PRÁCTICA 3:
Teorema de Liouville

Laura Cano Gómez (U2)

22 de marzo de 2024

1. Introducción

El objetivo de esta práctica es obtener información sobre un oscilador no lineal concreto, cuyo hamiltoniano describe una variedad simpléctica $(q(t), p(t)) \in \mathbb{R}^2$, con $H(q, p) = p^2 + \frac{1}{4} \cdot (q^2 - 1)^2$, para estudiar cómo es y qué propiedades satisface.

En particular, este trabajo consiste en representar el espacio fásico $D_{(0, \infty)}$ del sistema, con condiciones iniciales dadas y considerando 20 órbitas diferentes, obtener el valor del área de $D_{1/3}$ y una estimación del error, determinar si se cumple el teorema de Liouville entre D_0 y D_t, y generar una animación GIF del diagrama de fases D_t para $t \in (0, 5)$.

2. Material utilizado

Los recursos utilizados en este trabajo se han obtenido del campus virtual de la asignatura, en concreto, los ficheros *Plantilla de la practica 3.py* y *Plantilla de una animacion GIF.py*, de los que se han extraído código e ideas para la implementación de esta práctica.

Por otro lado, se han utilizado las librerías de Python *numpy* para estructuras de datos, *matplotlib* para representar los resultados y crear el GIF y *scipy.spatial* para los cálculos de las componentes convexas.

A continuación, se presentan las funciones incorporadas:

- **orb()**, **deriv()** y **F()**: obtiene la orbita de la ecuación dinámica y define la función oscilador no lineal pedida.
- **simplectica()**: funcion auxiliar para representar gráficamente el diagrama de fases.
- **phase_diagram()**: obtiene las condiciones iniciales y muestra gráficamente el diagrama de fases llamando a la función *simplectica()*.
- **area_aux()**: funcion auxiliar para calcular el área en la región determinada de entrada.
- **calculate_area()**: calcula el área total aplicando la función *area_aux* a cada componente convexa del cuadrado deformado.
- **animate()**: genera la animación para crear del gif, calculando los puntos para cada valor concreto de t. Para generar el gif, utilizamos fuera de la función el paquete *Animation* de la librería *Matplotlib*, que guarda el .gif en la carpeta donde se encuentre el archivo .py que ejecutamos.
- **main()**: devuelve el resultado de los tres apartados pedidos en la práctica al ejecutar el *.py*, imprimiendo por pantalla los enunciados y sus resultados.

3. Resultados

3.1. Apartado i)

Representamos gráficamente el diagrama de fases para 20 órbitas con las condiciones iniciales $D_0 := [0, 1] \times [0, 1]$, y una granularidad del parámetro temporal $t = n\delta$, tal que $\delta \in \left[10^{-4}, 10^{-3}\right]$ y $n \in \mathbb{N} \cup \{0\}$, obteniendo la *Figura 1*.

