Métodos Numéricos para la Ciencia e Ingeniería
Informe Tarea

Benjamín Guerrero
18.862.370-0
8 de Octubre, 2015

1. Pregunta 1

1.1. Introducción

La pregunta pide integrar la ecuación de movimiento del oscilador de van der Pol, que es la siguiente:

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

Haciendo cambios de variable, se llega a la siguiente ecuación, más sencilla:

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

Primero, se pide determinar cuáles fueron los cambios de variables usados.

Luego, siendo μ^* los 3 dígitos del RUT antes del guión, se pide usar el método Runge Kutta de orden 3, usando un algoritmo creado por el usuario, con los siguientes casos base:

$$1)\frac{dy}{ds} = 0; y = 0.1$$

$$2)\frac{dy}{ds} = 0; y = 4.0$$

Finalmente, se pide graficar y en función de s, y $\frac{dy}{ds}$ en función de y.

1.2. Procedimiento

Primero, antes de crear el programa, vamos a hacer unos cambios de variable para simplificar la ecuación. Estos son:

$$y = \frac{x}{a}, \qquad s = t\sqrt{k}, \qquad \mu^* = \frac{\mu a^2}{\sqrt{k}}$$

Para el ejercicio, el valor de μ^* será 1.370.

Ahora, se pide usar RK3. Este método es el siguiente:

Sea $f(x_n, y_n)$ la función que equivale a la EDO de la función $y(x)$. Sea $h = \Delta x$.

Se definen:

$$k_1 = h * f(x_n, y_n)$$

$$k_2 = h * f\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right)$$

$$k_3 = h * f(x_n + h, y_n - k_1 + 2k_2)$$

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 4k_2 + k_3) + O(h^4)$$

Para este sistema, se aproxima $O(h^4) \approx 0$.

Como la ecuación es un EDO de segundo orden, se debe aplicar RK3 a y , y a $\frac{dy}{ds}$. Para esto, se define una variable de nombre v tal que $v = \frac{dy}{ds}$, y $\frac{dv}{ds} = \frac{d^2y}{ds^2}$.

Ahora, se crea un programa de nombre *RK3Pol.py*, que realizará lo pedido. Primero, se importan las librerías necesarias. Luego, se define la función adhoc *f*, que recibe (y, v) y retorna $(v, -y - \mu^*(y^2 - 1)v)$, sus derivadas según la ecuación. Para sacar los k, se definen 3 funciones de tipo *get_k*, que reciben (y_n, v_n, h, f) y entregan los valores de k_1, k_2, y k_3 para los parámetros y, v. Usando esos valores, se define *RK3*, que recibe (y_n, v_n, h, f) y entrega los valores de y_{n+1} y de v_{n+1}.

Con las funciones definidas, se ven las condiciones iniciales. Se define h como 0,1. Considerando que el período definido es 20π, se aproximan la cantidad de pasos (n) a $\frac{20*3,14}{0,1} = 628$. Luego, para cada condición, se realiza lo siguiente:

Se crean 2 arrays de n ceros usando *numpy.zeros(n).* Uno representa y, y el otro representa v. Se determina el primer término de los array como la condición inicial entregada. Luego, usando un for loop (*for i in range(1,n))*, se aplica RK3 para determinar los valores de todos los y_i y v_i(insertando los valores en los array). Una vez hecho esto, se pueden graficar los valores dados para ambas condiciones usando *matplotlib.pyplot*. Estos se guardan en 2 archivos .png.

Se usa color rojo para la primera condición, y azul para la segunda.

1.3. Resultados

Al usar RK3, y graficar $\frac{dy}{ds}$ en función de y, se obtiene lo siguiente:

