Universidad del Estado de Sonora

División de Ciencias Exactas y Naturales.

Licenciatura en Física.

Física Computacional 1

El Espacio Fase.

Olga María Fimbres Morales

17 de Marzo de 2016

Introducción.

Nuevamente abordamos el modelo físico de péndulo, pero esta vez de una forma distinta a la que se ha venido haciendo, ya que durante el desarrollo de esta práctica lo que se busco lograr fue el generar un espacio fase de múltiples soluciones para el modelo que ya se tenia, otra vez, el péndulo.

Sabemos, que una ecuación diferencial posee lo que se llama una familia de soluciones que se generan a partir de aquellas mas elementales que surgen a partir de encontrar una solución general al problema, a esto, a esta familia de soluciones es a lo que nos referimos con espacio fase.

En el desarrollo de esta actividad, se generaron a aquellas posibles soluciones a partir de ciertas condiciones iniciales, dígase ángulo y velocidad angular inicial, durante un periodo de tiempo, permitiéndonos de esta forma observar la forma en que se comportaría el péndulo teniendo distintas condiciones iniciales.

Espacio Fase.

El espacio pase es una representación geométrica de las trayectorias de una sistema dinámico en el plano fase. Cada grupo de condiciones iniciales representa una curva o punto diferente en el.

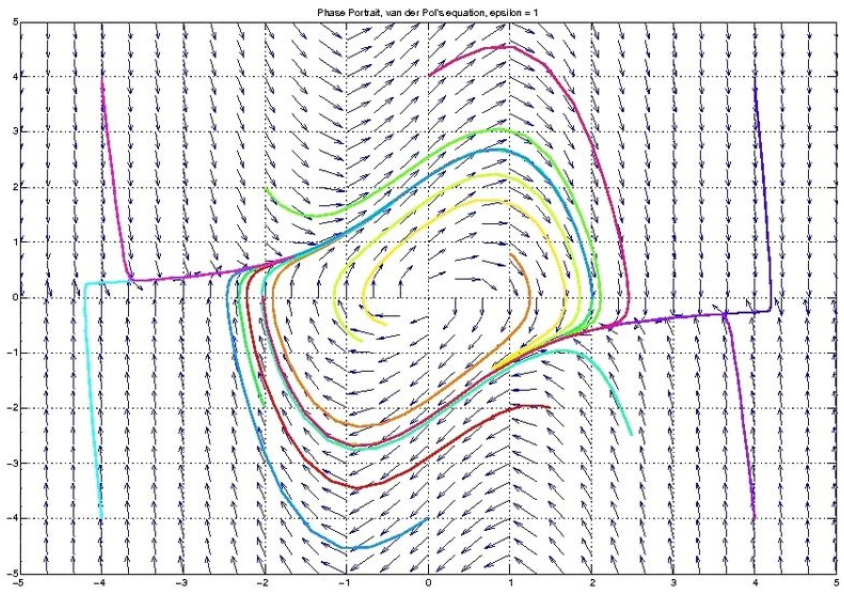
Son sumamente útiles al momento de estudiar sistemas dinámicos, consisten en un gráfico de las trayectorias en el espacio; nos permiten conocer diferente información como la velocidad que lleva en un determinado momento, el desplazamiento o los límites entre los cuales se desarrolla el sistema.

Utilizan diferente simbología por ejemplo, la trayectoria se representa con flechas, los puntos estables con puntos y los puntos inestables con círculos.

Existen diferentes ejemplo de donde podemos utilizar los espacios fases, algunos de ellos son:

- Péndulo simple.
- Oscilador armónico simple.

- Oscilador de Van der Pol.
- Diagrama de bifurcación.
- Plano de parámetro.



Espacio fase del oscilador de Van der Pol [2].

