diaFases

October 22, 2018

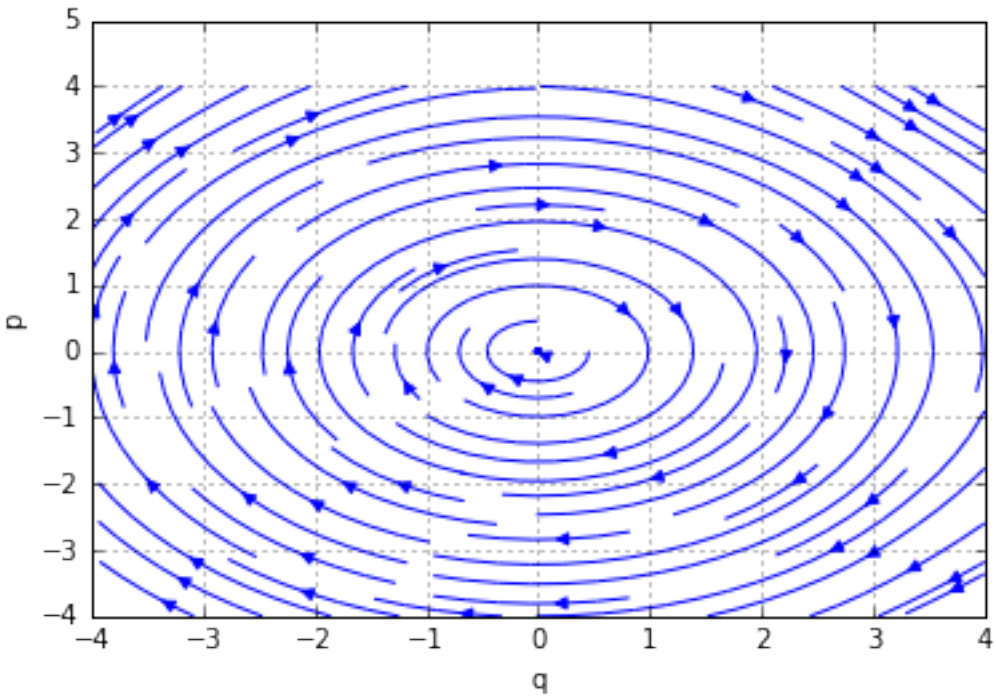
In [15]:
```
import matplotlib.pyplot as plt
import numpy as np
```

In [16]:
```
import sympy as sy
sy.init_printing()
```

1 Problema 1

1.1 a) Oscilador armónico

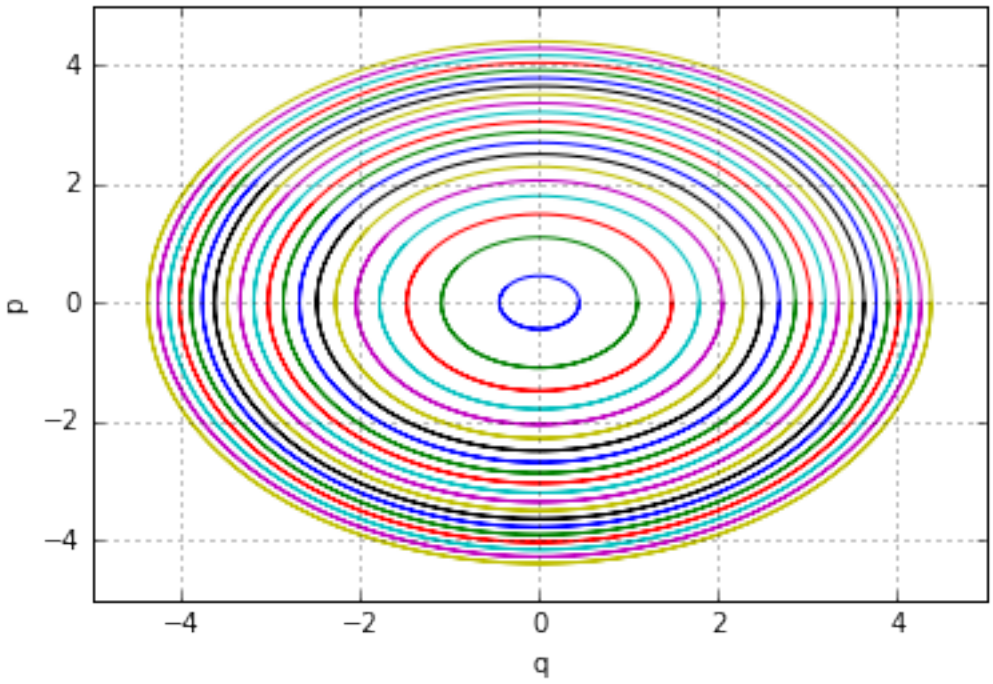
In [17]:
```
# Usando meshgrid y streamplot
w = 4 # limites para p y q
pm, qm = np.mgrid[-w:w:10j, -w:w:10j] # grilla matricial de valores de p y q equiespaci
m_numerico = 1 # parámetros del sistema
k_numerico = 1
dqm = pm/k_numerico # resultados de ecuaciones canónicas
dpm = -k_numerico* qm
plt.xlabel('q')
plt.ylabel('p')
plt.grid()
plt.streamplot(qm,pm,dqm,dpm)
plt.show()
```