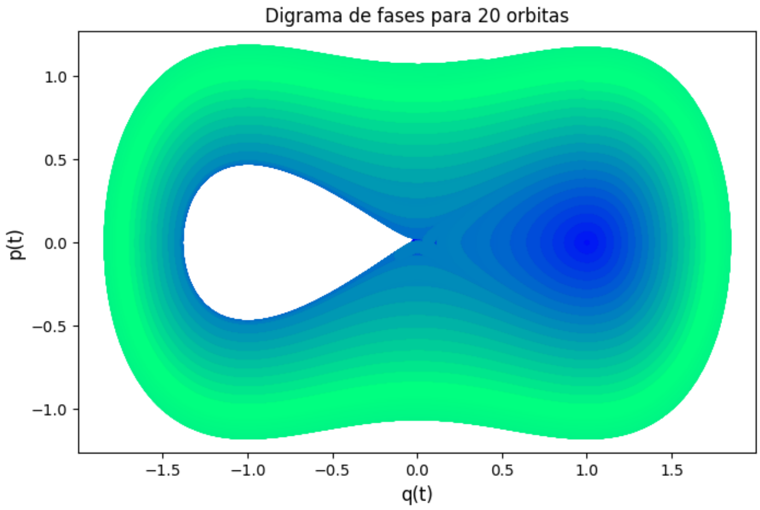


Figura 1: Diagrama de fases

3.2. Apartado ii)

Obtenemos el valor del área de $D_{1/3}$ para $\delta = 10^{-3}$ y $\delta = 10^{-4}$, estimando el error cometido. Para ello, calculamos el área de la componente convexa de la poligonal, *Figura 2*, y le restamos el área de la componente convexa inferior, *Figura 3*, y la componente convexa derecha, *Figura 4*, obteniendo los siguientes resultados:

- **Para $\delta = 10^{-3}$:** 1.0003 ± 0.0004
- **Para $\delta = 10^{-4}$:** 0.9999 ± 0.0004

Por tanto, podemos concluir que el área de $D_{1/3}$ es 1.

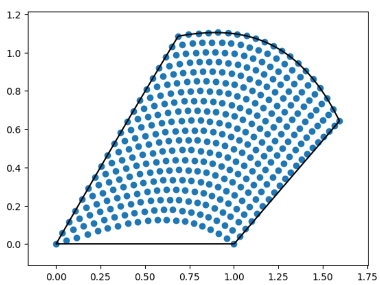


Figura 2: Comp. Convx. Pol.

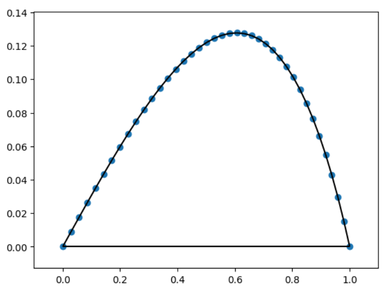


Figura 3: Comp. Convx. Inf.

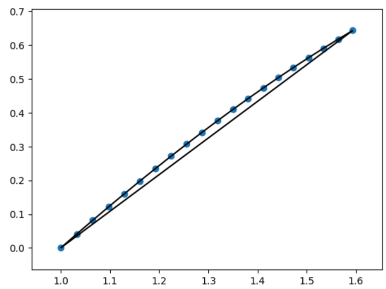


Figura 4: Comp. Convx. Der.

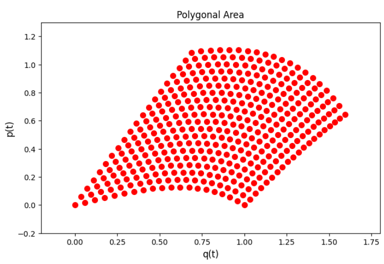


Figura 5: Área Pol.

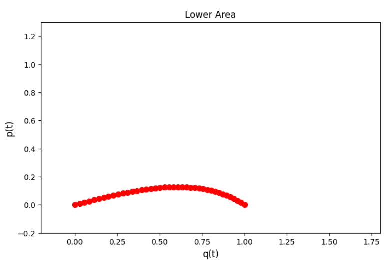


Figura 6: Área Inf.

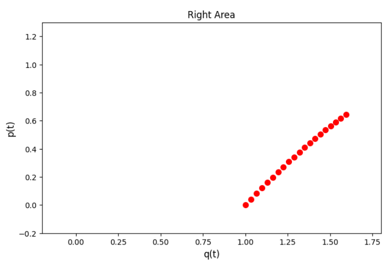


Figura 7: Área Der.

El Teorema de Liouville nos dice que el 2n-volumen de la distribucion de fases es invariante en el tiempo. En este caso, al estar en $\mathbb{R}^2$, el volumen es equivalente al área. Para D_0 el área es 1, por ser un cuadrado $[0,1] \times [0,1]$, y por los resultados obtenidos, $D_{1/3}$ es también 1, por tanto, concluimos que se satisface el Teorema de Liouville.

