Relativista

La aproximación newtoneana a el potencial relativista que se usó es

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

, que para efectos prácticos que simplificó por $U(r) = -\frac{GMm}{r}(\frac{\alpha}{r} - 1)$, cabe notar que alfa es un número pequeño (es más, se utilizó en la mitad de los casos $\alpha = 0$ y en las otras $\alpha = 10^(- 2.067)$. Se completó la clase planeta y fue importada al programa principal. Luego se observó la orbita del planeta (ploteada con python) para 5 periodos orbitales, con las siguientes condiciones

iniciales

$$\begin{aligned} x_0 &= 10 \\ y_0 &= 0 \\ v_x &= 0 \\ v_x &= 0.3 \\ GMm &= 1 \end{aligned}$$

Finalmente, se consideró el caso en que $\alpha = 10^(-2.067)$[2] y se integró utilizando Verlet para 30 órbitas.

3 Resultados

3.1 Parte 1 - Tutorial de Software Carpentry

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

Al escribir el main driver primero se puede tener una estructura sólida desde el comienzo, poder darle una secuencialidad de manera más sencilla. Al escribir el programa como una serie de funciones se ahorra tiempo ya que es más fácil encontrar bugs, agregarle cosas al código o incluso no hacer cosas que el problema no lo requiera.

¿Cuál es la idea detrás de la función mark filled? ¿Por qué es buena idea crearla en vez del código original al que reemplaza?

Es una función antibug. Verifica que los parametros a ingresar sean correctos, y si no lo son, entrega información especifica de cuál es el problema, evitando que el programa se caiga. Es mucho más fácil corregir errores con funciones de este tipo.

¿Qué es refactoring?

Es reorganizar un programa de manera tal que sea más facil probarlo. Sería el equivalente a 'pulirlo'. Limpiándolo y dejando claro que hacen todas las partes del código. En esta parte es útil probar el programa con casos extremos para ver si tira errores. Si es así, hay que retocar algunas partes entonces.

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

Porque si no lo son, hay peligro de enredarse y complicarse más la vida, agregando errores adicionales al programa o modificando partes que no son necesarias de modificar. La estrategia del tutorial es crear una lista de strings con todos los tests a realizar y sus resultados. Funcionaría como una especie de check in. De esta manera se pueden agregar o sacar tests fácilmente. El

único contra es que requiere una enorme cantidad de tiempo adicional para la manufactura del programa.

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

La primera idea es el 'Asymptotic Analysis',que es analizar funciones para números grandes y estudiar el tiempo que tarda el ordenar en procesar la data, y así medir su eficacia. Lo genial de esto es que se guarda la información en la memoria, por lo que no tiene que recalcular después lo mismo. En otras palabras, se puede intercambiar memoria por tiempo de ejecución. La segunda idea es el 'Binary Search', que es simplemente reducir la cantidad de datos a procesar al estar buscando en una lista o en un array. Se hace primero ordenando la pila de datos, luego dividiendola en la mitad y comparando el dato buscado con ese. Si es mayor o menor, se repite el proceso en esa pila de datos. La cantidad de iteraciones con este método son mucho menos que de manera aleatoria o secuencial.

¿Qué es lazy evaluation?

El lazy evaluation es una táctica de optimización. Consiste en no evaluar un dato hasta que sea necesario. De esta manera se evitan cálculos innecesarios. Es la versión virtual de la Ley del Mínimo Esfuerzo.

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

La other moral es que para escribir un programa veloz, hay que hacerlo simple. Luego de verificar su correcto funcionamiento, ir agregando cosas de a una a la vez, con la idea en mente de dejarlo lo más simple posible. De esta forma se evitan errores enormes por arrastre e incluso permite que crezca infinitamente (el código).

3.2 Parte 2 - Órbitas Planetarias Relativistas

Para la ecuación de movimiento se derivó el potencial, resultando para x:

$$\frac{GMx}{(x^2+y^2)^2}(2\alpha - \sqrt{x^2+y^2})$$

y para y:

$$\frac{GMy}{(x^2+y^2)^2}(2\alpha - \sqrt{x^2+y^2})$$

La resolución de las edos es bastante lineal, similar a lo visto en clases e informes anteriores. Sin embargo los resultados son bastantes distintos entre cada método de resolución de la EDO. Se encuentra destacable el hecho de que para 5 vueltas, con dt=1, con el método de euler se hicieron 2500 iteraciones, con el RK4 5000 y con el de Verlet tan sólo 800.