1.2 A la antigüa - oscilador armónico

```
In [19]: # trayectoria en base a posición, momento inicial y sus derivadas temporales
         def pares(p0, q0, pasos=int(1E4), dt=1E-3):
             qi = np.empty(pasos)
             pi = np.empty(pasos)
             qi[0]= q0
             pi[0]= p0
             for i in range(pasos-1):
                 pi[i+1]= pf(pi[i], qi[i], dt)
                 qi[i+1]= qf(pi[i+1], qi[i], dt) # usa el p actualizado
             return pi,qi

         # Para el problema en particular
         def pf(p, q, dt):
             k = 1
             dpdt = -k* q # aqui va p punto obtenido de la ecuación canónica
             return p+ dt* dpdt

         def qf(p, q, dt):
             m = 1
             dqdt = p/m # aqui va q punto obtenido de la ecuación canónica
             return q+ dt* dqdt
```

```
In [29]: # graficación
         plt.xlabel('q')
         plt.ylabel('p')
         plt.grid()
         limx = 5
         limy = 5
         plt.xlim([-limx, limx])
         plt.ylim([-limy, limy])

         # secuencia principal
         k = 1
         p0 = 0 # inicial
         for E in np.arange(0.1,10,0.5): # diferentes valores de energía
             q0 = np.sqrt(2*E/k) # inicial para p=0
             pi, qi = pares(p0, q0)
             plt.plot(qi, pi, '-')
         plt.show()
```



1.3 b) Potencial central (aún no funciona bien)

```
In [4]: p,q, m, k = sy.symbols('p q m k')
        rho, theta, p_rho, p_theta = sy.symbols('rho theta p_rho p_theta')

In [5]: T = (1/(2*m))*(p_rho**2+ p_theta**2/rho**2)
        T
```

Out[5]:

$$\frac{p_\rho^2 + \frac{p_\theta^2}{\rho^2}}{2m}$$

In [6]:
```
U = -k**2/rho
U
```

Out[6]:

$$-\frac{k^2}{\rho}$$

In [7]:
```
H = T+ U
H
```

Out[7]:

$$-\frac{k^2}{\rho} + \frac{p_\rho^2 + \frac{p_\theta^2}{\rho^2}}{2m}$$

In [8]:
```
# ecuaciones canónica para theta
thetapunto = sy.diff(H,p_theta) # p punto
thetapunto
```

Out[8]:

$$\frac{p_\theta}{m\rho^2}$$

In [9]:
```
# ecuaciones canónica para rho
# pp = sy.symbols('\dot{p}')
rhopunto = sy.diff(H,p_rho) # p punto
p_rhopunto = -sy.diff(H,rho) # p punto
p_rhopunto, rhopunto
```

Out[9]:

$$\left(-\frac{k^2}{\rho^2} + \frac{p_\theta^2}{m\rho^3}, \quad \frac{p_\rho}{m}\right)$$

In [10]:
```
k_numerico = 1
m_numerico = 1
p_theta_numerico = 1
p_rhopunto_subs = p_rhopunto.subs({k:k_numerico, m:m_numerico, p_theta:p_theta_numerico
p_rhopunto_subs
```

