Universidad de Sonora
División de Ciencias Exactas y Naturales
Departamento de Física

Actividad 8: Oscilador de Van Der Pol

Física Computacional I

Rolando Abdel Fimbres Grijalva

15 de abril de 2018

1. Introducción

En ésta actividad se estudió el oscilador de Van der Pol, el cual sirve pra comprender una clase de oscilación suave en señales generadas por circuitos particulares. Se pide replicar las imagenes que acompañan al artículo de wikipedia de éste oscilador. Para eso necesitaremos hacer uso de la integración numérica tal y como en las actividades pasadas.

2. Modelo de Van der Pol

El oscilador de Van der Pol es una clase de oscilacion amortiguada que no es lineal. Ésta puede ser modelada mediante una ecuación diferencial de segundo orden:

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

El modelo fue propuesto por Balthasar Van der Pol en 1920 mientras estaba empleado por la compañía Philips.

2.1. Historia

Van der Pol fue un ingeniero y físico. Él encontró oscilaciones estables que decidió llamar de relajación. Hoy en día éstas son comunmente llamadas cíclos límite. Los circuitos empleados en aquella época contaban con tubos de vacío, éstos se hacían funcionar cerca del cíclo límite pues entraban en acoplamiento y entraba en fase con la corriente.

2.2. Forma bidimensional

Aplicándo el teorema de Liénard se pueden encontrar dos formas, aplicando dos transformaciones. Una de ellas es: $y = x - x^3/3 - \dot{x}/\mu$, $y = x - x^3/3 - \dot{x}/\mu$, obteniendo así las ecuaciones:

$$\begin{aligned} \dot{x} &= \mu(x - \tfrac{1}{3}x^3 - y) \\ \dot{y} &= \tfrac{1}{\mu}x \end{aligned}$$

La forma se basa en la transformación $y = \dot{x}$, llevando a la ecuación:

$$\begin{aligned} \dot{x} &= y \\ \dot{y} &= \mu(1 - x^2)y - x \end{aligned}$$

3. Soluciones del modelo

Para poder analizar el comportamiento del oscilador es necesario utilizar un código bastante parecido al de lass actividades anteriores ya que el oscilador se modela con ecuaciones diferenciales que pueden integrarse numéricamente. A continuación se mostrarán las gráficas de posición contra tiempo, fase y cíclo límite así como el código utilizado para llegar a ella.
Primero se comienza con la definición de los parámetros:

[H]

```
from numpy import *
import pylab as p
# Definition of parameters
mu= 2.0
def dX_dt(X, t=0):
    xp=X[1]
    yp=mu*(1-X[0]**2)*X[1]-X[0]
    return array([xp, yp ])
```

Después necesitamos ingresar todas las condiciones iniciales para replicar en el diagrama de flujo de campo en wikipedia. En éste caso las condiciones son los definidos por x y v en el código:

[H]

```
from scipy.integrate import odeint
from scipy import array
import matplotlib.pyplot as plt

stoptime = 6.0
numpoints = 5000

#Tiempo
t = [stoptime * float(i) / (numpoints - 1) for i in range(numpoints)]

#Condiciones iniciales
x0 = -3.0
v0 = -3.0

#Resolver ecuaciones
x, y = odeint(dX_dt,(x0,v0),t).T

with open('Fase1.dat', 'w') as archivo:
    for t1, x1,y1 in zip(t, x, y):
        print (t1, x1,y1 ,file=archivo)
```

[H]

```
from scipy.integrate import odeint
from scipy import array
import matplotlib.pyplot as plt

stoptime = 6.0
numpoints = 5000

#Tiempo
t = [stoptime * float(i) / (numpoints - 1) for i in range(numpoints)]

#Condiciones iniciales
x1 = -1.0
v1 = -2.0

#Resolver ecuaciones
x, y = odeint(dX_dt,(x1,v1),t).T

with open('Fase2.dat', 'w') as archivo:
    for t2, x2,y2 in zip(t, x, y):
        print (t2, x2,y2 ,file=archivo)
```

