1. Introducción

En esta práctica, dado el hamiltoniano de un oscilado no lineal $H(q,p)$ estudiaremos la variedad simpléctica con coordenadas $(q(t),p(t)) \in \mathbb{R}$. Estudiaremos el espacio fásico D obtenido con las condiciones iniciales con $q_0, p_0 \in D_0$ con $D_0 := [0,1] \times [0,1]$. Puesto que tendremos que discretizar el parámetro temporal t utilizaremos una granularidad de este $t = n\delta$ con $\delta \in (10^{-4}, 10^{-3})$ y $n \in \mathbb{Z}^+$. Las ecuaciones de Hamilton-Jacobi conducen desde el hamiltoniano a la siguiente ecuación diferencial de segundo orden:

$$H(q,p) = p^2 + \frac{1}{4}(q^2-1)^2 \quad \Rightarrow \quad \ddot{q} = -2q(q^2-1)$$

2. Material empleado

Para el estudio del espacio fásico D en el primer apartado representaremos las órbitas finales del sistema con las condiciones iniciales D_0. Para ello primero tenemos que discretizar la ecuación para obtener un sistema dinámico discreto:

$$q_{n+2} = \delta^2 F(q_n) - q_n + 2q_{n+1}$$

donde $F(q) = -2q(q^2-1)$. Con este sistema dinámico ya podemos calcular los diferentes pares (q,p) para cada condición inicial y pintar las órbitas. Así obtendríamos una representación del espacio fásico.

Para calcular el área del espacio fásico del oscilador seguiremos el algoritmo de los apuntes. Como ya tenemos discretizado el sistema, para cada par de condiciones iniciales calculamos la órbita final a la que tiende. Una vez tenemos esa órbita tomamos una única onda completa que será la que integraremos. Como las ondas son simétricas respecto al eje x tomaremos únicamente la mitad superior a la hora de integrar para calcular el área.

Una vez obtenemos el área para cada una de las condiciones iniciales, la estimación del área $A(\delta)$ en función de la granularidad δ es $A(\delta) = a_{max}(\delta) - a_{inf}(\delta)$. El área máxima es la máxima de todos los valores obtenidos. Para calcular el área ínfima hay que tener en cuenta el hueco que se genera en la parte izquierda, como para calcular las ondas de cada condición inicial solo hemos tenido en cuenta aquellas con dos máximos, aproximamos el área ínfima como la mitad del "lazo" con menor área.

Para estimar el error, repetimos el proceso con 11 granularidades δ con $\delta \in (10^{-4}, 10^{-3})$ y nos quedamos con el percentil 90.

Para probar que se cumple el teorema de Liouville, tomamos varias instantáneas de como son las órbitas de todas las condiciones iniciales. En estas imágenes se puede apreciar que la forma del área va cambiando pero no el área en si. Con esto mostramos que el Teorema de Liouville se cumple entre D y D_0.

3. Resultados

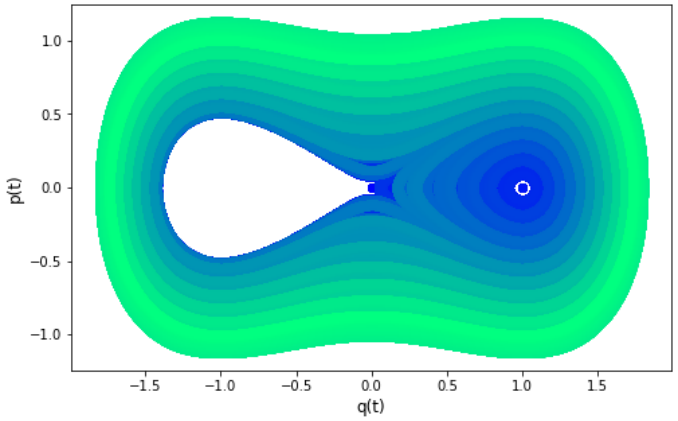

Figura 1: Espacio fásico con $(q_0, p_0) \in D_0$

```
El área calculada es: 6.182 con un error de 0.171
```

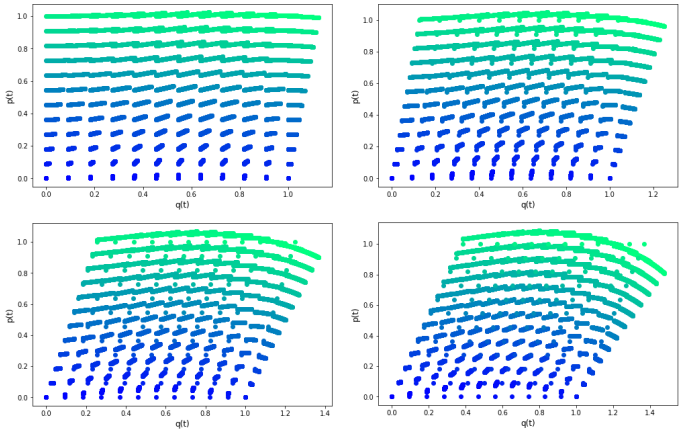

Figura 2: Evolución del espacio fásico con $(q_0, p_0) \in D_0$ en los primeros instantes

4. Conclusión

Como conclusión comentaremos que son los puntos $(0, 0)$ y $(1, 0)$. Las raíces del polinomio $-2q(q^2-1)$ en el intervalo [0,1] son 0 y 1 y la p indicaría la cantidad de movimiento. Por tanto una partícula en esos puntos estaría quieta tratándose entonces de puntos de equilibrio. Estudiando el signo de la derivada del polinomio en un entorno de cada punto podemos ver que el $(0, 0)$ es un punto de equilibrio inestable mientras que el $(1, 0)$ es un punto de equilibrio estable.

