Teorema de Liouville

Sergio González Montero (U2)

2 de abril de 2024

1. Objetivo

Dado el hamiltoniano de un oscilador no lineal, las ecuaciones de Hamilton-Jacobi nos llevan a $H(q,p) = p^2 + \frac{1}{4}(q^2-1)^2$, $\ddot{q} = -2q(q^2-1)$, describiendo la evolución de $q(t)$ y $p(t) = \frac{\dot{q}}{2}$, con condiciones iniciales $D_0 = [0,1]^2$ y parámetro temporal $t = n\delta$, $\delta \in [10^{-4}, 10^{-3}], n \in \mathbb{N} \cup \{0\}$, se representará el espacio fásico, el valor del área en el tiempo $t = \frac{1}{3}$ con una estimación del error y se verificará que el *Teorema de Liouville* se cumple durante la evolución de la ecuación diferencial anterior. Además, se creará un *GIF* del diagrama de fases D_t para $t \in (0,5)$.

2. Material y datos

Se usaron los siguientes apartados de las notas de Robert Monjo de la asignatura de Geometría Computacional, a día 25/03/2024: Definición 3.2.1. Espacio de fases; Teorema 3.2.1. Liouville; Sección 3.2.3. Computación de simplectomorfismos, Algoritmo 3.2.1. Área del espacio fásico de un oscilador y Ejemplo 3.2.4. Oscilador con dos mínimos.

En cuanto al código, se usaron como base las plantillas *GCOM2024-practica3_plantilla* y *GCOM2023-ejemplo_animacion_esfera*. En él, se han utilizado las librerías siguientes:

1. numpy: operaciones matemáticas y cálculos numéricos
2. matplotlib: para visualización de gráficos y *GIF*
3. scypy.spatial: añade el cálculo y representación del diagrama de la envolvente convexa

3. Resultados

Figura 1: **Phase space and time granularity**

Para $\delta = 10^{-4}$, la granularidad más fina de nuestro intervalo y a priori mejor, nos arroja el resultado de área de $D_{\frac{1}{3}} = 0{,}9999 + -0{,}0004$, con resultados redondeados a la diezmilésima. Esto supone un error inferior al 0,1 %. Tanto para esta aproximación como para las siguientes se ha tenido en cuenta la deformación de las condiciones iniciales, calculando la aproximación real del

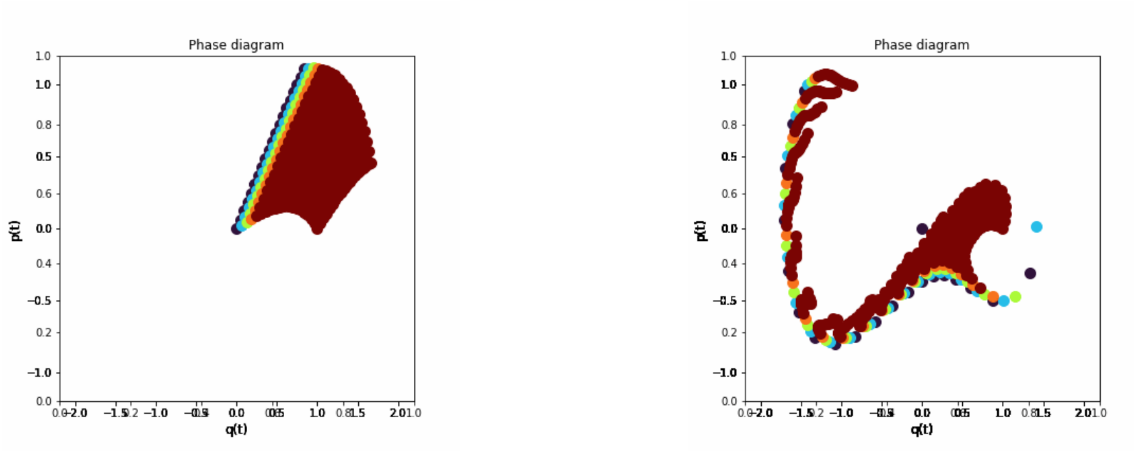

Figura 2: **Phase diagram evolution**

área a través de la envoltura convexa total y haciendo la diferencia con las envolturas de los lados derecho e inferior con el fin de reducir el error de medición.

Se aprecia que el teorema de Liouville se cumple ya que las áreas desde $t = 0$ hasta $t = \frac{1}{3}$ son:

[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.9999, 0.9999, 0.9999, 0.9999, 0.9999], con un error máximo de 0.0001 lo que, de nuevo, supone un error inferior al 0,1 %.