3.3. Apartado iii)

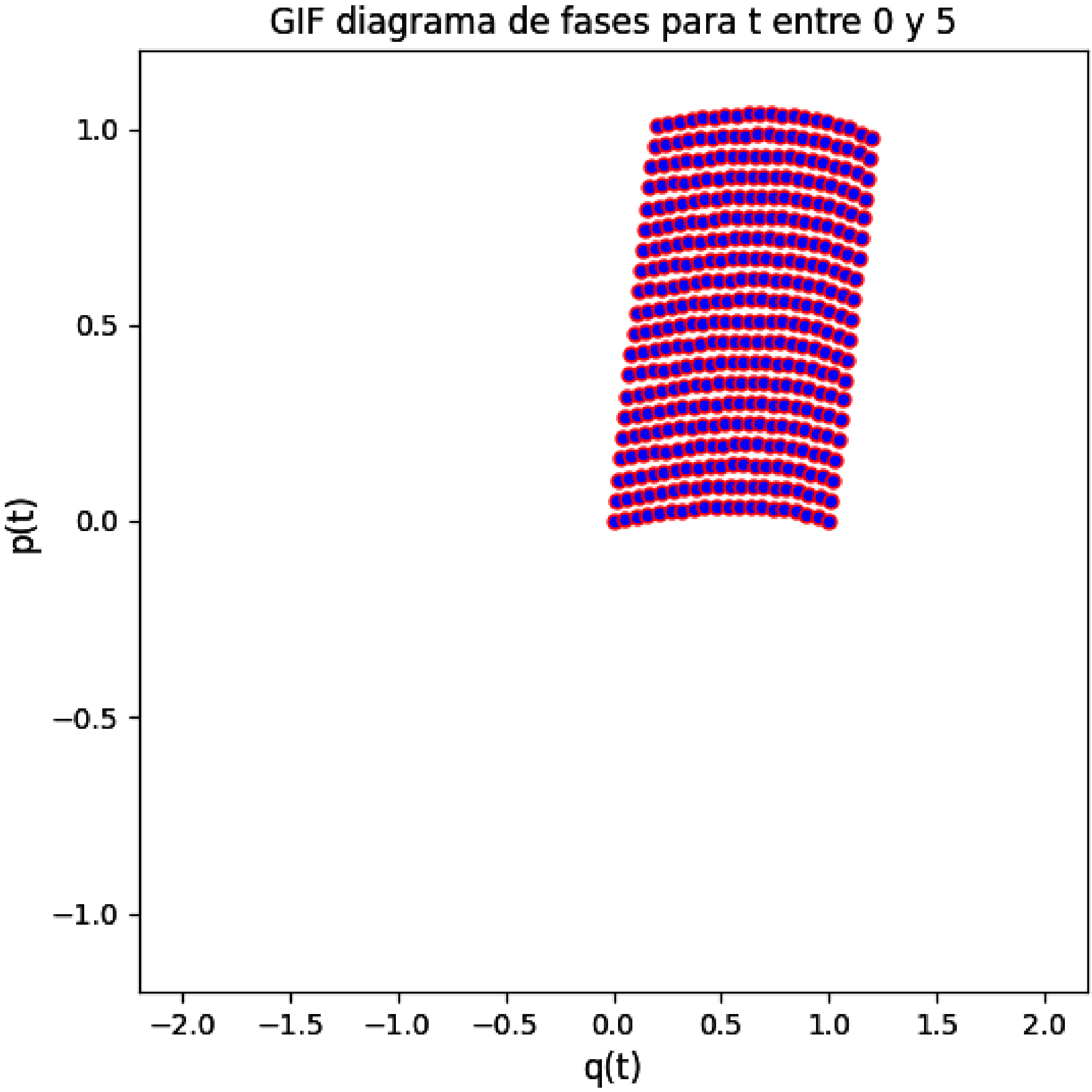
Se obtiene el gif del diagrama de fases para $t \in (0,5)$, usando el paquete *Animation* de la librería *Matplotlib* y la función implementada *animate()*. Se adjunta como archivo .gif y como *Anexo 1* en esta memoria (debe abrirse con *Adobe Acrobat* para poder visualizarlo correctamente).

