Métodos Numéricos para la Ciencia y la Ingeniería

Camila Castillo Pinto. RUT 18.889.762-2

20 de Octubre, 2015.

1 Introducción.

2 Pregunta 1

La pregunta 1 consistió en responder una serie de preguntas en base a un tutorial desarrollado por Software Carpentry.

2.1 Respuestas

- El *main driver* corresponde al código que se está implementando, pero en una etapa inicial, donde sólo se escriben strings que recuerden el objetivo del código y se va implementando de a poco. La idea de escribir un *main driver* es que se vaya implementando desde lo más general, para luego pasar a los detalles; por ejemplo, escribir un pedazo de código que realice cierta acción y use una función que todavía no haya sido definida (general), para que una vez que se termine con este código se defina las funciones necesarias (particular, detalles). Esto permite que la implementación sea ordenada y coherente, evitando problemas como olvidar definir una función o que partes del código completo no calcen.

- La idea de crear la función *mark_filled* consiste en, además de marcar la casilla como llena, entrega mensajes más claros si es que llegase a haber un error (no el típico error que entrega Python) y es mucho más ordenado.

- *Refactoring* es el término que se utiliza para referirse al procedimiento de cambiar la estructura del programa de manera que mejore la calidad de éste, sin modificar su comportamiento ni su objetivo.

- Es importante implementar tests sencillos de escribir ya que se pretende que éstos ayuden a crear un programa que esté correcto, entonces si se implementa un test que es complicado probablemente tendrá errores que llevarán a más complicaciones al momento de querer llegar a un código que funcione correctamente.

  La estrategia usada en el tutorial es buscar situaciones en donde el programa que se desea testear probablemente pueda fallar, de esta forma implementamos 'casos límites',

se les pone a prueba usando el programa implementado y se observan los resultados. En el caso particular del ejemplo del tutorial se escriben los 'casos límites' y los resultados como strings.

- La primera idea para optimizar un programa consiste en guardar la información que esté repetida, acción que requiere memoria de computador; en vez de volver a calcular esta información, acción que requiere tiempo importante.

  La segunda idea para optimizar utiliza algo llamado 'búsqueda binaria' y consiste en dividir el tramo inicial en dos, luego se determina si lo que se está buscando se encuentra en la primera o en la segunda mitad y, dependiendo de esto, se divide la nueva sección en la mitad y se repite el proceso hasta llegar al valor buscado. El tramo inicial contiene toda la información que se encuentra repetida, por lo que el proceso de recuperar esta información se hace mucho más rápida.

- *Lazy Evaluation* es una forma muy común y utilizada para optimizar los programas y consiste en calcular valores sólo en la medida en que sea necesarios.

  En el ejemplo particular del tutorial consiste en generar sólo los números de las casillas vecinas a las de aquellas casillas llenas. De esta forma no es necesario generar una grilla completa con números al azar, reduciendo así el tiempo de ejecución del programa.

- La *other moral* del tutorial dice que si se quiere escribir un programa que sea rápido, se debe en primer lugar escribir un programa simple, luego debe ser probado (usando tests y 'casos límites') y luego debe ser mejorado parte por parte, volviendo a chequear con los tests cada vez que se modifique algo.

2.2 Conclusiones

De este tutorial se puede concluir que el proceso de programación no debe hacerse de cualquier forma; existe un proceso de implementación que puede permitir hacer mejoras y encontrar errores en el programa de manera sencilla. Este proceso consiste en ir desde un código que sea simple y directo, para luego ir ahondando en los detalles y uniendo cada parte del código.

3 Pregunta 2

Se presenta la forma del potencial para planetas que orbitan cerca del Sol, con la corrección de la relatividad de Einstein:

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

con α pequeño.

3.1 Parte 1

Esta parte consistió en definir por completo la clase Planeta en el archivo *planeta.py*, definiendo sus atributos y métodos.

3.1.1 Procedimiento

Se completaron los siguientes métodos de la clase Planeta:

- *ecuacion_de_movimiento*: Se definió la ecuación de movimiento como un sistema de ecuaciones.

