Actividad 8: Oscilador de Van der Pol

Barajas Ibarria Juan Pedro

12 de abril del 2018

1. Introducción

En el presente reporte se estudia lo que es el **Oscilador de van der Pol**, su solución y el como esta puede variar con el cambio de parámetros que influyen en la ecuación, ademas se hablara de algo de historia y se graficarán algunas soluciones. El oscilador de van der Pol es un oscilador con amortiguamiento no lineal. Su evolución temporal obedece a la ecuación diferencial de segundo orden:

$$\ddot{x} - \epsilon(1 - x^2)\dot{x} + x = 0$$

en la que x es la posición, función del tiempo t y ϵ es un parámetro no lineal.

Antecedentes

El oscilador de van der Pol fue descrito por el ingeniero y físico Balthasar van der Pol mientras trabajaba. Van der Pol encontró oscilaciones estables, que llamó oscilaciones de relajación, conocidas en la actualidad como ciclos límite, en circuitos que usaban válvulas de vacío. Cuando esos circuitos se hacen funcionar cerca del ciclo límite entran en acoplamiento y la señal entra en fase con la corriente.Van der Pol y su colega, van der Mark,

informaron en el número de septiembre de 1927 de Nature que para determinadas frecuencias aparecía un ruido irregular, siempre cerca de las frecuencias de acoplamiento. Fue uno de los primeros descubrimientos experimentales de la Teoría del caos.

2. Modelo de Van der Pol

EL teorema de Liénard prueba que el sistema tiene un ciclo limite. Aplicando la transformación de Liénard $y = x - x^3/3 - \dot{x}/\epsilon$ Estas ecuaciones se utilizaron para buscar una aproximación numérica resolviendo en Python, en donde se gráfico el Plano fase de un oscilador de van der Pol no forzado. A continuación se presenta el código utilizado para generar la gráfica del espació vectorial donde se calcularon varios valores iniciales para recrear a gráfica de la wikipedia con $\epsilon = 4$ y variando las condiciones iniciales para poder generar los diferentes datos, después se genero el campo vectorial de la solución para graficar dentro de ella las soluciones. Continuando se generaron de igual manera varias soluciones con distintos valores de ϵ que van desde 0.01 hasta 4 en la segunda imagen

```
In [1]: # Importación de librerias
        from scipy.integrate import odeint
        from scipy import array, arange,pi, sin
        import matplotlib.pyplot as plt

        #Definición de la ecuación diferencial
        def VanDerPol(X,t):
            x = X[0]
            x_point = X[1]
            dx = x_point
            dx_point = epsilon*(1.0 - x*x)*dx - x
            return array([dx, dx_point])

        # Definición de vector de tiempo
        t0 = 0.0
        tmax = 10.0
        pastemps = 0.005
        time = arange(t0, tmax, pastemps)

        # Condiciones iniciales
        x0 = -2.0
        v0 = 0.0

        # Parametros
        epsilon = 4.0

        # Solución de la ecuación diferencial
        x, x_point = odeint(VanDerPol,(x0,v0),time).T
```