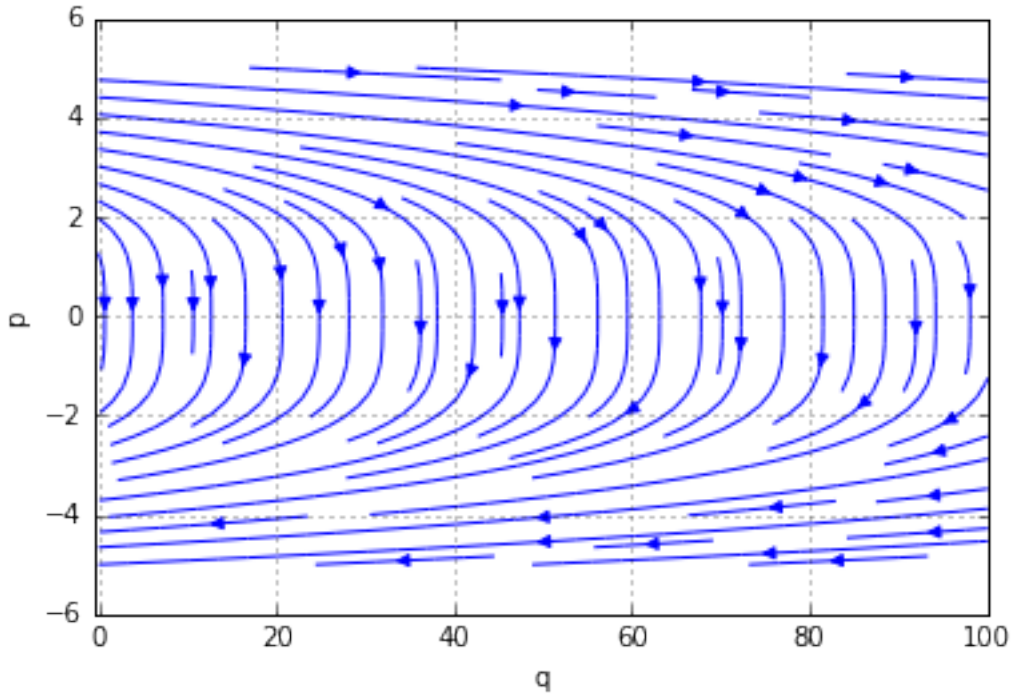
Out[10]:

$$-\frac{1}{\rho^2} + \frac{1}{\rho^3}$$

In [11]:
```
p_rhopunto_num = sy.lambdify(p_rhopunto_subs,rho) # solo en función de rho
p_rhopunto_num(3)
```

```
Traceback (most recent call last):

  File "/usr/local/lib/python3.5/dist-packages/IPython/core/interactiveshell.py", line 2
exec(code_obj, self.user_global_ns, self.user_ns)

  File "<ipython-input-11-18280908f8e9>", line 1, in <module>
p_rhopunto_num = sy.lambdify(p_rhopunto_subs,rho) # solo en función de rho

  File "/usr/lib/python3/dist-packages/sympy/utilities/lambdify.py", line 376, in lambdi
func = eval(lstr, namespace)

  File "<string>", line 1
lambda -1/rho**2 + rho**(-3): (rho)
       ^
SyntaxError: invalid syntax
```

In [14]:
```
# Usando meshgrid y streamplot
lim_p = 5 # limites para p y q
lim_q = 1E2
num_puntos_grilla = 100j
pm, qm = np.mgrid[-lim_p:lim_p:num_puntos_grilla,0:lim_q:num_puntos_grilla] # grilla ma
m_numerico = 1 # parámetros del sistema
k_numerico = 1
p_theta_numerico = 1E-6

dqm = pm/m_numerico # resultados de ecuaciones canónicas
dpm = -(k_numerico**2/pm**2)+(p_theta_numerico**2/(m_numerico*pm**3))

plt.xlabel('q')
plt.ylabel('p')
plt.grid()
plt.xlim(-0.5,lim_q)
plt.streamplot(qm,pm,dqm,dpm)
plt.show()
```


6
4
2
0
−2
−4
−6
p
0
20
40
60
80
100
q