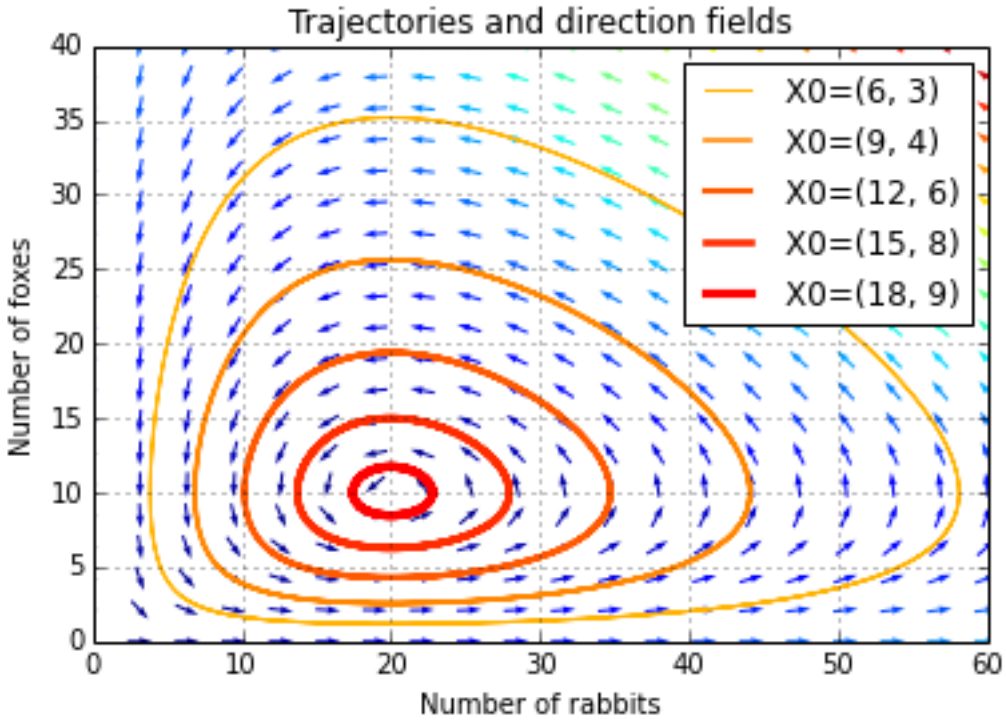
Como podemos ver, son una herramienta que nos permite conocer el estado del modelos que estemos estudiando, por lo que para ciertas condiciones iniciales nos es posible saber que trayectoria seguirá, que velocidad tendrá y como se comportara a lo largo del tiempo.

Problema.

El desarrollo de esta actividad consistió en realizar la gráfica de un espacio fase de las soluciones del péndulo simple a partir de un programa para el espacio fase del crecimiento de las poblaciones de zorros y conejos, el cual se encontraba disponible en la biblioteca de SciPy Cookbook[3].

Nuevamente se utilizó la herramienta *odeint* utilizada en la práctica número 5 para realizar la integración de nuestra funciones y de esta forma encontrar las diferentes soluciones a partir de distintas condiciones iniciales.

Primeramente fue necesario comprender la forma en que el código que se nos proporcionó trabajaba y la forma en que graficaba los resultados obtenidos de las integraciones de las funciones para crear el siguiente espacio fase:



Que como podemos observar nos muestra la forma en que se comporta el número de conejos y zorros vivos en una cierta área en distintas condiciones iniciales, donde los conejos corresponden al eje X y los zorros al eje Y.

Apoyándonos en actividades pasadas donde ya habíamos desarrollado la integración de la ecuación diferencial del péndulo simple solamente fue necesario tomar como referencia el ámbito que crea la gráfica del espacio fase del código que se nos proporcionó y utilizando condiciones iniciales adecuadas para crear el espacio fase deseado fue que se logro crear un diagrama adecuado. Generando de esta forma el siguiente código:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
import pylab as p
from scipy import integrate

b = 0.0 #fricción
g = 9.8 #gravedad
l = 1. #longitud de la cuerda
c = g/l

#condiciones iniciales
X_f0 = array([ -4*np.pi , 4*np.pi])
X_f1 = array([ -2*np.pi, np.pi*0])
t = np.linspace(0, 20, 300) #rango de tiempo

def pend(y, t, b, c):
     theta, omega = y
     dydt = [omega, -b*omega - c*np.sin(theta)]
     return dydt

values  = linspace(-1, 1, 60)
# position of X0 between X_f0 and X_f1
vcolors = p.cm.terrain_r(linspace(0.5, 1., len(values)))
# colors for each trajectory
vcolors2 = p.cm.winter_r(linspace(0.5, 1., len(values)))
f2 = p.figure()
```