Para Euler:

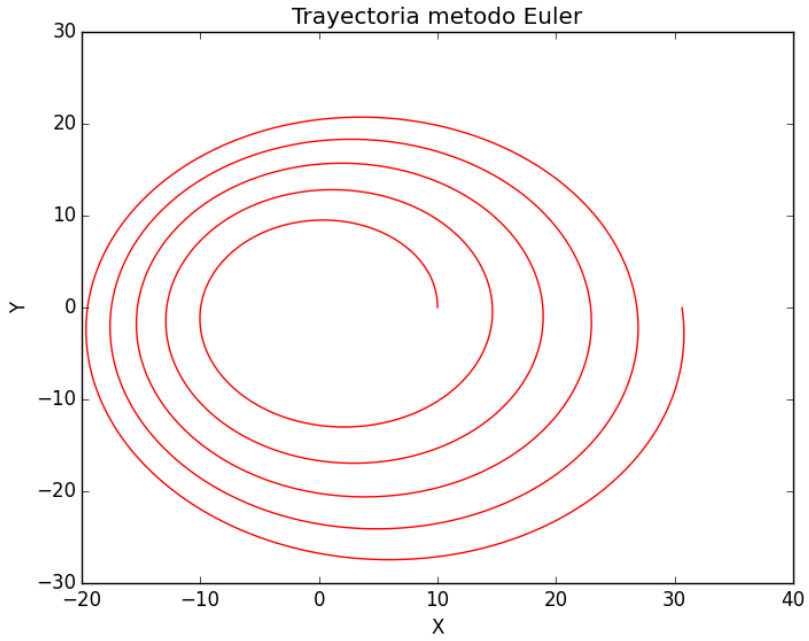

Figure 1: Gráfico de la trayectoria, con $\alpha = 0$ utilizando el método de Euler

