PHYS2600: Homework 5*

Due on Thursday, Mar, 22nd 2018

Yangrui Hu

*This Latex template is from http://www.latextemplates.com

Contents

Problem 1

Problem: Simulate a random walk in three dimensions allowing the walker to make steps of unit length in random directions; dont restrict the walker to sites on a discrete lattice. Show that the motion is diffusive, i.e. $\langle r^2 \rangle \propto t$. Find the value of the proportionality constant.

Solution: I simulate an ensemble of random walkers in three dimensions as Listing 1. Because the walker can make steps of unit length in random directions, in each step there are two random numbers: θ and ϕ. And the new position can be expressed as:

$$x_{new} = x_{old} + \sin\theta\cos\phi \quad (1)$$
$$y_{new} = y_{old} + \sin\theta\sin\phi \quad (2)$$
$$z_{new} = z_{old} + \cos\theta \quad (3)$$
$$r^2 = x^2 + y^2 + z^2 \quad (4)$$

Figure 1 shows the simulation result. I set the number of ensemble as $N = 1000$ and the number of step as $nsteps = 1000$. It's obvious that there is a linear relationship between $\langle r^2 \rangle$ and time step. The linear fitting result is $\langle r^2 \rangle = 1.017x - 5.01$. So the proportionality constant is 1.

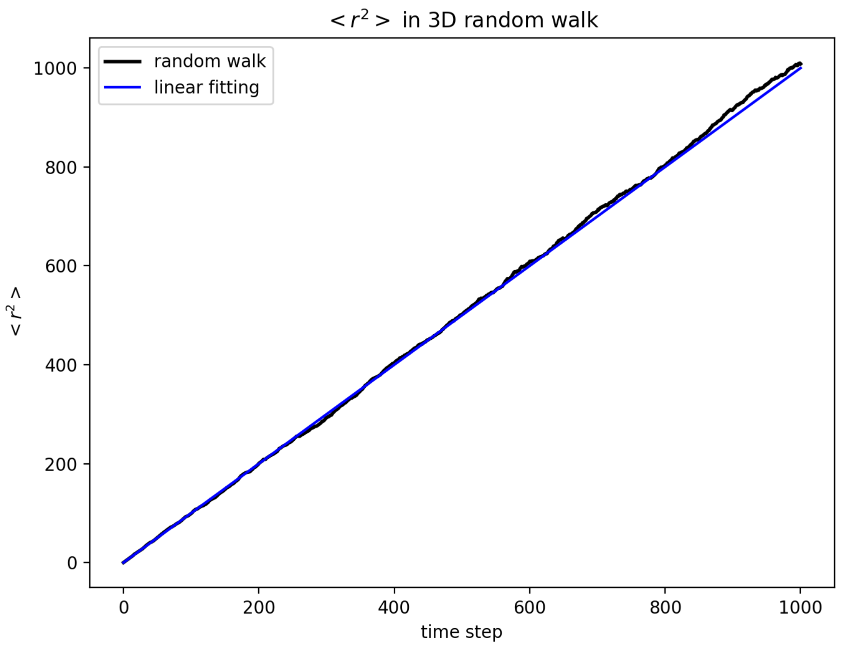

Figure 1: $\langle r^2 \rangle(t)$ in 3D random walk

Problem 1 continued on next page...

Listing 1: Original Script of Problem 1

```
import numpy as np
import matplotlib.pyplot as plt
from numba import jit
import random

@jit(nopython=True)
def step_3D():
     theta = np.pi*random.random()
     phi = 2*np.pi*random.random()
     delta_x = np.sin(theta)*np.cos(phi)
     delta_y = np.sin(theta)*np.sin(phi)
     delta_z = np.cos(theta)
     return [delta_x,delta_y,delta_z]

@jit(nopython=True)
def random_walk_3D(nsteps):
     x = np.zeros(nsteps+1, np.double)
     y = np.zeros(nsteps+1, np.double)
     z = np.zeros(nsteps+1, np.double)
     r_square = np.zeros(nsteps+1, np.double)
     for i in range(nsteps):
          delta = step_3D()
          x[i+1] = x[i] + delta[0]
          y[i+1] = y[i] + delta[1]
          z[i+1] = z[i] + delta[2]
          r_square[i+1] = x[i+1]**2+y[i+1]**2+z[i+1]**2
     return r_square

@jit(nopython=True)
def ensemble_3D(N,nsteps):
     r_square = np.zeros(nsteps+1, np.double)
     for i in range(N):
          r_update = random_walk_3D(nsteps)
          r_square[:] = r_square[:] + r_update[:]
     r_square[:] = r_square[:]/N
     return r_square

N = 1000
nsteps = 1000
r_square = ensemble_3D(N,nsteps)
step_array = np.linspace(0.0, nsteps, nsteps+1, endpoint = True)
fit=np.polyfit(step_array, r_square, 1)
print(fit)
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(step_array,r_square,'k',lw=2, label='random walk')
ax.plot(step_array,step_array,'blue',label='linear fitting')
ax.set_xlabel('time step')
ax.set_ylabel('$<r^2>$')
ax.legend()
ax.set_title("$<r^2>$ in 3D random walk")
plt.show()
```

