VP7 Spring-Ball Wave and Dispersion of 1D phonon [numpy array]

I. Numpy Array

Python has a very powerful module called "numpy", in which there is a type called array. Array can help us to speed up program dramatically (30 times to 300 times faster). First, let's see some operations about array.

```
>>> from numpy import *
>>> a = arange(4)                               # array range
>>> a
array([ 0, 1, 2, 3])
>>> a = arange(0, 4.0, 0.5)                     # array range
>>> a
array([ 0. ,  0.5,  1. ,  1.5,  2. ,  2.5,  3. ,  3.5])
>>> 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 before the last element, generate a new array with squared value
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 5th element to the end, change each of them by multiplying 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, and generate a new array to store the values
```

**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 numpy import *

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: Longitudinal Spring-Ball Wave.

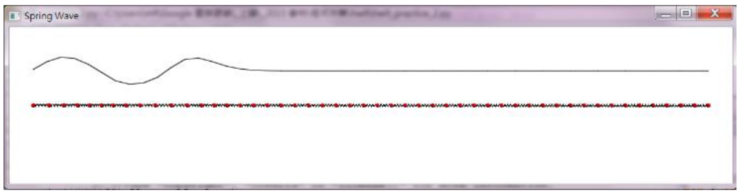
```
from vpython import *
A, N, omega = 0.10, 50, 2*pi/1.0
size, m, k, d = 0.06, 0.1, 10.0, 0.4
scene = canvas(title='Spring Wave', width=800, height=300, background=vec(0.5,0.5,0), center = vec((N-1)*d/2, 0, 0))
balls = [sphere(radius=size, color=color.red, pos=vector(i*d, 0, 0), v=vector(0,0,0)) for 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((springs[-1].axis.mag-d),0,0)/m*dt
      else: balls[i].v += k* vector((springs[i].axis.mag-d),0,0)/m*dt - k* vector((springs[i-1].axis.mag-d),0,0)/m*dt
      balls[i].pos += balls[i].v*dt
```

1. Array based simulation

Change the above codes to the following array-based codes. We also need to complete the 2 lines commented by '##'. Notice that, in the very first line, we import numpy as a shortened name 'np'. Later at the line marked #5, we use numpy's class (arange) to generate the arrays to store the positions of the balls, the original positions of the balls, the velocity of the balls, and the spring lengths. We need to use np.arange() as the proper syntax for such purpose.

```
import numpy as np
from vpython import *
A, N, omega = 0.10, 50, 2*pi/1.0
size, m, k, d = 0.06, 0.1, 10.0, 0.4
scene = canvas(title='Spring Wave', width=800, height=300, background=vec(0.5,0.5,0), center = vec((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 = np.arange(N)*d, np.arange(N)*d, np.zeros(N), np.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[:-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 clearer view of the wave, we plot the longitudinal displacement of the balls from their original positions by a transverse displacement.

at #1, add code

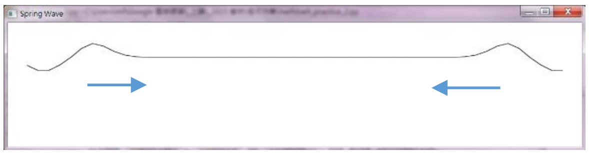
```
c = curve([vector(i*d, 1.0, 0) for i in range(N)], color=color.black)
```

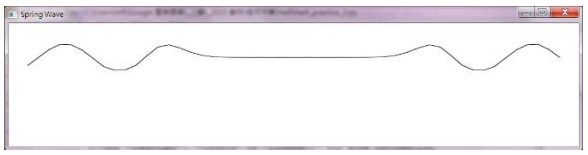
at #2, add code

```
   ball_disp = ball_pos - ball_orig
   for i in range(N):
      c.modify(i, y = ball_disp[i]*4+1)
```

3. Since we already have the clear representation of the wave, we can remove the ball-spring plotting, marked by #3. Notice that although the wave shown is with transverse motion, but it is indeed a longitudinal wave with a transverse wave representation.

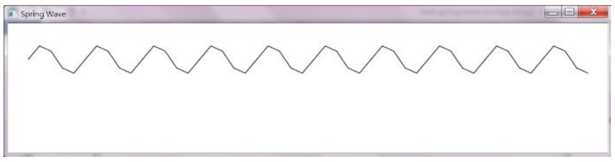
4. You may have seen such things in a video game that an object moves out of the right end of the screen and comes back from the left end of the screen. This is called "periodically boundary condition", a very often used assumption in physics and engineering. In our system here, it means that the last 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. As you saw in Practice 4, the wave amplitude at times gets larger and at times gets smaller. This means that this wave we try to create is not a "normal mode" of the system. The normal modes of the system are those waves whose amplitude can keep constant. This means that after going a full circle (Notice that we have the tail connected to the head, so we can view this system as a circle), the wave repeats itself, or we can say that wave satisfies the periodically boundary condition. These are waves with wavelength $\lambda = Nd/n$, where $n = 1, 2, 3, \ldots$ (i.e. wavenumber $K = 2\pi/\lambda = 2\pi n/(Nd)$). 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 = np.arange(N)*d + A*np.sin(phase), np.arange(N)*d, np.zeros(N), np.ones(N)*d
```

Notice in the last line, ball_pos is set to **np.arange(N)*d + A*np.sin(phase)**, in which **np.arange(N)*d** is the array for the balls' original positions at balance points. A in **A*np.sin(phase)** is the amplitude of the longitudinal wave, and each ball is originally displaced by A*sin(phase) before the simulation begins. phase in **phase = Wavevector * arange(N) * d** is the array for the phase of each ball's position in the longitudinal wave. We use np.sin(phase) because we want an array, elements of which are the sin function of phase, which is also an array. Then this array can be added to another array, np.arange(N)*d.
You also need to remove #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. Try to obtain the period (T) of the standing wave and the corresponding angular frequency (omega = 2*pi / T). For higher accuracy, please change dt = 0.001 to dt = 0.0003.

III. Homework (must)

In a crystal, only certain normal 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 (omega =2 pi/T) 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 lesson 5 homework (must) to plot the the angular frequency versus the wavevector.