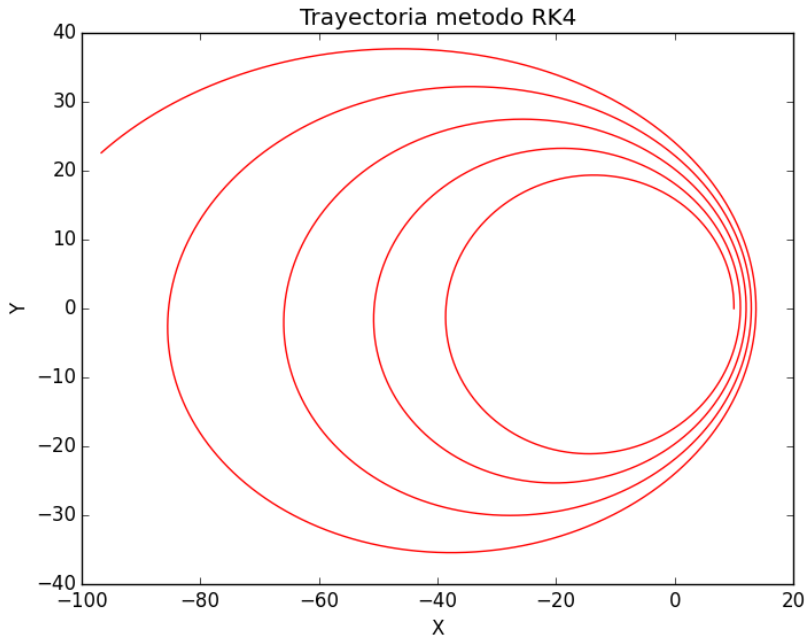

Figure 2: Gráfico de la trayectoria, con $\alpha = 0$ utilizando el método RK4

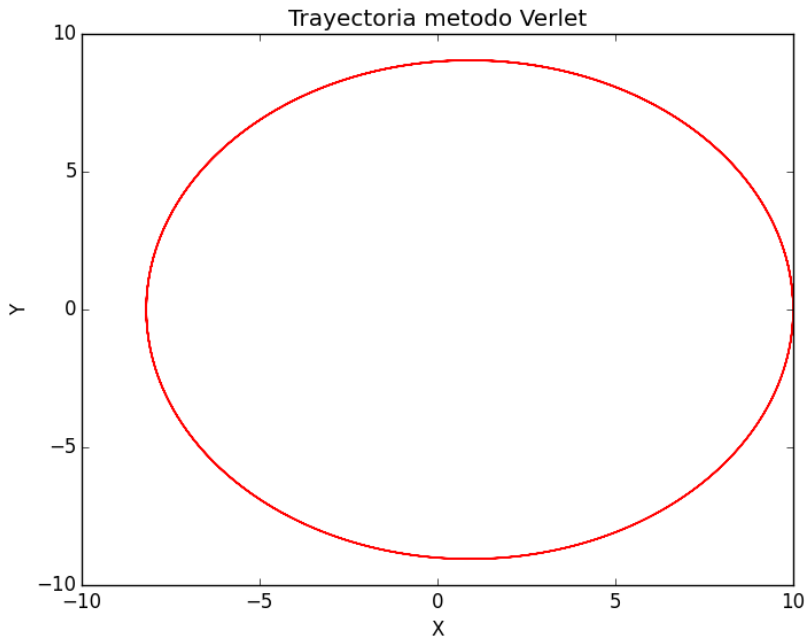

Figure 3: Gráfico de la trayectoria, con $\alpha = 0$ utilizando el método de Verlet

Los resultados para la energía basta tomar la suma de la energía cinética y la energía potencial gravitatoria:

$$E_(total) = \frac{1}{2}(v_x^2 + v_y^2) + \frac{GM}{x^2+y^2}(\alpha - \sqrt{x^2+y^2})$$

. Los resultados fueron los siguientes:

Finalmente, cabe destacar que para la precesión con alfa= $\alpha = 10^(- 2.067)$ y 30 vueltas se tuvo las siguientes gráficas:

4 Conclusiones

Para la primera parte, realmente resulta muy útil escribir todo el programa como funciones, porque es mucho más sencillo de analizar. La función de debugging también es útil, aunque demandaría aun más tiempo en hacer el programa. Cabe destacar que el uso de strings para darle múltiples usos a la función acortó mucho el tiempo de escribir el programa. (de la siguiente manera: Para la segunda parte, es muy llamativo la diferencia de iteraciones que necesitaron para las 5 vueltas los tres distintos métodos. Siendo RungeKutta4 el que más tardaba (5000 iteraciones), Euler el intermedio (2500 iteraciones) y Verlet el más veloz (¡800 iteraciones!). Las órbitas también son muy distintas, en el caso de Euler se tiene que la órbita es altamente inestable, para el caso de RK4 se tiene que la órbita es semiestable y en Verlet es altamente estable. Para el balance energético

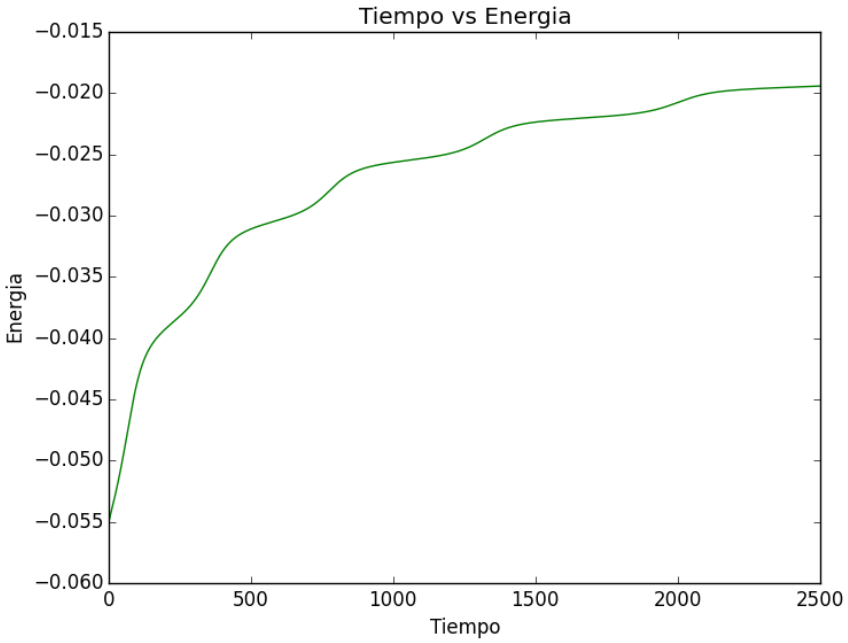

Figure 4: Gráfico de Energía vs Tiempo, con $\alpha = 0$ utilizando el método de Euler

cabe destacar que: para euler, si bien el planeta parece que terminará cayendo en su estrella, se tiene que la energía apenas tiene fluctuaciones anómalas (aparte del incremento de energía potencial, que es un incremento estable, es decir, semilineal). Para RK4 se tiene una conducta más extraña, con fluctuaciones periódicas para cada vuelta, lo que tiene sentido de acuerdo a su trayectoria, siendo el punto de inicio de la trayectoria los 'mínimos' de la gráfica. Para verlet se observa que el comportamiento de la energía es periódico, al igual que su trayectoria, por lo que tiene sentido. Finalmente, para la solución verlet de 30 vueltas, se tiene que apenas se mueve de la órbita el planeta, por lo mismo su energía apenas tiene fluctuaciones (del orden de $10^(- 6)$

