PRÁCTICA 3 – GEOMETRÍA COMPUTACIONAL – 2023

PABLO FERNÁNDEZ DEL AMO

1. INTRODUCCIÓN:

El objetivo de este ejercicio es aplicar los conocimientos del Tema 3 para estudiar el comportamiento de un oscilador no lineal. Se debe graficar el espacio fásico del sistema con ciertas condiciones iniciales y granularidad temporal dada, calcular el área de una región del espacio fásico para un tiempo dado y estimar su intervalo de error para verificar si se cumple el teorema de Liouville, y realizar una animación GIF que muestre la evolución del espacio fásico en el tiempo.

2. MATERIAL USADO:

Para resolver el problema se han utilizado las matemáticas y la física involucradas en el sistema, junto con las librerías NumPy, Matplotlib y Scipy.spatial. NumPy para realizar cálculos numéricos, Matplotlib para la visualización de datos y la creación del gif y Scipy.spatial para realizar operaciones espaciales como la estimación de áreas. Además, también se ha utilizado las plantillas de Python proporcionadas por Robert para facilitar la implementación de las ecuaciones y la visualización de los resultados.

3. RESULTADOS:

i) La Figura 2 muestra el espacio fásico D(0, ∞), que representa todas las órbitas que parten del estado inicial D0 hasta un tiempo t = 10.

Se observan las 10 órbitas solicitadas, cada una representada por un color diferente. Es importante destacar que las órbitas corresponden a los puntos en el espacio fásico que se alcanzan en el tiempo t = 10.

Cabe mencionar que el espacio fásico es un espacio abstracto utilizado en mecánica clásica, que permite representar de forma compacta y completa el estado dinámico de un sistema físico. En este contexto, una órbita se define como la trayectoria que describe un punto en el espacio fásico durante su evolución temporal.

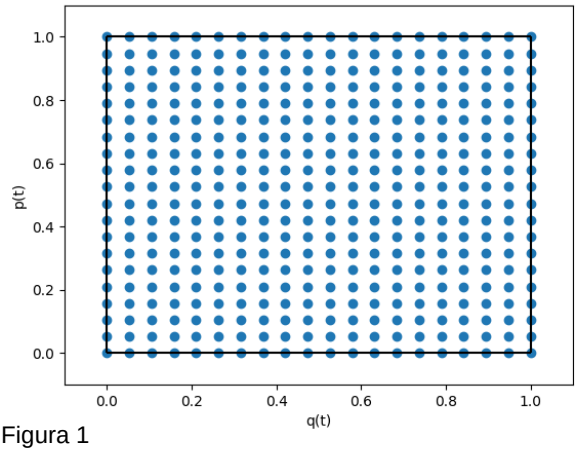

Figura 1

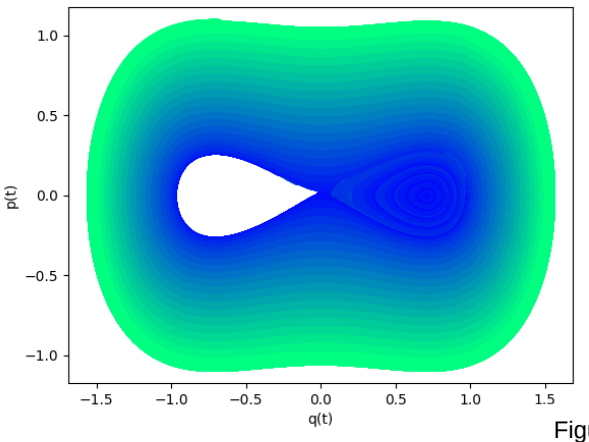

Figura 2

ii) El área de la región D_0 (Figura 1), definida como el rectángulo unitario [0,1] x [0,1], es trivialmente igual a 1. Esto es coherente con el área obtenida a partir de la envolvente convexa de D_0. Sin embargo, el cálculo del área de la región D_t, para un valor de t específico, es más complicado debido a que la envolvente convexa contiene puntos que no pertenecen a D_t. En consecuencia, para t = ¼ (Figura 5), debemos restar el área de la envolvente convexa total menos las envolventes convexas de los lados inferior (Figura 3) y derecho (Figura 4).

Para reducir el error de cálculo, se aplicará una técnica de aproximación iterativa que consiste en disminuir el valor de delta de manera geométrica hasta que la diferencia de áreas utilizando dos valores de delta consecutivos sea menor que un umbral de 0.7, elegido en este caso.

