Práctica 3: Discretización de sistemas continuos

Adrián Pérez Peinador (Grupo U1)

Introducción

El objetivo de esta práctica era obtener información de una ecuación diferencial mediante su discretización y representación de su espacio fásico. Asimismo, se pide ver la evolución de una región (rectángulo) y verificar que se cumple el teorema de Liouville.

Material usado

Para la realización de esta práctica se hace uso de librerías de python comunes como *numpy*, *matplotlib* o *scipy*.

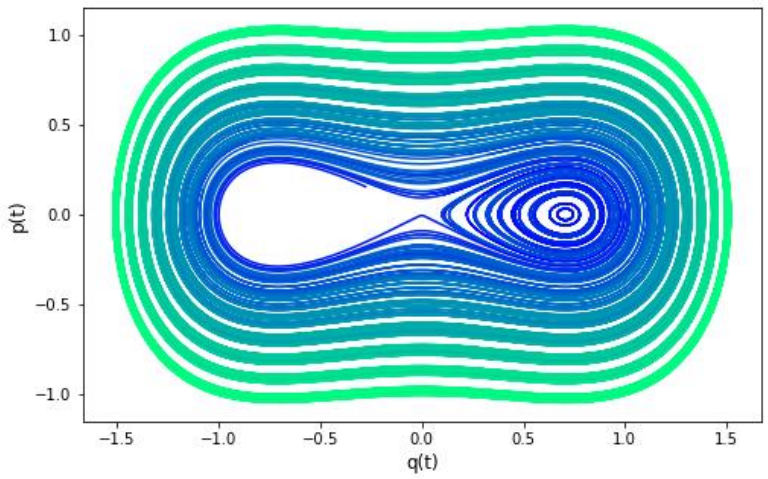
Además de estos recursos externos, en el código se definen varias funciones. Algunas de ellas provienen de la plantilla provista para la práctica, como *orb*, *deriv*, *simplectica* o *animate*, mientras que la función *evoluciondf* ha sido definida por nosotros para calcular y mostrar la evolución del rectángulo pedido dados el tiempo, la granularidad del parámetro temporal y el número de putos a calcular por fila y columna.

Resultados

A continuación, se exponen los resultados obtenidos para los diferentes apartados que se proponen en el enunciado de la práctica.

Apartado i:

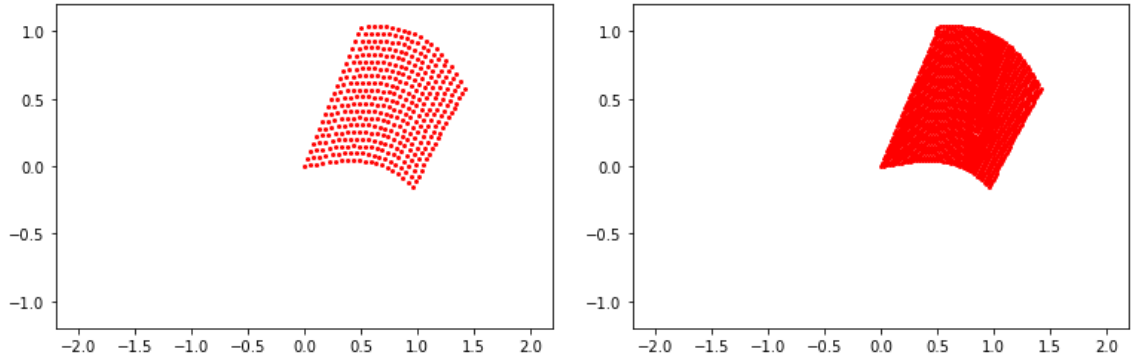
Para este apartado se define la función simplectica, que hace uso de la función orb para dibujar una órbita del espacio fásico. De esta forma, con una granularidad de 10^{-4} obtenemos 100 órbitas para representar el espacio fásico.



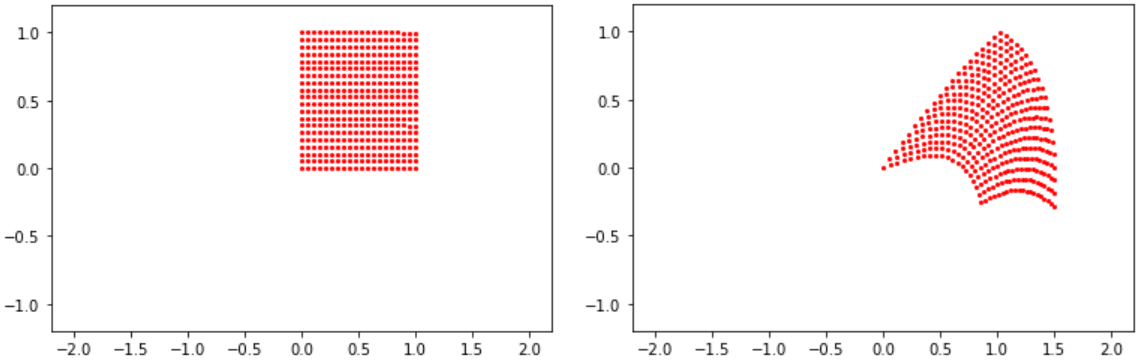
Apartado ii:

Para la realización de este apartado definimos la función *evoluciondf*, que recibe como parámetros el tiempo t, la granularidad d y el número de puntos por fila y columna N. Esta función calcula y representa D_t con el número de puntos y granularidad dados y calcula su área aproximada.