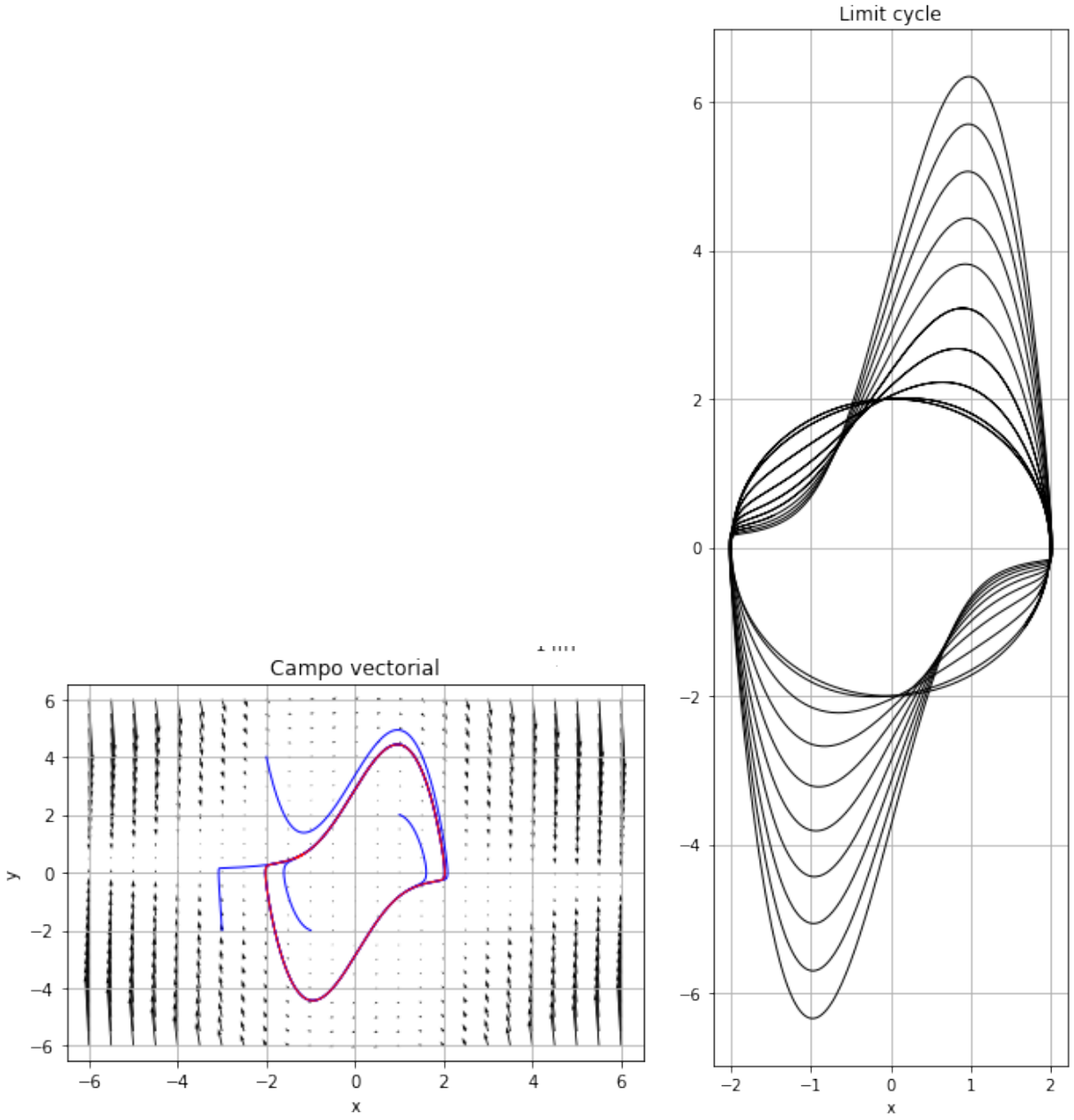
```
        with open('graf(b).dat', 'w') as f:
            # Print & save the solution.
            for t1, x1,y1 in zip(time, x, x_point):
                print (t1, x1, y1, file = f)
```

