Informe Tarea 3: Métodos Numéricos

Ignacio Andrés Sánchez Barraza
Rut: 18933808-2

October 5, 2015

1 Pregunta 1

1.1 Introducción

- El oscilador de Van der Pol fue propuesto para describir la dinámica de algunos circuitos eléctricos, cuya ecuación de movimiento corresponde a la siguiente:

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

donde k es la constante elástica y μ es un coeficiente de roce. Si $|x| > a$ el roce amortigua el movimiento, pero si $|x| < a$ el roce inyecta energía. Dada esta ecuación se puede hacer el cambio de variable $x(t) = ay(s) = ay(t\sqrt{k})$ con lo que $\frac{dx}{dt} = a\frac{dy}{ds}\frac{ds}{dt} = a\frac{dy}{ds}\sqrt{k}$ y $\frac{d^2x}{dt^2} = a\frac{d}{dt}(\frac{dy}{ds}\frac{ds}{dt}) = a\frac{d}{dt}(\frac{dy}{ds}\sqrt{k}) = a(\frac{d^2y}{ds^2}\frac{ds}{dt}\sqrt{k}) = ak\frac{d^2y}{ds^2}$ se puede convertir la ecuación a:

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

con lo cual ahora la ecuación sólo depende de un parámetro, $\mu^* = a^2\mu\frac{\sqrt{k}}{k}$.
Se pide integrar la ecuación de movimiento usando el método de Runge-Kutta de orden 3 usando $\mu^* = 1.RRR$, donde RRR son los 3 últimos dígitos del RUT antes del guión y con condiciones iniciales:

$$1)\frac{dy}{ds}|_{t=t_0} = 0; y(t_0) = 0.1$$
$$2)\frac{dy}{ds}|_{t=t_0} = 0; y(t_0) = 4.0$$

Además de ello se pide graficar la solución $y(s)$ vs s y dy/ds vs $y(s)$.

1.2 Procedimiento

- Primeramente se definió entonces un algoritmo que realizara el método de Runge-Kutta de orden 3 para resolver una EDO de orden 2 a través dle siguiente código:

```
def rk3(f,h,CT0,CI1,CI2):
    v0=CI1
    x0=CI2
    t0=CT0
    k1=h*v0
    l1=h*f(x0,v0,t0)
    k2=h*(v0+l1/2.0)
    l2=h*f(x0+k1/2.0,v0+l1/2.0,t0+h/2.0)
    k3=h*(v0-l1+2.0*l2)
    l3=h*f(x0-k1+2.0*k2,v0-l1+2.0*l2,t0+h)
    xn=x0+(k1+4*k2+k3)/6.0
    vn=v0+(l1+4*l2+l3)/6.0
    return xn,vn
```

Esta función recibía como argumento una función de la forma $\frac{dx}{dt} = f(x,t)$, un paso de tiempo h (que en este código sera utilizado como $h = 0.1$), y condiciones iniciales sobre tiempo, derivada y la posición ($CT0$, $CT1$ y $CT2$ respectivamente). Dicha función entonces consistía en aplicar el método numérico nombrado para la EDO anterior, pero con un cambio de variable se convierte en un sistema de ecuaciones compuesto por:

$$3)\frac{dy}{ds} = v$$
$$4)\frac{dv}{ds} = -y - \mu^*(y^2 - 1)v$$

Con este sistema de EDO's de primer orden, entonces se procedió a definir las variables auxiliares que permitirían realizar el algoritmo, ie., k_i y l_i, teniendo estas últimas dependencias entrelazadas entre k_i y l_i y con las variables para un $i-1$. Ahora, como el método de Runge-Kutta de orden 3 esta descrito por la siguiente ecuacion en general:

$$5)y_{n+1} = y_n + h\sum_{i=1}^{3} b_i k_i \rightarrow (correspondientealasolucioneneltiempot_{n+1})$$
$$6)k_i = f(tn + c_i h, y_n + h\sum_{j}^{3} a_{ij}k_j)$$

y dada la matriz Butcher tableau:

$$FormaGeneral \rightarrow \begin{array}{c|ccc} c_1 & a_{11} & a_{12} & a_{13} \\ c_2 & a_{21} & a_{23} & a_{23} \\ c_3 & a_{31} & a_{32} & a_{33} \\ \hline & b_1 & b_2 & b_3 \end{array}$$

$$Runge-Kutta\ orden\ 3 \rightarrow \begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 \\ 1 & -1 & 2 & 0 \\ \hline & 1/6 & 2/3 & 1/6 \end{array}$$

entonces:

$$k_i = h(v_0 + \tfrac{1}{2}l_{i-1})$$
$$l_i = hf(x0 + \tfrac{1}{2}k_{i-1}, v_0 + \tfrac{1}{2}l_{i-1}, t_0 + \tfrac{1}{2}h)$$

y se procedió a crear el cuerpo del código para dar solución al problema.

