Práctica 3: Discretización de sistemas dinámicos y teorema de Liouville

Enrique Carro Garrido

I. Introducción (motivación/objetivo de la práctica)

En el estudio de los sistemas dinámicos se presentan problemas interesantes, tales como entender la evolución geométrica de variedades. Muchas veces el comportamiento de estos sistemas dinámicos, dado que no tienen una solución analítica explícita, se puede analizar computacionalmente. En esta práctica pretendemos modelizar esa evolución geométrica de una variedad asociado a un tipo particular de sistema dinámico, el de un oscilador no lineal.

El objetivo de modelizar la evolución geométrica del oscilador es poder representarlo con herramientas de visualización y comprobar que se cumple el Teorema de Liouville mediante aproximaciones del cálculo de áreas.

Se hace hincapié en los problemas del proceso de discretización que se han de tener en cuenta a la hora de trasladar un sistema continuo a un modelo computable, analizando así los errores de medida de los métodos numéricos utilizados tanto en la resolución del sistema dinámico como en el cálculo de áreas.

Finalmente, perseguimos familiarizarnos con herramientas de visualización para poder analizar el proceso de deformación de una variedad por la acción de un sistema dinámico.

II. Material usado (método y datos)

El sistema dinámico que estudiaremos es el de un oscilador lineal. Consideramos así el siguiente hamiltoniano:

$$H : \mathbb{R}^2 \to \mathbb{R}, \tag{1}$$

$$(q, p) \mapsto p^2 + \frac{1}{3}(q^2 - \frac{1}{2})^2 \tag{2}$$

A partir de las ecuaciones de Hamilton-Jacobi se obtiene la siguiente ecuación diferencial

$$\ddot{q} = -\frac{8}{3}q(q^2 - \frac{1}{2}) \tag{3}$$

que describe la evolución de la posición del oscilador, $q(t)$ y del momento, $p(t) = \frac{\dot{q}}{2}$, en función del tiempo, $t \in (0, \infty)$.

Asumimos que disponemos de un conjunto de condiciones iniciales $D_0 := [0, 1] \times [0, 1]$ y una granularidad del parámetro temporal $t = n\delta$ con $\delta \in [10^{-3}, 10^{-4}]$.

El método utilizado para resolver esta ecuación diferencial y obtener así las órbitas del sistema, a partir de las condiciones iniciales $(q(0), p(0)) \in D_0$, es el **Método de Runge-Kutta de orden 4**. La elección de este método se debe al alto orden de precisión a pesar de ser un método explícito de un solo paso, es decir, solo necesita saber $q(n\delta)$ para obtener $q((n + 1)\delta)$, haciendo que sea un método rápido y preciso.

Para calcular el área de D_t para $t = \frac{1}{4}$ se ha utilizado el algoritmo que calcula la **envolvente convexa** implementada en ConvexHull de la librería scipy.spatial.

Finalmente, para realizar la animación GIF con la evolución del diagrama de fases se ha utilizado FuncAnimation de la librería matplotlib.animation.

Los datos utilizados en la práctica son:

- *"GCOM2023_practica3_plantilla.py"*, que contiene ejemplos de representación gráfica del espacio fásico y cálculo de áreas de regiones convexas.
- *"GCOM2022_practica_animacion_esfera.py"*, que aplica un ejemplo de animación GIF con la evolución de la proyección estereográfica y que se utiliza como modelo para el último apartado de esta práctica.

III. Resultados

i) Representa gráficamente el espacio fásico $D_{(0,\infty)}$ de las órbitas finales del sistema con las condiciones iniciales D_0. Considera al menos 10 órbitas finales diferentes.

Para representar el espacio fásico dibujamos como mucho 100 órbitas finales distintas resultantes de tomar una malla de
10×10 puntos equiespaciados en el conjunto
$D_0 := [0,1] \times [0,1]$ como condiciones iniciales de la ecuación diferencial (3). Como las órbitas son periódicas, $\exists n \in \mathbb{N}$ tal que

$$\| \begin{pmatrix} q_n \\ \dot{q}_n \end{pmatrix} - \begin{pmatrix} q_0 \\ \dot{q}_0 \end{pmatrix} \| < \epsilon \tag{4}$$

Tomando $\epsilon = 10^{-3}$ se obtienen al menos 10 órbitas finales, como se puede ver en la Figura 1.