[H]

```
from scipy.integrate import odeint
from scipy import array
import matplotlib.pyplot as plt

stoptime = 6.0
numpoints = 5000

#Tiempo
t = [stoptime * float(i) / (numpoints - 1) for i in range(numpoints)]

#Condiciones iniciales
x2 = 1.0
v2 = 2.0

#Resolver ecuaciones
x, y = odeint(dX_dt,(x2,v2),t).T

with open('Fase3.dat', 'w') as archivo:
    for t3, x3,y3 in zip(t, x, y):
        print (t3, x3,y3 ,file=archivo)
```

[H]

```
from scipy.integrate import odeint
from scipy import array
import matplotlib.pyplot as plt

stoptime = 6.0
numpoints = 5000

#Tiempo
t = [stoptime * float(i) / (numpoints - 1) for i in range(numpoints)]

#Condiciones iniciales
x3 = -2.0
v3 = 4.0

#Resolver ecuaciones
x, y = odeint(dX_dt,(x3,v3),t).T

with open('Fase4.dat', 'w') as archivo:
    for t4, x4,y4 in zip(t, x, y):
        print (t4, x4,y4 ,file=archivo)
```

[H]

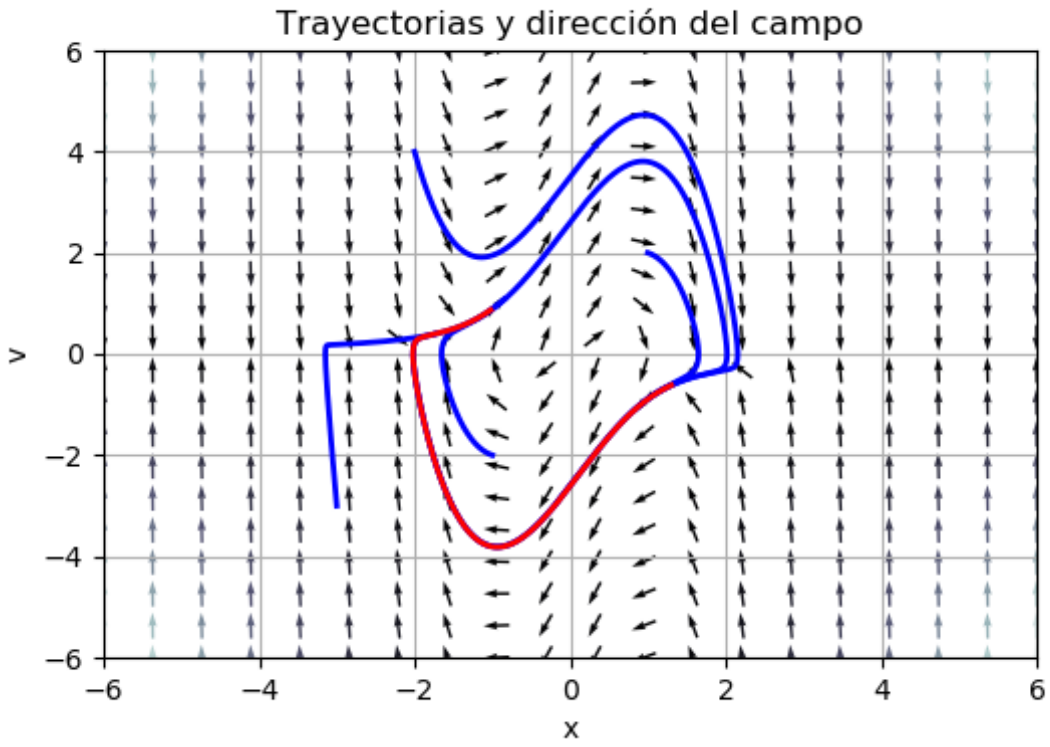
```
from scipy.integrate import odeint
from scipy import array
import matplotlib.pyplot as plt

stoptime = 6.0
numpoints = 5000

#Tiempo
t = [stoptime * float(i) / (numpoints - 1) for i in range(numpoints)]

#Condiciones iniciales
x4 = 2.0
v4 = 0.0

#Resolver ecuaciones
x, y = odeint(dX_dt,(x4,v4),t).T

with open('Fase.dat', 'w') as archivo:
    for t5, x5,y5 in zip(t, x, y):
        print (t5, x5,y5 ,file=archivo)
```

Ahora se introducirá el código que genera el diagrama de campo.

