Tarea 3 Métodos Numéricos para la Ciencia y la Ingeniería

Javier Silva Lafaurie
RUT:18.928.657-0

Octubre 2015

1. Introducción

En esta ocación se busca solucionar dos problemas de ecuaciones diferenciales utilizando métodos de integración numéricos. El primer caso se trata de la ecuación del oscilador de Van der Pool:

$$\frac{d^2x}{dt^2} = -kx - \mu(x^2 - a^2)\frac{dx}{dt} \quad (1)$$

Mientras que el segundo consiste en solucionar el sistema de ecuaciones del atractor de lorenz:

$$\begin{aligned} \frac{dx}{ds} &= \sigma(y - x) \\ \frac{dy}{ds} &= x(\rho - z) - y \quad (2) \\ \frac{dz}{ds} &= xy - \beta z \end{aligned}$$

Para parámetos ρ, σ y β conocidos y tales que la solución es caótica.
Se implementa el método de Runge-Kutta de orden 3 para el primer problema. En cuanto que para el segundo se utiliza del método de Runge-Kutta orden 4, esta vez utilizando el algoritmo guardado en la librería *scipy.integrate* de python.

2. Procedimiento

2.1. Oscilador de Van der Pool

Para integrar la primera EDO (1) se procede a realizar un cambio de variables para que la ecuación que depende de tres parámetros, solo dependa de uno. Los cambios de variables usados fueron:

$$\begin{aligned} y = \frac{x}{a} &\Rightarrow dy = \frac{1}{a}dx \\ s = \sqrt{k}t &\Rightarrow ds = \sqrt{k}dt \end{aligned}$$

Reemplazando en la ecuación (1), ésta nos queda finalmente:

$$\frac{d^2y}{ds^2} = -y - \frac{\mu a^2}{\sqrt{k}}(y^2 - 1)\frac{dy}{ds}$$

Tomamos $\mu^* = \frac{\mu a^2}{\sqrt{k}}$, de esta forma la ecuación tiene asociada solo éste parámetro, tomando un valor, en este caso, de 1,657.

Notemos que la ecuación anterior es del tipo:

$$\frac{d^2y}{ds^2} = f(s, y, \frac{dy}{ds})$$

Por lo que se procede a diseñar dos procesos simultáneos para ir obteniendo $\frac{dy}{ds}$ y luego $y(s)$.

Para comenzar a resolver el problema es necesario definir la función f como:

```
def funcion(y,v,mu=1.657):
    return v, -y-mu*((y**2) -1)*v
```

De esta forma será más fácil invocarla. Notar que la función recibe el parámetro $v = \frac{dy}{ds}$. De igual forma entrega dos valores, el primero corresponde a la velocidad y el segundo al valor de la función evaluada en los argumentos entregados.

La idea de este método es dar condiciones iniciales, s_0, y_0 y v_0, y luego iterar, en un intervalo ds hasta un s_n conocido. De esta forma el problema es encontrar los y_{i+1} y v_{i+1} en función de valores ya calculados.

Para resolver numéricamente la ecuación con Runge-Kutta orden 3, se plantea construir funciones para calcular los coeficientes de la resolución ($k1$,$k2$,$k3$). Sin embargo como es una ecuación de segundo orden el proceso de integración es doble, por lo que necesitamos 6 coeficientes. Estos se pueden expresar matemáticamente, para un y_i y v_i como:

$$\begin{gathered} k1 = v_i ds \\ k1' = f(y_i, v_i) ds \\ k2 = (v_i + \tfrac{k1'}{2}) ds \\ k2' = f(y_i + \tfrac{k1}{2}, v_i + \tfrac{k1'}{2}) ds \\ k3 = (v_i - k1' - 2k2') ds \\ k3' = f(y_i - k1 - 2k2, v_i - k1' - 2k2') ds \end{gathered}$$

Notemos que el orden en que se calculan es crucial, pues cada factor necesita valores previos. Finalmente el método de Runge-Kutta nos permite encontrar el y_{i+1} y el v_{i+1} a través de las siquientes relaciones:

$$\begin{aligned} y_{i+1} &= y_i + \tfrac{1}{6}(k1 + 4k2 + k3) \\ v_{i+1} &= v_i + \tfrac{1}{6}(k1' + 4k2' + k3') \end{aligned}$$

Teniendo condiciones iniciales el problema queda solucionado. Tomamos dos casos, el primero con $y_0 = 0{,}1$ y $v_0 = 0$ y el segundo con $y_0 = 4$ y $v_0 = 0$. Para ambos casos se considera un $s_0 = 0$ y un $s_n = 20\pi$, con $n = 500$ el número de veces a integrar.