Problem 1 continued on next page. . .

Problem 2

Problem: Simulate an ensemble of random walkers in two dimensions, constrained to a 50 by 50 square lattice, and starting with an initial distribution in which all of the particles are near the center. Let one of the walls of the container have a small hole of 10 units in length along the edge through which particles can leave the container. Once a particle has left the container, it cannot reenter. Calculate the number of particles in the container as a function of time and show that it varies as $\exp(t/\tau)$, where τ is an effective time constant for the escape time.

Solution: I simulate the ensemble described in the problem as Listing 2. Figure 2 shows the simulation result. The initial number of particles is 1000. I use exponential function to fit the curve and get $N(t) = 1005.3e^{-t/6803.2}$. So the effective time constant for the escape time $\tau = 6803.2$

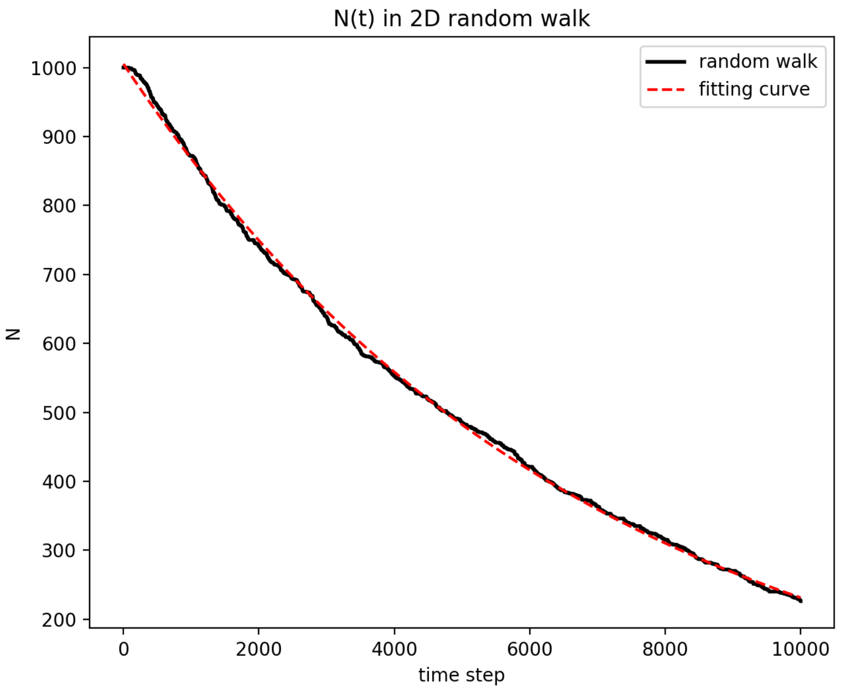

Figure 2: $N(t)$ in 2D random walk constrained to a 50 by 50 square lattice

Problem 2 continued on next page. . .