[H]

```
from numpy import loadtxt
from numpy import *
import pylab as p
from matplotlib.font_manager import FontProperties
from scipy import integrate
%matplotlib inline


#-----------------------------------------------------------
# define a grid and compute direction at each point
nb_points   = 20

x = linspace(-6, 6, nb_points)
y = linspace(-6, 6, nb_points)

X1 , Y1  = meshgrid(x, y)                       # create a grid
DX1, DY1 = dX_dt([X1, Y1])                      # compute growth rate on the gridt
M = (hypot(DX1, DY1))                           # Norm of the growth rate
M[ M == 0] = 1.                                 # Avoid zero division errors
DX1 /= M                                        # Normalize each arrows
DY1 /= M

#-----------------------------------------------------------
# Draw direction fields, using matplotlib 's quiver function
# I choose to plot normalized arrows and to use colors to give information on
# the growth speed

Q = p.quiver(X1, Y1, DX1, DY1, M, pivot='mid', cmap=p.cm.bone)
p.grid()

lw=2

t1, x1, y1 = loadtxt('Fase1.dat', unpack=True)
t2, x2, y2 = loadtxt('Fase2.dat', unpack=True)
t3, x3, y3 = loadtxt('Fase3.dat', unpack=True)
t4, x4, y4 = loadtxt('Fase4.dat', unpack=True)
t5, x5, y5 = loadtxt('Fase.dat', unpack=True, skiprows=1500)

p.plot(x1, y1, 'blue', linewidth=lw)
p.plot(x2, y2, 'blue', linewidth=lw)
p.plot(x3, y3, 'blue', linewidth=lw)
p.plot(x4, y4, 'blue', linewidth=lw)
p.plot(x5, y5, 'red', linewidth=lw)
```

[H]



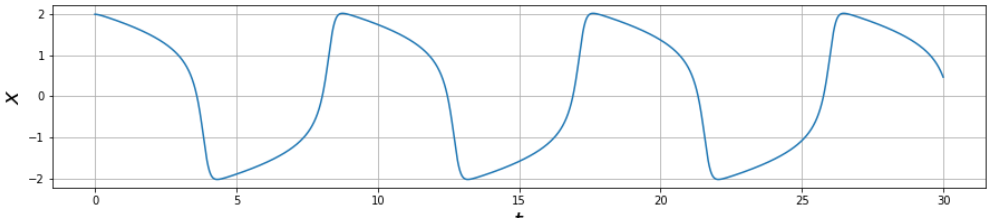
Después necesitamos el código para mostrar el desplazamiento con respecto al tiempo. El primero son fuerza externa y el segundo con fuerza externa. En el caso del primero:

[H]

```
t0 = 0
tmax = 30
pastemps = 0.01
t = arange(t0, tmax, pastemps)

#Condiciones iniciales
x0 = 2.0
y0 = 0.0

##Mu
u = 3.0

#Solución a la ecuación diferencial
x, y = odeint(dX_dt,(x0,y0),t).T

#Trayectoria del ocilador
pl = plt.figure(figsize=(15,3))
plt.grid()
plt.xlabel('$t$', fontsize = 20)
plt.ylabel('$x$', fontsize = 20)
plt.plot(t, x)
plt.show()
```

Obteniendo:

[H]



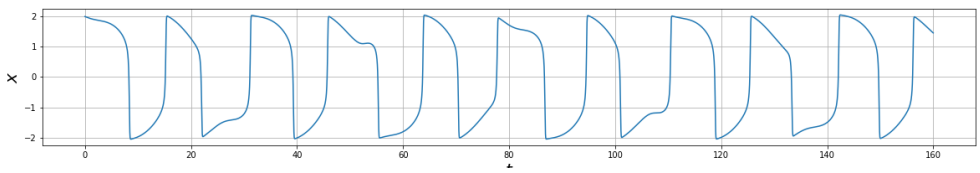
En el segundo caso:

[H]

```
t0 = 0
tmax = 160
pastemps = 0.01
t = arange(t0, tmax, pastemps)

#Condiciones iniciales
x0 = 2.0
y0 = 0.0

#Parametros
u = 8.53
M = 1.2
w = np.pi/5.0

#Solución a la ecuación diferencial
x, y = odeint(dX_dt,(x0,y0),t).T

#Trayectoria del ocilador
pl = plt.figure(figsize=(20,3.0))
plt.grid()
plt.xlabel('$t$', fontsize = 20)
plt.ylabel('$x$', fontsize = 20)
plt.plot(t, x)
plt.show()
```

Obteniendo:

[H]



También necesitabamos ver el comportamiento cuando μ varía desde 0.01 hasta 4.