2.2. Atractor de Lorenz

Nos encontramos frente al problema del sistema de ecuaciones (2). Como se puede observar este corresponde un conjunto de EDOs acopladas. La solución depende de los parámetros σ, β y ρ, para los cuales tomamos valores de:

$$\begin{aligned} \sigma &= 10 \\ \beta &= 8/3 \\ \rho &= 28 \end{aligned}$$

Se toman estos valores debido a que ya se ha comprobado que corresponden a la solución caótica del problema.

Para solucionar este problema se usa el método de Runge-Kutta orden 4, el cual es parecido al método anterior, pero en vez de tres coeficientes, acá se calculan 4. Sin embargo para este problema se utiliza el módulo *ode* de la librería *scipy.integrate* de python:

```
from scipy.integrate import ode
```

Luego se define la función
textitsistema en el código, la cual devuelve las ecuaciones del sistema (2) dado un $w = (x, y, z)$ o variables del problema, un parámetro t del cual dependen las variables, y otras constantes del sistema, en este caso σ, β y ρ:

```
def sistema(t,w,sigma=sigma,beta=beta,rho=rho):
    x, y, z =w
    return [sigma*(y-x), x*(rho-z)-y, x*y-beta*z]
```

Al igual que con el método pasado, necesitamos condiciones iniciales, las cuales para $t_0 = 0$ tomamos como $(x_0, y_0, z_0) = (10, 20, 30)$. Ahora solo es necesario invocar a la función *ode* y darle el método de integración requerido y las condiciones iniciales:

```
w0= x0, y0, z0
r=ode(sistema)
r.set_integrator('dopri5')
r.set_initial_value(w0)
```

Notemos que el comando *'dropi5'* le indica al módulo *ode* que queremos solucionar las EDOs usando Runge-Kutta orden 4. A continuación es necesario crear arreglos donde guardemos los valores que vaya obteniendo el método. Para esto nos damos un tiempo final de integración de $t_f = 40$ y creamos un vector de $n = 5000$ valores equispaciados en el intervalo $[t_0, t_f]$. En cuanto a los valores de (x, y, z) creamos tres arreglos, también de 5000 valores, pero solo con 1s, los cuales se irán cambiando a medida que el programa integre. La forma para obtener estos valores es iterando como se indica:

```
for i in range(0,len(t)):
    r.integrate(t[i])
    x[i], y[i], z[i] = r.y
```

Ahora que se conocen los valores de $(x(t), y(t), z(t))$ para t en $[t_0, t_f]$, se procede a graficar estos en un espacio tridimencional para esto necesitamos importar:

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

Con esto se implementan los comandos del archivo *3D.py* para graficar en 3-dimensiones, estos son:

```
fig=plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot(x, y, z)
plt.show()
```

Con la primera linea guardamos una nueva ventana para graficar, con la segunda declaramos que sea un gráfico en tres dimensiones, con la tercera se grafica en la ventana y finalmente con la cuarta se muestra el gráfico.

3. Resultados

3.1. Oscilador de Van der Pool

En el primer problema se grafica la trayectoria $y(s)$ en función de s, como el del espacio de fase ($\frac{dy}{ds}$ en función de $y(s)$) para condiciones iniciales de $y_0 = 0{,}1$ y $v_0 = 0$, estos se pueden apreciar en la Figura 1 y Figura 2.

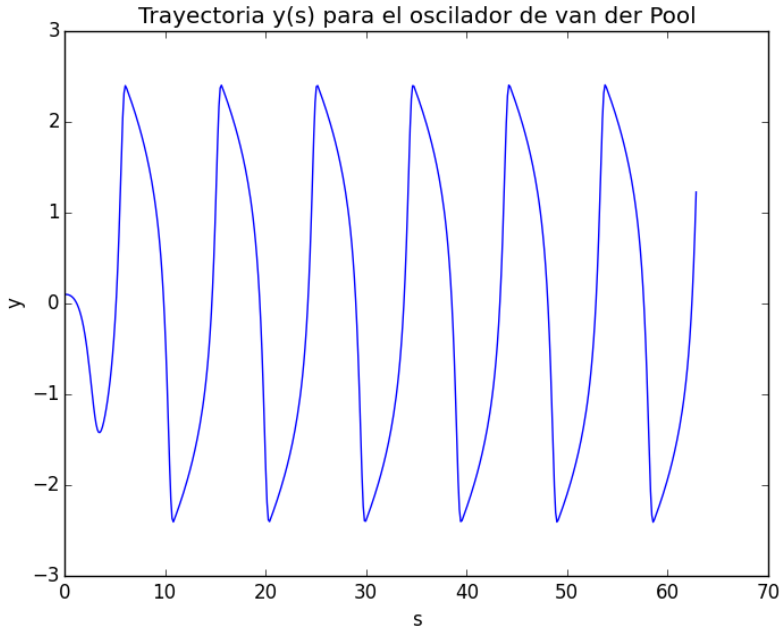

Figura 1. Trayectoria oscilador de Var der Pool con condiciones iniciales $y(0) = 0{,}1$ y $(\frac{dy}{ds})_{s=0} = 0$