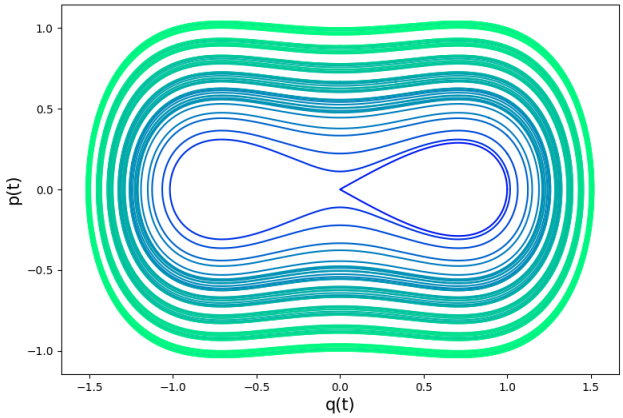
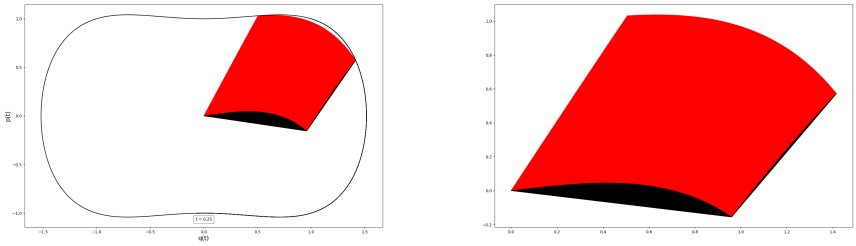


Figura 1: Espacio de fases

ii) Obtén el valor del área de D_t 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 D_0 y D_t o bien entre D_0 y $D_{(0,\infty)}$? Razona la respuesta.

Para calcular $V_{\frac{1}{4}}$:= Área de $D_{\frac{1}{4}}$ se ha de tener en cuenta que el método numérico que calcula el área de la evolución de D_0 hasta $t = \frac{1}{4}$ depende de como de fino sea la partición de D_0 y cuál es el error global de discretización del método de resolución de la ecuación diferencial, que depende del paso δ. Si fijamos la partición de D_0 el error de medida dependerá de la elección de δ, que dado que estamos realizando envolturas convexas basta con realizar la partición en los puntos de la frontera (haciendo que la división pueda ser más fina en menos tiempo, siendo esta alrededor de 2000 puntos por cada lado).



(a) (b)

Figura 2: Procedimiento para calcular el área de $D_{\frac{1}{4}}$ con $\delta = 10^{-4}$

Para estimar el error de la aproximación, ϵ, variamos $\delta \in [10^{-1}, 10^{-2}, ..., 10^{-4}]$ para así obtener una cota superior de ϵ restando la máxima medición obtenida con la mínima.

Para calcular el área de $D_{\frac{1}{4}}$ (que aparece en rojo en la Figura 2), calculamos primero el área de la envolvente convexa de $D_{\frac{1}{4}}$ para después restar las envolventes convexas que sobran (que aparecen en negro en la Figura 2).

Obtenemos los siguientes resultados:

10^{-i}	1	2	3	4
$V_{\frac{1}{4}}$	$V_{\frac{1}{4}}^{min} = 0{,}9999903902847965$	$0{,}9999999920891072$	$0{,}9999999925047366$	$V_{\frac{1}{4}}^{max} = 0{,}9999999925434159$

Por lo tanto, $V_{\frac{1}{4}} = V_{\frac{1}{4}}^{max} \pm (V_{\frac{1}{4}}^{max} - V_{\frac{1}{4}}^{min}) = 0{,}999999993 \pm 9{,}6 \times 10^{-6}$. Como $V_0 = 1$, **se satisface el Teorema de Liouville.**

Por otro lado, el Teorema de Liouville no afirma que $D_{(0,\infty)}$ conserve el área de D_0, por lo que no nos podemos asegurar que esto ocurra. De hecho, como podemos ver en la Figura 3 $D_{\frac{1}{4}} \subset D_{(0,\infty)}$ estrictamente.

