Métodos Numéricos para la Ciencia e Ingeniería

# Informe Tarea 4

Nelson Soto Medina RUT 18.694.312-0

## Parte 1 (Sección de Preguntas):

- **Describa la idea de escribir el *main driver* primero y llenar los huecos luego. ¿Por qué es buena idea?**
  La idea de escribir el main driver primero es para crear la estructura a nuestro programa, para así tener un orden en él a la hora de rellenar con funciones y códigos. Es buena idea porque permite tener todo mas organizado a la hora de buscar errores o bugs.
- **¿Cuál es la idea detrás de la función mark_filled? ¿Por qué es buena idea crearla en vez del código original al que reemplaza?**
  La idea de la función es ayudarnos en la detección de errores. Cuando uno ingresa datos que pueden producir un error, la función los detecta y entrega mensajes de errores propios. Es útil ya que si dejamos correr el programa python puede hacer caso omiso a los datos y generar un error a lo largo del programa el cual tal vez no se pueda identificar fácilmente, ocupando mas tiempo.
- **¿Qué es *refactoring*?**
  El refactoring se refiere a modificar nuestro programa haciendo cambios a su estructura pero sin cambiar su comportamiento, para así mejorarlo y hacerlo más eficiente para testearlo y legible a la hora de buscar errores.
- **¿Por qué es importante implmentar tests que sean sencillos de escribir? ¿Cuál es la estrategia usada en el tutorial?**
  Porque si el test es complicado podrían producirse errores en su propia implementación, en cambio si es sencillo es menos probable de que presente bugs y podrá ayudarnos a ver si el programa hace lo que debiera. En el tutorial se usan los “fixtures”. Para cada fixture hay un resultado correcto, el que se compara con el resultado que entrega el test y así comprobar el funcionamiento de nuestro programa.
- **El tutorial habla de dos grandes ideas para optimizar programas, ¿cuáles son esas ideas? Descríbalas.**
  La primera se trata de ocupar memoria a cambio de tiempo. Se guarda información redundante para evitar volver a hacer cálculos y ahorrar tiempo. La otra idea es utilizar más tiempo de uno mismo para optimizar el

funcionamiento del programa, y así no gastar tiempo en el futuro por errores inesperados.

- **¿Qué es lazy evaluation?**
  Se refiere a evaluar o calcular los valores de las variables solo cuando se pide. Así se reduce el tiempo de trabajo en la programación a cambio de eficiencia en el programa.
- **Describa la *other moral* del tutorial (es una de las más importantes a la hora de escribir buen código).**
  La other moral nos dice que para hacer un programa rápido, primero debemos partir por algo simple, y luego ir mejorándolo y haciéndolo más eficiente utilizando pequeños test para verificar que funcionen las mejoras, hasta tener algo más complejo pero eficiente.

# Parte 2 (Problema):

En esta parte se introduce la ecuación del potencial para planetas que orbitan en torno al Sol. Es una corrección a la ley de Newton sobre la gravedad derivada de la relatividad general de Einstein:

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

Con esta fórmula ahora las órbitas precesan.

Se nos pide integrar la ecuación de movimiento utilizando los métodos de Euler explícito, Runge-Kutta 4 y el de Verlet, con las condiciones iniciales:

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

Con Vy a elección y con GMm=1.

Para llevar a cabo todo esto es necesario trabajar la ecuación de potencial para obtener la ecuación de movimiento.

Dado que:

$$F_x = -\frac{\partial U}{\partial x}, F_y = -\frac{\partial U}{\partial y}$$

Se puede concluir, al dividir por la masa, que las ecuaciones de movimiento son:

$$\ddot{x} = -xGM(\frac{1}{r^3} - \frac{2\alpha}{r^4}) \qquad \ddot{y} = -yGM(\frac{1}{r^3} - \frac{2\alpha}{r^4})$$

Y notamos:

$$r = \sqrt{x^2 + y^2} \qquad \ddot{x} = f_x \qquad \ddot{y} = f_y$$

Con esto podemos definir las funciones para integrar y obtener posiciones, velocidades y así también la energía del sistema (Todo presente en el archivo de python de la clase Planeta). Así se completa la **primera parte**.