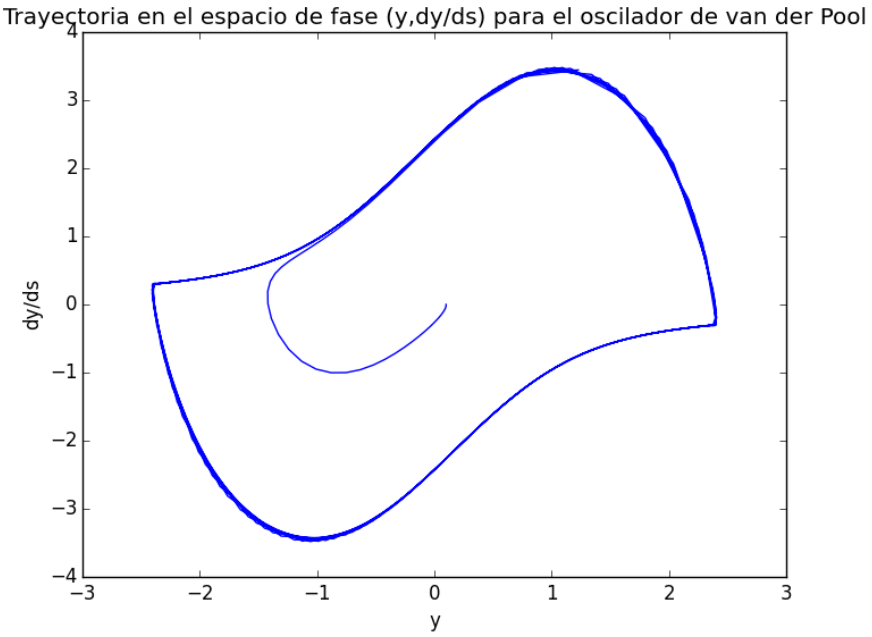

Figura 2. Espacio de fase del oscilador de Var der Pool con condiciones iniciales $y(0) = 0{,}1$ y $(\frac{dy}{ds})_{s=0} = 0$

Tomando el oscilador de Van der Pool, pero ahora con condiciones iniciales de $y_0 = 4$ y $v_0 = 0$ obtenemos los gráficos de la Figura 3 y Figura 4, correspondientes a la trayectoria y al espacio de fase reespectivamente.

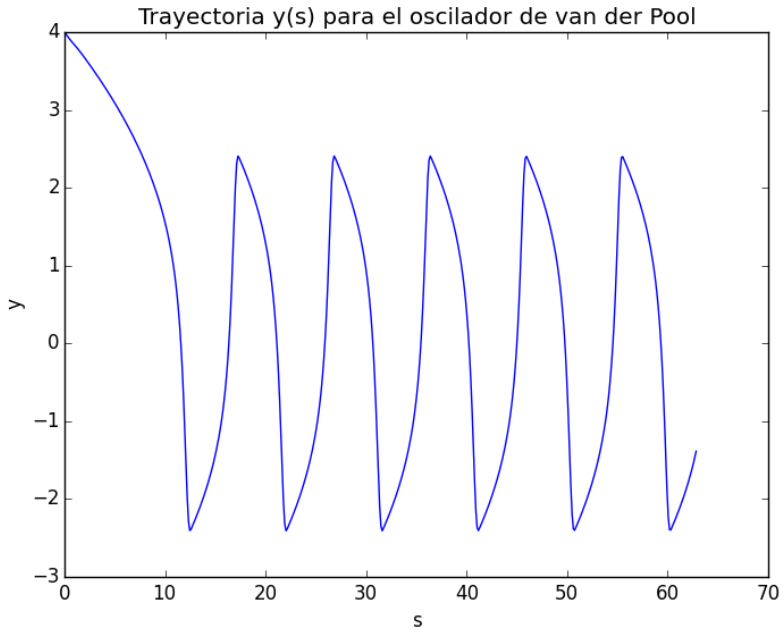

Figura 3. Trayectoria oscilador de Var der Pool con condiciones iniciales $y(0) = 4$ y $(\frac{dy}{ds})_{s=0} = 0$