Primeramente, la ejecutamos en t = ¼, d = 10^{-4} y N = 20 y obtenemos la imagen de la izquierda, con un área de 0.9999804270062724. Después, con t = ¼, d = 10^{-5} y N = 40, aumentamos la precisión y obtenemos la imagen de la derecha y un área de 0.9999867080519315. Podemos asumir entonces que la precisión nos permite tomar solo hasta el cuarto decimal pues la diferencia es 6.28 10^{-6}. De esta forma, obtenemos el intervalo de error [0.999973, 0.999993].



De la misma manera, a modo de prueba, calculamos en t = 0.001, d = 10^{-4} y N = 20 y el t = ½, d = 10^{-4} y N = 20, obteniendo las imágenes de abajo y áreas de 1 y 1.00003 respectivamente.



En definitiva, como la precisión solo alcanza el cuarto decimal podemos decir que el área calculada es en D_0 igual que en $D_{¼}$ y que en $D_{½}$ y vale 1, cumpliéndose el teorema de Liouville.

Por otro lado, como se puede apreciar al tomar $D_{(0,\infty)}$, el área es mucho mayor que 1, por lo que en este caso no se cumple.

Apartado iii:

La animación GIF generada por nuestro código se adjunta a la entrega junto con el propio código.

Conclusiones

En definitiva, podemos concluir que en este caso se cumple el teorema de Liouville pues se mantiene constante el área del rectángulo cuando se va deformando con el tiempo. Podemos concluir también empíricamente que se trata de un sistema conservativo, pues en el diagrama de fases se observa que las órbitas son cerradas.

Código

A continuación, se incluyen capturas del código en Python usado para la práctica. Asimismo, este código será subido en un archivo .py junto con este documento.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial import ConvexHull, convex_hull_plot_2d

def F(q):
    ddq = -8/3*q*(q**2-1/2)
    return ddq

def orb(n,q0,dq0,F, args=None, d=0.001):
    #q = [0.0]*(n+1)
    q = np.empty([n+1])
    q[0] = q0
    q[1] = q0 + dq0*d
    for i in np.arange(2,n+1):
        args = q[i-2]
        q[i] = - q[i-2] + d**2*F(args) + 2*q[i-1]
    return q #np.array(q),

#q = variable de posición, dq0 = \dot{q}(0) = valor inicial de la derivada
#d = granularidad del parámetro temporal
def deriv(q,dq0,d):
   #dq = np.empty([len(q)])
   dq = (q[1:len(q)]-q[0:(len(q)-1)])/d
   dq = np.insert(dq,0,dq0) #dq = np.concatenate(([dq0],dq))
   return dq


#gráfico del oscilador no lineal
q0 = 0.
dq0 = 1.
fig, ax = plt.subplots(figsize=(12,5))
#plt.ylim(0, 1)
plt.rcParams["legend.markerscale"] = 6
ax.set_xlabel("t = n $\delta$", fontsize=12)
ax.set_ylabel("q(t)", fontsize=12)
iseq = np.array([3,4])
horiz = 32
for i in iseq:
    d = 1./10**i
    n = int(horiz/d)
    t = np.arange(n+1)*d
    q = orb(n,q0=q0,dq0=dq0,F=F,d=d)
    plt.plot(t, q, 'ro', markersize=0.5/i,label='$\delta$ ='+
             str(np.around(d,4)),c=plt.get_cmap("winter")(i/np.max(iseq)))
    ax.legend(loc=3, frameon=False, fontsize=12)

d = 1./10**4
n = int(horiz/d)
t = np.arange(n+1)*d
q = orb(n,q0=q0,dq0=dq0,F=F,d=d)
dq = deriv(q,dq0=dq0,d=d)
p = dq/2
```

```
#grafica derivada de q(t)
fig, ax = plt.subplots(figsize=(12,5))
#plt.ylim(-1.5, 1.5)
plt.rcParams["legend.markerscale"] = 6
ax.set_xlabel("t = n $\delta$", fontsize=12)
ax.set_ylabel("dq(t)", fontsize=12)
plt.plot(t, dq, '-')

#Ejemplo de diagrama de fases (q, p) para una órbita completa
fig, ax = plt.subplots(figsize=(5,5))
#plt.xlim(-1.1, 1.1)
#plt.ylim(-1, 1)
plt.rcParams["legend.markerscale"] = 6
ax.set_xlabel("q(t)", fontsize=12)
ax.set_ylabel("p(t)", fontsize=12)
plt.plot(q, p, '-')
plt.show()