Para la **segunda parte** simplemente integramos, pero con el alfa de la función potencial igual a cero y mostrar la trayectoria del planeta tras aproximadamente 5 vueltas. El Vy que elegí es Vy=0.27.

Se setea la condición inicial y se discretiza para cada método.

Tras varias pruebas modificando el tiempo se puede aproximar el tiempo necesario para que se muestren las 5 vueltas.

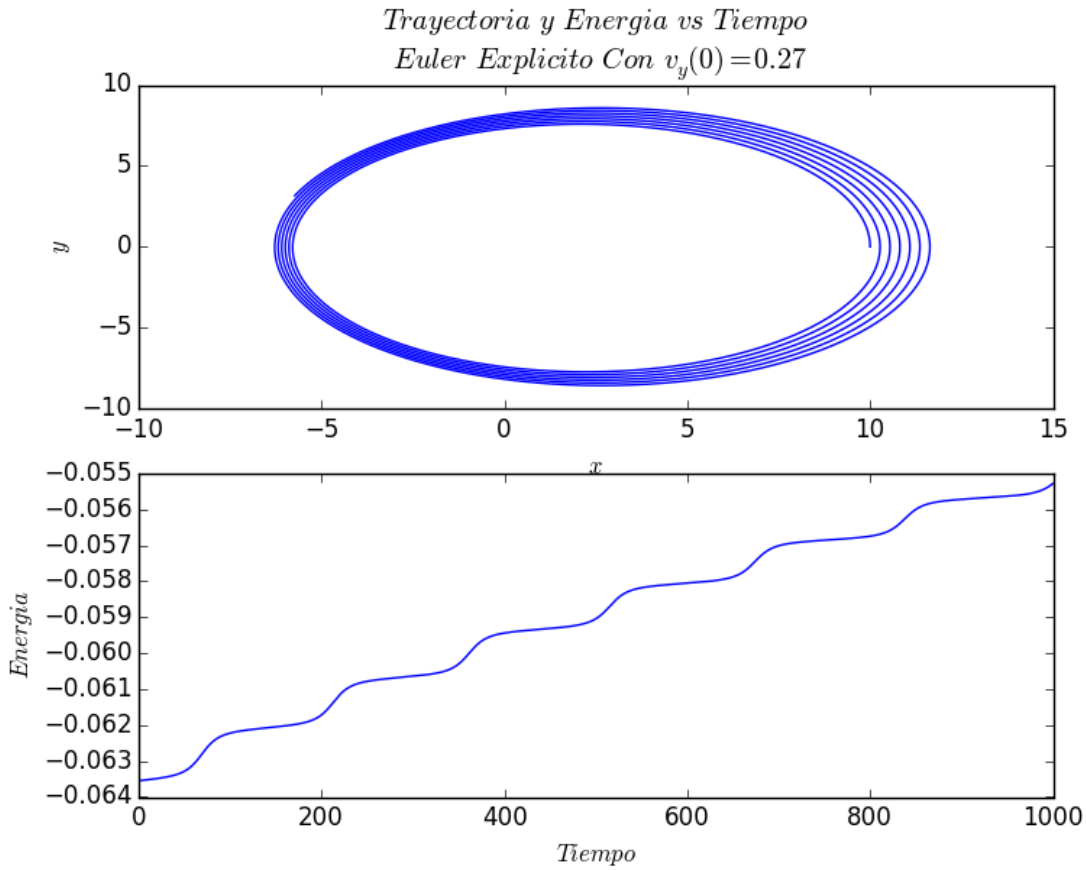
Se utiliza el siguiente método iterativo para integrar:

```
N=1
while N<Pasos:
    p.avanza_METODO(h)
    N+=1
    x.append(p.y_actual[0])
    y.append(p.y_actual[1])
    p.energia_total()
    Ene.append(p.EneActual)
```

Donde lo único que cambia es METODO, el que puede ser ‘euler’, ‘rk4’ y ‘verlet’.