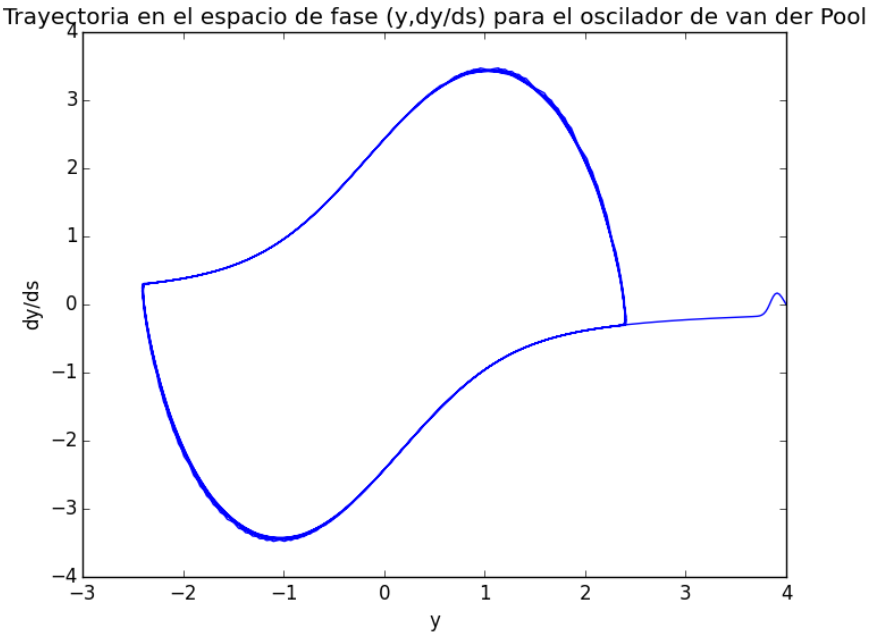

Figura 4. Espacio de fase del oscilador de Var der Pool con condiciones iniciales $y(0) = 4$ y $(\frac{dy}{ds})_{s=0} = 0$

3.2. Atractor de Lorenz

Para el segundo problema se grafica la trayectoria $(x(t), y(t), z(t))$ en un espacio tridimensional como se observa en la Figura 5.

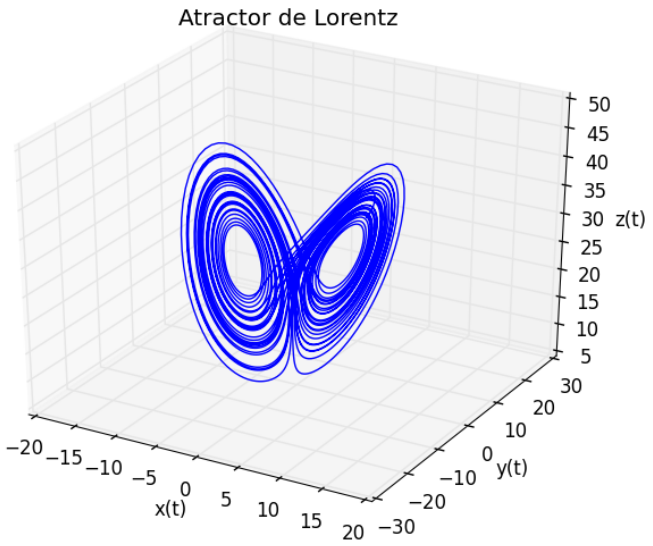

Figura 5. Trayectoria del problema del atractor de Lorenz con condiciones iniciales $x_0 = 10$, $y_0 = 20$ y $z_0 = 30$. Los valores de los parámetros corresponden a $\sigma = 10$, $\beta = 8/3$ y $\rho = 28$

4. Conclusiones

Dentro de lo obtenido anteriormente hacemos notar que tanto los gráficos como las rutinas creadas y otras prediseñadas, hacen de Python una plataforma ideal para muchas aplicaciones dentro del campo científico.

Para el oscilador de Van der Pool notamos que el coeficiente μ^* inyecta y quita energía, dependiendo de la posición en donde se encuentre la partícula. Esto se puede ver en las figuras 2 y 4, donde para velocidades iniciales

nulas, pero posiciones iniciales diferentes, podemos ver como ambas tienden al mismo diagrama de fase en el espacio (y, v). De esto se concluye que las posiciones fuera de este grafo son puntos inestables y que el sistema definido tenderá a la solución estable aunque para esto deba inyectar o quitar energía al sistema. Ésto se demuestra también con las trayectorias en las Figuras 1 y 3, debido a que aunque tengan posiciones iniciales diferentes ambas tienden a oscilar con la misma amplitud y período.

Para el atractor de Lorenz, en cambio, si variamos las condiciones iniciales va a cambiar notariamente la trayectoria. Sin embargo se tendrá la misma forma mostrada en la Figura 5. Esto es debido a que los parámetros σ, β y ρ estan elegidos sabiendo que darán la solución caótica del problema, pero además son estos que determinan la forma de la trayectoria.

Finalmente se concluye esta tarea sin mayores dificultades, a exepción de la implementación de los métodos de resolución de EDOs en python, los cuales tuvieron que adaptarse en comparación a como se presentan en la librería.