Utilizando esta metodología, se obtiene que el área de Dt para t = ¼ es :

1.00054 ± 1.56676e-04 unidades de área.

Se confirma así que el teorema de Liouville se cumple, ya que se verifica que Área(D_0) = Área($D_{1/4}$).

Sin embargo, se concluye que el teorema de Liouville no se cumple para D(0, ∞), ya que su área, como mínimo, es mayor que la de D_t y D_0, al contener ambos conjuntos.
Se puede comprobar gráficamente comparando las Figuras 1 y 2.

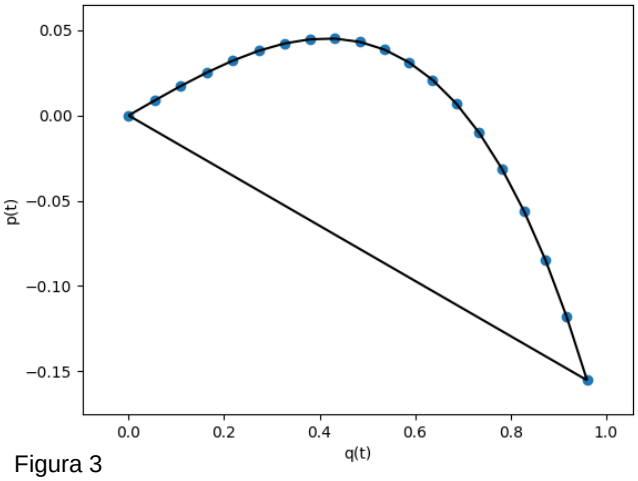

Figura 3

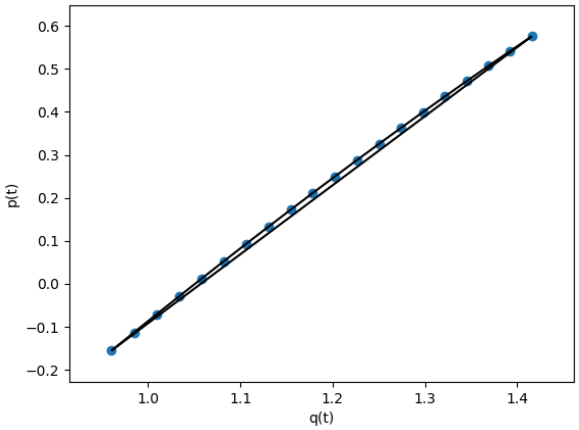

Figura 5

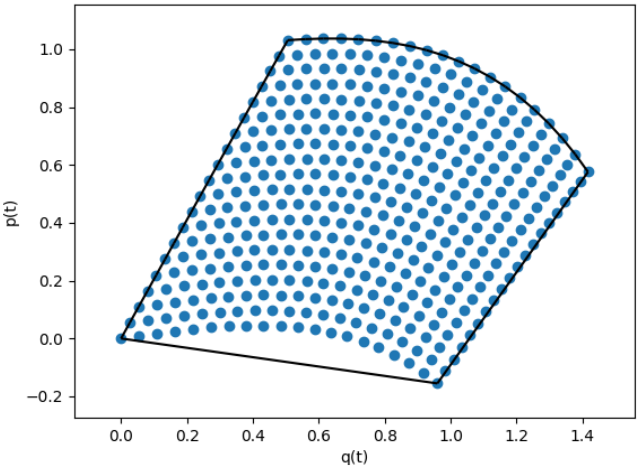

Figura 6

iii) El GIF que corresponde a la resolución del apartado 3 se encuentra adjunta a esta memoria y el código utilizado para confeccionarla.

4. CONCLUSIÓN:

En resumen, el ejercicio trata sobre un oscilador no lineal, cuyas ecuaciones de Hamilton-Jacobi describen la evolución de q(t) y p(t) en función de ciertos parámetros. Se pide representar gráficamente el espacio fásico D(0,∞) de las órbitas finales del sistema con las condiciones iniciales D0, obtener el valor del área de Dt para t=1/4 y estimar su intervalo de error, y realizar una animación GIF con la evolución del diagrama de fases Dt para t en el intervalo (0,5). Los resultados obtenidos indican que se cumple el teorema de Liouville entre D0 y Dt, pero no entre D0 y D(0,∞),

En esta práctica hemos obtenido resultados interesantes sobre el comportamiento del oscilador no lineal descrito por el hamiltoniano H, utilizando diferentes técnicas de análisis y visualización. Estos resultados pueden ser útiles para entender mejor las propiedades de este sistema y para realizar predicciones sobre su comportamiento en diferentes condiciones.