Más cosas que destacar! Lo difícil que es ordenar las imágenes con látex... siempre se desordenan, sin importar los [!h].

References

[1] http://swcarpentry.github.io/v4/invperc/index.html

[2] Mi RUT es 19.077.067-2

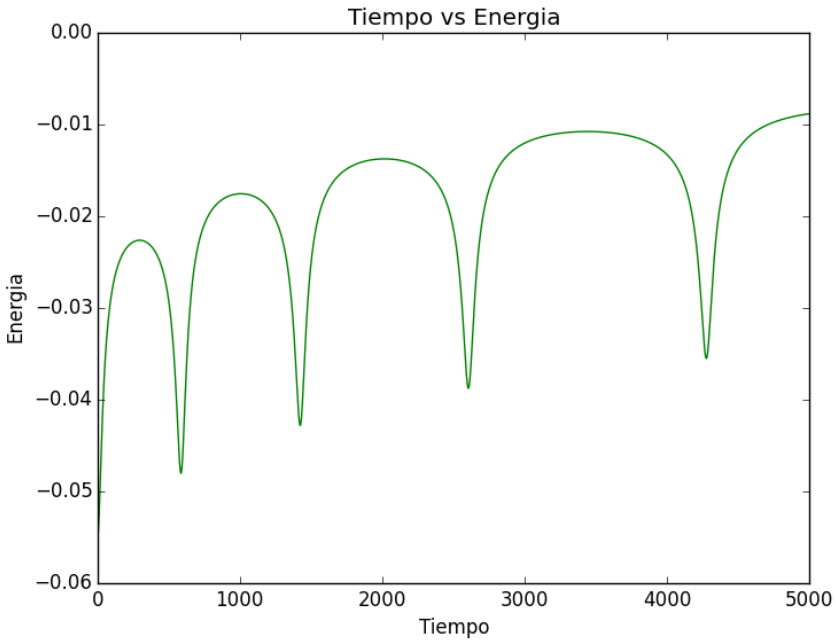

Figure 5: Gráfico de Energía vs Tiempo, con $\alpha = 0$ utilizando el método RK4

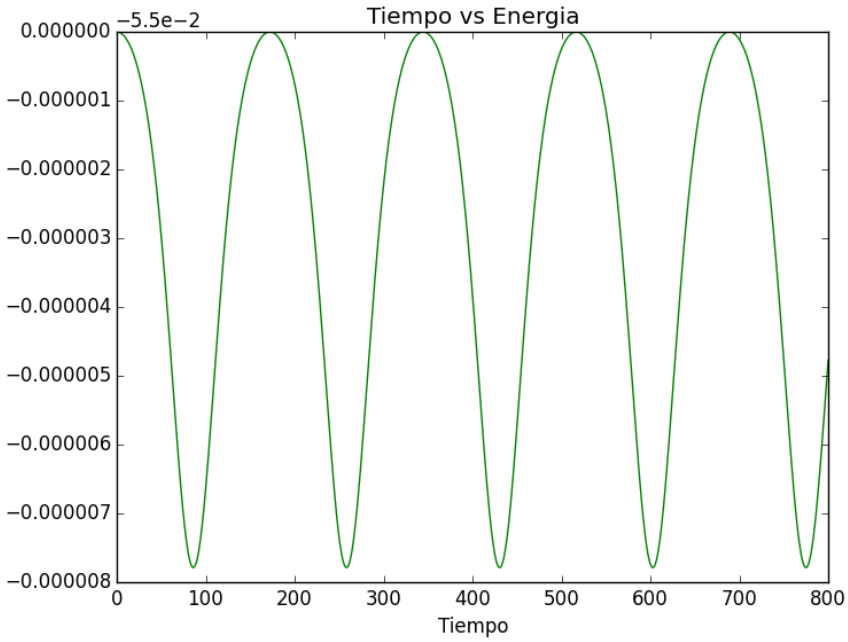

Figure 6: Gráfico de Energía vs Tiempo, con $\alpha = 0$ utilizando el método de Verlet