In [2]:
```
        #Campo vectorial
        #Importamos librerias de matplotlib
        from numpy import loadtxt
        from pylab import figure, plot, xlabel, grid, hold, legend, title, savefig
        from matplotlib.font_manager import FontProperties
        %matplotlib inline
        from pylab import *

        #Se define la granuladidad de las celdas (que tan pequenias son las celdas)
        #Maximo y minimo del eje x
        xmax = 6.0
        xmin = -xmax
        #Cuantos puntos se buscan, por eje x o y
        NX = 25
        NY = 25
        #Maximo y minimo del eje y
        ymax = 6
        ymin = -ymax
        #Se crea la red y se calculan los componentes del vector
        x = linspace(xmin, xmax, NX)
        y = linspace(ymin, ymax, NY)

        #Con meshgrid creamos 2 redes donde 'x' y 'y' contiennen las coordenadas de 'x'
        X, Y = meshgrid(x, y)
        #Estas son las ecuaciones diferenciales, que son sustituidos por sus respectivos
        # es para x y By es para y
        mu = 0.5
        Bx = Y
        By = mu*(1 - X**2)*Y - X

        figure()
        #Llamamos quiver de pylab, para hacer pintar nuestro campo vectorial en 2-D el c
        QP = quiver(X,Y,Bx,By)
        #quiverkey agrega una llave el cual mostrarala escala en la que sera mostrado el
        #Donde toma como primer argumento  QP, que es lo que se va a pintar, el segundo
        #dan la posicion de la llave en la direccion horizontal y vertical desde la part
        #derecha como fracciones del tamanio a pintar. El cuarto y quinto argumento son
        #y de las etiquetas. Como argumento final es la posicion de la etiqueta donde
        # 'N' es arriba
```

```
        # 'S' es abajo
        # 'W' es izquierda
        # 'E' es derecha
        quiverkey(QP, 0.85, 1.05, 1.0, '1 mT', labelpos='N')

        dx = (xmax - xmin)/(NX - 1)
        dy = (ymax - ymin)/(NY -1)

        #axis es usado para colocar a la izquierda, derecha, abajo y arriba los limites
        #cruces el cual ira en el orden antes mencionado
        axis([xmin-dx, xmax+dx, ymin-dy, ymax+dy])

        # Plot the solution that was generated
        xlabel('x')
        ylabel('y')

        time, x1, y1 = loadtxt('Datos1.dat', unpack=True)
        xlabel('x')
        grid(True)
        #hold(True)
        lw = 1
        plot(x1, y1, 'b', linewidth=lw)

        time, x1, y1 = loadtxt('Datos2.dat', unpack=True)
        plot(x1, y1, 'b', linewidth=lw)

        time, x1, y1 = loadtxt('Datos3.dat', unpack=True)
        plot(x1, y1, 'b', linewidth=lw)

        time, x1, y1 = loadtxt('Datos4.dat', unpack=True)
        plot(x1, y1, 'b', linewidth=lw)

        time, x1, y1 = loadtxt('Datos5.dat', unpack=True,skiprows=300)
        plot(x1, y1, 'red', linewidth=lw)

        figure(1, figsize=(8, 8))
        #legend((r'$x_1$'), prop=FontProperties(size=16))
        title('Campo vectorial')
        savefig('Vectores.png', dpi=100)
```

In [5]:
```
        # Plot the solution that was generated

        from numpy import loadtxt
        from pylab import figure, plot, xlabel, grid, hold, legend, title, savefig
        from matplotlib.font_manager import FontProperties
```

```
%matplotlib inline
figure( figsize=(4, 12))
time, x1, y1 = loadtxt('graf1.dat', unpack=True)
plot(x1, y1, 'k', linewidth=lw)

time, x1, y1 = loadtxt('graf2.dat', unpack=True)
plot(x1, y1, 'k', linewidth=lw)

time, x1, y1 = loadtxt('graf3.dat', unpack=True)
plot(x1, y1, 'k', linewidth=lw)

time, x1, y1 = loadtxt('graf4.dat', unpack=True)
plot(x1, y1, 'k', linewidth=lw)

time, x1, y1 = loadtxt('graf5.dat', unpack=True)
plot(x1, y1, 'k', linewidth=lw)

time, x1, y1 = loadtxt('graf6.dat', unpack=True)
plot(x1, y1, 'k', linewidth=lw)

time, x1, y1 = loadtxt('graf7.dat', unpack=True)
plot(x1, y1, 'k', linewidth=lw)

time, x1, y1 = loadtxt('graf8.dat', unpack=True)
plot(x1, y1, 'k', linewidth=lw)

time, x1, y1 = loadtxt('graf9.dat', unpack=True)
plot(x1, y1, 'k', linewidth=lw)

time, x1, y1 = loadtxt('graf10.dat', unpack=True)
plot(x1, y1, 'k', linewidth=lw)

xlabel('x')
grid(True)
#hold(True)
lw = 1
#legend((r'$x_1$'), prop=FontProperties(size=16))
title('Limit cycle')
savefig('ciclo.png', dpi=100)
```



Resultados del oscilador no forzado

Hay dos regímenes de funcionamiento interesantes para el oscilador no forzado:

- Cuando $\mu = 0$, no hay amortiguamiento, y la ecuación queda:

$$\frac{d^2x}{dt^2} + x = 0$$

Es la fórmula del oscilador armónico que no pierde energía.