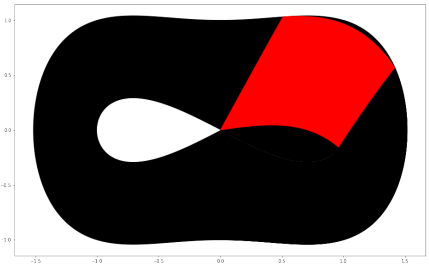


Figura 3: $D_{(0,\infty)}$ contiene a $D_{\frac{1}{4}}$

iii) Realiza una animación GIF con la evolución del diagrama de fases D_t para $t \in (0,5)$.

Reutilizamos la partición de D_0 del apartado anterior, es decir, computamos soluciones únicamente en el borde de D_0, dado que la partición tiene que ser muy fina para que se visualice lo mejor posible la evolución geométrica de D_0. La animación aparece en el vídeo adjunto.

IV. Conclusión

A partir de los resultados obtenidos se puede concluir que los métodos numéricos de resolución de ecuaciones diferenciales y de cálculo de áreas nos permiten comprobar resultados experimentalmente con bastante precisión, tal y como hemos hecho en el apartado ii) al ver que se satisfacía el Teorema de Liouville.

V. Anexo con el código utilizado

```
    """
Enrique Carro Garrido

Práctica 3: Discretización de sistemas dinámicos y Teorema de Liouville
"""

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

# Constante delta que necesitaremos para el primer y último apartado, que define
# la malla de tiempos que utilizaremos para encontrar la solución del sistema
# dinámico utilizando como método de resolución el método de Runge-Kutta de orden 4.
delta = 10**(-3)

# Constante épsilon para obtener la órbita final.
eps = 10**(-3)

# Funciones que utilizaremos en todos los apartados para obtener las órbitas del
# sistema dinámico dadas unas condiciones iniciales.
def F(q):
  """
  Función que define el sistema dinámico que describe la evolución de la posición q(t)
  y del momento p(t) = \dot{q}(t)/2 de un oscilador no lineal.
  """
  return -8/3*q*(q*q - 1/2)

def orb(T, q0, p0, F, delta, eps):
  """
  Resolvemos la ecuación dinámica \ddot{q} = F(q) a partir de las condiciones iniciales
  de la posición, q0 = q(0), y de la velocidad, dq0 = \dot{q}(0).

  La solución (q(t), \dot{q}(t)) describe la órbita (q(t), p(t)) con p(t) = \dot{q}(t)/2.

  Para la resolución de la ecuación dinámica se ha utilizado el método de Runge-Kutta de
  orden 4.

  Args:
      T: Tiempo final del intervalo [0, T] donde está definida la solución.
      q0: q(0)
      p0: p(0)
      F: Función que define el sistema dinámico del oscilador.
      delta: Paso o distancia entre los puntos de la malla.
  """
  # En primer lugar pasamos el sistema a una ecuación diferencial de primer orden
  # equivalente \dot{y} = G(y) con y(t) = (q(t), \dot{q(t)}), t \in [0, T].
  def G(y):
    return np.array([y[1], F(y[0])])

  # Número de puntos en los que se divide de manera equiespaciada el
  # intervalo [0, T] donde está definida la solución, en su proceso de discretización.
  # Si T = 'INF' entonces vamos insertando los puntos hasta que se haya calculado la
  # órbita final, es decir, se acerca al valor inicial para t > 0.
  N = 0 if T == 'INF' else int((T + delta) / delta)
```

```
  # PASO 1: Inicializamos con las condiciones iniciales
  y = np.empty([2, N + 1])
  y[:,0] = [q0, 2*p0]
  # PASO 2: Obtenemos la solución de manera iterativa utilizando en cada paso
  # únicamente el elemento anterior.
  i, final = 0, False
  while (T == 'INF' and not(final)) or (T != 'INF' and i < N):
    f1 = G(y[:,i])
    f2 = G(y[:,i] + delta/2 * f1)
    f3 = G(y[:,i] + delta/2 * f2)
    f4 = G(y[:,i] + delta * f3)
    next = y[:,i] + delta/6*(f1 + 2*f2 + 2*f3 + f4)
    if T == 'INF':
      y = np.insert(y, i + 1, next, axis=1)
      final = np.linalg.norm(y[:,i + 1] - y[:,0]) < eps
    else:
      y[:,i + 1] = next
    i += 1
  return np.array([y[0,:], y[1,:]/2])

def D(D0, F, T, delta, eps):