Listing 2: Original Script of Problem 2

```
import numpy as np
import matplotlib.pyplot as plt
from numba import jit
import random
from scipy.optimize import curve_fit

@jit(nopython=True)
def step_2D():
     d = np.random.randint(1,5)
     if d == 1:
          x_s, y_s = 1, 0
     if d == 2:
          x_s, y_s = 0, 1
     if d == 3:
          x_s, y_s = -1, 0
     if d == 4:
          x_s, y_s = 0, -1
     return [x_s, y_s]

@jit(nopython=True)
def ensemble_2D(N0, nsteps):
     Number = np.zeros((nsteps+1,N0))
     N = np.zeros(nsteps+1)
     Number[0,:]=1
     x = np.zeros((nsteps+1,N0))
     y = np.zeros((nsteps+1,N0))
     for i in range(N0):
          for j in range(nsteps):
               delta = step_2D()
               x[j+1,i] = x[j,i] + delta[0]
               y[j+1,i] = y[j,i] + delta[1]
               if x[j+1,i]>25 and abs(y[j+1,i])<10:
                    break
               if abs(x[j+1,i]) > 25:
                    x[j+1,i] = x[j,i]
               if abs(y[j+1,i]) > 25:
                    y[j+1,i] = y[j,i]
               Number[j+1,i] = 1
     for i in range(nsteps+1):
          for j in range(N0):
               N[i] = N[i] +Number[i,j]
     return N

def func(x,a,tau):
     return a*np.exp(-x/tau)

N0 = 1000
nsteps = 10000
N = ensemble_2D(N0, nsteps)
step_array = np.linspace(0.0, nsteps, nsteps+1, endpoint = True)
popt, pcov = curve_fit(func, step_array, N)
y2 = [func(i, popt[0], popt[1]) for i in step_array]
```

Problem 2 continued on next page. . .

```
print(popt)
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(step_array,N,'k',lw=2, label='random walk')
ax.plot(step_array,y2,'r--',label = 'fitting curve')
ax.set_xlabel('time step')
ax.set_ylabel('N')
ax.legend()
ax.set_title("N(t) in 2D random walk")
plt.show()
```

Problem 3

Problem: Carry out an analysis of the entropy for the nonlinear damped driven pendulum. Divide the possible range for $\theta(t)$ into 100 separate bins. Simulate the pendulum and calculate a histogram for $\theta(t)$. When constructing your histogram, only sample θ in sync with the driving force, as was done when calculating Poincare sections. Calculate the entropy as a function of the driving force using

$$S = -\sum_i P_i \ln P_i \tag{5}$$

where P_i is the probability of finding the system in state i. What is S in the period-2 and period-4 regimes? How does the entropy change as the driving force is increased from the periodic regime to the chaotic regime?

Solution: I simulate the nonlinear damped driven pendulum, construct a histogram for θ, and calculate the entropy when $F_d = 1.4$ to $F_d = 1.5$ as Listing 3. As the previous homework shown, when $F_d = 1.4$, the behavior is period 1; when $F_d = 1.44$, the behavior is period 2; when $F_d = 1.465$, the behavior is period 4; when $F_d = 1.481$, the behavior is chaos. So the entropy at different situations is as Table 1:

Table 1: Entropy at Different Situations

F_d	Period	Entropy
1.4	1	0.704
1.44	2	1.437
1.465	4	1.821
1.481	chaos	2.306

Figure 3 shows the entropy as a function of F_d. When I sample θ, I choose two points per period here. According to Figure 3, we can find that as the driving force increasing, the periodic regime changes to the chaotic regime and the corresponding entropy also increases. Because I just simulate one pendulum, the entropy fluctuates a little bit. When F_d is large enough, the system changes back to periodic regime and the entropy decreases.
Figure 4 shows the histogram of θ when $F_d = 1.4,\ 1.44,\ 1.465,\ 1.481$. When I draw the histograms, I only choose one point per period to show the period behavior. From Figure 4, we can clearly state that when $F_d = 1.4$, the behavior is period 1; when $F_d = 1.44$, the behavior is period 2; when $F_d = 1.465$, the behavior is period 4; when $F_d = 1.481$, the behavior is chaos.

Problem 3 continued on next page...