4. Conclusión

Según el diagrama de fases obtenido en la *Figura 1*, para $D_{(0,\infty)}$, el área no es 1. Esto se debe a que el área no se conserva en todos los instantes simultáneamente, sin embargo, según hemos comprobado en el segundo apartado, sí se satisface el Teorema de Liouville entre D_0 y $D_{1/3}$, es decir, que el 2n-volumen de la distribución de fases es invariante en el tiempo.

Por último, observamos que se obtiene mejor precisión del área para un δ menor, por tanto, el resultado experimenta una tendencia a converger hacia un valor concreto, que en nuestro caso es 1.

5. Anexo 1 - Gif del apartado iii)



6. Anexo 2 - Script/código utilizado

```
"""
Practica 3. TEOREMA DE LIOUVILLE

Alumna: Laura Cano Gomez
Subgrupo: U2
"""

#import os
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial import ConvexHull, convex_hull_plot_2d
from matplotlib.animation import FuncAnimation
#os.getcwd()


##########################################################################
#  Funciones auxiliares de la plantilla
##########################################################################

def orb(n, q0, dq0, F, args=None, d=0.001):
    """
    Obtiene la orbita de la ecuacion dinamica

    Returns a array object (q)

    Arguments:
        n   -> int (numero de variables de estado)
        q0  -> float (variable de s)
        dq0 -> float (valor inicial de la derivada)
        F   -> function (funcion oscilador no lineal)
        d   -> float (granularidad del parametro temporal)
    """
    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


def deriv(q, dq0, d):
    """
    Returns an array object (dq)

    Arguments:
        q   -> array (variable de posicion)
        dq0 -> array (valor inicial de la derivada)
        d   -> float (granularidad del parametro temporal)
    """
    dq = ( q[1 : len(q)] - q[0 : (len(q)-1)] ) / d
    dq = np.insert(dq, 0, dq0)

    return dq


def F(q):
    """
    Funcion oscilador no lineal pedida

```

```
    Returns a array object

    Arguments:
        q   -> array (variable de posicion)
    """
    k = 2
    ddq = - k * q * (q**2 - 1)

    return ddq



#################################################################################
#  APARTADO I)
#################################################################################

def simplectica(q0, dq0, F, col= 0, d= 10**(-4), n = int(16/(10**(-4))),
    marker='-'):
    """
    Representa el diagrama de fases

    Arguments:
        q0     -> float (variable de s)
        dq0    -> float (valor inicial de la derivada)
        F      -> function (funcion oscilador no lineal)
        col    -> int
        d      -> float (granularidad del parametro temporal)
        n      -> int (numero de variables de estado)
        marker -> string (representacion de los puntos en la grafica)
    """
    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))


def phase_diagram(d, Horiz, n_orbit, show_it= False):
    """
    Obtiene las condiciones iniciales y dibuja el diagrama de fases llamando
        a simplectica()

    Arguments:
        d        -> float (granularidad del parametro temporal)
        Horiz    -> float
        n_orbit -> int
    """
    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= n_orbit)
    seq_dq0 = np.linspace(0., 2, num= n_orbit)
    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))

            simplectica(q0= q0, dq0= dq0, F= F, col= col, marker= 'ro', d=
                10**(-3), n = int(Horiz/d))

    if show_it == True:
```

```
        ax.set_xlabel("q(t)", fontsize= 12)
        ax.set_ylabel("p(t)", fontsize= 12)
        plt.title(f"Digrama de fases para {n_orbit} orbitas")

        plt.show()



#############################################################################
#  APARTADO II)
#############################################################################

def area_aux(t, seq_q0, seq_dq0, d, title, show_it= True): # Plantilla
    """
    Calcula el area en la region determinada de entrada

    Returns a float object (subarea)

    Arguments:
        seq_q0  -> list (numero de variables de estado)
        seq_dq0 -> float (variable de s)
        d       -> float (granularidad del parametro temporal)
        title   -> string (titulo de la grafica)
        show_it -> bool (si queremos que lo represente o no)
    """
    if show_it:
        fig, ax = plt.subplots(figsize=(8,5))

    horiz = t

    # Area
    q2 = np.array([])
    p2 = np.array([])
    for i in range(len(seq_q0)):
        for j in range(len(seq_dq0)):
            q0 = seq_q0[i]
            dq0 = seq_dq0[j]
            n = int(horiz/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 show_it:
                plt.xlim(-0.2, 1.8)
                plt.ylim(-0.2, 1.3)
                plt.rcParams["legend.markerscale"] = 6
                ax.set_xlabel("q(t)", fontsize=12)
                ax.set_ylabel("p(t)", fontsize=12)
                plt.title(title)
                plt.plot(q[-1], p[-1], marker="o", markersize= 7,
                   markeredgecolor="red",markerfacecolor="red")

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

    if show_it:
        #plt.show()
        convex_hull_plot_2d(hull)

    return(hull.volume)