- Cuando $\mu > 0$, el sistema alcanzará un ciclo límite, en el que se conservará la energía. Cerca del origen $x = dx/dt = 0$ el sistema es inestable, y lejos del origen hay amortiguamiento.

El oscilador de van der Pol forzado

Utilizando una fuente de excitación sinusoidal $A \sin \omega t$ la ecuación diferencial queda:

$$\frac{d^2x}{dt^2} - \mu(1 - x^2)\frac{dx}{dt} + x - A \sin \omega t = 0,$$

Donde A es la amplitud y ω la frecuencia angular asociada.

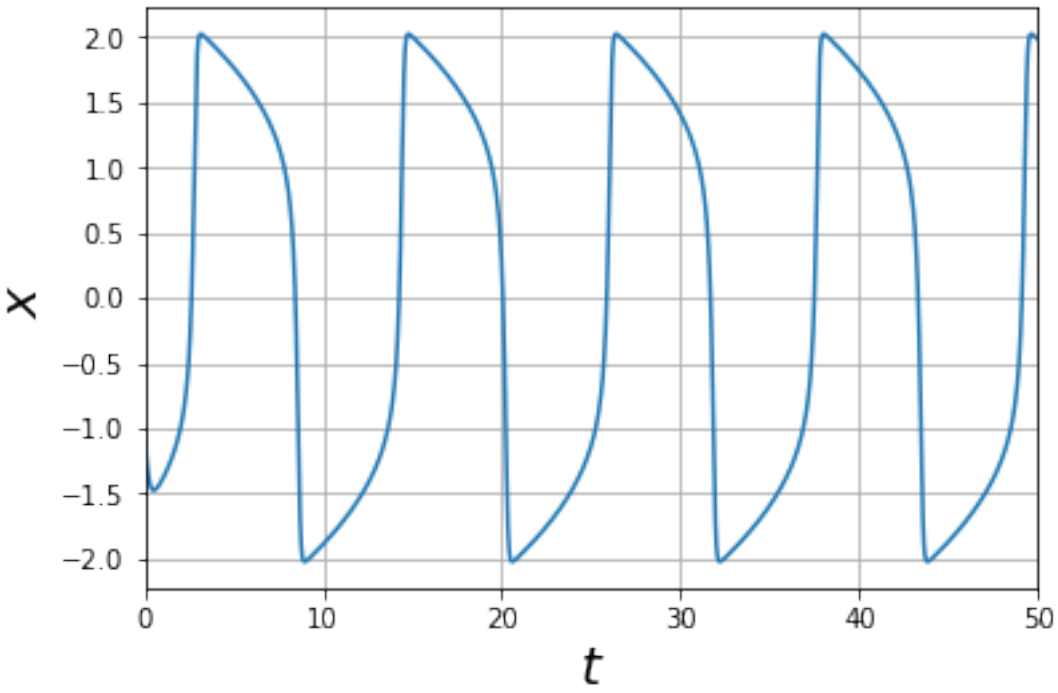
3. Resultados

Como parte del análisis de las soluciones y las ecuaciones de movimiento del oscilador, encontramos primero resolviendo para el oscilador no forzado donde $\epsilon = 5$, donde se gráfico la posición con respecto al tiempo.