5. **ANEXO:**
   Código de Python utilizado:

```
# -*- coding: utf-8 -*-

"""
PRÁCTICA 3
GEOMETRÍA COMPUTACIONAL

PABLO FERNÁNDEZ DEL AMO"""

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

from matplotlib import animation



a = 3
b = 0.5

# Define a function to compute the derivative of q with respect to time
def deriv(q, dq0, d):
    """
    Compute the derivative of q with respect to time.
    q: numpy array, representing the orbit
    dq0: float, initial derivative
    d: float, time granularity
    """
    dq = (q[1:len(q)] - q[0:(len(q)-1)]) / d
    dq = np.insert(dq, 0, dq0)
    return dq

# Define the system function
def F(q):
    """
    Define the system function.
    q: numpy array, representing the orbit
    """
    return -(8 / a) * q * (q*q - b)

# Compute the first n elements of the orbit q(t), solving dq = F(q)
# with initial conditions q0 and dq0, and time granularity delta
def orb(n, F, q0, dq0, d):
    """
    Compute the first n elements of the orbit q(t), solving dq = F(q)
    n: int, number of elements
    F: function, representing the system
    q0: float, initial position
    dq0: float, initial derivative
    d: float, time granularity
    """
```

```
    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

d = 10 ** (-4)

def simplectica(D, q0, dq0, F, col=0, d=10**(-4), n=int(16/d), marker='-'):
    """
    Compute the simplectic solution for the given parameters.
    D: list of lists, to store the computed values
    q0: float, initial position
    dq0: float, initial derivative
    F: function, representing the system
    col: int, color index
    d: float, time granularity
    n: int, number of elements
    marker: str, marker style
    """
    q = orb(n, F, q0=q0, dq0=dq0, d=d)
    dq = deriv(q, dq0=dq0, d=d)
    p = dq/2
    plt.plot(q, p, marker, c=plt.get_cmap("winter")(col))
    for k in range(n):
        D[k].append([q[k], p[k]])


def figura_Dt(D0, t, d):
    """
    Compute the curve Dt for the given parameters.
    D0: list of lists, representing the initial data
    t: float, time limit
    d: float, time granularity
    """
    n = int(t / d)
    Dt = []
    for q0, p0 in D0:
        dq0 = 2 * p0
        q = orb(n, F, q0, dq0, d)
        dq = deriv(q, dq0, d)
        p = dq / 2
        Dt.append([q[-1], p[-1]])
    return Dt

def area_Dt(D, q0s, p0s, d):
    # Obtain the length of the input lists
    t = 1/4
    length_q0s = len(q0s)
    length_p0s = len(p0s)

    # Calculate the area of the convex hull of Dt
```

```
    hullDt = ConvexHull(D[int(t/d)])

    Area_Dt_ConvHull = hullDt.volume


    D0 = [[q0s[i], p0s[j]] for i in range(length_q0s) for j in range(length_p0s)]
    Dt = figura_Dt(D0, t, d)
    Dt_ConvHull = ConvexHull(Dt)
    Area_Dt_ConvHull = Dt_ConvHull.volume
    """
    ax = plt.axes(xlabel = 'q(t)',ylabel = 'p(t)')
    fig = convex_hull_plot_2d(Dt_ConvHull,ax)
    fig.savefig('Dt.pdf',format='png')
    plt.show()
    """

    # Calculate the area resulting from transforming the line p = 0
    lower_edge0 = [[q0s[i], p0s[0]] for i in range(length_q0s)]
    lower_edget = figura_Dt(lower_edge0, t, d)
    hull_loweredget = ConvexHull(lower_edget)
    area_loweredget = hull_loweredget.volume
    """
    ax = plt.axes(xlabel = 'q(t)',ylabel = 'p(t)')
    fig = convex_hull_plot_2d(hull_loweredget,ax)
    fig.savefig("Lower.pdf",format='png')
    plt.show()
    """

    # Calculate the area resulting from transforming the line q = 1
    right_edge0 = [[q0s[length_q0s - 1], p0s[i]] for i in range(length_p0s)]
    right_edget = figura_Dt(right_edge0, t, d)
    hull_rightedget = ConvexHull(right_edget)
    area_rightedget = hull_rightedget.volume
    """