  En primer lugar se definieron las constantes $G = M = m = 1$ y se obtienen los valores de las condiciones iniciales para x, y, v_x, v_y mediante el comando *self.y_actual* (ya implementado en *__init__*).

  Luego se definieron el radio $r = \sqrt{x^2 + y^2}$ y las funciones:

  $f_x = \frac{xGMm}{r^3} - \alpha\frac{2xGMm}{r^4}$

  $f_y = \frac{yGMm}{r^3} - \alpha\frac{2yGMm}{r^4}$,

  que fueron calculadas mediante la fórmula $f_i = -\frac{\partial U}{\partial i}$ para $i = x$ o $i = y$.

- *avanza_euler*: Corresponde a la implementación del método de Euler explícito para calcular la posición y velocidad de un planeta luego de una fracción de tiempo dt.

  Para ello se obtienen los valores de las velocidades v_x y v_y, y los valores de las funciones f_x y f_y, y se ocupan para calcular las nuevas posiciones x e y, y velocidades v_x y v_y, con la fórmula (Euler Explícito):

  $y_{n+1} = y_n + hf(t_n, y_n)$,

  donde y_{n+1} corresponde a los nuevos valores, y_n los antiguos valores, h es el paso (en este caso es el dt) y $f(t_n, y_n)$ corresponde a la función (en este caso será las velocidades y las funciones f_x y f_y).

  Finalemente se actualizan los valores de las posiciones y velocidades.

- *avanza_rk4*: Implementación del método de Runge Kutta 4 visto en clases, cuyas fórmulas se muestran a continuación:

$$\begin{cases} k_1 &= f\left(x_i, y_i\right) \\ k_2 &= f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}hk_1\right) \\ k_3 &= f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}hk_2\right) \\ k_4 &= f\left(x_i + h, y_i + hk_3\right) \end{cases}$$

$$y_{i+1} = y_i + \frac{1}{6}h\left(k_1 + 2k_2 + 2k_3 + k_4\right)$$

  Donde para calcular cada k_i se deben obtener los valores de las velocidades y de f_x y f_y, y luego actualizar los valores de las condiciones iniciales según lo obtenido con las fórmulas.

- *avanza_verlet*: Se implementó el método de Verlet con las fórmulas vistas en clases (Verlet que requiere de la velocidad):

$$x_{n+1} = x_n + v_n h + \frac{a_n h^2}{2} \tag{2}$$

$$v_{n+1} = v_n + \frac{(a_n + a_{n+1})h^2}{2} \tag{3}$$

  Donde x_{n+1} y v_{n+1} son la posición y velocidad que se quiere calcular, en función de las anteriores x_n y v_n. h es el paso y a_n corresponde, en este caso, a f_x (para calcular x y v_x) y a f_y (para calcular y y v_y).

  Notar que para calcular las posiciones se deben obtener los valores de la condición inicial y con ello los valores de f_x y f_y, mientras que para calcular las velocidades los valores de la condición inicial deben ser actualizados con las posiciones encontradas y a partir de ahí se debe calcular nuevamente f_x y f_y.

- *energia_total*: Se implementó este método de manera que obtuviese las condiciones iniciales del problema, calculase r, para luego calcular la energía como: $E = T + U$, donde T corresponde a la energía cinética y U corresponde a la energía potencial.

3.2 Parte 2

En esta parte se pedía implementar la integración de la ecuación de movimiento con los 3 métodos: Euler, Runge Kutta y Verlet; con las siguientes condiciones iniciales:

$$x_0 = 10$$
$$y_0 = 0$$
$$v_x = 0$$

y $\alpha = 0$.
Además se utiliza $GMm = 1$.

3.2.1 Procedimiento

Se crean archivos donde se soluciona la ecuación mediante los distintos métodos. Se le asigna la clase Planeta a una variable.