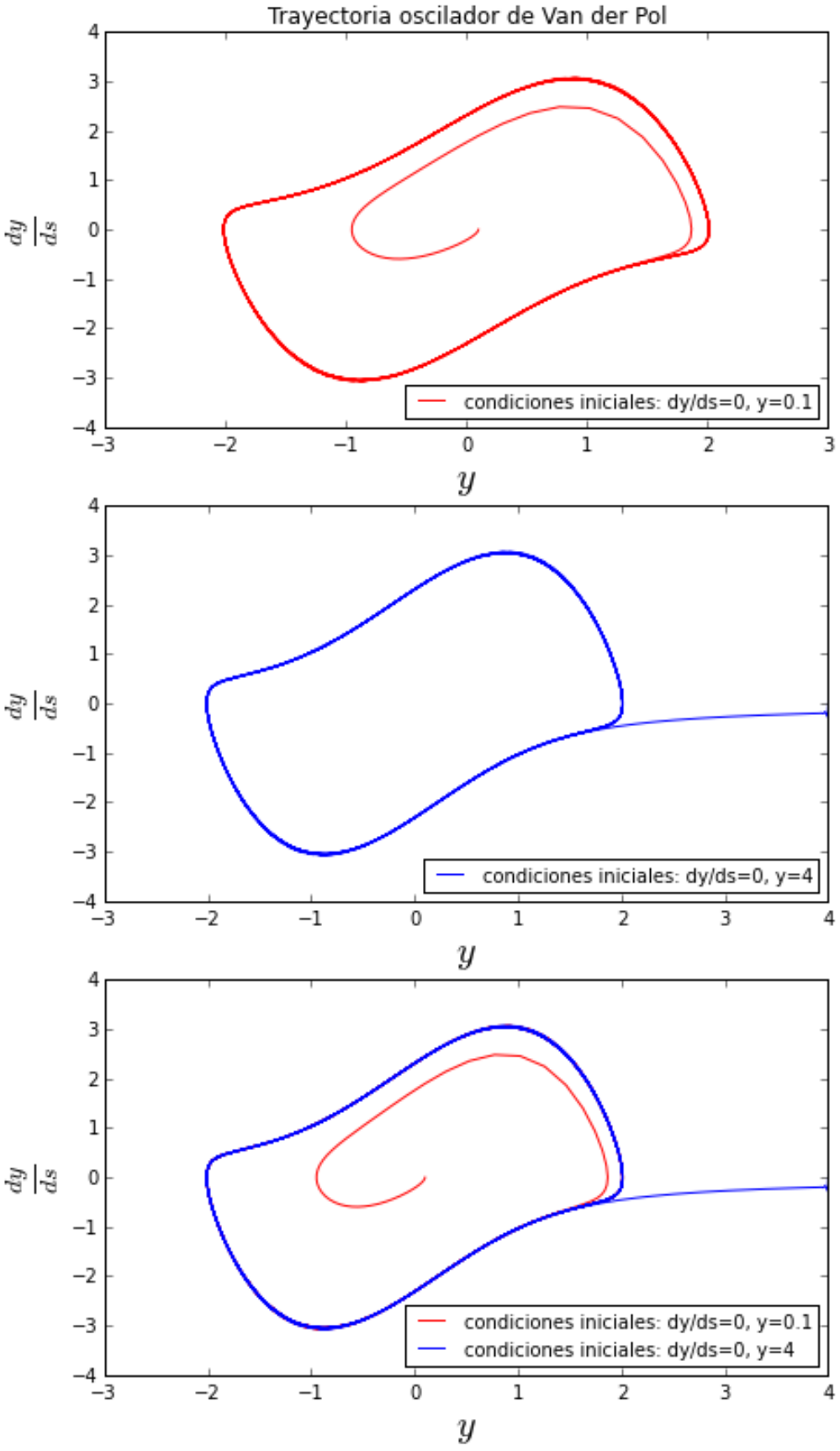

Fig 1: Trayectoria oscilador de Van der Pol

Y al graficar y en función de s, se tiene:

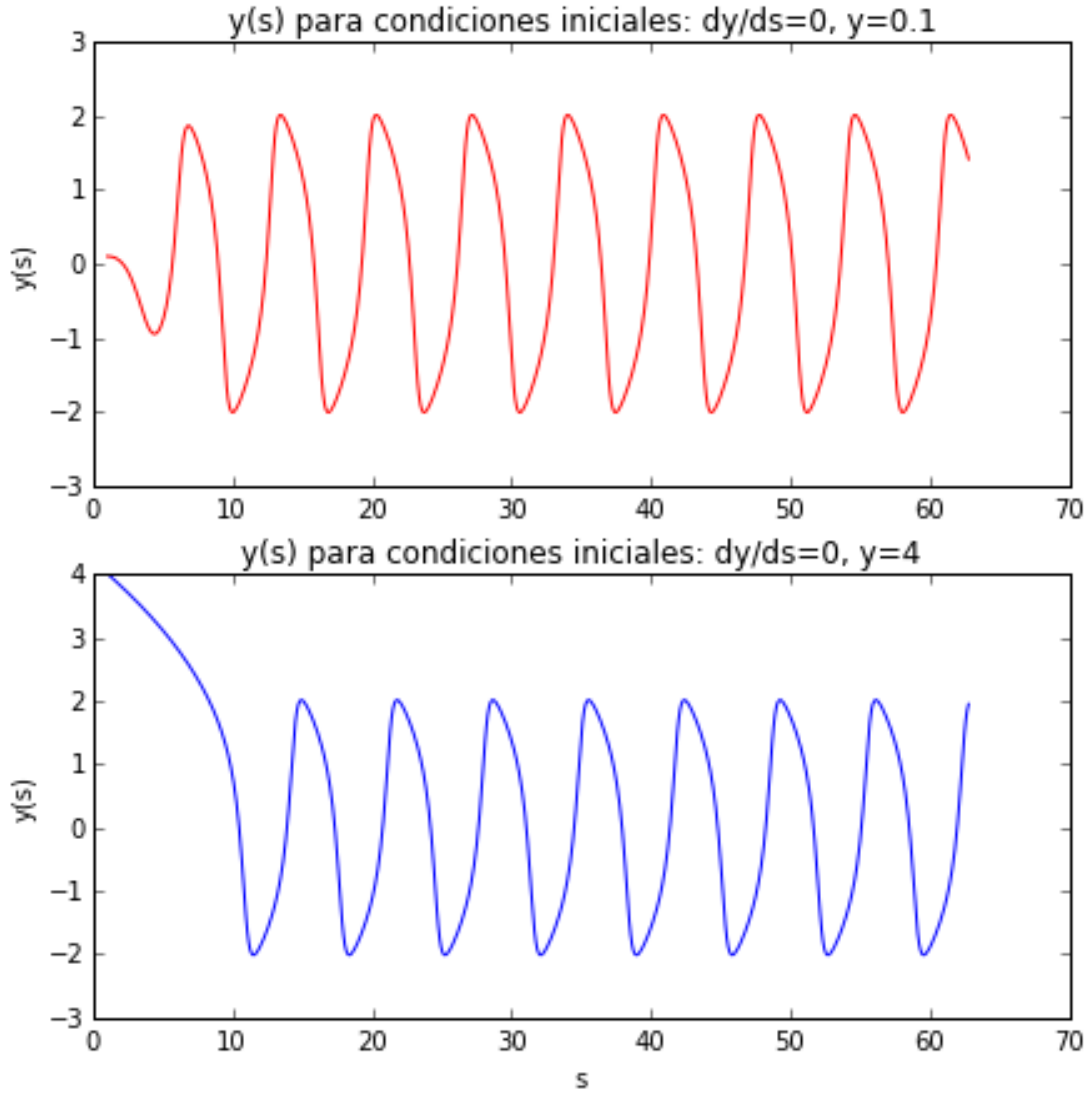

Fig 2: Gráfico y v/s s.

2. Pregunta 2

2.1. Introducción

Se pide integrar las ecuaciones del sistema de Lorentz:

$$\frac{dx}{dt} = \sigma(y - x)$$

$$\frac{dy}{dt} = x(\rho - z) - y$$

$$\frac{dz}{dt} = xy - \beta z$$

Con $\sigma = 10, \rho = 28$, y $\beta = \frac{8}{3}$. Aquí, se pide usar Runge Kutta de orden 4, pero no se necesita implementarlo, se puede usar un algoritmo disponible en *scipy.integrate*, por ejemplo. Las condiciones iniciales son a elección.
Una vez integrado, se pide graficar en 3D la solución $(x(t), y(t), z(t))$. El repositorio incluye un programa llamado *3D.py*, que crea un gráfico en 3D simple, como guía de ayuda.