    ax = plt.axes(xlabel = 'q(t)',ylabel = 'p(t)')
    fig = convex_hull_plot_2d(hull_rightedget,ax)
    fig.savefig('Right.pdf',format='png')
    plt.show()
    """
    # The actual area of Dt is that of its convex hull minus the surplus due to
    # the curvature of the p = 0 and q = 1 edges
    return Area_Dt_ConvHull - area_loweredget - area_rightedget




"""
I) Representa gráficamente el espacio fásico D(0,∞) de las órbitas finales del sistema con
las condiciones iniciales D0. Considera al menos 10 órbitas finales diferentes.
"""

import numpy as np
import matplotlib.pyplot as plt
```

```
# Define the time granularity
d = 10**(-3)

# Define the total time for the simulation
t = 10

# Calculate the number of points for each orbit based on time granularity
n = int(t/d)

# Define the number of initial conditions to explore in the phase space
gran_q0s = 20
gran_p0s = 20

# Create an array of time points for the simulation
ts = np.arange(n)*d

# Create an empty list to store the points for each orbit
D = [[] for i in range(n)]

# Define the initial conditions for the entire phase space
q0s = np.linspace(0,1,gran_q0s)
p0s = np.linspace(0,1,gran_p0s)

# Create a figure to plot the phase space
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

# Plot a curve for each initial condition in the phase space
for i in range(gran_q0s):
    for j in range(gran_p0s):
        # Define a color for the curve based on the initial conditions
        col = (1+i+j*(len(q0s)))/(len(q0s)*len(p0s))

        # Define the initial conditions for the q component of the orbit
        q0 = q0s[i]
        dq0 = 2*p0s[j]

        # Simulate the orbit and plot the points on the phase space
        simplectica(D, q0=q0, dq0=dq0, F=F, d=d, n=n, col=col, marker='ro')



# Add labels to the x and y axes of the phase space plot
plt.xlabel('q(t)')
plt.ylabel('p(t)')

# Save the figure as a PDF and display it
fig.savefig('espacioFasico.pdf', format='png')
plt.show()

"""
II) Obtén el valor del área de Dt para t = 1/4 y una estimación del su intervalo de error,
```

```
presentando los valores de forma científicamente formal. ¿Se cumple el teorema de Liouville
entre D0 y Dt o bien entre D0 y D(0,∞)? Razona la respuesta.
"""

t = 1/4

# Define delta as the time granularity
d = 10**(-3)



# Compute the convex hull of D0 and Dt and calculate their areas
hullD0 = ConvexHull(D[0])

areaD0 = hullD0.volume

areaDt = area_Dt(D, q0s, p0s, d)

hullD0 = ConvexHull(D[0])
ax = plt.axes(xlabel = 'q(t)',ylabel = 'p(t)')
fig = convex_hull_plot_2d(hullD0,ax)
fig.savefig('D0.pdf',format='png')
plt.show()

# Define a function to calculate the error in the approximation of the area of Dt
def error(d, area_fig, diff):
    d /= 2
    areaDtant = area_fig
    areaDt = area_Dt(D, q0s, p0s, d)
    prev_diff = diff
    diff = abs(areaDt - areaDtant)

    if prev_diff == None:
        return error(d, areaDt, diff)
    elif diff / prev_diff >= 0.7:
        return error(d, areaDt, diff)

    else:
        return diff

print("El área de Dt es : {:.5f} y su error asociado es: {:.5e}".format(areaDt,error(d,
areaDt, None)))
"""
III) Realiza una animación GIF con la evolución del diagrama de fases Dt para t ∈ (0, 5).
"""
# Define a function to animate the points of D for each time step
def animate(t,D,m1,m2):
    ax = plt.axes()
    ax.set_xlim(-2,2)
    ax.set_ylim(-2,2)
    for q,p in D[t]:
        ax.plot(q,p,marker ='o',markerfacecolor = 'tab:blue',markeredgecolor = 'tab:blue')
```

```
d = 10**(-3)

fig = plt.figure(figsize = (10,10))
ani = animation.FuncAnimation(fig, animate,range(0,int(5//d), 200), fargs =
(D,gran_q0s,gran_p0s), interval = 20)
ani.save("Final.gif", fps = 5)
```