La condición que faltaba, v_y, fue escogida luego de varias pruebas con distintos valores, pero la que mostraba resultados que se podían analizar, y que finalmente fue fijada, fue la del valor $v_y = 0.4$.

En primer lugar se ecogió el número de pasos a dar, el dt y el arreglo de tiempos (todos fijados luego de probar varios valores para cada uno).

Luego se crearon arreglos para x, y y la Energía, de manera de ir guardando los valores que resultaban en la iteración.

Se guardan los primeros valores, determinados por las condiciones iniciales, luego se procede a iterar. Para esto se llaman a los métodos implementados en la clase Planeta

(*avanza_euler, avanza_rk4, avanza_verlet*) y luego se obtienen los valores de x y de y que quedaron determinados. Además se calcula la energía. Se guardan todos los valores obtenidos en los arreglos correspondientes.

Finalmente se grafican: la trayectoria (órbita del planeta, x versus y), y la energía en función del tiempo.

Se utiliza esta misma estructura para integrar la ecuación con cada método.

Nota: Se plotearon los gráficos y al intentar guardarlos, se guardaron imágenes en blanco. No se logró localizar el error, pero al correr los códigos los gráficos aparecen de forma normal. Es por esto que no se adjuntaron en el pull request los archivos .png que contenían los gráficos.

3.2.2 Resultados y Análisis

Se muestran los gráficos resultantes en las figuras 1, 2 y 3.

