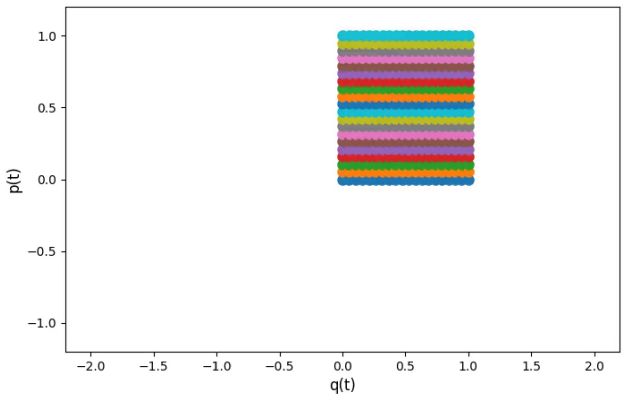
Práctica 6 - GCOMP

Celia Rubio Madrigal

3 de mayo de 2022

Índice



*Para ver este gráfico en movimiento, se recomienda abrir este PDF en Adobe Reader u Okular.

1. Introducción

En esta práctica calcularemos, analizaremos y trazaremos sobre una gráfica la discretización de un sistema dinámico continuo. El sistema que vamos a tratar es el oscilador con dos mínimos, que forma parte de la familia de los osciladores. Son unas ecuaciones muy relevantes y útiles para el mundo de la física, usadas, por ejemplo, en sistemas de muelles, en electrónica, o en cuántica, entre otros campos.

El sistema describe la evolución de la posición y el momento con respecto al tiempo; se trata, por tanto, de una variedad simpléctica con coordenadas $(q(t), p(t)) \in \mathbb{R}^2$. El hamiltoniano de nuestro sistema es: $H(q,p) = p^2 + \frac{1}{4}(q^2-1)^2$. Y se puede deducir que el sistema cumple estas dos ecuaciones: $\ddot{q} = -2q(q^2-1)$, y $p = \frac{\dot{q}}{2}$.

2. Material usado y metodología

Partimos de un conjunto de condiciones iniciales $D_0 = [0,1] \times [0,1]$, que discretizaremos de distintas maneras para cada apartado. Además, discretizaremos el parámetro temporal $t = n\delta \in [0, \infty)$, donde $n \in \mathbb{N}_0$ y δ también dependerá para cada apartado.

Discretizamos las derivadas de q mediante diferencias finitas. Llamando $q_k \coloneqq q(k\delta)$, tenemos expresado q_{n+2} a partir de q_{n+1} y q_n, solo teniéndolo que despejar de la siguiente ecuación:

$$-2q_n \cdot (q_n^2 - 1) = \ddot{q}(t) = \lim_{h \to 0} \frac{\dot{q}(t+h) - \dot{q}(t)}{h} \approx \frac{q(t+2\delta) - 2q(t+\delta) + q(t)}{\delta^2} = \frac{q_{n+2} - 2q_{n+1} + q_n}{\delta^2}$$

2.1. Apartado *i)*

En el primer apartado, fijamos $\delta = 10^{-3}$. Hemos escogido 16 valores en D_0 como nuestras condiciones iniciales. En concreto, una malla equidistante de 4 por 4 en las variables q y p. Consideramos sus órbitas en el sistema.

Llamando al método `simplectica`, se calcula la órbita de cada punto inicial hasta el paso $n = \frac{20}{\delta}$ (ya que no podemos calcular sus valores hasta el infinito) y se dibuja en una gráfica. De esta manera, estamos representando una discretización del espacio fásico $D_{(0,\infty)}$.

2.2. Apartado *ii)*

En el segundo apartado, calculamos el valor del área D_t que ocupan los puntos iniciales, que comenzaron en D_0, en el instante $t = 0.25$. Ahora vamos a tomar 10 puntos por variable, luego tendremos 100 puntos en total.

El teorema de Liouville enuncia que el área de D_t es invariante con respecto a la variable temporal. Como el área de D_0 es 1 (es el cuadrado unidad), el valor que calculemos tendrá que ser también 1, con un posible margen de error.

Para calcular el área, tomaremos el área de la envoltura convexa de los puntos. Le restaremos las dos áreas de las envolturas convexas de las aristas superior y derecha, que se curvan y hacen que la figura no sea convexa.

