009x_43

March 18, 2017

Stochastic Processes: Data Analysis and Computer Simulation

Brownian motion 2: computer simulation
-Making Animations-

1 Simulation code with on-the-fly animation

1.1 Import libraries

```
In [1]: % matplotlib nbagg
        import numpy as np # import numpy library as np
        import matplotlib.pyplot as plt # import pyplot library as plt
        import matplotlib.mlab as mlab # import mlab module to use MATLAB commands
        import matplotlib.animation as animation # import animation modules from ma
        from mpl_toolkits.mplot3d import Axes3D # import Axes3D from mpl_toolkits.m
        plt.style.use('ggplot') # use "ggplot" style for graphs
```

1.2 Define `init` function for `FuncAnimation`

```
In [2]: def init():
            global R,V,W,Rs,Vs,Ws,time
            R[:,:] = 0.0 # initialize all the variables to zero
            V[:,:] = 0.0 # initialize all the variables to zero
            W[:,:] = 0.0 # initialize all the variables to zero
            Rs[:,:,:] = 0.0 # initialize all the variables to zero
            Vs[:,:,:] = 0.0 # initialize all the variables to zero
            Ws[:,:,:] = 0.0 # initialize all the variables to zero
            time[:]   = 0.0 # initialize all the variables to zero
            title.set_text(r'') # empty title
            line.set_data([],[]) # set line data to show the trajectory of particle
            line.set_3d_properties([]) # add z-data separately for 3d plot
            particles.set_data([],[]) # set position current (x,y) position data fo
            particles.set_3d_properties([]) # add current z data of particles to ge
            return particles,title,line # return listed objects that will be drawn
```

1.3 Define `animate` function for `FuncAnimation`

```
In [3]: def animate(i):
            global R,V,W,Rs,Vs,Ws,time # define global variables
            time[i]=i*dt # store time in each step in an array time
            W = std*np.random.randn(nump,dim) # generate an array of random forces
            V = V*(1-zeta/m*dt)+W/m # update velocity via Eq.(F9)
            R = R+V*dt # update position via Eq.(F5)
            Rs[i,:,:]=R # accumulate particle positions at each step in an array Rs
            Vs[i,:,:]=V # accumulate particle velocitys at each step in an array Vs
            Ws[i,:,:]=W # accumulate random forces at each step in an array Ws
            title.set_text(r"t = "+str(time[i])) # set the title to display the cur
            line.set_data(Rs[:i+1,n,0],Rs[:i+1,n,1]) # set the line in 2D (x,y)
            line.set_3d_properties(Rs[:i+1,n,2]) # add z axis to set the line in 3D
            particles.set_data(R[:,0],R[:,1]) # set the current position of all the
            particles.set_3d_properties(R[:,2]) # add z axis to set the particle in
            return particles,title,line # return listed objects that will be drawn
```

1.4 Set parameters and initialize variables

```
In [4]: dim  = 3 # system dimension (x,y,z)
        nump = 1000 # number of independent Brownian particles to simulate
        nums = 1024 # number of simulation steps
        dt   = 0.05 # set time increment, \Delta t
        zeta = 1.0 # set friction constant, \zeta
        m    = 1.0 # set particle mass, m
        kBT  = 1.0 # set temperatute, k_B T
        std  = np.sqrt(2*kBT*zeta*dt) # calculate std for \Delta W via Eq.(F11)
        np.random.seed(0) # initialize random number generator with a seed=0
        R = np.zeros([nump,dim]) # array to store current positions and set initial
        V = np.zeros([nump,dim]) # array to store current velocities and set initia
        W = np.zeros([nump,dim]) # array to store current random forcces
        Rs = np.zeros([nums,nump,dim]) # array to store positions at all steps
        Vs = np.zeros([nums,nump,dim]) # array to store velocities at all steps
        Ws = np.zeros([nums,nump,dim]) # array to store random forces at all steps
        time = np.zeros([nums]) # an array to store time at all steps
```

1.5 Perform and animate the simulation using `FuncAnimation`

```
In [5]: fig = plt.figure(figsize=(10,10)) # set fig with its size 10 x 10 inch
        ax = fig.add_subplot(111,projection='3d') # creates an additional axis to t
        box = 40 # set draw area as box^3
        ax.set_xlim(-box/2,box/2) # set x-range
        ax.set_ylim(-box/2,box/2) # set y-range
        ax.set_zlim(-box/2,box/2) # set z-range
        ax.set_xlabel(r"x",fontsize=20) # set x-lavel
        ax.set_ylabel(r"y",fontsize=20) # set y-lavel
        ax.set_zlabel(r"z",fontsize=20) # set z-lavel
```

```
        ax.view_init(elev=12,azim=120) # set view point
        particles, = ax.plot([],[],[],'ro',ms=8,alpha=0.5) # define object particle
        title = ax.text(-180.,0.,250.,r'',transform=ax.transAxes,va='center') # def
        line, = ax.plot([],[],[],'b',lw=1,alpha=0.8) # define object line
        n = 0 # trajectry line is plotted for the n-th particle
        anim = animation.FuncAnimation(fig,func=animate,init_func=init,
                    frames=nums,interval=5,blit=True,repeat=False)
        ## If you have ffmpeg installed on your machine
        ## you can save the animation by uncomment the last line
        ## You may install ffmpeg by typing the following command in command prompt
        ## conda install -c menpo ffmpeg
        ##
        # anim.save('movie.mp4',fps=50,dpi=100)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```