  """
  Función que calcula el espacio fásico D(0, T) de las órbitas definidas en [0, T] del
  sistema con las condiciones iniciales D0.

  Args:
      D0: Condiciones iniciales
      F: Función que describe el sistema dinámico \ddot{q} = F(q).
      T: Tiempo final del intervalo [0, T] donde está definida la solución.
      delta: El paso entre los nodos de la malla.
  """
  orbitas = list()
  for [q0, p0] in D0:
    new = orb(T, q0, p0, F, delta, eps)
    # Solo añadimos la órbita si no es un solo punto
    orbitas.append([[],[]] if new.shape == (2, 2) else new)
  # Si T = 'INF' entonces las listas tienen distinta dimensión con lo que no
  # pueden ser arrays.
  return np.array(orbitas) if T != 'INF' else orbitas

def D_t(D0, F, delta, t, eps):
  """
  Función que dado un instante t y las condiciones iniciales D0, obtiene el conjunto
  D_t que consiste en evolucionar las condiciones iniciales hasta el instante t.

  Args:
      D0: Condiciones iniciales.
      F: Función que describe el sistema dinámico \ddot{q} = F(q).
      delta: El paso entre los nodos de la malla.
      t: Instante en el que se evalúa el diagrama fásico.

  Return:
      D_t: Array de puntos (q(t), p(t))
  """
  return np.array([[q[-1], p[-1]] for [q,p] in D(D0, F, t, delta, eps)])

# Dado que utilizaremos la órbita más externa en los dos últimos apartados
# para no calcularla más de una vez lo calculamos aquí
orbita_externa = orb('INF', 1, 1, F, delta, eps)

# SOLUCIONES
```

```
# PRIMER APARTADO: Representa gráficamente el espacio fásico D(0,inf) de las órbitas
# finales del sistema con las condiciones iniciales D0. Considera al menos 10 órbitas
# finales diferentes.
fig = plt.figure(figsize=(9,6))
divisiones = 10
seq_q0, seq_p0 = np.linspace(0.,1.,divisiones), np.linspace(0.,1.,divisiones)
D0 = [[q0, p0] for q0 in seq_q0 for p0 in seq_p0]
plt.xlabel("q(t)", fontsize = 15)
plt.ylabel("p(t)", fontsize = 15)
D_inf = D(D0, F, T='INF', delta=delta, eps=eps)
for i in range(divisiones):
  for j in range(divisiones):
    col = (1+i+j*divisiones)/(divisiones*divisiones)
    plt.plot(D_inf[i*divisiones + j][0], D_inf[i*divisiones + j][1], '-',
             c=plt.get_cmap("winter")(col))
fig.savefig('Espacio_fasico.png')

# SEGUNDO APARTADO: Obtén el valor del área de Dt para t = 1/4 y una estimación de
# 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.

# Utilizamos la frontera de D0 para que la partición sea más fina, ya que
# la región convexa de D0 es la de la frontera de D0. Lo utilizaremos también en el
# tercer apartado.
divisiones = 2000
D0 = np.array([(q0, 0) for q0 in np.linspace(0.,1.,num=divisiones)] + \
     [(1, p0) for p0 in np.linspace(0.,1.,num=divisiones)] + \
     [(q0, 1) for q0 in np.linspace(1.,0.,num=divisiones)] + \
     [(0, p0) for p0 in np.linspace(1.,0.,num=divisiones)])

# Variamos la delta para realizar una estimación de su intervalo de error
deltas = [10**(-i) for i in np.arange(1., 5., 1)]
# Fijamos el valor de t=1/4 que es el instante que vamos a estudiar
t = 1/4
# Fijamos el valor del área de D0 := [0,1]x[0,1], que es 1
V0 = 1
# Inicializamos el intervalo de error
areas = []
for d in deltas:
  # Creamos la figura sobre la que dibujaremos el área que estamos calculando y que
  # además ayuda a visualizar el procedimiento que seguimos, esto es, mediante cálculo
  # de áreas de regiones convexas.
  fig, ax = plt.subplots(figsize=(9, 6))
  ax.set_xlabel("q(t)", fontsize = 15)
  ax.set_ylabel("p(t)", fontsize = 15)
  ax.text(0.5, 0.03, "t = 0.25", bbox={'facecolor':'w', 'alpha':0.5, 'pad':5}, transform=ax.transAxes, ha="ce