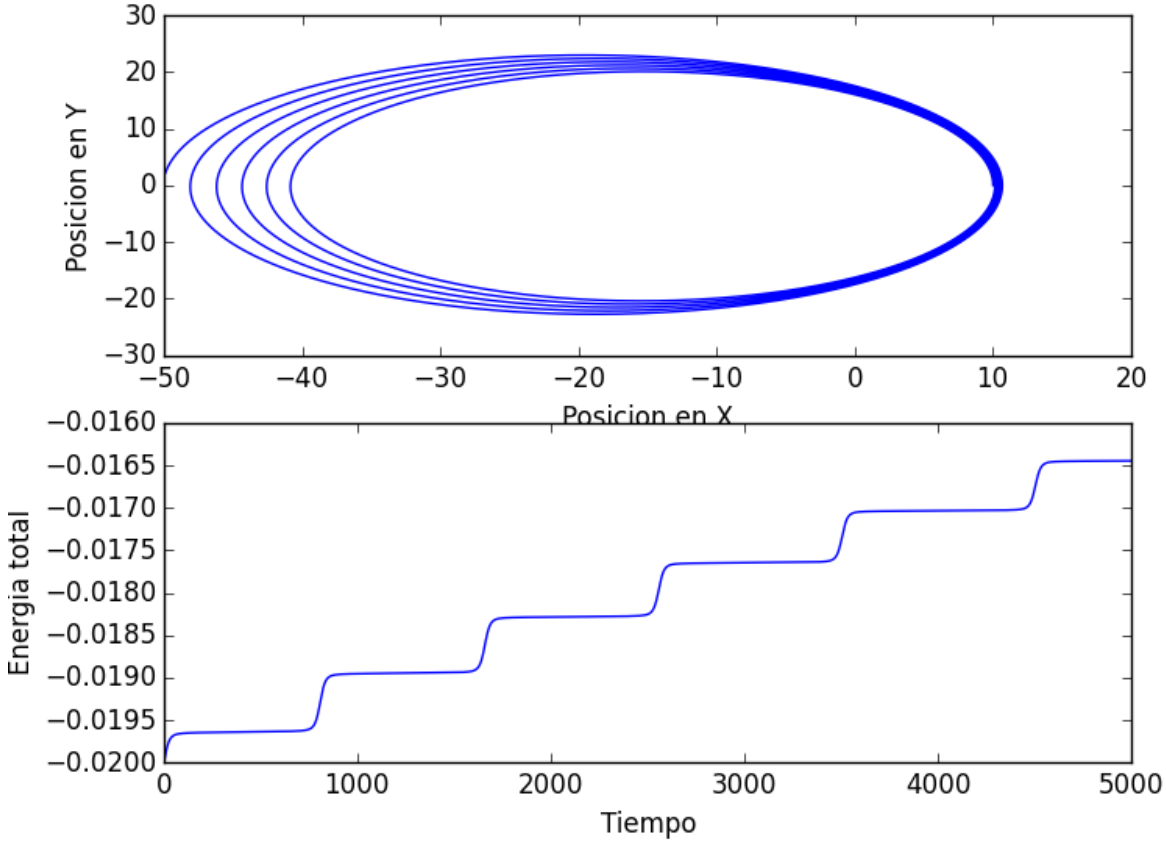

Figure 1: Método de Euler. Gráfico de la Trayectoria: x versus y. Gráfico de la Energía versus tiempo.

De las Figuras 1, 2 y 3 podemos apreciar que el método que mejor conserva la energía es el método de Verlet.

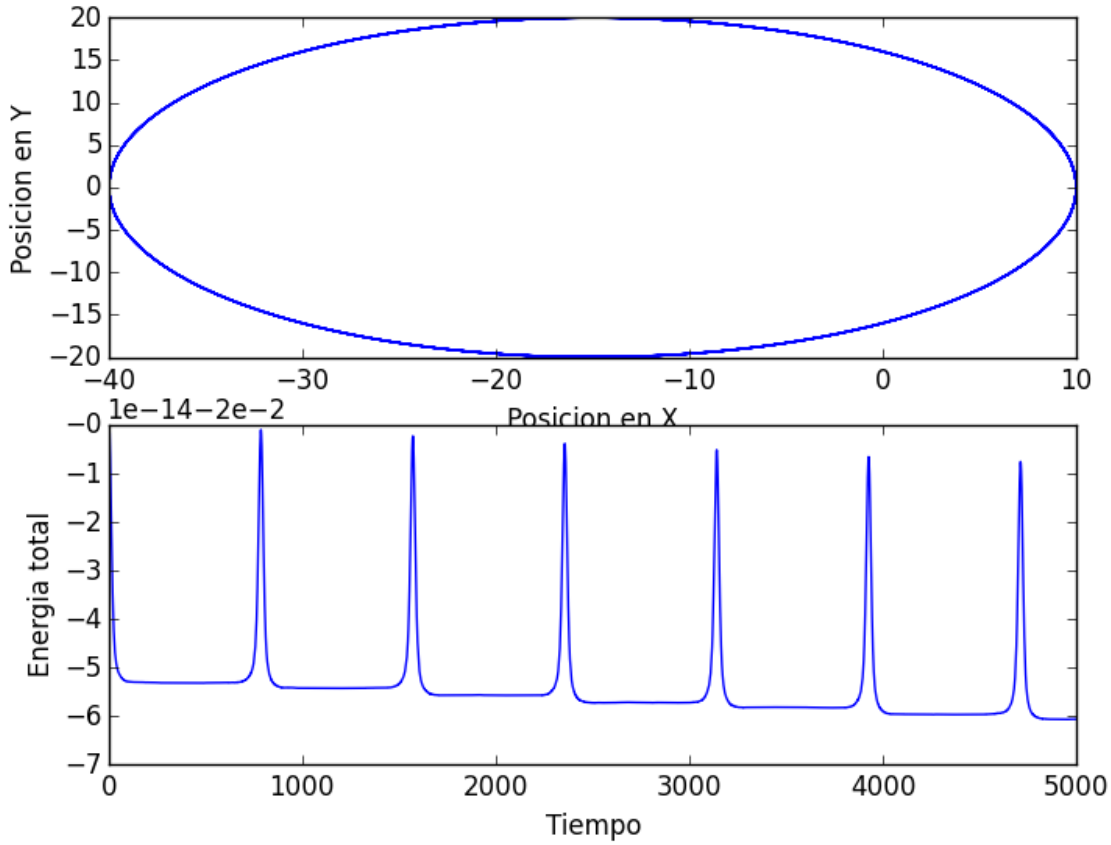

Figure 2: Método de Runge Kutta 4. Gráfico de la Trayectoria: x versus y. Gráfico de la Energía versus tiempo.