Utilizando el código en python obtenido del ejemplo de lotka volterra

```
In [3]: # Importación de librerias
        from scipy.integrate import odeint
        from scipy import array, arange,pi, sin
        import matplotlib.pyplot as plt

        #Definición de la ecuación diferencial
        def VanDerPol(X,t):
            x = X[0]
            x_point = X[1]
            dx = x_point
            dx_point = epsilon*(1.0 - x*x)*dx - x
            return array([dx, dx_point])

        # Definición de vector de tiempo
        t0 = 0.0
        tmax = 50.0
        pastemps = 0.005
        time = arange(t0, tmax, pastemps)

        # Condiciones iniciales
        x0 = -1.0
        v0 = -2.0

        # Parametros
        epsilon = 5.0
```

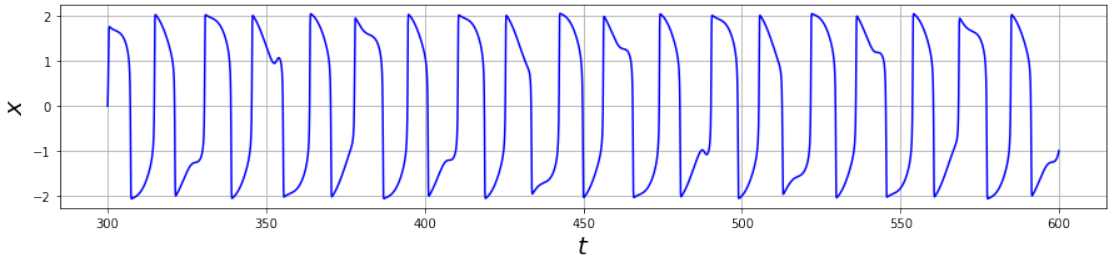
```
# Solución de la ecuación diferencial
x, x_point = odeint(VanDerPol,(x0,v0),time).T
```



También se obtuvo solución para un oscilador forzado con $\epsilon = 8.83$, teniendo la amplitud $A = 1.2$ y una frecuencia angular $\omega = 2\pi/10$. De igual manera se tomo el mismo código para resolver y graficar. Se obtuvo lo siguiente.

```
In [15]: # Importación de librerias
         from scipy.integrate import odeint
         from scipy import array, arange,pi, sin
         import matplotlib.pyplot as plt
         import numpy as np
         %matplotlib inline
         from pylab import figure, plot, xlabel, grid, hold, legend, title, savefig, yla

         #Definición de la ecuación diferencial
         def VanDerPol(X,t):
             x = X[0]
             w = (2*np.pi)/10
             a = 1.2
             x_point = X[1]
             dx = x_point
```

```
    dx_point = epsilon*(1 - x*x)*dx - x + a*np.sin(w*t)
    return array([dx, dx_point])

# Definición de vector de tiempo
t0 = 300
tmax = 600
pastemps = 0.05
time = arange(t0, tmax, pastemps)

# Condiciones iniciales
x0 = 0.0
v0 = 1.0

# Parametros
epsilon = 8.53

# Solución de la ecuación diferencial
x, x_point = odeint(VanDerPol,(x0,v0),time).T
```



4. Conclusiones

Como podemos apreciar el oscilador de van der Pol se convierte en un modelo de gran importancia al ser uno de los primeros modelos que entraban en categoría de caótico, ademas de funcionar de manera muy aceptable en circuitos eléctricos.

Referencias

Apéndice

1. Este ejercicio pareciera similar al desarrollado en las actividades 6 y 7. ¿Qué aprendiste nuevo?

-

2. ¿Qué fue lo que más te llamó la atención del oscilador de Van der Pol?
    - Sus soluciones tienen una forma muy interesante.
3. Has escuchado ya hablar de caos. ¿Por qué sería importante estudiar este oscilador?
    - Por ser fácil al momento de la resolución de las ecuaciones ademas de ser uno de los primeros da mucha información del comportamiento caótico.
4. ¿Qué mejorarías en esta actividad
    - Podríamos resolver las ecuaciones en clase
5. ¿Algún comentario adicional antes de dejar de trabajar en Jupyter con Python?
    - Se pudiera implementar una inducción al lenguaje.
6. Cerramos la parte de trabajo con Python ¿Que te ha parecido?
    - Bien.