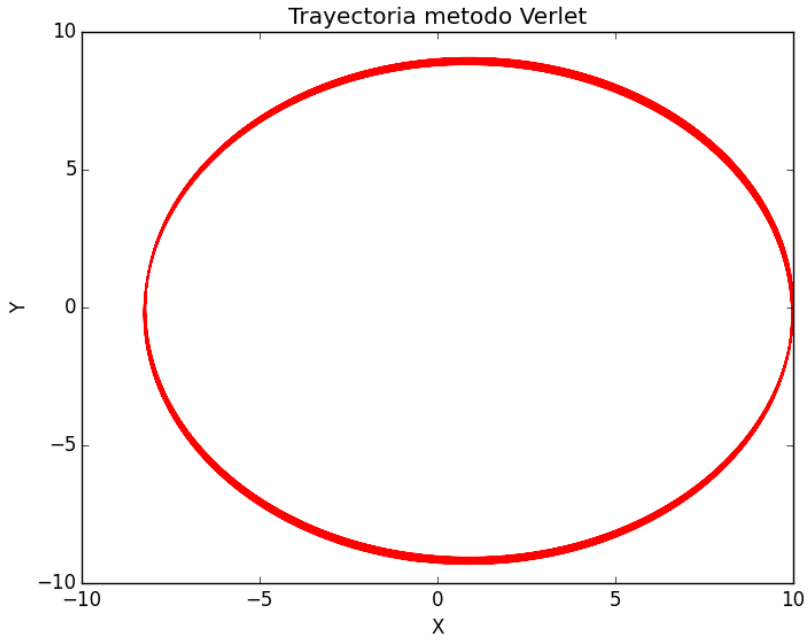

Figure 7: Gráfica de la trayectoria, con $\alpha = 10^(- 2.067)$ utilizando el método de Verlet

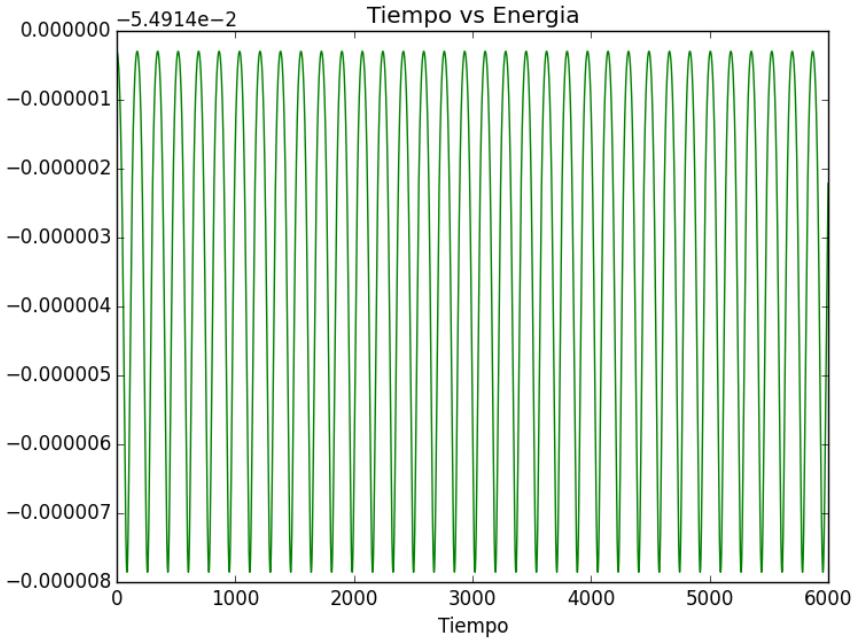

Figure 8: Gráfico de Energía vs Tiempo, con $\alpha = 10^(- 2.067)$ utilizando el método de Verlet

```
def orbitar(CI,solucion, prt=True):

    if solucion=='euler':
        n= 2500 #grande y dsps se arregla
    if solucion=='rk4':
        n= 5000 #grande y dsps se arregla
    if solucion=='verlet':
        n=800
    Aiur= Planeta(CI) #se crea el planeta
    if solucion=='verlet_reloaded': #caso especial para la ultima parte
        n=6000
        Aiur= Planeta(CI,alpha=alpha2) #si es el verlet bkn se parcha
    dt= 1 #error si es muy chico
    '''
```