Además, calcularemos el error que estamos cometiendo al tomar la aproximación discreta, mediante el ajuste de $\delta \in [10^{-4}, 10^{-3}]$, de cuyo intervalo tomaremos 10 valores equidistantes para comparar sus áreas resultantes entre sí.

2.3. Apartado *iii*)

En el tercer apartado, fijaremos de nuevo $\delta = 10^{-3}$. De nuevo, dibujaremos los puntos del sistema para distintos tiempos. Tomaremos una malla equidistante de 20 por 20 en D_0 como condiciones iniciales.

Esta vez los instantes t serán equidistantes en el intervalo $[0, 5]$. Con las 20 gráficas que resulten, realizaremos una animación.

3. Resultados y conclusiones

3.1. Apartado *i*)

En la figura 1 se muestra la gráfica de las 16 órbitas finales de nuestro sistema. Se puede observar que parece tener dos mínimos, como bien indica su nombre.

3.2. Apartado *ii*)

En la figura 2 se muestra la gráfica de los 100 puntos del sistema, que comenzaron como una malla en el cuadrado unidad, en el instante $t = 0.25$ y con $\delta = 10^{-3}$. Los puntos verdes y naranjas forman las dos áreas convexas de los laterales que se han restado al valor del área total.

El valor del área para este δ es de 0.9999 ± 0.0001. Para $\delta = 10^{-4}$, es de 0.9997 ± 0.0001, y entre medias los valores decrecen al decrecer δ. Tomaremos, por tanto, el área total como 0.9998 ± 0.0002, con el valor siendo la media entre ambos valores extremos, y el error su resta. El resultado es muy cercano al valor de 1 que esperábamos, así que hemos verificado que el teorema de Liouville se cumple.

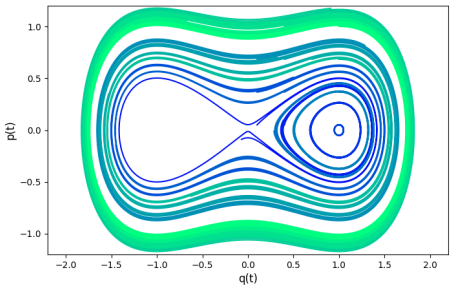


Figura 1: Espacio fásico del sistema.

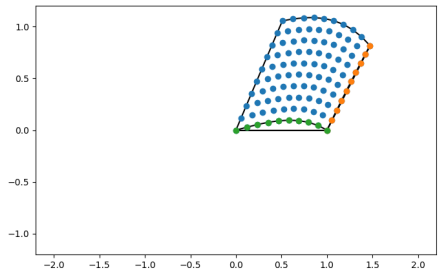


Figura 2: $D_{0.25}$ ($\delta = 10^{-3}$).

3.3. Apartado *iii*)

En los archivos adjuntos, así como en la portada de este trabajo, se encuentra la animación de la sucesión de gráficas de D_t representadas a lo largo de $t \in [0, 5]$.

Gracias a estos cálculos y estas representaciones visuales del espacio de fases, comprendemos mejor cómo se comporta este sistema físico. Con nuestros datos podríamos modelar, comparar y predecir un sistema del mundo real con amplias aplicaciones. Por otro lado, hemos comprobado que se cumple el teorema de Liouville para este sistema simpléctico.

4. Código

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

delta = 10 ** (-3)

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


# Ecuación de un sistema dinámico continuo
def F(q):
    ddq = -2 * q * (q ** 2 - 1)
    return ddq


# Resolución de la ecuación dinámica \ddot{q} = F(q), obteniendo la órbita q(t)
# Los valores iniciales son la posición q0 := q(0) y la derivada dq0 := \dot{q
    }(0)
def orb(n, q0, dq0, F, args=None, d=delta):
    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


######################################################################
#  CÁLCULO DE ÓRBITAS
######################################################################

## Pintamos el espacio de fases
def simplectica(q0, dq0, F, col=0, d=delta, n=int(10 / delta)):
    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, c=plt.get_cmap("winter")(col))