Los errores se han calculado variando la granularidad desde 3.0 hasta 3.9 , calculando el área del diagrama de fases para cada una de ellas y haciendo la diferencia con la medición tomada en $t = \frac{1}{3}$, tomando el máximo de ellas.

Se añaden también dos momentos del *GIF* para apreciar la evolución del sistema.

4. Conclusión

Por un lado, una granularidad más fina conduce a mejores resultados al asemejarse más al caso continuo. Nótese que con un paso aún menor podría darse la situación de que, computacionalmente, hayan oscilaciones y producir inestabilidad en la evolución del sistema debido a los errores de redondeo.

Por otro lado, para verificar la satisfacción del teorema de Liouville, ha bastado con observar la evolución del diagrama de fases para encontrar dónde aplicar de nuevo las envolturas convexas y conseguir una aproximación más fina. Sin embargo, otra evolución más caótica quizá dificulte esta tarea y sea más conveniente usar otras librerías como Polygon, de shapely.

Como se apreciará en el *GIF* y en las capturas anteriores, los puntos $(0, 0)$ y $(1, 0)$ de la malla quedan invariantes con el tiempo, de donde se deduce que se tratan de puntos de equilibrio.

5. Código

Programa 1: practica3.py

```
# Sergio Gonzalez Montero
# Victor Martin Martin

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

from matplotlib import animation

os.getcwd()

# q = variable de posicion, dq0 = \dot{q}(0) = valor inicial de la derivada
# d = granularidad del parametro 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)
   return dq

# Oscilador no lineal
def F(q):
    ddq = -2*q*(q**2-1)
    return ddq

# Resolucion de la ecuacion dinamica \ddot{q} = F(q), obteniendo la orbita q(t)
# Los valores iniciales son la posicion q0 := q(0)
# y la derivada dq0 := \dot{q}(0)
def orb(n, q0, dq0, F, args=None, d=0.001):
    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


print("----------------Espacio fasico----------------\n")
"""
Grafico del espacio de fases
"""
```

```python
def simplectica(q0, dq0, F, col=0, d=10**(-4), n=0, marker='-', plot=False):
    """
    q0 : float; variable de estado
    dq0 : float; derivada de q0
    F : function; funcion del oscilador no lineal
    col : int; controla el color de la linea del grafico
    d : float; longitud de paso en el mallado
    n : int; numero de subintervalos
    marker : patron de graficado
    plot : bool; control de graficado
    """
    q = orb(n, q0=q0, dq0=dq0, F=F, d=d)
    dq = deriv(q, dq0=dq0, d=d)
    p = dq/2
    if plot: plt.plot(q, p, marker, c=plt.get_cmap("turbo")(col))


def espacio_fasico(F, d, plot=False):
    """
    F : function; ecuacion diferencial
    d : float; longitud de paso en el mallado
    plot : bool; control de graficado
    Se plotea el espacio fasico completo
    """
    if plot:
        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))
            simplectica(q0=q0, dq0=dq0, F=F, col=col, d=d, n=int(16/d),
                        marker='o', plot=plot)
    if plot:
        ax.set_xlabel("q(t)", fontsize=12)
        ax.set_ylabel("p(t)", fontsize=12)
        plt.title("Phase space")
        plt.savefig('Phase space.png', dpi=250)
        plt.show()

# CALCULO DE ORBITAS
```

```
"""
Grafico del oscilador con d = [10^-4, 10^-3]
Se busca la mayor granularidad del mallado
"""
q0 = 0.
dq0 = 1.
fig, ax = plt.subplots(figsize=(12, 5))
plt.ylim(-2, 2)
plt.rcParams["legend.markerscale"] = 6
ax.set_xlabel("t = n $\delta$", fontsize=12)
ax.set_ylabel("q(t)", fontsize=12)
iseq = np.linspace(3.,4., num=5)
horiz = 32
for i in iseq:
    d = 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, 'o', markersize=0.5/i,
             label='$\delta$ ='+str(np.around(d, 4)),
             c=plt.get_cmap("turbo")(i/np.max(iseq)))
    plt.rcParams["legend.markerscale"] = 35
    ax.legend(loc=3, frameon=False, fontsize=12)
    plt.title("Time granularity")

plt.savefig('Time granularity.png', dpi=250)
plt.show()
"""
Se detecta en la grafica que el delta que