## Euler Explícito

Para el método de Euler Explícito se obtuvo un buen resultado con 30000 pasos y un tiempo igual a 1000:



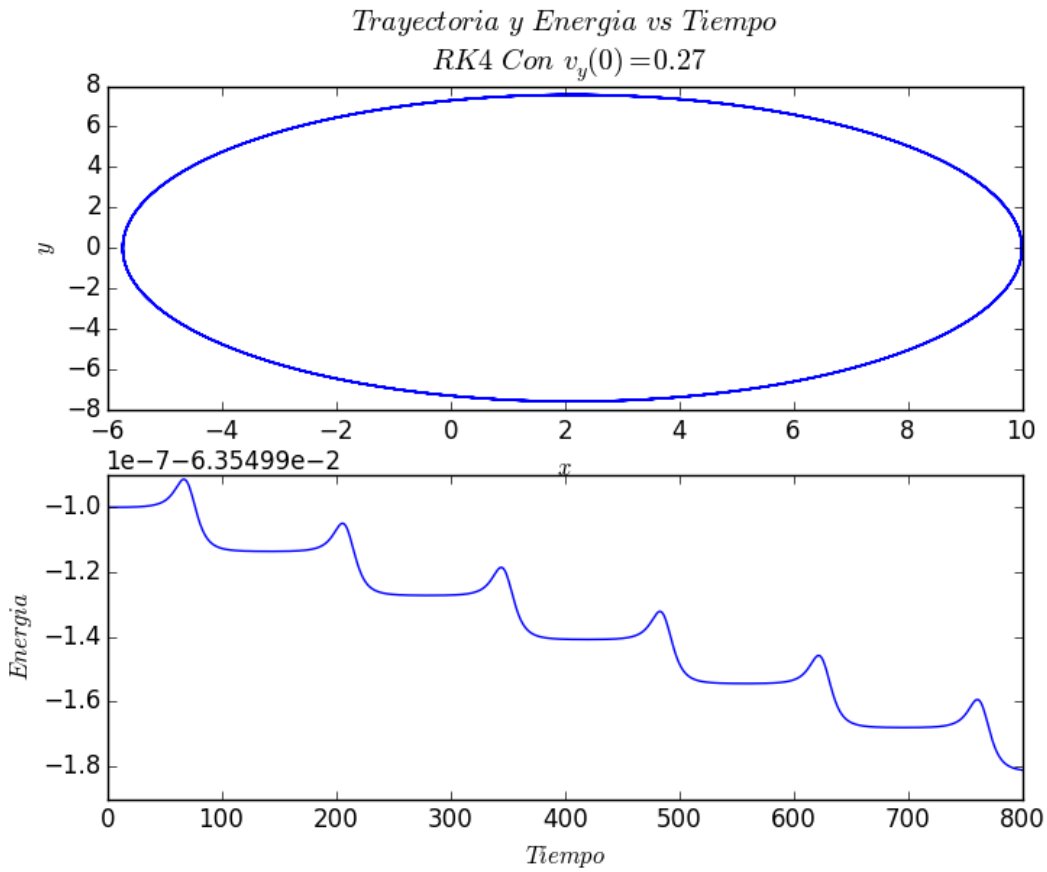

Se puede apreciar que la energía va ‘aumentando’ lo que demuestra que este método no la conserva. La órbita empieza a agrandarse a medida que pasa el tiempo, ya que recordemos que parte en x=10

Gracias a las oscilaciones que muestra el grafico energía vs tiempo se puede ver cuantas vueltas da el planeta.

## Runge-Kutta 4

Para este método un buen resultado me dio con pocos pasos (800) y con un tiempo igual a 800 para tener las 5 vueltas. Al parecer, como la discretización depende de los pasos, al tener un ‘h’ muy pequeño, acumula errores en la integración.

El gráfico resultante fue:



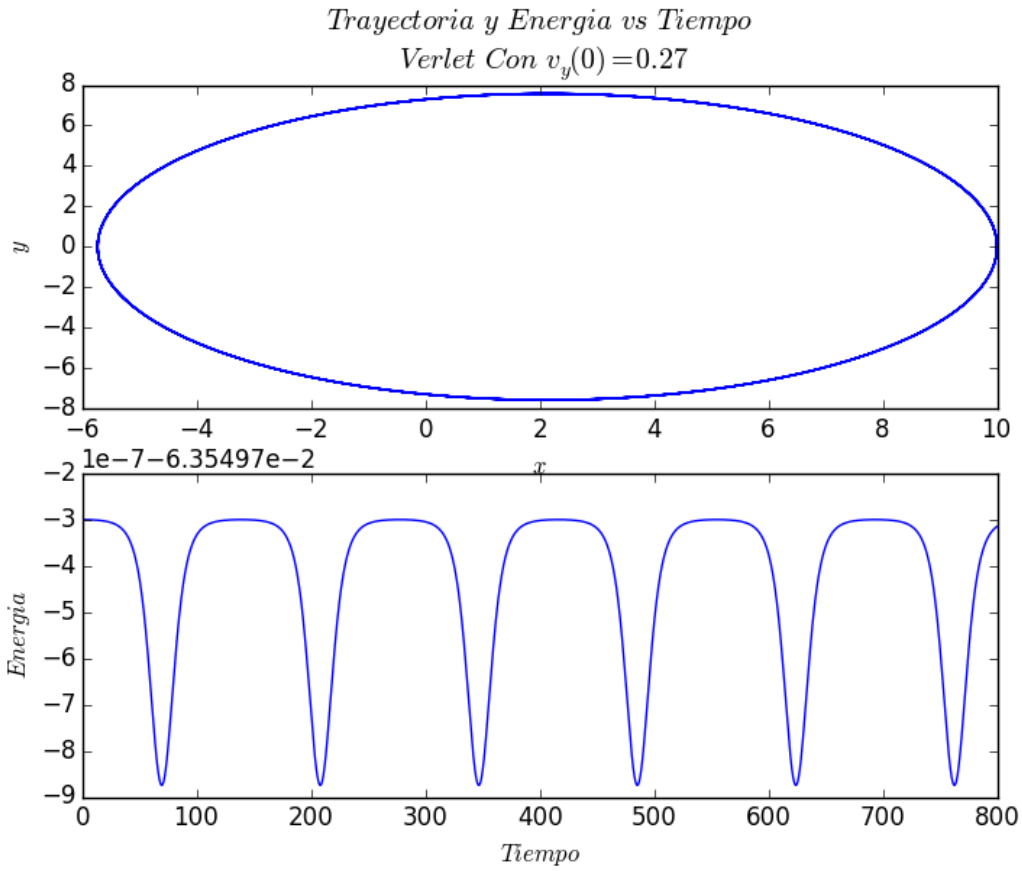

La órbita del planeta se mantiene sin cambios, pero se puede ver que la energía (aun que bastante poco) va disminuyendo a medida que da vueltas.

Parte de 10 pero no llega a -10 en el eje x, al igual que en el caso pasado.

También gracias a las oscilaciones de energía se puede ver cuantas vueltas da el planeta.

## Verlet

Acá también se ve que con tiempo igual a 800 da las 5 vueltas. La cantidad de pasos ahora no importa mucho, se puede variar harto sin cambiar la dinámica (excepto si se trata de pocos pasos, mientras más pasos mayor precisión)