```
#-----------------------------------------------------
# plot trajectories
for v, col in zip(values, vcolors):
    X0 = v * X_f0
    # starting point
    X = integrate.odeint( pend, X0, t, (b, c) )
    p.plot( X[:,0], X[:,1], lw=3.5*v, color=col,
    label='X0=(%.f, %.f)' % ( X0[0], X0[1]) )

for v, col in zip(values, vcolors):
    X0 = v * X_f1
    # starting point
    X = integrate.odeint( pend, X0, t, (b, c) )
    p.plot( X[:,0], X[:,1], lw=3.5*v, color=col,
    label='X0=(%.f, %.f)' % ( X0[0], X0[1]) )

#-----------------------------------------------------
p.title('Espacio fase del pendulo simple.')
p.grid()
p.xlim((-4*np.pi)/2, (4*np.pi)/2)
p.ylim(-4*np.pi, 4*np.pi)
f2.savefig('rabbits_and_foxes_2.png')
```

Podemos observar que comenzamos por especificar todas aquellas herramientas que utilizaremos durante el desarrollo del programa, todas ellas previamente utilizadas.

Continuamos especificando las constantes para nuestro sistema del péndulo simple, como siempre la fricción sera nula, la gravedad la constante terrestre y la longitud de la cuerda un metro. Ahora, las condiciones iniciales de nuestro sistema serán en dos distintas direcciones, f1 correspondiente al ángulo inicial y f2 correspondiente a la velocidad angular inicial, además de la variante del tiempo; las primeras dos las construimos como arreglos entre dos diferentes posiciones dependientes de π y el último como una secuencia de 300 números igualmente espaciados entre 0 y 20.

Ahora, procedemos a especificar la función con la que vamos a trabajar, que como recordara, es la misma utilizada en actividades anteriores. Después, definimos la posición inicial, el cual resulta una sucesión de números

entre -1 y 1, y los colores que se utilizaran para graficar; Python cuenta con una amplia gama de colores solidos y degradados para realizar las gráficas, por lo que esto resulta decisión del programador.

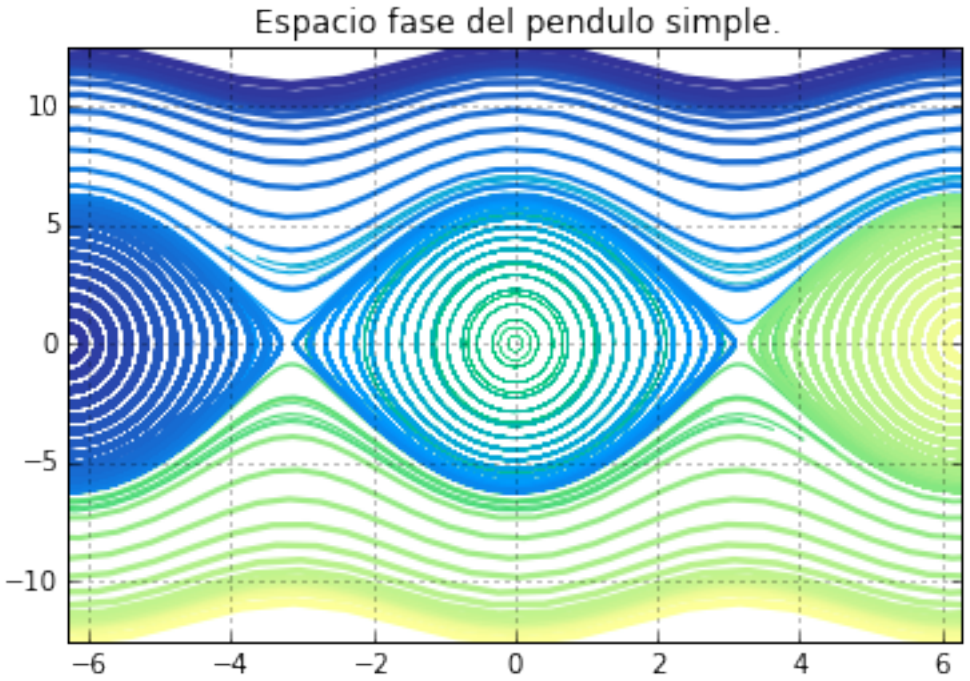
Pasamos ahora a realizar graficar las trayectorias, pero primero resulta necesario generar los datos a graficar; lo cual se logra con un ciclo que toma los arreglos que especificamos como condiciones iniciales. Ahora, pasamos a realizar la integración de la ecuación del péndulo simple junto con todos aquellos parámetros que le conciernen, que como ya sabemos logramos con la herramienta *odeint*.

Esto debe realizarse dos veces, una con nuestras condiciones f1 y otra con las condiciones f2 para que de esta forma seamos capaces de cubrir un mayor número de casos para el péndulo.

Finalmente, graficamos los arreglos formados por la integración de forma que una columna sera las ordenadas y la otra las abscisas.

Resultados

Una vez que logramos que nuestro código funcionara de la manera adecuada, para lo cual puede resultar necesario jugar un poco con los límites utilizados en las condiciones iniciales y el punto inicial, obtenemos el espacio fase que nos muestra las posibles soluciones para la ecuación diferencial del péndulo simple:



Utilizando esto como apoyo podemos conocer la posición y la velocidad angular que poseerá en diferentes momentos y para distintas condiciones iniciales, así como conocer sus puntos de equilibrio, por ejemplo, a simple vista podemos observar que uno de sus puntos de equilibrio es el punto (0,0) lo cual nos dice que para un ángulo inicial cero y una velocidad angular cero el péndulo no se moverá.

De igual modo podemos seleccionar un punto cualquiera, dentro de los límites de nuestro diagrama de fase, formando sus coordenadas por ángulo y velocidad angular inicial y conocer el comportamiento que el péndulo tendrá para dichas condiciones iniciales.

Anexos

Código Zorros y Conejos

