Universidad de Chile, Facultad de Ciencias Fisicas y Matematicas
Metodos Numericos para la Ciencia y Ingenieria

Tarea 4: Diseño de programas y Órbitas

Maria Jose Hernandez Pozo

October 22, 2015

1 Introduccion

En este trabajo analizaremos la importancia del diseño de programa, tanto en su construcción, revisión y ejecución mediante la revisión de un tutorial desarrollado por Software Carpentry en el cual se estudia un problema ejemplo: invasion percolation.
Posteriormente se aplicará lo aprendido en el análisis de órbitas cerca del sol con la corrección a la ley de gravitación de Newton, la cuál es una aproximación derivada de la teoría de la relatividad general de Einstein.

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

donde α es un número pequeño. Bajo este potencial, las órbitas siguen siendo planas pero ya no son cerradas, sino que precesan, es decir, el perihelio (punto más lejano de la órbita) gira alrededor del Sol.
Adjunto a este informe se encuentra el archivo planeta.py es cuál contiene la Clase PLaneta de los cuales se implementarán los métodos.
Por otra parte se estudiarán distintan órbitas con alpha igual cero y distinto; en el segundo caso se determinará con los datos la velocidad angular de precesión.

2 Procedimiento

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

EL tutorial esta construido en torno al ejemplo invasion percolation, que es una modelación de filtración de contaminantes a traves de rocas. Después de analizar por partes el problema y una posible solución a cada uno, se procede a construir el cuerpo del programa o main driver, utilizando funciones inexistentes, esto se hace en gran medida para ahorrar tiempo, pues muchas veces se construyen primero las funciones y durante la construccion del cuerpo nos damos cuenta que no nos sirven o no nos da el valor de la forma que se requiere. Por otra parte también es buena idea para tener presente que funciones y variables necesitamos para construirlas todas después, de forma más ordenada.

¿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 interesante ver como implementa la función mark_filled la cuál nos dice si un pedazo de nuestra roca se filtró o no, lo que le da un plus es a esta es primero, saber lo que nos entrega respecto a las coordenadas X e Y; segundo que nos proporciona mensajes en el caso que las coordenadas entregadas a la función no cumplan con alguna condición esencial para que el programa funcione bien, se podría pensar que esto es innecesario pues python también nos dice si existe algún error, sin embargo, algunas coordenadas que nosotros podemos consirar erroneas como -1, no lo son para python como un error, además nuestro mensajes serán muchos más significativos que los entregados por Python, es decir sabremos a que se debe especificamente el error.

¿Qué es refactoring?

Luego de construir tanto el cuerpo como las funciones se desea probar el programa sin embargo, se requiere hacer un orden de este para hacerlo más testeable esto recibe el nombre de Refactoring que significa "cambiar la estructura de un programa sin modificar su comportamiento o funcionalidad con el fin de mejorar su calidad". Esto va desde hacer más eficientes algunas funciones, agregar comentarios a las funciones para especificar que hacen, que requieren y que entregan, condiciones iniciales para realizar pruebas fáciles, hasta cambiarle el nombre a la funciones que puedan ser muy símiles.

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

Hecho esto, es la hora de probar el programa, por lo que es importante realizar una prueba que sea sencilla de escribir, ya que un código complejo puede contener errores, entonces no confiaremos en este test y por otra parte si el código de prueba es fácil y confiable podemos observar una cantidad importante de casos.
La estrategia usada en el tutorial plantea un fixture, que es lo que el test usará como caso para probar determinada parte del programa, el resultado esperado para determinado caso para poder comparar con el reasultado real y el test en sí.

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

Para optimizar tanto el programa como las pruebas existen dos puntos importantes: cambiar espacio por tiempo, es decir, guardar datos para despues no volver a calcularlos y hacer el programa más eficiente. O hacer un código más complejo para que sea más rápido y hacer testearlo varias veces.

¿Qué es lazy evaluation?

Un ejemplo de esto es Lazy evaluation, una estrategia de evaluación que retrasa el cálculo de una expresión hasta que su valor sea necesario, y que también evita repetir la evaluación en caso de ser necesaria en posteriores ocasiones, esto se hace con el fin de que sea más rápido el realizar pruebas.