""" Apartado i) """
```

```
fig = plt.figure(figsize=(8, 5))
plt.xlim(-2.2, 2.2)
plt.ylim(-1.2, 1.2)
ax = fig.add_subplot(1, 1, 1)
ax.set_xlabel("q(t)", fontsize=12)
ax.set_ylabel("p(t)", fontsize=12)
# Condiciones iniciales:
seq_q0 = np.linspace(0.1, 1.0, num=4)
seq_dq0 = np.linspace(0.1, 2.0, num=4)
for i, q0 in enumerate(seq_q0):
    for j, dq0 in enumerate(seq_dq0):
        col = (1 + i + j * (len(seq_q0))) / (len(seq_q0) * len(seq_dq0))
        simplectica(
            q0=q0,
            dq0=dq0,
            F=F,
            col=col,
            d=delta,
            n=int(20 / delta),
        )
plt.savefig("1")

""" Apartado ii) """

fig = plt.figure(figsize=(8, 5))
plt.xlim(-2.2, 2.2)
plt.ylim(-1.2, 1.2)
ax = fig.add_subplot(1, 1, 1)
ax.set_xlabel("q(t)", fontsize=12)
ax.set_ylabel("p(t)", fontsize=12)
# Condiciones iniciales:
seq_q0 = np.linspace(0.0, 1.0, num=10)
seq_dq0 = np.linspace(0.0, 2.0, num=10)
t = 0.25
ds = np.linspace(10 ** (-3), 10 ** (-4), num=10)
for d in ds:
    n = int(t / d)
    qall, pall, qbi, pbi, qbj, pbj = (
        np.array([]),
        np.array([]),
        np.array([]),
        np.array([]),
        np.array([]),
        np.array([]),
    )
    for i, q0 in enumerate(seq_q0):
        for j, dq0 in enumerate(seq_dq0):
            q = orb(n, q0=q0, dq0=dq0, F=F, d=d)
```

```
            dq = deriv(q, dq0=dq0, d=d)
            p = dq / 2
            qall = np.append(qall, q[-1])
            pall = np.append(pall, p[-1])
            if i == len(seq_q0) - 1:
                qbi = np.append(qbi, q[-1])
                pbi = np.append(pbi, p[-1])
            if j == 0:
                qbj = np.append(qbj, q[-1])
                pbj = np.append(pbj, p[-1])
    ax.clear()
    totalhull = ConvexHull(np.array([qall, pall]).T)
    convex_hull_plot_2d(totalhull,ax=ax)
    ihull = ConvexHull(np.array([qbi, pbi]).T)
    convex_hull_plot_2d(ihull,ax=ax)
    jhull = ConvexHull(np.array([qbj, pbj]).T)
    convex_hull_plot_2d(jhull, ax=ax)
    area = totalhull.volume - ihull.volume - jhull.volume
    print("Área %f:\t" % d, area)
    plt.xlim(-2.2, 2.2)
    plt.ylim(-1.2, 1.2)
    plt.savefig("figs/2-%f.png" % d)

""" Apartado iii) """

seq_q0 = np.linspace(0.0, 1.0, num=20)
seq_dq0 = np.linspace(0.0, 2.0, num=20)
t = 5
n = int(t / delta)
# Guardar puntos:
q3 = np.empty([len(seq_q0), len(seq_dq0), n + 1])
p3 = np.empty([len(seq_q0), len(seq_dq0), n + 1])
for i, q0 in enumerate(seq_q0):
    for j, dq0 in enumerate(seq_dq0):
        q = orb(n, q0=q0, dq0=dq0, F=F, d=delta)
        dq = deriv(q, dq0=dq0, d=delta)
        p = dq / 2
        q3[i][j] = q
        p3[i][j] = p


def animate(t):
    n = int(t / delta)
    ax.clear()
    ax.set_xlabel("q(t)", fontsize=12)
    ax.set_ylabel("p(t)", fontsize=12)
    for i, q0 in enumerate(seq_q0):
        for j, dq0 in enumerate(seq_dq0):
            qt = q3[i][j][n]
```

```
            pt = p3[i][j][n]
            plt.plot(
                qt,
                pt,
                marker="o",
                markersize=8,
            )
    plt.xlim(-2.2, 2.2)
    plt.ylim(-1.2, 1.2)
    return (ax,)


fig = plt.figure(figsize=(8, 5))
ax = fig.add_subplot(1, 1, 1)
ani = animation.FuncAnimation(fig, animate, np.linspace(delta, 5, 20))
ani.save("3.mp4", fps=5)
```