Computer Work Unit 9 spring-ball wave and dispersion relationship of 1D phonon modes [numpy array]

I. Python (array)

“visual” module contains a “numpy” module, which has a type called array. Array can help us to speed up program dramatically (10 times to 100 times faster). Here, let’s see how to use array.

```
>>> from numpy import *
>>> a = arange(0, 4.0, 0.5)                          # array range
>>> a
array([ 0. ,  0.5,  1. ,  1.5,  2. ,  2.5,  3. ,  3.5])   # change an list to an array
>>> a[0] = 5                                         # change the 0th element of a to 5
>>> a
array([ 5. ,  0.5,  1. ,  1.5,  2. ,  2.5,  3. ,  3.5])
>>> b = array(range(10))                             # b is an array from a list generated by range(10)
>>> b
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a[1:-1] **2                                      # from the 1st to the on before the last elements, do the square
array([ 0.25,  1.  ,  2.25,  4.  ,  6.25,  9.  ])
>>> a[-1] = 100                                      # change the last element (-1) to 100
>>> a
array([   5. ,    0.5,    1. ,    1.5,    2. ,    2.5,    3. ,  100. ])
>>> a[5:] *= 0.5                                     # from the fifth element, multiply each of them by 0.5
>>> a
array([  5.  ,   0.5 ,   1.  ,   1.5 ,   2.  ,   1.25,   1.5 ,  50.  ])
>>> a[:-1] + b[-7:]                                  # add array a from 0th to before the last element by array b from the last 7th
array([  8.  ,   4.5 ,   6.  ,   7.5 ,   9.  ,   9.25,  10.5 ])   # element to the end
```

Notice the index starts from 0. The following is to compare code execution speed for different methods:

```
from timeit import default_timer as timer
from visual import *                          # this also import numpy
                    # compare the time to execute 100 times of generating a list or an array for square of 1 to 10000
start = timer()
for x in range(100):
   j = []
   for i in range(10000): j.append(i**2)
end = timer()
print end-start

start = timer()
for x in range(100): j=[i**2 for j in range(10000)]
end = timer()
print end-start

start = timer()
for x in range(100):
   j=arange(10000)**2
end = timer()
print end-start
```

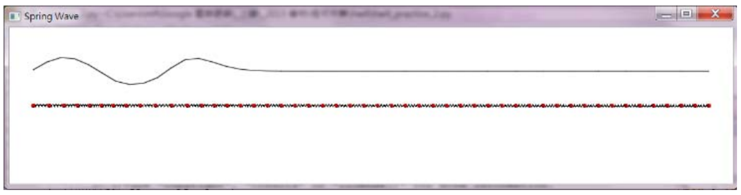
II. Practice: The following codes generates a longitudinal spring-ball wave.

```
from visual import *
A, N, omega = 0.10, 50, 2*pi/1.0
size, m, k, d = 0.06, 0.1, 10.0, 0.4
scene = display(title='Spring Wave', width=1200, height=300, background=(0.5,0.5,0), range = N*d/2+0.5, center = ((N-1)*d/2, 0, 0))
balls = [sphere(radius=size, color=color.red, pos=vector(i*d, 0, 0), v=vector(0,0,0)) for i in range(N)]
springs = [helix(radius = size/2.0, thickness = d/15.0, pos=vector(i*d, 0, 0), axis=vector(d,0,0)) for i in range(N-1)]

t, dt = 0, 0.001
while True:
   rate(1000)
   t += dt

   balls[0].pos = vector(A * sin(omega * t ), 0, 0)
   for i in range(N-1):
      springs[i].pos = balls[i].pos
      springs[i].axis = balls[i+1].pos - balls[i].pos
   for i in range(1, N):
      if i == N-1: balls[-1].v += - k * vector((abs(springs[-1].axis)-d),0,0)/m*dt
      else: balls[i].v += k* vector((abs(springs[i].axis)-d),0,0)/m*dt - k* vector((abs(springs[i-1].axis)-d),0,0)/m*dt
      balls[i].pos += balls[i].v*dt
```