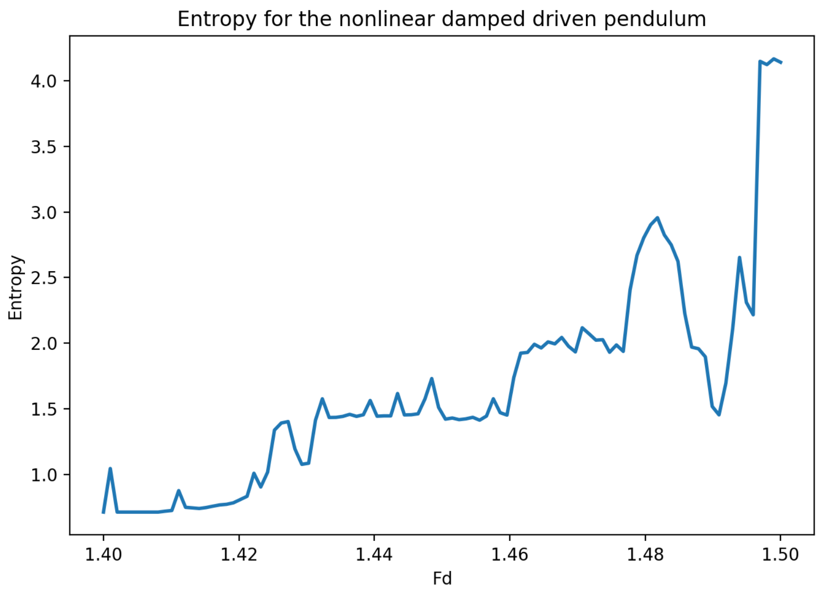

Figure 3: Entropy of the nonlinear damped driven pendulum

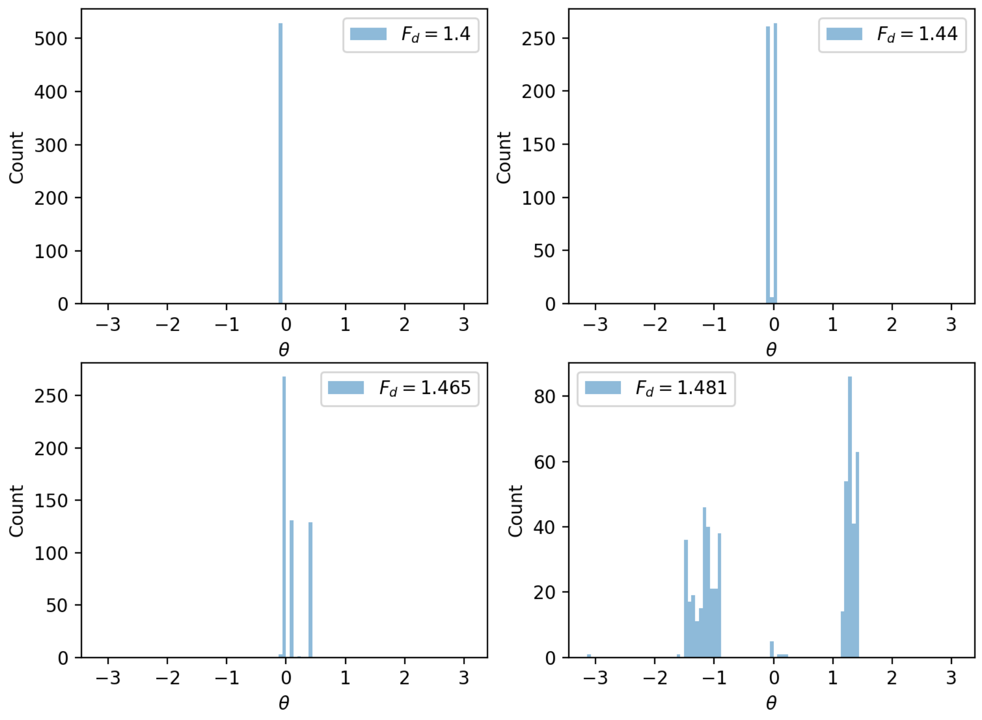

Figure 4: The histogram of of θ when $F_d = 1.4,\ 1.44,\ 1.465,\ 1.481$

Problem 3 continued on next page. . .

Listing 3: Original Script of Problem 3

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
from numba import jit
import random

class Pendulum(object):

    def __init__(self, l = 9.8, nu = 0.5, Fd  = 1.4, omega_d = 2/3,
     m = 1.0, x0 = 0.2 ,v0 = 0.0, tf = 10000.0, dt = 0.005):
        self.g = 9.8
        self.m = m
        self.x = x0
        self.v = v0
        self.l = l
        self.Fd = Fd
        self.omega_d = omega_d
        self.nu = nu
        self.omega0 = np.sqrt(self.g/l)

        self.t = 0.0
        self.tf = tf
        self.dt = dt

        npoints = int(tf/dt)
        self.tarray = np.linspace(0.0, tf, npoints, endpoint = True)
        self.xv0 = np.array([self.x, self.v])

