Métodos numéricos para la Ciencia e Ingeniería
Tarea 3

Felipe Toledo B.
Rut: 17.519.820-2

October 8, 2015

1 Problema 1

1.1 Introducción y objetivos

El objetivo de esta experiencia es integrar numéricamente la ecuación diferencial del Oscilador de Van der Pool, implementando el método de Runge Kutta de orden 3. La ecuación original se observa en (1), con k una constante elástica, a una constante de largo y mu un coeficiente de roce.

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

1.2 Metodología

Para simplificar el problema, primero se reduce la ecuación usando el cambio de variables $\frac{x}{a} = y$, $\sqrt{k}t = s$. Tras esto se obtiene (2), con $\mu^* = \frac{\mu a^2}{\sqrt{k}}$.

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

Como existe una derivada de primer orden a la derecha de la ecuación, para poder resolver la EDO usando el método de Runge Kutta es necesario expresarla como el siguiente sistema de ecuaciones:

$$\begin{cases} \dfrac{dy}{ds} = v \\ \\ \dfrac{dv}{ds} = -y - \mu^*(y^2 - 1)v \end{cases} \tag{3}$$

Definiendo las funciones $\frac{dy}{ds} = f(v)$ y $\frac{dv}{ds} = g(y, v)$, se pueden calcular los valores de 'y' y 'v' para todo 's' (coordenada temporal) usando el algoritmo (4).

$$
\begin{aligned}
y_{n+1} &= y_n + \tfrac{1}{6}(k_1 + 4k_2 + k_3) \\
v_{n+1} &= v_n + \tfrac{1}{6}(l_1 + 4l_2 + l_3)
\end{aligned}
\tag{4}
$$

En (5) se puede observar cómo se calcula cada constante k_i, l_i, donde h es el parámetro de paso temporal entre cada punto calculado.

$$
\begin{aligned}
k_1 &= hf(v_n) \\
l_1 &= hg(y_n, v_n) \\
\\
k_2 &= hf(v_n + \tfrac{1}{2}l_1) \\
l_2 &= hg(y_n + \tfrac{1}{2}k_1, v_n + \tfrac{1}{2}l_1) \\
\\
k_3 &= hf(v_n - l_1 - 2l_2) \\
l_3 &= hg(y_n - k_1 - 2k_2, v_n - l_1 - 2l_2)
\end{aligned}
\tag{5}
$$

Los métodos descritos se encuentran implementados en la clase *oscilador_van_der_pool.py*.

1.3 Resultados

Se realizaron dos resoluciones con parámetros iguales pero condiciones iniciales distintas. Tanto los resultados como las figuras pueden obtenerse ejecutando el script *P1.py*. El valor de los parámetros es:

- $mu^* = 1.820$
- $h = 10^-4$
- $T_{final} = 20\pi$

1.3.1 Caso 1

En este caso las condiciones iniciales son:

$$
\frac{dy}{ds} = 0; y = 0.1 \tag{6}
$$

Los resultados puede verse en las Figuras 1 y 2.

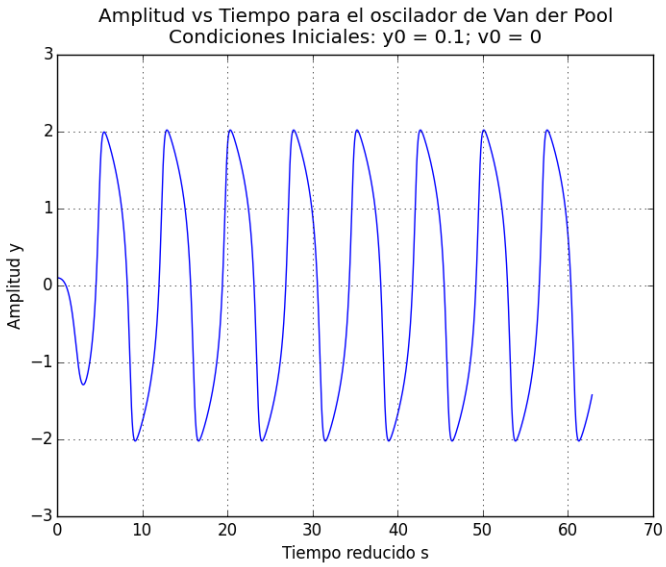

Figure 1: Amplitud vs Tiempo para el oscilador de Van der Pool.