[H]

```
from scipy.integrate import odeint
from scipy import array, arange,pi, sin
import matplotlib.pyplot as plt

def dX_dt(X,t):
    x = X[0]
    y = X[1]
    a = X[0]
    b = X[1]
    c = X[0]
    d = X[1]
    e = X[0]
    f = X[1]
    g = X[0]
    h = X[1]
```

[H]

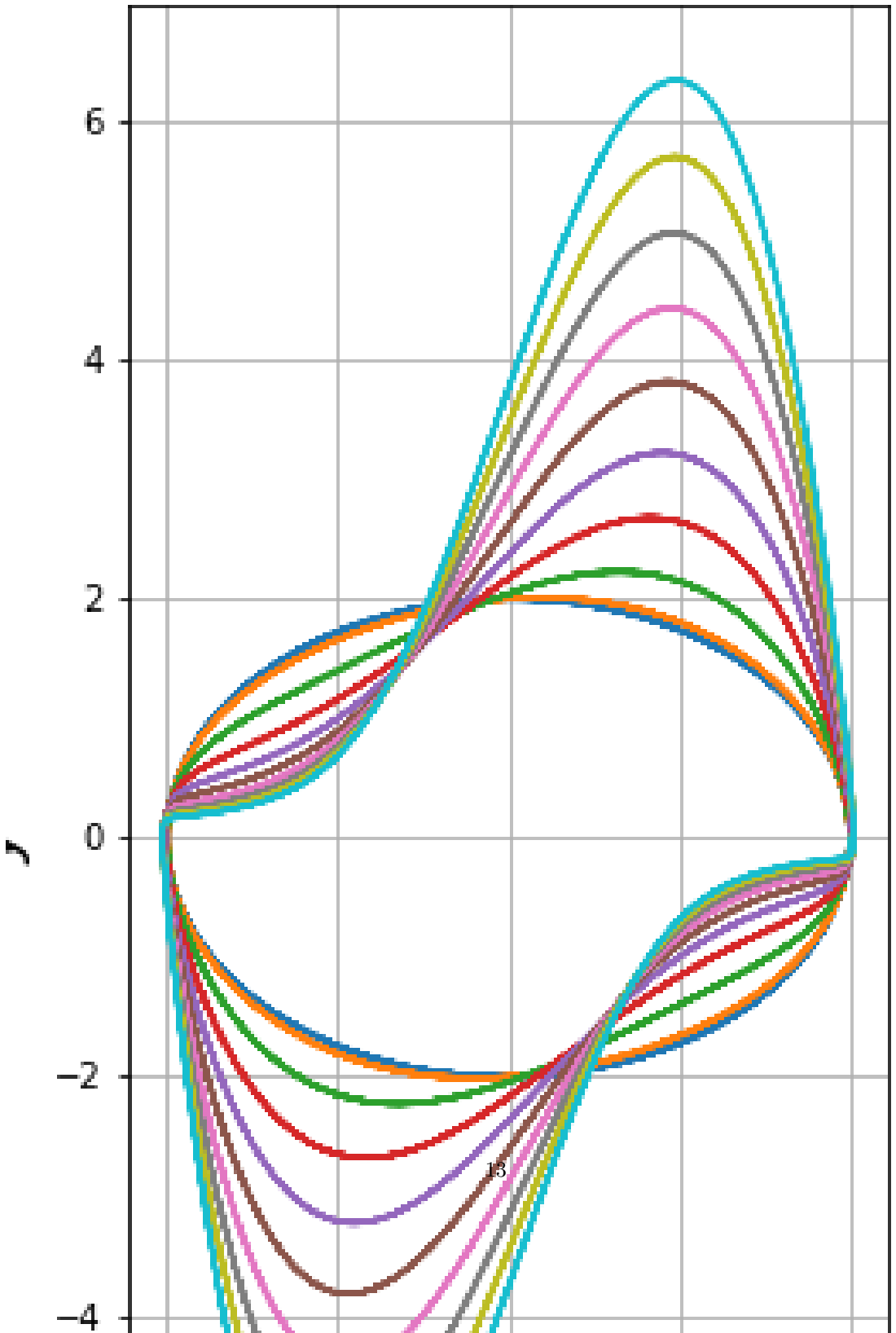
```
#Solución a la ecuación diferencial
a, b = odeint(dX_dt,(x0,y0),t).T

u = 0.1

#Solución a la ecuación diferencial
c, d = odeint(dX_dt,(x0,y0),t).T

u = 0.5

#Solución a la ecuación diferencial
e, f = odeint(dX_dt,(x0,y0),t).T
```