    def F(self, x, v, t):
        return self.Fd*np.cos(self.omega_d*t) - self.nu*v - self.g/self.l
        *np.sin(x)

    def scipy_trajectory(self):
        self.xv = odeint(self.derivative, self.xv0, self.tarray)
        x = self.xv[:,0]
        x_new = np.zeros(np.shape(x))
        x_new[0] = x[0]
        dx = np.diff(x)
        nx = np.shape(x)[0]
        for ii in range(1,nx):
            x_new[ii] = x_new[ii-1]+dx[ii-1]
            if x_new[ii] > np.pi:
                x_new[ii] -= 2*np.pi
            elif x_new[ii] < -np.pi:
                x_new[ii] += 2*np.pi
        self.xv_unwrap = self.xv
        self.xv[:,0] = x_new

    def derivative(self, xv, t):
        x =xv[0]
        v =xv[1]
        a = self.F(x, v, t) / self.m
```

Problem 3 continued on next page. . .

```
            return np.ravel(np.array([v, a]))

    def poincare(self):
            T = 2*np.pi/self.omega_d
            nsamples = 1 #calculate entropy, nsamples=2; draw histogram, nsamples=1
            sample = T/nsamples
            index = np.nonzero((self.tarray%sample)<(self.dt/2.))[0]
            xv_poincare = self.xv[index,:]
            t_poincare = self.tarray[index]
            theta_poincare = xv_poincare[:,0]
            return t_poincare, theta_poincare

def entropy():
    S = []
    L = 2*np.pi/100
    theta_bin = np.arange(-np.pi,np.pi,L)
    dFd = 0.001
    Fd_int = 1.4
    Fd_final = 1.5
    Fd_step = int((Fd_final-Fd_int)/dFd)
    Fd_array = np.linspace(Fd_int, Fd_final, Fd_step, endpoint = True)
    for i in range(Fd_step):
        Fd = Fd_int + i*dFd
        P = Pendulum(Fd = Fd)
        P.scipy_trajectory()
        t_poincare, theta_poincare = P.poincare()
        h, x_edge =np.histogram(theta_poincare, bins = theta_bin, normed = True)
        S.append(-1*np.sum(L*h*np.log(L*h+1e-40)))

    print(np.sum(h)*L)
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.plot(Fd_array,S, lw = 2)
    ax.set_xlabel('Fd')
    ax.set_ylabel('Entropy')
    ax.set_title("Entropy for the nonlinear damped driven pendulum")
    plt.show()

#entropy()
def plot_Histogram():
    fig = plt.figure()
    ax1 = fig.add_subplot(221)
    ax2 = fig.add_subplot(222)
    ax3 = fig.add_subplot(223)
    ax4 = fig.add_subplot(224)
    L = 2*np.pi/100
    theta_bin = np.arange(-np.pi,np.pi,L)
    P1 = Pendulum(Fd = 1.4)
    P1.scipy_trajectory()
    t1_poincare, theta1_poincare = P1.poincare()
    ax1.hist(theta1_poincare, bins = theta_bin, alpha=0.5, label='$F_d=1.4$')
    ax1.set_xlabel("$\\theta$")
    ax1.set_ylabel("Count")
```

Problem 3 continued on next page. . .

```
     ax1.legend()
     P2 = Pendulum(Fd = 1.44)
     P2.scipy_trajectory()
     t2_poincare, theta2_poincare = P2.poincare()
     ax2.hist(theta2_poincare, bins = theta_bin, alpha=0.5, label='$F_d=1.44$')
     ax2.set_xlabel("$\\theta$")
     ax2.set_ylabel("Count")
     ax2.legend()
     P3 = Pendulum(Fd = 1.465)
     P3.scipy_trajectory()
     t3_poincare, theta3_poincare = P3.poincare()
     ax3.hist(theta3_poincare, bins = theta_bin, alpha=0.5, label='$F_d=1.465$')
     ax3.set_xlabel("$\\theta$")
     ax3.set_ylabel("Count")
     ax3.legend()
     P4 = Pendulum(Fd = 1.481)
     P4.scipy_trajectory()
     t4_poincare, theta4_poincare = P4.poincare()
     ax4.hist(theta4_poincare, bins = theta_bin, alpha=0.5, label='$F_d=1.481$')
     ax4.set_xlabel("$\\theta$")
     ax4.set_ylabel("Count")
     ax4.legend()

     plt.show()

plot_Histogram()
```