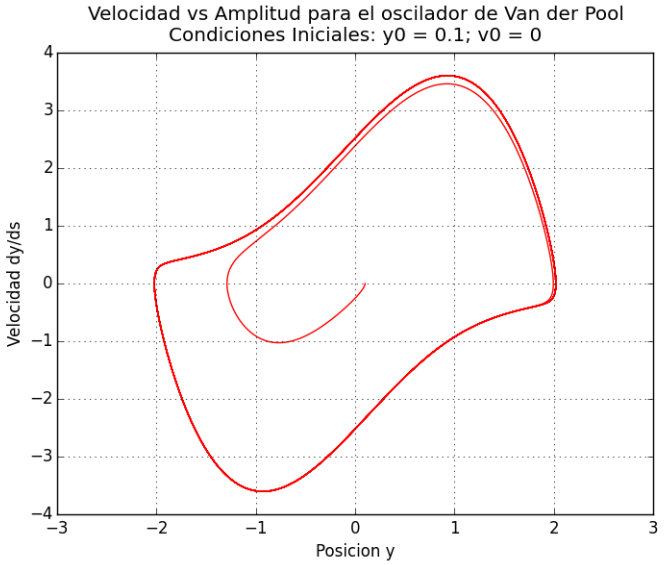

Figure 2: Velocidad vs Posición para el oscilador de Van der Pool.

1.3.2 Caso 2

Los resultados puede verse en las Figuras 3 y 4. Las condiciones iniciales son:

$$\frac{dy}{ds} = 0; y = 4.0 \tag{7}$$

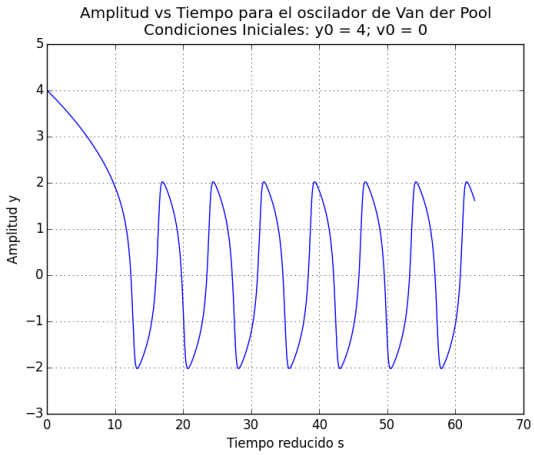

Figure 3: Amplitud vs Tiempo para el oscilador de Van der Pool.

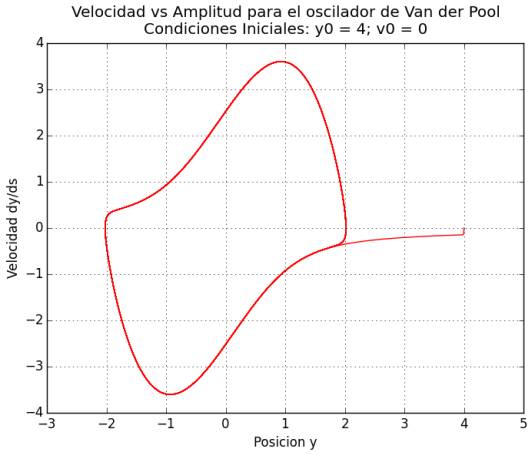

Figure 4: Velocidad vs Posición para el oscilador de Van der Pool.