- Terminada la definición de la función $rk3$, se procede entonces a integrar hasta el número de periodos pedidos ($T = 20\pi$) con un paso de tiempo $h = 0.1$ y $\mu^* = 1.808$:

```
uast=1.808
v0=0
y0=0.1
h=0.1
y=[y0]
v=[v0]
n=600 #aprox. 10 periodos (20 pi aprox. 60 => n*h=600*0.1=60)
for i in range(n):
    y0=(rk3(lambda y,v,t:-y-uast*(y**2-1)*v,h,0,v0,y0))[0]
    v0=(rk3(lambda y,v,t:-y-uast*(y**2-1)*v,h,0,v01,y01))[1]
    y.append(y0)
    v.append(v0)
```

1.3 Resultados

- Hecho entonces el algoritmo, los gráficos obtenidos para las condiciones (1) y (2) respectivamente fueron:

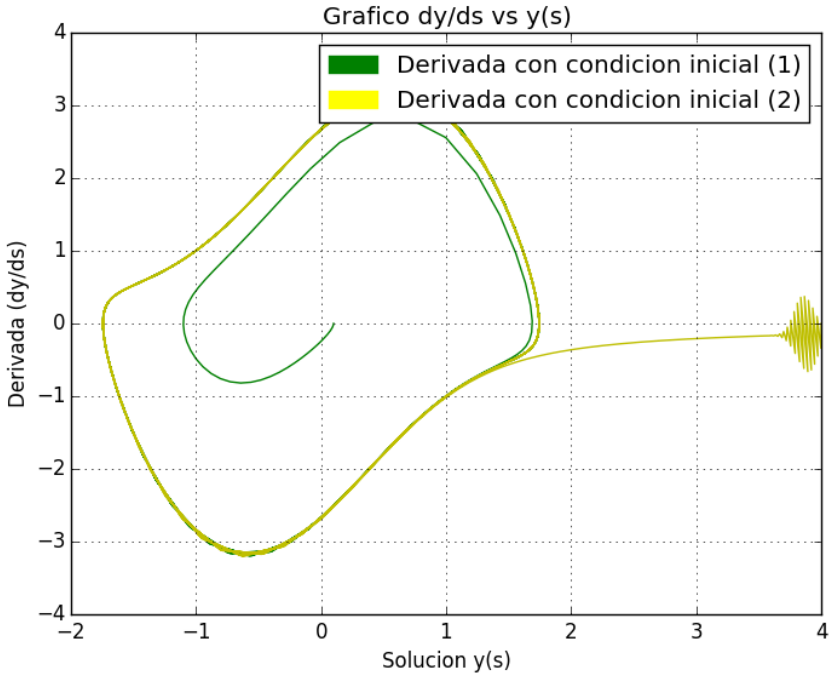

Figure 1: Gráfico derivada (dy/ds) vs solución (y(s)).

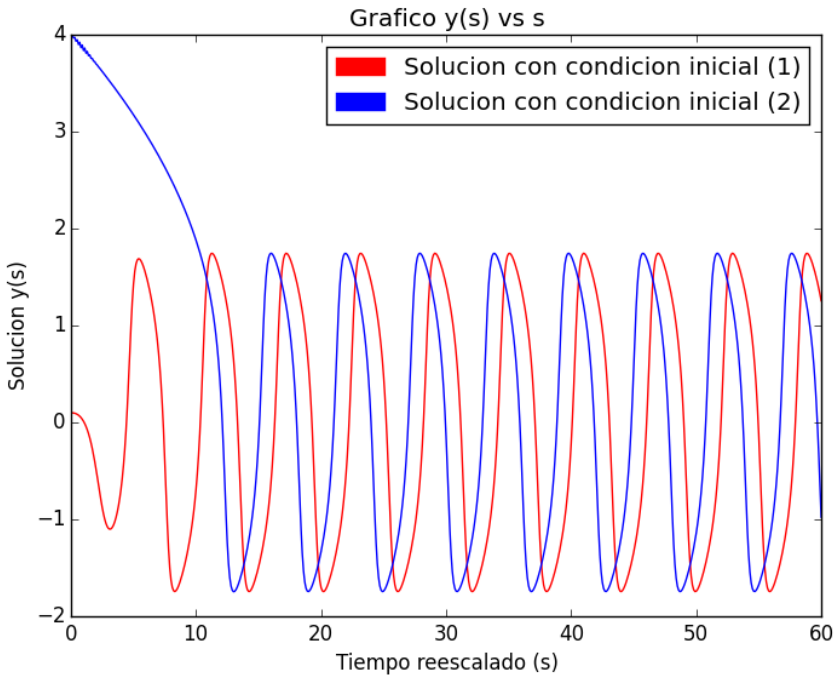

Figure 2: Gráfico solución (y(s)) vs tiempo reescalado (s).