Además se puede observar que los métodos de Runge Kutta 4 y Verlet, a diferencia del método de Euler Explícito, presentan soluciones de la trayectoria que son estables.

3.3 Parte 3

Para esta parte se debía realizar la integración con el método de Verlet pero con $alpha = 10^{-2.XXX}$ con XXX últimos dígitos del RUT. Además determinar la velocidad angular de precesión.

3.3.1 Procedimiento

Se usó la misma estructura que en la parte anterior para integrar y se cambió $alpha$, con $alpha = 10^{-2.762}$.

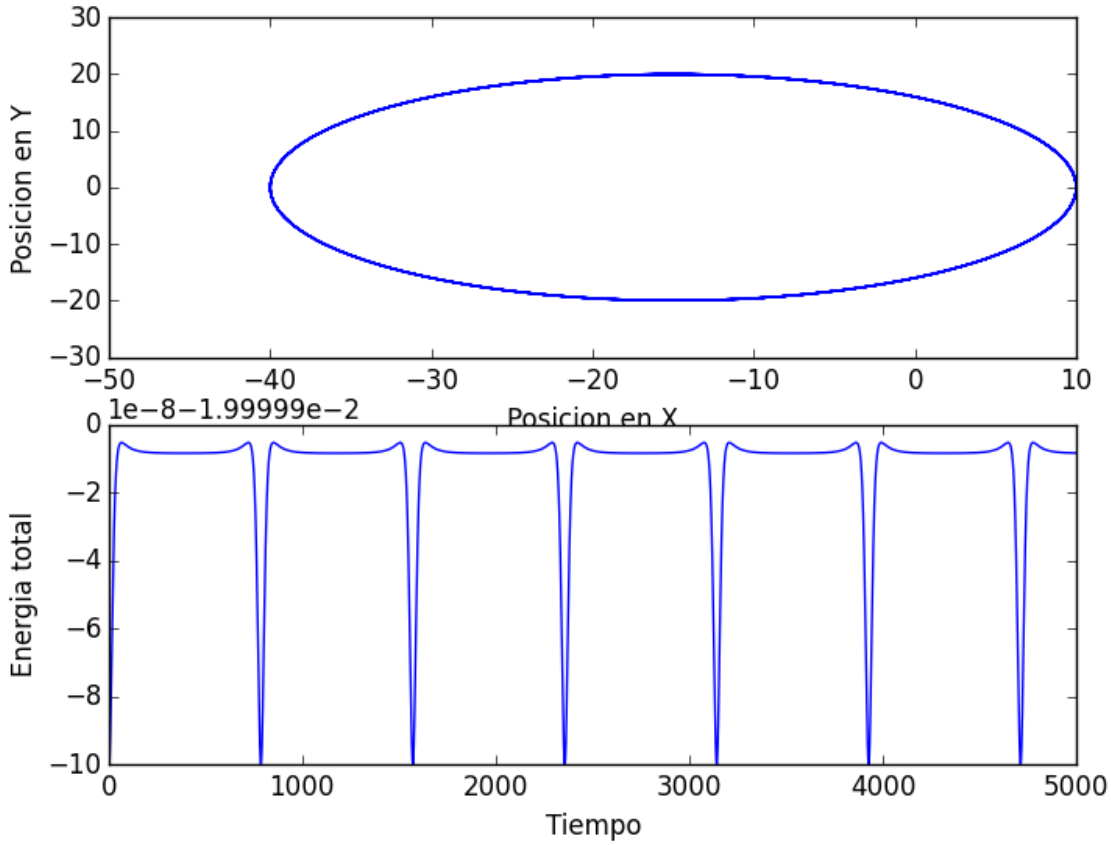

Figure 3: Método de Verlet. Gráfico de la Trayectoria: x versus y. Gráfico de la Energía versus tiempo.