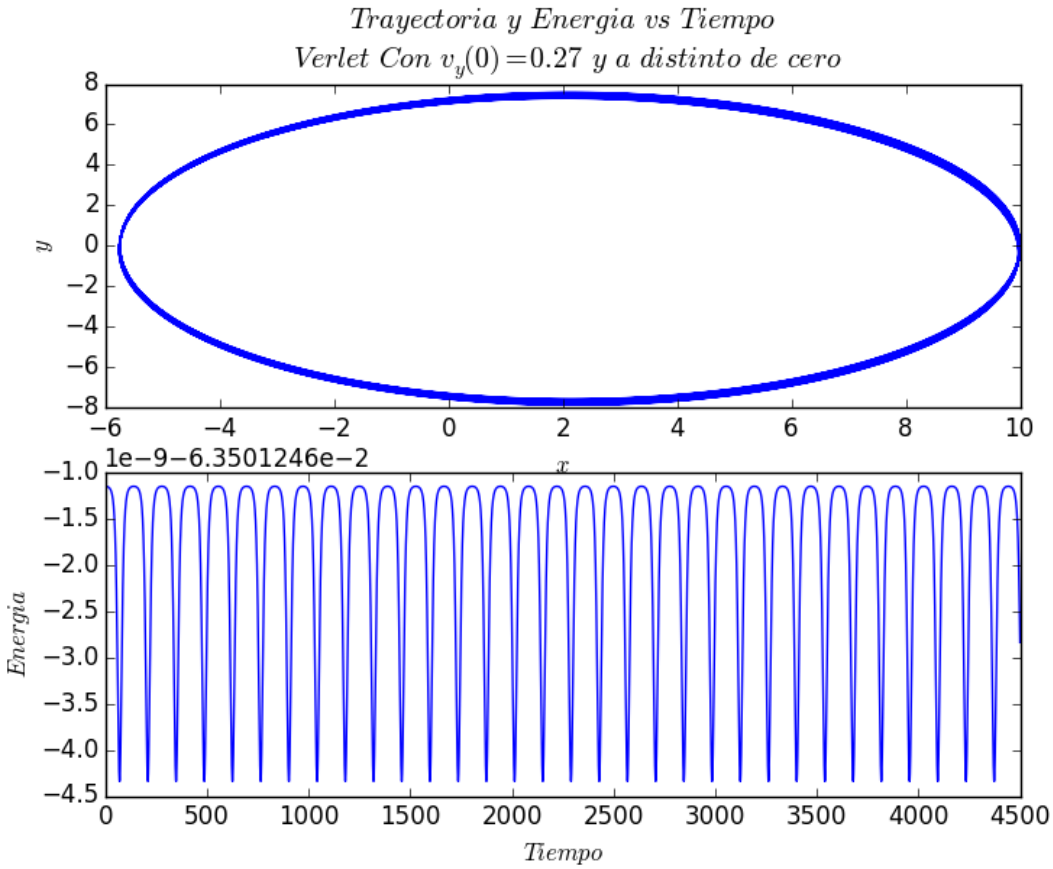

Al igual que con Runge-Kutta 4, la trayectoria se mantiene, pero además en este método se aprecia cómo se mantiene la energía. Aunque se ve que oscila, la diferencia es bastante baja, así que es prácticamente constante  y es aun más fácil ver la cantidad de vueltas que da.

En la **tercera parte** se nos pide integrar solo con el método de Verlet, pero ahora tomando en cuenta alfa en la ecuación de potencial, valor que depende del Rut del estudiante. En mi caso alfa toma el valor $10^{-2.312}$.

Además se pide integrar para al menos 30 vueltas, y obtener también la velocidad angular de precesión, particularmente la posición del perihelio.

De partida se hace exactamente lo mismo que con el experimento pasado pero agregando las modificaciones en el alfa y en las vueltas.

Experimentando note que con tiempo igual a 4500 se cumplen 32 vueltas.




Se puede seguir apreciando la conservación de energía de la misma forma.

Ahora si se nota algo diferente, y es que la órbita se ve más ‘ancha’. La razón de eso es porque la misma se ha ido girando a medida que pasa el tiempo, y es lo que llamamos precesión. Aunque es ligero el cambio, se nota. Tal vez si expandimos el tiempo lo suficiente se note como puede cambiar la orientación del eje mayor hacia el eje x.

Para calcular el perihelio simplemente para cada valor de radio compare con los valores de los 2 puntos adyacentes, identificando los mínimos.

El método implementado en python es:

```
#Ajuste de variables para rellenar con datos
r=[]
rMin=[]
Angulos=[]
Tiempos=[]
W=[]
Wtot=0
for a in range(len(x)):
    r.append(np.sqrt(x[a]**2+y[a]**2))
for a in range(len(x)-2):
    if r[a+1]<r[a] and r[a+1]<r[a+2]:
        rMin.append(r[a+1])
        alfa=np.arctan((abs(y[a+1]))/abs(x[a+1]))
        Angulos.append(alfa)
```

Donde se definen variables vacías para rellenar luego se crea el vector que contiene todos los radios. Luego para cada radio se compara con los más próximos para identificar los mínimos, obteniendo los ‘perihelios’.

Después identificando los x e y de dados radios, con trigonometría se obtienen los ángulos, que junto a la diferencia de tiempo dada por los índices se puede calcular la velocidad con que se gira el eje del perihelio, o sea la velocidad angular de precesión, cuyo resultado final que entrega el programa es 3.021e-05.

Cabe destacar que para obtener una mayor precisión a la hora de obtener estos valores, elevé la cantidad de pasos a 600000, lo que retrasa mas la respuesta del programa, pero entrega mejores resultados ya que no se enreda al calcular con números muy pequeños.