5. Código

```
# -*- coding: utf-8 -*-
"""
Created on Wed Apr  8 23:53:36 2020
@author: Jorge Sainero y Lucas de Torre con la plantilla de Robert
"""
import os
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import simps
#https://matplotlib.org/3.1.0/tutorials/colors/colormaps.html

os.getcwd()

#q = variable de posición, dq0 = \dot{q}(0) = valor inicial de la derivada
#d = granularidad del parámetro 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) #dq = np.concatenate(([dq0],dq))
   return dq

#Ecuación de un sistema dinámico continuo
#Ejemplo de oscilador simple
def F(q):
    #k = 1
    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=0.001,n0=0):
    #q = [0.0]*(n+1)
    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[n0:] #np.array(q),

def periodos(q,d,max=True):
```

```
    #Si max = True, tomamos las ondas a partir de los máximos/picos
    #Si max == False, tomamos los las ondas a partir de los mínimos/valles
    epsilon = 5*d
    dq = deriv(q,dq0=None,d=d) #La primera derivada es irrelevante
    if max == True:
        waves = np.where((np.round(dq,int(-np.log10(epsilon))) == 0) & (q >0))
    if max != True:
        waves = np.where((np.round(dq,int(-np.log10(epsilon))) == 0) & (q <0))
    diff_waves = np.diff(waves)
    waves = waves[0][1:][diff_waves[0]>1]
    pers = diff_waves[diff_waves>1]*d
    return pers, waves

d = 10**(-3.5)
## Pintamos el espacio de fases
def simplectica(q0,dq0,F,col=0,d=10**(-3.5),n = int(16/d),marker='-',n0=0):
    q = orb(n,q0=q0,dq0=dq0,F=F,d=d,n0=n0)
    dq = deriv(q,dq0=dq0,d=d)
    p = dq/2
    plt.plot(q, p, marker,c=plt.get_cmap("winter")(col))

def print_espacio_fasico(n=int(16/d),n0=0):
    fig = plt.figure(figsize=(8,5))
    fig.subplots_adjust(hspace=0.4, wspace=0.2)
    seq_q0 = np.linspace(0.,1.,num=12)
    seq_dq0 = np.linspace(0.,2.,num=12)
    for i in range(len(seq_q0)):
        for j in range(len(seq_dq0)):
            q0 = seq_q0[i]
            dq0 = seq_dq0[j]
            ax = fig.add_subplot(1,1, 1)
            col = (1+i+j*(len(seq_q0)))/(len(seq_q0)*len(seq_dq0))
            #ax = fig.add_subplot(len(seq_q0), len(seq_dq0), 1+i+j*(len(seq_q0)))
            simplectica(q0=q0,dq0=dq0,F=F,col=col,marker='ro',d=d,n=n,n0=n0)
    ax.set_xlabel("q(t)", fontsize=12)
    ax.set_ylabel("p(t)", fontsize=12)
    #fig.savefig('Simplectic.png', dpi=250)
    plt.show()

#####################################################################
#  CÁLCULO DE ÓRBITAS
#####################################################################
def apartado1():
```

```
    print("Apartado 1\n")

    ######################################################################
    #  ESPACIO FÁSICO
    ######################################################################

    print_espacio_fasico()




######################################################################
#  CÁLCULO DEL ÁREA DEL ESPACIO FÁSICO
######################################################################
#Tomamos un par (q(0), p(0)) y nos quedamos sólo en un trozo/onda de la órbita,
sin repeticiones
#Para eso, tomaremos los periodos de la órbita, que definen las ondas

#Paso1: Buscamos las condiciones iniciales que maximizan/minimizan en área
#q0 = 0.
#dq0 = 2.
#d = 10**(-3.5)
def area_espacio_fasico(d):
    areas=[]
    for q0 in np.linspace(0.,1.,num=11):
        for dq0 in np.linspace(0.,2.,num=11):

            n = int(32/d)
            #t = np.arange(n+1)*d
            q = orb(n,q0=q0,dq0=dq0,F=F,d=d)
            dq = deriv(q,dq0=dq0,d=d)
            p = dq/2
            #Tomaremos los periodos de la órbita, que definen las ondas
            T, W = periodos(q,d,max=False)
            if len(W)>1:

                #Tomamos la mitad de la "curva cerrada" para integrar más
                fácilmente
                mitad = np.arange(W[0],W[0]+np.int((W[1]-W[0])/2),1)

                # Regla de Simpson
                area = simps(p[mitad],q[mitad])
                areas.append([q0,dq0,area])
```

```
    sort_areas=sorted(areas,key=lambda x:x[2])
    min_a=sort_areas[0][2]
    max_a=2*sort_areas[len(sort_areas)-1][2]
    return max_a-min_a

def apartado2():
    #Definimos distintos valores de d
    iseq=np.linspace(3.01,3.99,num=11)
    #Hallamos el área de cada espacio fásico para cada d
    areas=[area_espacio_fasico(10**(-d)) for d in iseq]
    #Hallamos el valor absoluto de las diferencias de cada área
    comparada con "la mejor"
    resta_areas=[abs(areas[i]-areas[10]) for i in range(len(areas)-1)]
    sort_resta_areas=sorted(resta_areas)
    #Cogemos el cuantil de orden 0,9
    print("El área calculada es:",round(areas[10],3),
          "con un error de", round(sort_resta_areas[8],3))

    ###################################################################
    ################## TEOREMA DE LIOUVILLE ###########################
    ###################################################################
    for i in range(4):
        print_espacio_fasico((i+1)*200,i*200)

apartado1()
apartado2()
```