2.2. Procedimiento

Primero, se crea un programa de nombre *RK4Lorentz.py* que realizará lo pedido. En éste primero se importa lo siguiente:

Se importan, además de *numpy* y *matplotlib.pyplot*, las librerías *scipy.integrate.ode* y *mpl_toolkits.mplot3d.Axes3D*.

Luego, tal como en el caso anterior, se define una función adhoc *f* que retorna las derivadas de (x,y,z), según las ecuaciones de Lorentz y las constantes dadas. Luego, de define el tiempo final (100), el número de pasos (10000), y el valor h=($\frac{t_f}{N_{steps}}$).

Se crean 3 arrays de (*N_steps+1*) ceros usando *numpy.zeros(N_steps+1)*. Estos representan x,y,z. Ahora, se definen las condiciones iniciales como:

$$(x_o, y_o, z_o) = (1,1,1)$$

Luego, se define t como *t=numpy.linspace(0, 10, N_steps+1)*.

Luego, llamando a *solver = ode(f)*, se define *solver.set_integrator('dopri5', atol=1E-6, rtol=1E-4)*. Esto determina que se usará RK4, con una tolerancia relativa de 10^{-4}, y una tolerancia absoluta de 10^{-6}.

Con *solver.set_initial_value(xyz_inicial)* se determina la c. inicial. Después, usando un while loop (mientras la función funcione, y el tiempo y los pasos no lleguen a los máximos), se obtienen los valores de (x, y, z) que se necesitan.

Finalmente, usando el archivo *3D.py* como guía, se grafican los parámetros en un gráfico 3D, que se guarda en un archivo llamado *Lorentz.png*.

2.3. Resultados

Una vez resuelto el programa, con tiempo final 100, el gráfico resultante es el siguiente:

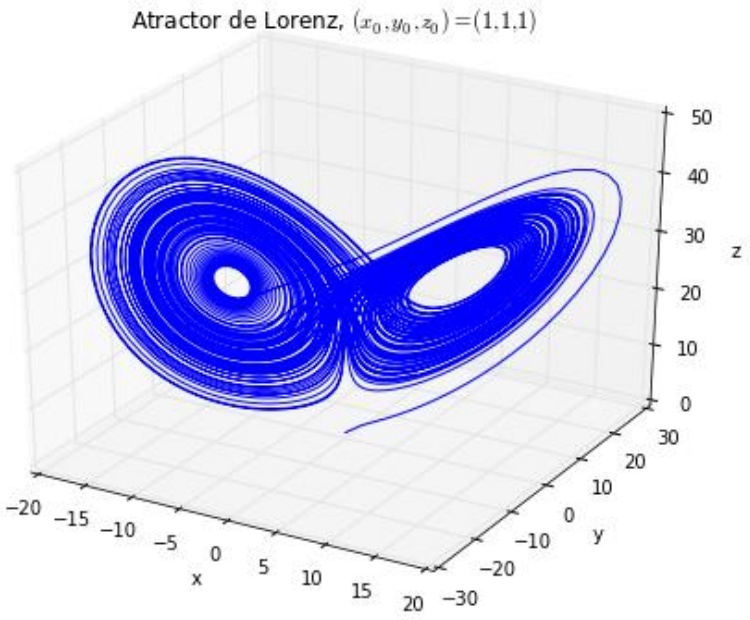

Fig. 3: Trayectoria del Atractor de Lorentz.

3. Conclusiones

A partir de la figura 2, se puede observar que, si importar la condición inicial, el valor *y* va a volverse periódico y va a oscilar entre -2 y 2. Esto es reflejado en la figura 1, en el que el valor de $\frac{dy}{ds}$ cae en el mismo loop sin importar las condiciones iniciales de *y*.

En la figura 3, se aprecia que la trayectoria en el espacio del atractor da vueltas en torno a 2 puntos diferentes, variando entre los 2 puntos cada cierto tiempo. Se puede apreciar la condición inicial, en que la trayectoria es "absorbida" por uno de los "pozos", y entra a dar vueltas. Así, es posible deducir que la forma se va a dar sin importar las condiciones iniciales.

Se concluye, también, que el método de Runge Kutta funciona para resolver EDOs de forma numérica. El error asociado al método no afectó los ejercicios (ya que no había que compararlos a soluciones analíticas).