3.3.2 Resultados y Análisis

El gr'afico resultante se muestra en la Figura 4.

En el gráfico de la figura 4 podemos notar que el contorno de la trayectoria se observa más grueso. Al hacer un acercamiento se observó que esto era provocado porque la órbita se fue desplazando, fenómeno llamado precesión. Además la energía se logró conservar de excelente forma.

3.4 Conclusiones

Se concluye, pues, que los métodos que mejor conservan la energía son el Runge Kutta 4 y el de Verlet. Además se pudo observar que cuando se varía el valor de α, la órbita comienza a desplazarse, moviéndose así su perihelio, fenómeno conocido como Precesión.

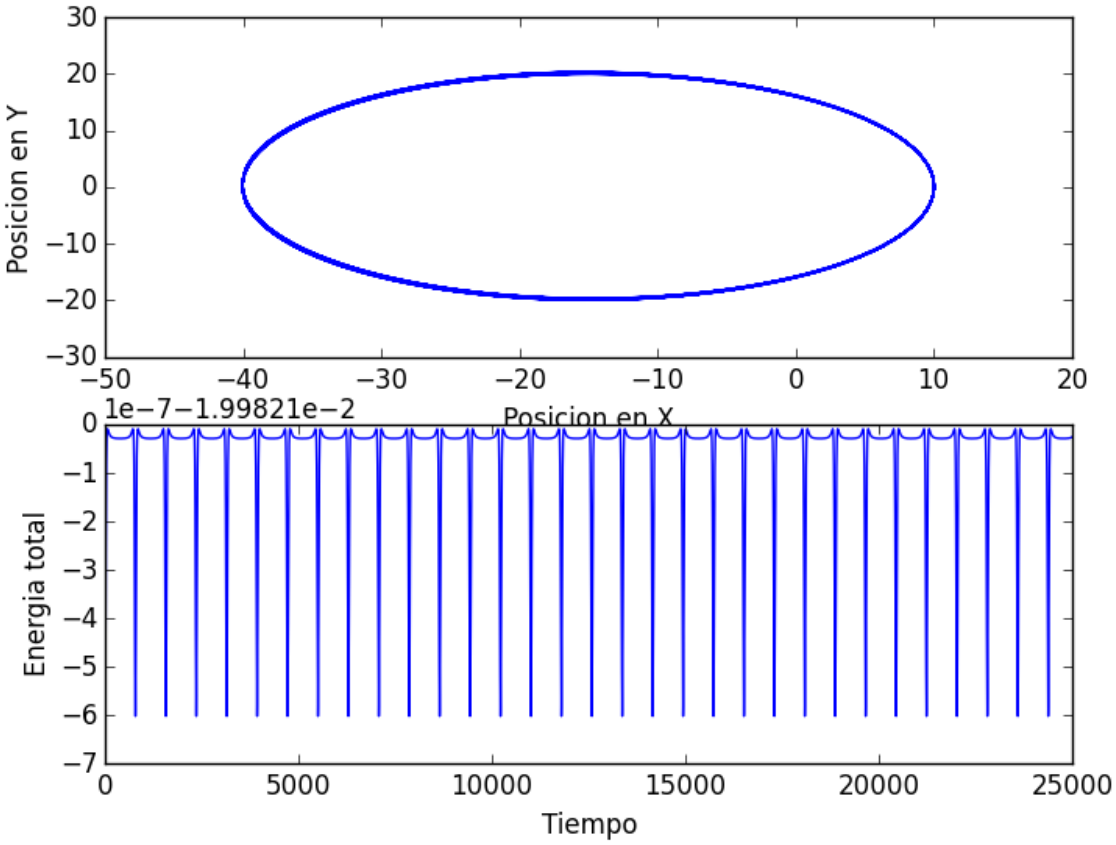

Figure 4: Método de Verlet, $\alpha = 10^{-2.762}$. Gráfico de la Trayectoria: x versus y. Gráfico de la Energía versus tiempo.