2 Pregunta 2

2.1 Introducción

- El sistema de Lorenz es un set de ecuaciones diferenciales ordinarias conocido por tener algunas soluciones caóticas, la más famosa el llamado atractor de Lorenz. El sistema de ecuaciones de

Lorens es el siguiente:

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

De este set, la solución más famosa se obtiene con los parámetros $\sigma = 10$, $\beta = 8/3$ y $\rho = 28$. Se pide Utilizar esos parámetros junto a un set de condiciones iniciales $(x0, y0, z0)$ e integrar la ecuación por un tiempo conveniente mediante algún módulo de *scipy* de uso libre y graficar la solución $(x(t), y(t), z(t))$ en un gráfico 3D.

2.2 Procedimiento

- Para comenzar, se dirá que se utilizó el método incluido en el módulo *scipy.integrate*, específicamente el módulo *scipy.integrate.ode*. Dicho módulo integra numéricamente una EDO por el método de Runge-Kutta de orden 4. Ahora bien para poder realizar lo pedido se usó una extensión de este módulo específicamente para este problema, el cual corresponde a $ode(f).set_integrator('vode', method='adams')$ que toma una función f de la forma $\frac{dx}{dt} = f(x,t)$ y realiza la integración numérica de dicha función. Ahora como se necesitan condiciones iniciales para resolver el sistema de EDO's, se utilizó el vector $w_0 = [5, -5, 10]$ con w un vector de (x, y, z) y un vector p con los valores dados en el enunciado de (σ, β, ρ). Hecho esto, se procedió a realizar la función que entregara el sistema de ecs. de Lorenz para no redefinirlas a cada momento en el algoritmo, como sigue:

```
def ecs_lorenz(t,w,p): #w es un vector de (x,y,z)
    sigma=p[0]
    beta=p[1]
    rho=p[2]
    dw = np.zeros([3])
    dw[0] = sigma*(w[1]-w[0]) #dx/ds
    dw[1] = w[0]*(rho - w[2])-w[1] #dy/ds
    dw[2] = w[0]*w[1]-beta*w[2] #dz/ds
    return dw
```

que recibe un arreglo de tiempo, uno de vectores, en este caso $w = (x, y, z)$ y uno de constantes $p = (\sigma, \beta, \rho)$ para luego retornar las ecs. de Lorenz (entrega las funciones $f(x,t) = \frac{dx}{dt}$ asociados a cada EDO).
Definido esto luego se construyó el cuerpo del algoritmo como sigue:

```
t0 = 0
tf = 100.0
dt = 0.01
w0 = [5,-5,10]
Y=[]
T=[]
sol = ode(ecs_lorenz).set_integrator('vode',method='adams')
p = [10.0,8.0/3.0,28.0]
sol.set_f_params(p).set_initial_value(w0,t0)
while sol.successful() and sol.t+dt < tf:
    sol.integrate(sol.t+dt)
    Y.append(sol.y)
    T.append(sol.t)
Y = np.array(Y)
```

donde se usa como condición inicial del tiempo a $t_0 = 0$ con un paso de tiempo $dt = 0.01$ hasta un tiempo final $t_f = 100$, un vector o set de condiciones iniciales de espacio w_0 y se definen los arreglos Y y T que contendrán a las 3 soluciones $(x(t), y(t), z(t))$ y a el vector de tiempo de las mismas dimensiones, respectivamente, para poder graficar. Ahora solo queda graficar y siguiendo la ayuda del enunciado se usa el submódulo *Axes3D* del módulo $mpl_toolkits.mplot3d$ y se grafica la solución Y vs T.

2.3 Resultados

- A raíz del procedimiento nombrado se obtiene como resultado el siguiente gráfico en 3D para las soluciones a las ecs. de Lorenz:

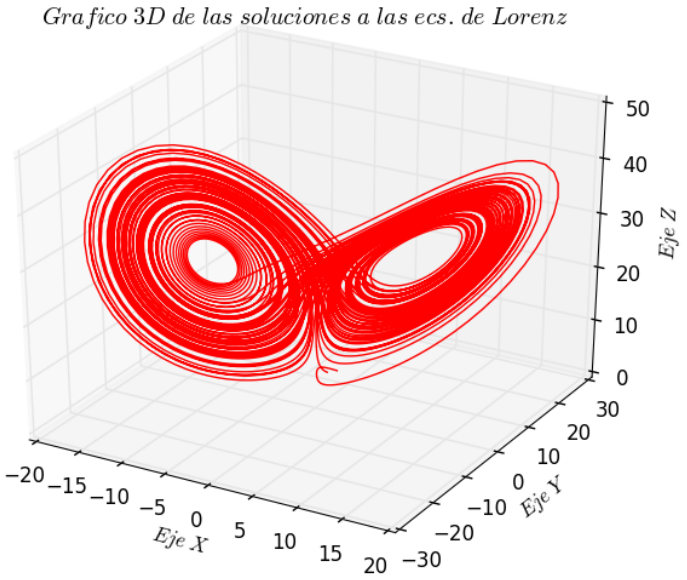

Figure 3: Gráfico en 3D de soluciones ecs. de Lorenz vs tiempo.