[H]

```
#Trayectoria de fase
p2 = plt.figure(figsize=(4,9))
plt.grid()
plt.xlabel('$x$', fontsize = 20)
plt.ylabel('$y$', fontsize = 20)
plt.plot(a, b)
plt.plot(c, d)
plt.plot(e, f)
plt.plot(g, h)
plt.plot(j, k)
plt.plot(l, n)
plt.plot(o, p)
plt.plot(q, r)
plt.plot(s, v)
plt.plot(x, y)
#plt.xlim(-2.5,2.5)
#plt.ylim(-6.0,6.0)
plt.show()
```

Obteniendo:

[H]


6
4
2
0
−2
−4

4. Resultados

Al concluir con la programación y mirar detalladamente los diagramas generados podemos ver que en el primer caso se conto con varias condiciones iniciales. Aun así miramos que siguen el patrón del cíclo límite. Además de como el campo direccional favorece a la trayectoria.
En el caso de las graficas de desplazamiento vemos como aun existe una periodicidad notable en la oscilación.
Finalmente se observa cómo afecta una fuerza externa a la oscilación aunque se vea muy ligera. Y al cambiarse los coeficientes como se notan picos.

5. Conclusiones

El tema fué bastante interesante pues se introdujo un nuevo fenómeno que puede ser resuelto con las mismas herramientas numéricas utilizadas con anterioridad. Debo decir que me costó bastante trabajo a comparacion de las prácticas anteriores pues tuve que recurrir a la ayuda de compañeros. Despejadas mis dudas me di cuenta que es muy importante definir los parametros y en mi caso mantener separadas los archivos de datos ya que cometía el error de asignarles un mismo nombre cosa que complicaba todo.

6. Bibliografía

Matplotlib: Lotka-Volterra tutorial -SciPy Cookbook doc. 2018.
Van der Pol oscillator. 2018., www.scholarpedia.org

7. Apéndice

1.Éste ejercicio pareciera similar al desarrollo en las cativideades 6 y 7. ¿Qué aprendiste nuevo?
A como hacer el campo direccional, ne pareció que fue lo más relevante.
2.¿Qué fue lo que más te llamó la atención del oscilador de Van der Pol?
El hecho de que los circuitos con los que Van der Pol trabajaba funcionaban cerca del cíclo límite.
3.Has escuchado ya hablar de caos. ¿Por qué sería importante estudiar éste oscilador?
Si. Lo considero importante porque me parece que los fenómenos más complejos y que mayor impacto hacia el humano tienen son sistemas caóticos.
4.¿Qué mejorarías de ésta actividad?
No sería una mejora, tal vez lo que yo haría sería separarla en dos actividades distintas. Me pareció muy extensa.
5.¿Algún comentario adicional antes de dejar de trabajar en Jupyter con Python?
Que me parece un lenguaje más sencillo y práctico que FORTRAN. Y jupyter fue nuevo para mi. Me parece muy bien y que solo necesitas acceso a internet si no cuenta tu equipo con la aplicación necesaria.
6.Cerramos la parte de trabajo con Python ¿qué te ha parecido?
Muy bien, práctico, fácil de aprender y me agrada que cuente con una gran cantidad de bibliotecas para realizar una gran variedad de trabajos. Me parece muy útil como herramienta de análisis de datos lo que la vuelve excelente en la física y muchas otras áreas.