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

Por último es importante recalcar "the other moral" que se menciona al final de tutorial: Si desea escribir un programa que sea rápido, empiece por escribir un programa que es simple. A continuación, pruébelo y mejórelo poco a poco, reutilice las pruebas para comprobar su trabajo en cada etapa.
Hecho el análisis del tutorial se procede a analizar el caso propuesto.
Nuestro potencial al derivarlo con respecto a r obtenemos

$$F(r) = \frac{GMm}{r^2} - 2\alpha\frac{GMm}{r^3}$$

Donde F es la fuerza producida por el potencial U, recordando que F=ma y derivando con respecto a x e y, obtenemos las componentes de la aceleración radial

$$a_x(x,y) = GM\frac{x}{\sqrt{x^2+y^2}}(\frac{1}{(x^2+y^2)} - 2\alpha\frac{1}{\sqrt[2/3]{x^2+y^2}})$$
$$a_y(x,y) = GM\frac{y}{\sqrt{x^2+y^2}}(\frac{1}{(x^2+y^2)} - 2\alpha\frac{1}{\sqrt[2/3]{x^2+y^2}})$$

Donde los componentes que vienen después de GM corresponden a un coseno y seno respectivamente.
Por otra parte, dado que se proporcionan las condiciones iniciales de posicion y velocidad en las coordenadas x e y, podemos definir un vector de estas condiciones y derivarlas obteniendo un vector con las velocidades y aceleraciones

$$\begin{pmatrix} x \\ y \\ v_x \\ v_y \end{pmatrix} = \frac{d}{dt}\begin{pmatrix} v_x \\ v_y \\ a_x \\ a_y \end{pmatrix}$$

De esta forma podemos escribir nuestra ecuación de movimiento usando esta derivada.

Para integrar mediante euler, al actualizar los valores de posición y velocidad, el valor n+1, será el valor n más un delta de tiempo multiplicado por la derivada de la función.
Para verlet por cada coordenada se utiliza el siguiente algoritmo:

$$y_{n+1} = 2 * y_n - y_{n-1} + (\Delta t^2)\frac{dy}{dt}$$

Donde Δt^2 es determinado por nosotros, en este caso será 0.1 o 0.3 y por último con el método Runge-Kutta, tiene la forma de

$$\begin{aligned} k_1 &= f(x_i, y_i) \\ k_2 &= f(x_i + \frac{1}{2}h, y_i + \frac{1}{2}hk_1) \\ k_3 &= f(x_i + \frac{1}{2}h, y_i + \frac{1}{2}hk_2) \\ k_4 &= f(x_i + h, y_i + hk_3) \\ y_{n+1} &= y_n + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6} \end{aligned}$$

Aplicando estas ecuaciones al código quedaría más menos así

$$\begin{aligned} K1 &= f(x, y, v_x, v_y) \\ K2 &= f((x, y, v_x, v_y) + \Delta t * K1/2) \\ K3 &= f((x, y, v_x, v_y) + \Delta t * K1/2 + \Delta t * K2/2) \\ K4 &= f((x, y, v_x, v_y) + \Delta t * K1/2 + \Delta t * K2/2 + \Delta t * K3/2) \\ y_{n+1} &= y_n + (K1 + 2K2 + 2K3 + K4)\Delta t/6. \end{aligned}$$

Donde f correspondeerá a la función de movimiento

3 Resultados

Figure 1: Grafica con $\alpha = 0$ usando la integración de euler, Condiciones Iniciales [10, 0, 0, 0.3]

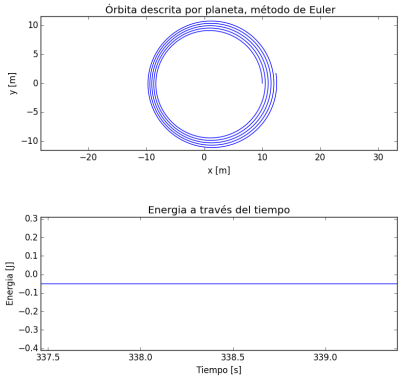
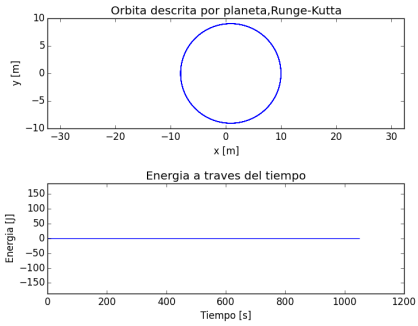


Figure 2: Grafica con $\alpha = 0$ usando la integración de Runge-Kutta, Condiciones Iniciales [10, 0, 0, 0.3]



4 Conclusiones

Primero el análisis hecho con respecto al tutorial hace reflexionar importantes puntos a la hora de construir un programa que nos entregue seguridad y rapidez. Se explican varias formas y "morales" para poder trabajar, lamentablemente no se pudo implementar de forma correcta en este problema, esto resalta en que no se pudo resolver un error en el método de verlet y por ende no se generó gráfica.

Sin embargo, observando los resultados, tenemos que para el metodo de euler el radio de la orbita va creciendo y con esto tambien suponemos que su energia va creciendo y no se conserva, lo que claramente es errónea al compararlo con datos reales y con los otros métodos; esto puede deberse a una inestabilidad del métodos haciendo crecer un error. Por su parte Runge-Kutta la órbita se conserva y por ende su energía también lo que representa un buen método para trabajar.