  # Dibujamos la órbita externa para localizar Dt
  ax.plot(orbita_externa[0], orbita_externa[1], 'black')
  # En primer lugar, obtenemos Dt
  Dt = D_t(D0, F, d, t, eps)
  # Después obtenemos el área de la región convexa, X, que recubre D_t
  X = ConvexHull(Dt)
  A = X.volume
  # Le restamos el área de la región convexa de la parte inferior de X, x1, que no
  # pertenece a D_t
  x1 = ConvexHull(Dt[0:divisiones,:])
  a1 = x1.volume
  # Le restamos el área de la región convexa de la parte derecha de X, x2, que no
  # pertenece a D_t
  x2 = ConvexHull(Dt[divisiones:2*divisiones])
  a2 = x2.volume
  # Dibujamos las regiones convexas que hemos creado en negroy en rojo el área de Dt.
```

```
  ax.add_artist(patches.Polygon(Dt, color='r'))
  ax.add_artist(patches.Polygon(Dt[0:divisiones,:], color='black'))
  ax.add_artist(patches.Polygon(Dt[divisiones:2*divisiones,:], color='black'))
  fig.savefig(f"Dt_con_delta_{d}.png")
  # Dibujamos también Dt pero sin la órbita externa
  fig, ax = plt.subplots(figsize=(9, 6))
  ax.fill(Dt.T[0], Dt.T[1], color = 'r')
  ax.add_artist(patches.Polygon(Dt[0:divisiones,:], color='black'))
  ax.add_artist(patches.Polygon(Dt[divisiones:2*divisiones,:], color='black'))
  fig.savefig(f"Dt_con_delta{d}_zoom.png")
  # Almacenamos el área
  areas.append(A - a1 - a2)

# Escribimos los resultados
print("Áreas obtenidas:", areas)
area_min, area_max = min(areas), max(areas)
print("V0 := Área de D0 =", V0)
print("Vtmáx := Área máxima calculada de Vt =", area_max)
print("Vtmíx := Área mínima calculada de Vt =", area_min)
eps = area_max - area_min
print("Estimación del error: {:.5e}".format(eps))
print("Vt := Área de Dt = {:.6f} +- {:.1e}".format(area_max, eps))
if (abs(V0 - area_max) < eps):
  print("Se satisface el Teorema de Liouville")
else:
  print("No se satisface el Teorema de Liouville")

# TERCER APARTADO: Realiza una animación GIF con la evolución del diagrama de fases
# Dt para t  (0, 5)

# Fijamos el tiempo máximo, T, que evolucionaremos el diagrama de fases
T = 5

# Fijamos los parámetros de la animación para que tarde exactamente 5s
FPS = 30
duration = 10
ani_delta = T / (FPS * duration)

# Obtenemos la evolución del diagrama de fases hasta T = 5, D(0,5)
D_0_5 = D(D0, F, T, delta, eps)

# Creamos la figura y el Axes sobre la que hacemos la animación.
fig, ax = plt.subplots(figsize = (16,10))
ax.set_xlabel("q(t)", fontsize = 15)
ax.set_ylabel("p(t)", fontsize = 15)

# Dibujamos la órbita más externa, que coincide con las condiciones iniciales
# (q0,p0) = (1,1)
ax.plot(orbita_externa[0], orbita_externa[1], 'black')

# Creamos los Artist que vamos a ir modificando a lo largo de la animación ya que,
# por razones de eficiencia, son solamente estas clases las que se modifican en el ax.
polygon = patches.Polygon(D0, color='r')
ax.add_artist(polygon)
title = ax.text(0.5, 0.03, "", bbox={'facecolor':'w', 'alpha':0.5, 'pad':5}, transform=ax.transAxes, ha="cent

# Definimos las funciones que describen la animación.
def animate(t):
  # Obtenemos Dt
  Dt = np.array([[q[int(t * (1/delta))], p[int(t * (1/delta))]]
                 for [q,p] in D_0_5])
  # Modificamos los Artists
  title.set_text("t = " + "{:.2f}".format(t))
```

```
  polygon.set_xy(Dt)
  return polygon, title,

def init():
  return animate(0)

# Creamos la animación
ani = animation.FuncAnimation(fig, animate, np.arange(FPS*duration)*ani_delta,
                              init_func=init,
                              blit= True, repeat=False, interval=1./FPS)

ani.save("Diagrama_de_fases.gif",fps=FPS)
```