

Reporte Actividad 8

Eduardo Hndz.

May 15, 2018

1 Introducción

2 Modelo de Van Der Pol

En dinámica, el oscilador de *Van der Pol* es un oscilador no conservativo con amortiguamiento no lineal. Evolucionan en el tiempo acorde con la ecuación diferencial de segundo orden:

$$\frac{d^2x}{dt^2} - \mu(1 - x^2)\frac{dx}{dt} + x = 0 \quad (1)$$

Donde x es la posición en función del tiempo t , y μ es un parámetro escalar que indica la no linealidad y la fuerza de amortiguamiento

2.1 Forma de dos dimensiones

El *Teorema de Liénard's* puede usarse para probar que el sistema tiene un ciclo límite. aplicando la siguiente *Transformación de Liénard's*. El *Oscilador de Van der Pol* puede ser escrito en su forma de dos dimensiones.

$$\dot{x} = y \quad (2)$$

$$\dot{y} = \mu(1 - x^2)y - x \quad (3)$$

2.1.1 Resultados del Oscilador sin fuerza

Hay dos regímenes para las características del oscilador sin fuerza:

- cuando $\mu = 0$ no hay función que dependa del amortiguamiento, la ecuación queda como sigue:

$$\frac{d^2x}{dt^2} + x = 0 \quad (4)$$

Esta es una forma del *Oscilador Armónico Simple*, y siempre hay conservación de la energía.

- cuando $\mu > 0$, el sistema entra en un ciclo límite. Cerca del origen $x = \frac{dx}{dt} = 0$, el sistema es inestable, y alejado del origen, el sistema es amortiguado.
- El *Oscilador de Van der Pol* no tiene solución analítica exacta. dicha solución del Ciclo límite existe si $f(x)$ en la *Ecuación de Liénard* es una función constante por partes.

2.2 Oscilador de Van der Pol Forzado

El Manejo de fuerzas en el *Oscilador de Van der pol* toma la forma original de la función y se le agrega la función $Asin(\omega t)$ lo cual nos presenta una ecuación diferencial de la forma siguiente:

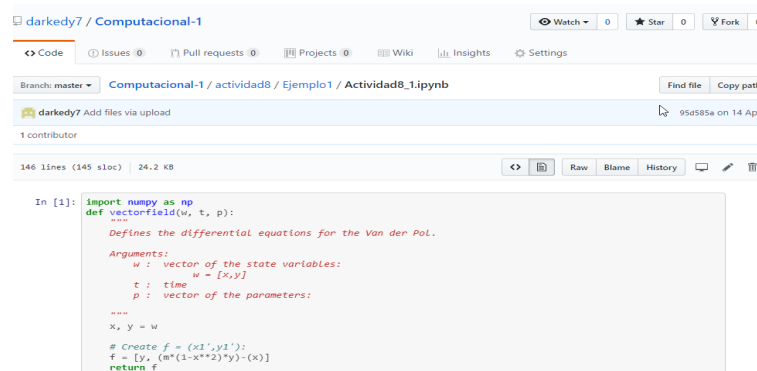
$$\frac{d^2x}{dt^2} - \mu(1 - x^2)\frac{dx}{dt} + x - Asin(\omega t) = 0 \quad (5)$$

Donde A es la *Amplitud* y ω es la *velocidad angular*

2.3 Ejemplos

2.3.1 Ejemplo1

En el primer ejemplo se debía reproducir el *Oscilador de Van der Pol* observando la fase de retraro del Oscilador sin fuerza, mostrando el ciclo límite. En dicho ejemplo se tuvieron que variar los valores correspondientes a las codiciones iniciales que se observaban en la ilustración otorgada por el profesor. El código quedo como sigue, siendo primero la función a utilizar:



```

In [1]: import numpy as np
def vectorfield(w, t, p):
    """
    Defines the differential equations for the Van der Pol.

    Arguments:
        w : vector of the state variables:
            w = [x,y]
        t : time
        p : vector of the parameters:
    """
    x, y = w
    # Create f = (x1', y1')
    f = [y, (mu*(1-x**2)*y)-(x)]
    return f
  