1.4 Conclusiones P1

Se puede concluir que el oscilador de Van der Pool es un sistema que siempre tiende a un mismo estado estable, al menos para condiciones de velocidad inicial nula.
El paso de Runge Kutta demostró ser apropiado ya que provee de una buena resolución temporal con tiempos de cómputo bajos. En caso de que se quieran procesar intervalos de tiempo mayores podría ser retribuyente dedicar esfuerzos a refinar su tamaño para integrar la ecuación más rápidamente.

2 Problema 2

2.1 Introducción y Metodología

El problema consiste en calcular la trayectoria de una partícula governada por las ecuaciones del Atractor de Lorenz, presentadas a continuación:

$$\frac{dx}{ds} = \sigma(y - x)$$
$$\frac{dy}{ds} = x(\rho - z) - y$$
$$\frac{dz}{ds} = xy - \beta z$$

Con los parámetros $\sigma = 10$, $\beta = 8/3$ y $\rho = 28$. La particularidad de estas ecuaciones y parámetros es que posee soluciones caóticas para las trayectorias. Se resuelven directamente las ecuaciones utilizando la función *odeint* de la librería *scipy.integrate*. El detalle de la implementación puede verse en la clase *atractor_de_lorenz.py*, la cual reutiliza muchas funciones de la clase *oscilador_van_der_pool.py* implementada en el problema 1.

2.2 Resultados

Se simularon dos partículas con condiciones iniciales levemente distintas, descritas a continuación:
Partícula 1:

- $x(t = 0) = 1$
- $y(t = 0) = 1$
- $z(t = 0) = 1$

Partícula 2:

- $x(t = 0) = 1.01$
- $y(t = 0) = 1$
- $z(t = 0) = 1$

Oteniéndose el gráfico de la Figura 6.

2.3 Conclusiones P2

En este caso se observa como una pequeña desviación en las condiciones iniciales genera trayectorias distintas donde se ve que se alcanzan diferencias de hasta 3 órdenes de magnitud respecto a la desviación original (la diferencia inicial de un centímetro llega a alcanzar cerca de diez metros en uno de los máximos).

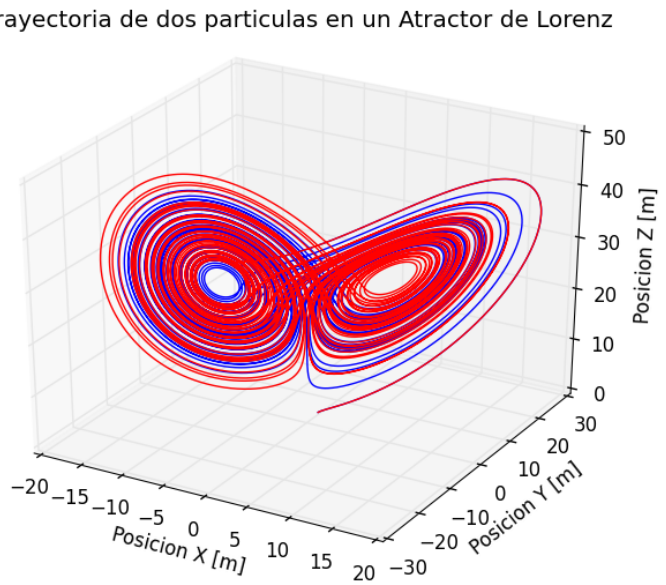

Figure 5: Trayectoria de dos partículas en el Atractor de Lorenz. La curva azul corresponde a la partícula 1 y la roja a la partícula 2.

Esto es característico de los sistemas caóticos, donde pequeñas variaciones en las condiciones iniciales pueden generar sistemas radicalmente distintos.
Desde el punto de vista de la implementación, se observa que la clase escrita para este problema utiliza muchas funciones replicadas directamente desde la solución del P1. Debido a ello se vuelve interesante el mejorar el uso de la herencia para futuros programas similares, lo que permitirá ahorrar código al facilitar la reutilizaci
on. En particular para este caso, cualquier programa de cálculo de trayectorias debiese tener, en general, un tratamiento similar al de las soluciones desarrolladas.