(1) Change the above codes to array-based simulation by completing the 2 lines commented by '##'

```
from visual import *
A, N, omega = 0.10, 50, 2*pi/1.0
size, m, k, d = 0.06, 0.1, 10.0, 0.4
scene = display(title='Spring Wave', width=1200, height=300, background=(0.5,0.5,0), range = N*d/2+0.5, center = ((N-1)*d/2, 0, 0))
balls = [sphere(radius=size, color=color.red, pos=vector(i*d, 0, 0), v=vector(0,0,0)) for i in range(N)]          #3
springs = [helix(radius = size/2.0, thickness = d/15.0, pos=vector(i*d, 0, 0), axis=vector(d,0,0)) for i in range(N-1)]     #3
#1
ball_pos, ball_orig, ball_v, spring_len = arange(N)*d, arange(N)*d, zeros(N), ones(N)*d          #5
t, dt = 0, 0.001
while True:
  rate(1000)
  t += dt
  ball_pos[0] = A * sin(omega * t )                      #4
##   spring_len[0:-1] =
##   ball_v[1:] +=                                       #6
  ball_pos +=  ball_v*dt

  for i in range(N): balls[i].pos.x = ball_pos[i]        #3
  for i in range(N-1):                                   #3
    springs[i].pos = balls[i].pos                        #3
    springs[i].axis = balls[i+1].pos - balls[i].pos      #3
  #2
```

(2) To get a more clear view of the wave, we plot the longitudinal displacement of the balls from their original positions by a transverse displacement.



at #1, add code

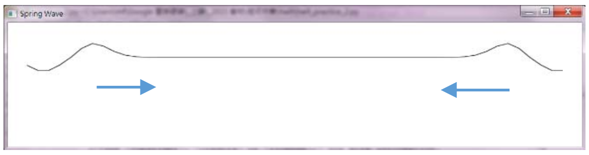
```
c = curve(display = scene)
```

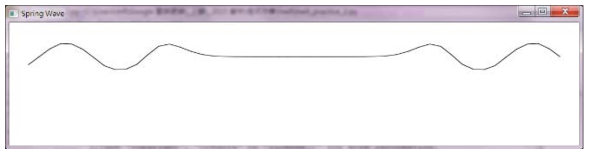
at #2, add code

```
  ball_disp = ball_pos - ball_orig
  c.x = ball_orig
  c.y = ball_disp * 4.0 + 1.0
```

(3) Since we already have the clear representation of the wave, we can remove the ball-spring plotting, marked by #3. You can clearly see that the wave propagating faster. This is because although we use rate(1000) to refresh the frame 1000 times per second, but the computer really cannot do that fast due to the slow plotting of the springs and the balls. Taking out the springs' and balls' plotting, the frame now is refreshing 1000 times per second, therefore it looks running faster.

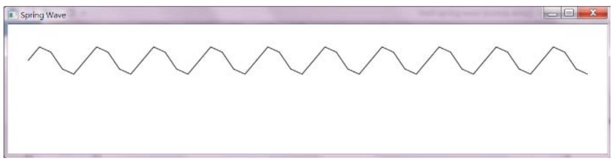
(4) You may have seen such things in a video game that an object moves out of the right end of the screen and comes back from the left end of the screen. This is called "periodically boundary condition", a very often used assumption in physics, engineering, and mathematics. In our system here, it means that the last ball and the zeroth ball is also connected by a spring in a fashion that looks like the zeroth ball is to the right of the last ball, or the last ball is to the left of the zeroth ball.





Now, modify the codes marked by '##', and add one or two more lines to achieve such "periodically boundary condition". You will see that the wave is generated from both ends towards the center because now the last one is also connected to the zeroth one, which is the source of the wave.

(5) Only certain wavelengths can satisfy the periodically boundary condition since they are the only waves that can repeat themselves after a complete cycle. These are waves with wavelength $\lambda = Nd/n$, where $n = 1, 2, 3, \ldots$ or with wavevector $K = 2\pi/\lambda = 2\pi n/(Nd)$. We can observe such wave by initially assigning balls to their proper positions, removing the external source, and letting the system to go by itself. This means changing #5 to



```
Unit_K, n = 2 * pi/(N*d), 10
Wavevector = n * Unit_K
phase = Wavevector * arange(N) * d
ball_pos, ball_orig, ball_v, spring_len = arange(N)*d + A*sin(phase), arange(N)*d, zeros(N), ones(N)*d
```

removing #4 and add after #6 a line to handle zeroth ball's velocity ball_v[0]. In the simulation, you will see clearly the oscillating standing wave with n = 10.

Now obtain the period (T) of the standing wave with n = 10 and its angular frequency (omega = 2*pi / T). For more accuracy, please change from dt = 0.001 to dt = 0.0001, and rate(1000) to rate(10000)

Homework submission:

In a crystal, only certain modes of waves can exist. These modes are called "Phonons". Practice (5) shows one of such modes in a simplified 1-dimensional crystal comsisted of only 50 balls (atoms). As you already see, when the wavevector is given, the angular frequency is decided by the system.

Now we want to know the relationship between the angular frequency and the wavevector, which is called the dispersion relationship. Modify you program from Practice (5) such that you can obtain the angular frequency (omega) for n from 1 to N/2. Use the graph plotting similar to homework 8 practice (4) to plot the the angular frequency versus the wavevector.