```
from scipy import integrate
import numpy as np
import matplotlib.pyplot as plt

l = 1. #longitud de la cuerda
g = 9.81

#periodo para oscilaciones pequeñas
To = 2*np.pi*sqrt(l/g)

#periodo para oscilaciones arbitrarias
#definir la función a integrar

#theta_0 = np.linspace(0, np.pi-0.001 ,100)

X = lambda theta: 1.0/(np.sqrt(np.cos(theta)-np.cos(theta_0)))
#ciclo para integrar
theta_0 = 0.0
while (theta_0 < np.pi-0.08):
#es necesario restringirlo a un numero antes de pi,
pero si este es muy pequeño
el programa tiene problemas para correr
    theta_0 = theta_0 + 0.05
    int2 = integrate.quad(X, 0.0, theta_0)

    #print(int2[0])
    #theta = (theta_0*360)/np.pi
    #print(theta)
    T = 4*sqrt(l/(2*g))*int2[0]
    #print(T)
    Z = T/To
    #print(Z)
    #print(theta_0)
    plt.plot(theta_0 ,Z,'o', label='Error relativo')
    plt.grid(True)
    plt.xlabel('Theta',fontsize=15)
    plt.ylabel('T/To',fontsize=15)
    v=[0,np.pi/2,1,1.2]
    axis(v)
    #plt.legend()
    #plt.show()
```

Espacio fase Zorros y Conejos.

```
values  = linspace(0.3, 0.9, 5)
# position of X0 between X_f0 and X_f1
vcolors = p.cm.autumn_r(linspace(0.3, 1., len(values)))
# colors for each trajectory

f2 = p.figure()

#-----------------------------------------------------------
# plot trajectories
for v, col in zip(values, vcolors):
    X0 = v * X_f1
    # starting point
    X = integrate.odeint( dX_dt, X0, t)
    # we don't need infodict here
    p.plot( X[:,0], X[:,1], lw=3.5*v, color=col,
     label='X0=(%.f, %.f)' % ( X0[0], X0[1]) )

#-----------------------------------------------------------
# define a grid and compute direction at each point
ymax = p.ylim(ymin=0)[1]
# get axis limits
xmax = p.xlim(xmin=0)[1]
nb_points   = 20

x = linspace(0, xmax, nb_points)
y = linspace(0, ymax, 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
```

Continuamos con el código.

```
#--------------------------------------------------------
# Drow direction fields, using matplotlib 's quiver function
# I choose to plot normalized arrows
and to use colors to give information on
# the growth speed
p.title('Trajectories and direction fields')
Q = p.quiver(X1, Y1, DX1, DY1, M, pivot='mid', cmap=p.cm.jet)
p.xlabel('Number of rabbits')
p.ylabel('Number of foxes')
p.legend()
p.grid()
p.xlim(0, xmax)
p.ylim(0, ymax)
f2.savefig('rabbits_and_foxes_2.png')
```

Referencias

[1] Wikipedia, The free encyclopedia; "Phase portait"; 2016content...

[2] By Ashraful; via Wikimedia Commons;"Phase portrait"

[3] SciPy Cookbook; Matplotlib: Lotka volterra tutorial.