```

```python

def calculate_area(d, t= 1/3, show_it= True):
    """
    Calcula el area total aplicando area_aux a cada componente convexa

    Returns a float object (the area)

    Arguments:
        d -> float (granularidad del parametro temporal)
        t -> float (parametro temporal)
        show_it -> bool (si queremos que lo represente o no)

    """
    polygonal_area = area_aux(t, np.linspace(0., 1., num=20), np.linspace
        (0., 2., num=20), d, "Polygonal Area", show_it)
    lower_area = area_aux(t, np.linspace(0., 1., num=20), [0], d, "Lower
        Area", show_it)
    right_area = area_aux(t, [1], np.linspace(0., 2., num=20), d, "Right
        Area", show_it)

    area = polygonal_area - (lower_area + right_area)

    return area



#############################################################################
#  APARTADO III)
#############################################################################

fig = plt.figure(figsize=(6,6))

def animate(t):
    """
    Funcion auxiliar para la creacion del gif, calcula los puntos concretos
    para cada valor de t.

    Arguments:
        t -> float (parametro temporal)
    """

    horiz = t

    ax = fig.add_subplot(1, 1, 1)
    seq_q0 = np.linspace(0.,1.,num=20)
    seq_dq0 = np.linspace(0.,2,num=20)
    q2 = np.array([])
    p2 = np.array([])
    for i in range(len(seq_q0)):
        for j in range(len(seq_dq0)):
            q0 = seq_q0[i]
            dq0 = seq_dq0[j]
            d = 10**(-3)
            n = int(horiz / 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])
            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.set_title("GIF diagrama de fases para t entre 0 y 5")
            ax.plot(q[-1], p[-1], marker="o", markersize= 5, markeredgecolor="red", markerfacecolor="blue")

    return ax


# Creacion del gif usando el paquete Animation de Matplotlib
anim = FuncAnimation(fig, animate, np.arange(0.1, 5.1, 0.1), interval= 60)
anim.save("GIF.gif", fps = 10)



############################################################################
#  RESULTADOS
############################################################################

def main():

    print('APARTADO I)')
    '''
    Representa graficamente el espacio fasico D(0, inf) de las orbitas
    finales del sistema S con las condiciones iniciales D0. Considera al
    menos 20 orbitas finales diferentes.
    '''
    n_orbit = 20

    d = 10**(-3)
    Horiz = 12
    phase_diagram(d, Horiz, n_orbit, True)


    print('APARTADO II)')
    '''
    - Obten el valor del area de D para t = 1/3 y una estimacion de su
    intervalo de error, presentando los valores de forma cientificamente
    formal.
    - Razona si se cumple el teorema de Liouville entre D0 y Dt.
    '''
    d1 = 10**(-3)
    d2 = 10**(-4)
    t = 1/3

    a1 = calculate_area(d1, t)
    a2 = calculate_area(d2, t, show_it= False)

    print("El area para t= 1/3 es:")
    print(f'    Para d = 10**(-3): {a1}')
    print(f'    Para d = 10**(-4): {a2}')

    print(f'\nEstimacion del error: {abs(a1-a2)}')


    '''
    APARTADO III)
    Realiza una animacion GIF del diagrama de fases Dt para t en (0, 5).
    '''
    # Se hace fuera de la funcion para generar el gif correctamente


if __name__ == '__main__':
```

```
    main()
```