```

```

# Use ODEINT to solve the differential equations defined by the vector field
from scipy.integrate import odeint
# Coeficiente no lineal
m = 2.0
# Initial conditions
# x1 and x2 are the initial displacements
# x = -3
# y = -3

# ODE solver parameters
abserr = 1.0e-8
relerr = 1.0e-6
stoptime = 300.0
numpoints = 5000

# Create the time samples for the output of the ODE solver.
# I use a large number of points, only because I want to make
# a plot of the solution that looks nice.
t = [stoptime * float(i) / (numpoints - 1) for i in range(numpoints)]

# Pack up the parameters and initial conditions:

numbers=[(-3.0,-3.0),(-2.0,4.0),(-1.0,-2.0),(1.0,2.0)]
for (x,y) in numbers:
    # Call the ODE solver.
    w0 = (x, y)
    wsol=odeint(vectorfield, w0, t, args=(m,),
               atol=abserr, rtol=relerr)

    with open('%(x,y)'+str((x,y))+'.dat', 'w') as f:
        # Print & save the solution.
        for ti, wi in zip(t, wsol):
            print (ti, wi[0], wi[1], file=f)

```

En esta segunda parte del código, es donde se resuelve la ecuación diferencial. Como en el material proporcionado por el maestro se necesitaban utilizar distintos valor iniciales, hice un loop en el cual se ingresaran los valores iniciales a utilizar y una vez usado cada cada valor, fueran expulsados los datos en un archivo de texto.

```

In [3]: import numpy as np
        from matplotlib.font_manager import FontProperties
        import math
        import numpy as np
        import matplotlib.pyplot as plt
        #Nueva Gráfica #Velocidad2 Vs Posición2
        import math
        import numpy as np
        from pylab import savefig
        import matplotlib.pyplot as plt
        #####

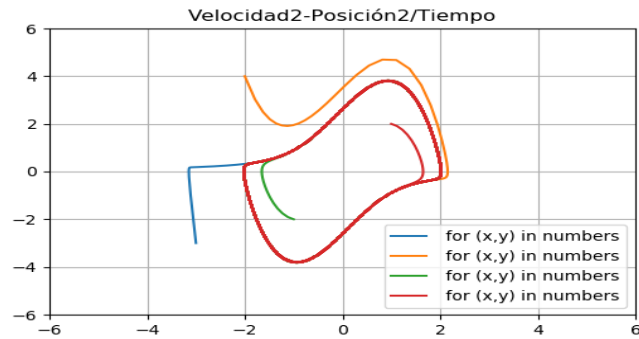
fig = plt.figure()

numbers=[(-3.0,-3.0),(-2.0,4.0),(-1.0,-2.0),(1.0,2.0)]
for (x,y) in numbers:
    t, x, y = np.loadtxt('%(x,y)'+str((x,y))+'.dat', unpack=True)
    plt.plot(x,y,label=' for (x,y) in numbers')
plt.legend(loc='lower right')
plt.xlim(-6,6)
plt.ylim(-6,6)
plt.title('Velocidad2-Posición2/Tiempo')
plt.grid()
plt.savefig('xD.png', dpi=100)
plt.show()

```

Por último de debían leer los datos y graficarlos, de igual forma ingresé un Loop para poder leerlos dato por dato.

El resultado obtenido fue el siguiente:



2.3.2 Ejemplo2

En el siguiente ejemplo se debía utilizar el mismo código pero era necesario variar el Valor de μ (coeficiente de amortiguamiento), teniendo los valores de condiciones iniciales fijos. En la parte media del código (solución de la ecuación diferencial) tuve que introducir nuevamente un loop, el cual fuera variando los valores de μ y expulsara los datos obtenidos con dicho valor.

```
: # Use ODEINT to solve the differential equations defined by the vector field
from scipy.integrate import odeint

#Coeficiente no lineal
mu = 0.0
# Initial conditions
# x1 and x2 are the initial displacements
x = -3
y = -3

# ODE solver parameters
abserr = 1.0e-8
relerr = 1.0e-6
stop_time = 300.0
num_points = 5000

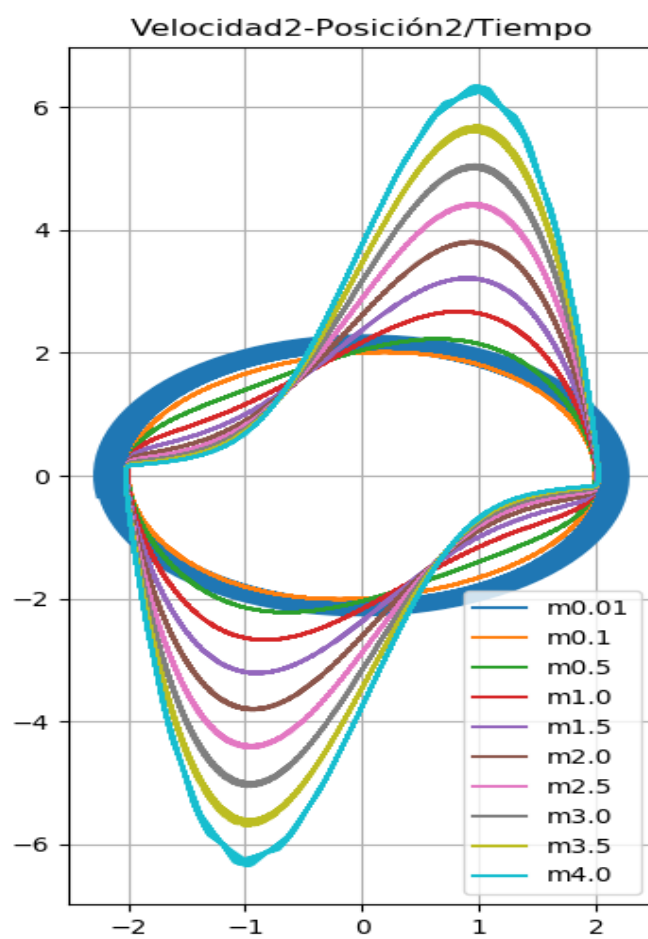
# Create the time samples for the output of the ODE solver.
# I use a large number of points, only because I want to make
# a plot of the solution that looks nice.
t = [stop_time * float(i) / (num_points - 1) for i in range(num_points)]

# Pack up the parameters and initial conditions:
w0 = [x, y]

numbers=[0.01,0.1,0.5,1.0,1.5,2.0,2.5,3.0,3.5,4.0]
for m in numbers:
    # Call the ODE solver.
    wsol=odeint(vectorfield, w0, t, args=(m,),
                atol=abserr, rtol=relerr)

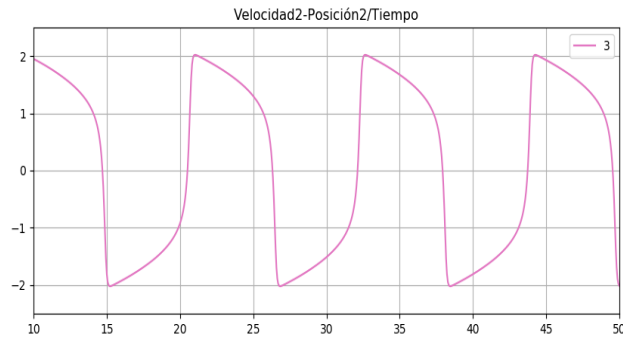
    with open('m'+str(m)+''.dat', 'w') as f:
        # Print & save the solution.
        for ti, wd in zip(t, wsol):
            print (ti, wd[0], wd[1], file=f)
```

El resultado obtenido en esta actividad es como sigue:



2.3.3 Ejemplo3

En este ejercicio, solo se nos proporcionaba la imagen del producto a graficar, en mi caso tuve que jugar con los valores de las *condiciones iniciales* para poder reproducir la gráfica de forma similar a la proporcionada, cabe resaltar que el valor de μ era fija (5.0) El valor que yo encontré para x fue de 2.0 y para y fue de -3.75 . La gráfica quedó como sigue:



2.3.4 Ejemplo4

Aquí se tuvo que cambiar parte de la función a utilizar ya que en este caso el sistema se veía afectado por una fuerza externa, el código quedó de la siguiente forma:

```
In [10]: import numpy as np
def vectorfield(w, t, p):
    """
    Defines the differential equations for the Van der Pol.

    Arguments:
        w : vector of the state variables:
            w = [x,y]
        t : time
        p : vector of the parameters:

    """
    X, y = w

    # Create f = (x1', y1'):
    f = [y, ((m*(1-x**2)*y) - {x}) + {a*(np.sin(wl*t))}]
    return f
```

```
n [17]: # Use ODEINT to solve the differential equations defined by the vector field
from scipy.integrate import odeint
import numpy as np
import math
#Coeficiente no lineal
m = 0.53
# Initial conditions
# x1 and x2 are the initial displacements
x = 0.0
y = 0.0
#Amplitud
a=(1.2)
#Velocidad Angular
wl=(2*(math.pi))/10
# ODE solver parameters
abserr = 1.0e-8
relerr = 1.0e-6
stoptime = 700.00
numpoints = 5000

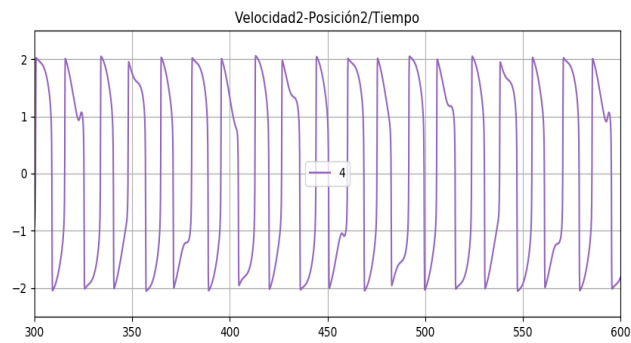
# Create the time samples for the output of the ODE solver.
# I use a large number of points, only because I want to make
# a plot of the solution that looks nice.
t = [stoptime * float(i) / (numpoints - 1) for i in range(numpoints)]

# Pack up the parameters and initial conditions:
w0 = [x, y]
# Call the ODE solver:
wsol=odeint(vectorfield, w0, t, args=(m,),
            atol=abserr, rtol=relerr)

with open('4.dat', 'w') as f:
    # Print & save the solution.
    for t1, w1 in zip(t, wsol):
        print (t1, w1[0], w1[1], file=f)
```

Una vez realizado el código y ejecutada cada celda, el resultado fue el siguiente.

- *Condiciones iniciales* $x = 0.0$, $y = 0.0$
- $A = 1.2$
- $\mu = 8.53$
- $\omega = \frac{\pi}{5}$



3 Conclusiones

Como pudimos observar el *Oscilador de Van der Pol* tiene variantes distintas las cuales depende de las condiciones iniciales, el valor que tenga el coeficiente de fricción y si es afectado por una fuerza o no.

4 Apéndice

- *Este ejercicio pareciera similar al desarrollado en las actividades 6 y 7. ¿Qué aprendiste nuevo?* Es similar, el cambio que noté fue el de ingresar datos distintos en forma de loop, aunque también la forma de las ecuaciones a utilizar era distinta.
- *¿Qué fue lo que más te llamó la atención del oscilador de Van der Pol?* Que es muy Complejo en el aspecto en el que puedes obtener distintos resultados incluso en la misma ecuación , al momento de cambiar el intervalo de los valores de μ , de las condiciones iniciales y de si es afectado por una fuerza externa. Además de que no puede hacer el reto del campo direccional. R.I.P
- *escuchado ya hablar de caos. ¿Por qué sería importante estudiar este oscilador?* No he escuchado mucho pero siento que este Oscilado es importante ya que tiene una fase de ciclo la cual creo que tiene algo que ver con el orden en el que se desarrolla la función.
- *¿Qué mejorarías en esta actividad?* Nada, es una actividad sencilla pero compleja
- *¿Algún comentario adicional antes de dejar de trabajar en Jupyter con Python?* Siento que debería haber una actividad más sobre esto.
- *Cerramos la parte de trabajo con Python ¿Que te ha parecido?* Muy padre, muy compleja y que es una gran herramienta.

5 Bibliografía

https://en.wikipedia.org/wiki/Van_der_Pol_oscillator