Force and dynamics with a spring, numerical approach

It may strike you as strange that one of the first forces we will discuss will be that of a spring. It is not one of the four Universal forces and we don’t use springs every day. Or do we? The basic motion of a spring is to oscillate. Many, many mechanical systems oscillate. However, they might not appear to be springs. Take a tree branch bouncing up and down in the wind, is it a spring? How about the vibrations between two atoms in a molecule or a cork bobbing up and down on the surface of water, are these spring systems? From a physics point of view, all of these can be modeled by the same equations as those that describe a spring. This is why examining spring systems is so important. Later in the class we will make this mathematically precise, but for now just remember that the study of springs is far more general and important than a coil of metal with a mass hanging from it.

The force law, called Hooke’s law, for an ideal spring is \( \vec{F} = -k(\vec{L} - \vec{L}_0) = -k\Delta\vec{L} \). This says that the force that a spring applies is proportional to the amount that the spring is stretched from its equilibrium length. The constant of proportionality is negative, meaning that the force is directed in the opposite direction from the extensive or compressive displacement \( \Delta L \). Let’s look at the simplest spring system:

In this figure the mass is displaced to the right and the spring exerts a restoring force to the left.

Writing Newton’s equation for motion gives \( \vec{F} = \frac{d\vec{p}}{dt} = -k(\vec{x} - \vec{L}_0) \), combining this with \( \frac{d\vec{x}(t)}{dt} = \frac{\vec{p}(t)}{m} \), using the nonrelativistic \( \vec{p}(t) = m\vec{v}(t) \), gives a coupled set of first order differential equations.

We have turned the physical problem of a spring-mass system, through Newton’s equations of motion, into a mathematical problem, solving a coupled set of differential equations. Now we need to solve the equations to learn about the behavior of the mass-spring system. There are a number of ways to solve differential equations; in this section we will solve the problem numerically.

Since we know the force law, the equations can be numerically integrated simply. The algorithm looks like this:
Initialize needed variables
Initialize graphics

Main loop

- Calculate Forces on object(s)
  - Force_total equals sum of forces on object(s)
- Calculate new momenta by adding impulse
  - \( \text{mom} = \text{mom} + (F_{\text{total}})^*(dt) \)
- Calculate new position(s) using momentum
  - \( \text{pos} = \text{pos} + (\text{mom/mass})^*(dt) \)
- \( \text{time} = \text{time} + \text{dt} \)

Display results

Analyze results, often with aid of graphics

The code that we wrote together in class looked like this; the physics is highlighted in yellow:

```python
from __future__ import division
from visual import *
from visual.graph import *  # import graphing features

#Simulation of a spring mass system, no gravity

time = 0  # start the system clock at zero
dt = .01   # time increment (sec)
mass = 1  # mass of object
k = 100  # spring constant
L_0 = vector(0,0,0)  # relaxed position of the spring

#graphics
position = gdisplay(title = ' Y Position vs time',
    xtitle = 'time (s)', ytitle = 'y position (m)',
    x=0, y=0,
    width=1000, height=400,
    foreground=color.black,
    background= color.white)  # window for position vs. time

velocity = gdisplay(title = ' Y Velocity vs time',
    xtitle = 'time (s)', ytitle = 'velocity (m/s)',
    x=0, y=450,
)```
window for velocity vs. time

acceleration = gdisplay(title = 'Y acceleration vs time',
                        xtitle = 'time (s)', ytitle = 'Y acceleration (m/s**2)',
                        x=0, y=0,
                        width=1000, height=400,
                        foreground=color.black,
                        background=color.white)                # window for velocity vs. time

pos_vs_time = gdots(gdisplay=position, color=color.black)   # a graphics curve for position vs. time
vel_vs_time = gdots(gdisplay=velocity, color=color.black)   # a graphics curve for velocity vs. time
acc_vs_time = gdots(gdisplay=acceleration, color=color.black) # a graphics curve for acceleration vs. time

#initialize all loop variables
initial_vel = vector(0,0,0)
initial_mom = mass*initial_vel
initial_pos = vector(0,0,0)

position = initial_pos
momentum = initial_mom

#main loop
while (time < 10):   # simulate time < 10 sec
    F_s = -k*(position - L_0) # force produced by spring, a vector
    momentum = momentum + (F_s)*dt # note using total force in momentum update
    position = position + momentum/mass * dt    # position update

    # graphics
    pos_vs_time.plot(pos=(time, position.y ))        # plot y position vs time
    vel_vs_time.plot(pos=(time, momentum.y/mass ))   # plot y velocity vs time
    accel = F_s/mass                                # calculate acceleration
    acc_vs_time.plot(pos=(time,accel.y))             # plot acceleration y vs. time

    time = time + dt                          # increment system time

If we have a vertical spring and want to include the force of gravity there are only a couple of changes, highlighted in red:

from __future__ import division
from visual import *
from visual.graph import * # import graphing features

#Simulation of a spring mass system with gravity

time = 0          # start the system clock at zero
dt = .01          # time increment (sec)
mass = 10         # mass of object
k = 100           # spring constant
L_0 = vector(0,0,0) # relaxed position of the spring
g = vector(0,-9.8,0)  # acceleration near Earth

# graphics
position = gdisplay(title = 'Y Position vs time',
                     xtitle = 'time (s)', ytitle = 'y position (m)',
                     x=0, y=0,
                     width=1000, height=400,
                     foreground=color.black,
                     background=color.white)                # window for position vs. time

velocity = gdisplay(title = 'Y Velocity vs time',
                     xtitle = 'time (s)', ytitle = 'velocity (m/s)',

That is all that needs to be changed!

Using the initial conditions of dropping the mass with zero velocity from the equilibrium position of the spring yields this trajectory.
The mass oscillates about the position where the spring would be extended with the mass hanging at rest, \( y_0 = \frac{mg}{k} \) for the conditions used \( m = 10 \), \( k = 100 \), yields \( y_0 \approx 1 \).

Graphing position, velocity and acceleration vs time shows the phase relationship between the three. Note that the amplitudes all have different units.

![Graph showing position, velocity, and acceleration vs time](image)

The velocity (cyan) leads the position by \( \frac{\pi}{2} \) and the acceleration (red) is out of phase by \( \pi \).

So we are able to integrate Newton’s law for a spring and produce oscillatory motion. **Note that nowhere have we used the sine or cosine functions; the oscillations emerge naturally from the Hooke’s law spring force and integrating the resulting differential equations found by applying Newton’s second law.**