#####################################################################
#  ESPACIO FÁSICO
#####################################################################

def simplectica(q0,dq0,F,col=0,d = 10**(-4),n = int(16/d),marker='-'):
    q = orb(n,q0=q0,dq0=dq0,F=F,d=d)
    dq = deriv(q,dq0=dq0,d=d)
    p = dq/2
    plt.plot(q, p, marker,c=plt.get_cmap("winter")(col))


Horiz = 12
d = 10**(-4)

fig = plt.figure(figsize=(8,5))
fig.subplots_adjust(hspace=0.4, wspace=0.2)
ax = fig.add_subplot(1,1, 1)
#Condiciones iniciales:
seq_q0 = np.linspace(0.,1.,num=10)
seq_dq0 = np.linspace(0.,2.,num=10)
for i in range(len(seq_q0)):
    for j in range(len(seq_dq0)):
        q0 = seq_q0[i]
        dq0 = seq_dq0[j]
        col = (1+i+j*(len(seq_q0)))/(len(seq_q0)*len(seq_dq0))
        #ax = fig.add_subplot(len(seq_q0), len(seq_dq0), 1+i+j*(len(seq_q0)))
        simplectica(q0=q0,dq0=dq0,F=F,col=col,marker='-',d= 10**(-4),n = int(Horiz/d))
ax.set_xlabel("q(t)", fontsize=12)
ax.set_ylabel("p(t)", fontsize=12)
#fig.savefig('Simplectic.png', dpi=250)
plt.show()
```

```
def evoluciondf (t, d, N):
    ax = fig.add_subplot(1,1, 1)
    seq_q0 = np.linspace(0,1,num=N)
    seq_dq0 = np.linspace(0,2,num=N)
    q2 = np.array([])
    p2 = np.array([])
    q3 = np.array([])
    p3 = np.array([])
    q4 = np.array([])
    p4 = np.array([])
    for i in range(N):
        for j in range(N):
            q0 = seq_q0[i]
            dq0 = seq_dq0[j]
            n = int(t/d)
            q = orb(n,q0=q0,dq0=dq0,F=F,d=d)
            dq = deriv(q,dq0=dq0,d=d)
            p = dq/2
            q2 = np.append(q2,q[-1])
            p2 = np.append(p2,p[-1])
            if(j == 0):
                q3 = np.append(q3,q[-1])
                p3 = np.append(p3,p[-1])
            if(i == N-1):
                q4 = np.append(q4,q[-1])
                p4 = np.append(p4,p[-1])

            plt.xlim(-2.2, 2.2)
            plt.ylim(-1.2, 1.2)
            plt.rcParams["legend.markerscale"] = 6
            ax.set_xlabel("q(t)", fontsize=12)
            ax.set_ylabel("p(t)", fontsize=12)
            plt.plot(q[-1], p[-1], marker="o", markersize= 2,
                     markeredgecolor="red",markerfacecolor="red")

    plt.show()

    X = np.array([q2,p2]).T
    hull = ConvexHull(X)
    #convex_hull_plot_2d(hull)

    Y = np.array([q3,p3]).T
    Y_hull = ConvexHull(Y)
    #convex_hull_plot_2d(Y_hull)

    Z = np.array([q4,p4]).T
    Z_hull = ConvexHull(Z)
    #convex_hull_plot_2d(Z_hull)

    print("Perímetro:", hull.area)
    print("Área:", hull.volume- Y_hull.volume- Z_hull.volume)
    #print("Vértices:", X[hull.vertices])

evoluciondf(0.25, 10**(-4), 20)
evoluciondf(0.25, 10**(-5), 40)
evoluciondf(0.001, 10**(-4), 20)
evoluciondf(0.5, 10**(-4), 20)
```

```
from matplotlib import animation

def animate(t):

    ax = plt.axes()
    ax.clear()
    d = 10**(-4)
    if (t == 0):
        t += 0.001
    n = int(t/d)
    N = 100
    seq_q0 = np.linspace(0,1,num=N)
    seq_dq0 = np.linspace(0,2,num=N)

    for i in range(N):
        for j in range(N):
            if i == 0 or i == N-1 or j == 0 or j == N-1:
                q0 = seq_q0[i]
                dq0 = seq_dq0[j]
                q = orb(n,q0=q0,dq0=dq0,F=F,d=d)
                dq = deriv(q,dq0=dq0,d=d)
                p = dq/2
                plt.xlim(-2.2, 2.2)
                plt.ylim(-1.2, 1.2)
                plt.rcParams["legend.markerscale"] = 6
                ax.set_xlabel("q(t)", fontsize=12)
                ax.set_ylabel("p(t)", fontsize=12)
                ax.plot(q[-1], p[-1], marker="o", markersize= 1,
                        markeredgecolor="red",markerfacecolor="red")
    return ax,

fig = plt.figure(figsize=(6,6))
ani = animation.FuncAnimation(fig, func=animate,
                              frames=np.arange(0,5,0.1), interval=100)
ani.save("animation.gif", fps = 8)
```