mejor ajusta es el menor, d = 10^-4
"""

# ESPACIO FASICO
"""
Representacion del espacio fasico para d = 10^-4
"""
d = 10**(-4)
espacio_fasico(F, d, True);


print("\n----------------Area D_(1/3)----------------\n")
"""
t = nd, con t = 1/3; d = 10^-4 --> n = t/d --> n = 3333.33
n = int(horiz/d), con n = 3333.33; d = 10^-4 -->
--> horiz ~ n*d --> horiz = 0.33
"""
```

```
def area_convexa(F, d=10**(-4), horiz=0.33, plot=False, verdad = True):
    """
    F : function; ecuacion diferencial
    d : float; longitud de paso en el mallado
    horiz : int; maximo de pasos temporales
    plot : bool; control de graficado
    return : area de la envolvente convexa
    """
    seq_q0 = np.linspace(0., 1., num=20)
    seq_dq0 = np.linspace(0., 2, num=20)
    q2 = np.array([])
    p2 = np.array([])

    if plot: ax = fig.add_subplot(1, 1, 1)

    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 plot:
        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=10,
                 c=plt.get_cmap("turbo")(i/np.max(iseq)))

    X = np.array([q2,p2]).T
    hull = ConvexHull(X)
    area = hull.volume

    if verdad:

        Y = []
        for i in range(0,400,20):
            Y.append(X[i])
        hull_inf = ConvexHull(Y)
        area_inf = hull_inf.volume
```

```

        Z = []
        for i in range(380,400):
            Z.append(X[i])
        hull_drch = ConvexHull(Z)
        area_drch = hull_drch.volume

        area_total = area-area_inf-area_drch


    if plot:
        convex_hull_plot_2d(hull)
        plt.show()

    if verdad:
        return (round(area_total, ndigits=4),round(area, ndigits=4),
               round(area_inf, ndigits=4),round(area_drch, ndigits=4))
    else:
        return round(area, ndigits=4)

"""
Calculo del area segun d = iseq en t = 1/3, devuelve la maxima
diferencia de cada area con el area del delta con mayor granularidad
"""
# CALCULO DEL AREA
area_tercio = area_convexa(F,10**(-4),0.33)
print(f"Whole convex hull area d = 10^-4, horiz = 0.33:\n\
{round(area_tercio[1], ndigits=4)}\n")
print(f"Lower convex hull area d = 10^-4, horiz = 0.33:\n\
{round(area_tercio[2], ndigits=4)}\n")
print(f"Right Convex hull area d = 10^-4, horiz = 0.33:\n\
{round(area_tercio[3], ndigits=4)}\n")
print(f"Real area approximation d = 10^-4, horiz = 0.33:\n\
{round(area_tercio[0], ndigits=4)}\n")

# CALCULO DEL ERROR
iseq = np.linspace(3., 3.9, num=10)
areas_esp_fas = [area_convexa(F,10**(-i),0.33)[0] for i in iseq]
areas_err = [abs(area_esp_fas-area_tercio[0])
             for area_esp_fas in areas_esp_fas]
area_max_error = np.max(areas_err)
# Area +- error
print("Area D_1/3:", round(area_tercio[0], ndigits=4), "+-",
round(area_max_error,ndigits=4))

# TEOREMA DE LIOUVILLE
```

```
print("\nLiouville theorem")
tseq = np.linspace(1.00969697e-02, 0.33, num=33)
areas_0_t = [area_convexa(F,10**(-4),t)[0] for t in tseq]
print(f"\nAreas from D_0 to D_1/3:\n{areas_0_t}\n")
print("Maximum error:", round(max(abs(np.max(areas_0_t) - 1),
                abs(np.min(areas_0_t) - 1)),ndigits=4))


print("\n----------------GIF animation----------------\n")
def animate(ft):
    seq_q0 = np.linspace(0., 1., num=20)
    seq_dq0 = np.linspace(0., 2, num=20)
    q2 = np.array([])
    p2 = np.array([])

    ax = fig.add_subplot(1, 1, 1)

    horiz = ft
    for i in range(len(seq_q0)):
        for j in range(len(seq_dq0)):
            q0 = seq_q0[i]
            dq0 = seq_dq0[j]
            d = 10**(-4)
            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)
            plt.plot(q[-1], p[-1], marker="o", markersize=10,
                     c=plt.get_cmap("turbo")(i/np.max(iseq)))

    return ax

def init():
    return animate(10)

# Representacion: animacion
fig = plt.figure(figsize=(6, 6))
ani = animation.FuncAnimation(fig, animate, np.arange(0.1, 5, 0.1),
```

```
                                init_func=init, interval=48)
plt.title("Phase diagram")
ani.save("Phase diagram.gif", fps = 30)
plt.close(fig)
print("Done")
```