# Computer Model of Spring-Mass System

## GOAL

In the minilab on coiled springs you measured the spring stiffness $k_s$ of a long, soft spring, the mass $m$ of a weight, and the period $T$ of the oscillations for a system consisting of a mass hung from a soft spring. Now you'll put your measurements of $m$ and $k_s$ into a computer model (based, of course, on applying the momentum principle) and see how well the model predicts the motion of your spring-mass system.

You have also studied an "analytical" solution for the motion of a spring-mass oscillator, obtained by guessing a sinusoidal solution and seeing what must be true in order that this solution satisfy the momentum principle. In this computer model you will see how the sinusoidal motion emerges from a step-by-step "numerical integration" of the momentum principle, and also how a computer model can deal with situations that are beyond the scope of an analytical solution.

## STARTING WITH A PREVIOUS PROGRAM

Consider your program that modeled the change in momentum of a fan cart on a track due to the constant force associated with the fan, which looked something like this (we've made the track longer, 1.5 m).

```
from visual import *
from   __future__   import division # makes 1/2 be 0.5, not 0
track = box(pos=vector(0,-.075,0), size=(1.5,0.05,0.10))
cart = sphere(pos=vector(-0.5,0,0), radius=0.05, color=color.green)
cart.m = 0.80
cart.p = cart.m*vector(0.9,0,0)
deltat = 0.01
t = 0
while t<3.0:
    rate(100)
    Fnet = vector(-0.4,0,0)
    cart.p = cart.p + Fnet*deltat
    cart.pos = cart.pos + (cart.p/cart.m)*deltat
    t = t + deltat
```

**Changing division:** We've added a statement that we'll use from now on. The statement about **import division** (with two underscores preceding and two underscores following the word **future**) changes the standard behavior of the Python programming language. Like many other programming languages, Python normally 1/2 evaluates as 0 rather than 0.5. With the new statement in your program, Python evaluates 1/2 to be 0.5.

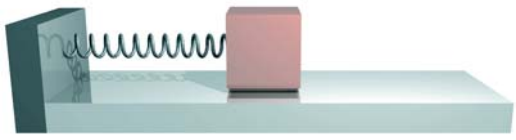
## WRITING A SPRING-MASS PROGRAM

The program above had a constant force law. The new program will be similar, but we will need to change the force law to match our mathematical description of the force exerted on a mass by a spring to which it is attached.

In the program, instead of hanging the mass on a spring, as you did in lab, we'll turn the setup sideways. We'll have the mass oscillating back and forth on a frictionless surface, such as an air table.

Why do we want to make the mass go sideways instead of up and down?

It's simpler - we only need to take into account the spring force (in the x direction); the gravitational force is cancelled out by the air supporting the mass. It turns out (see textbook) that the math describing the system comes out the same, so we should expect to see the same period in our computer model as we did in the lab.



## OBJECTS: SETTING UP THE SCENE

Using the fan cart program as a guide, write a similar program including the track but call the moving object "ball" rather than "cart" to emphasize that you're modeling the motion of a ball at the end of spring, which is what you observed in the lab.

We'll represent the spring by a cylinder. Like an arrow, the "pos" of a cylinder is at one end of the cylinder, and its "axis" is a vector from "pos" to the other end of the cylinder. Let the origin of the coordinate system be at the left end of the cylinder, where the spring is attached to the wall. Add the following at the start of the program, before the creation of the ball. Assume that the length of your spring when it was relaxed was 0.5 m, and call this `L0`:

```
L0 = 0.5
spring = cylinder(pos=vector(0, 0, 0), axis=vector(L0, 0, 0), radius=0.01)
```

For now we'll simply imagine a wall at the left end of the spring (you can make one yourself later if you like, using a box object like the box object used to display the track).

## IMPORTANT ISSUES

As usual, we need to make three important decisions:

- What initial conditions (position, momentum) should we use?
- What time step `deltat` ($\Delta t$) will give us acceptable results?
- How do we tell the computer to calculate the force vector each time through the loop?

We will also do something new: add a graph (of spring stretch vs. time) in a separate window.

### Initial conditions

You stretched the spring and started the ball from rest. So the initial position of the ball should be at the right end of the spring, where the spring is stretched in the x direction by the amount you used (probably about 0.15 m). You released the ball from rest, which tells you what the initial momentum of the ball should be. You also need to give values for the mass and the spring stiffness. (If you prefer, you can model the ball as a box rather than as a sphere.)

```
ball = sphere(pos=vector(?, ?, ?), radius=0.05, color=color.green)
ball.p = vector(?, ?, ?)
ball.m = ? # use the actual mass from your experiment
ks = ? # use the actual spring stiffness from your experiment
```

### Time step deltat

Your time step must be much shorter than the period of the oscillations you observed, or your calculations will be very inaccurate, since the force and velocity would not be even approximately constant during your large time step. Make `deltat` significantly less than the period you observed in lab.

### Calculating the force

We know that the magnitude of the force exerted by a spring on a mass attached to it depends on the stretch of the spring (the difference between the current length and the relaxed length of the spring) $|\vec{F}| = k_s|s|$. Further, we know that the spring is always tending to return to its relaxed length, so the direction of the force is toward the relaxed position, whether the spring is stretched or compressed.

In the fan cart program the force was a constant vector. Now the force is not constant, in either magnitude or direction. The quantity `ball.pos.x` is always the current x component of the position of the ball, and the relaxed length of the spring is `L0`. Remember that the stretch of the spring is the difference between the current length and the relaxed length of the spring, and that the direction of the force is to the left when the spring is stretched but to the right when the spring is compressed. Calculate the force in your program to express the force exerted by the spring on the ball.

### Updating the display of the spring

In your loop, update `spring.axis` each time, so that the "spring" is always displayed with its left end attached to the wall (which is the origin) and the right end attached to the ball.

**Run your program, and see what the motion looks like. If the behavior does not look like what you expect, revise your program, paying special attention to the force calculation.**

***Have an instructor check your program, then proceed to the next section.***

## MAKING A GRAPH OF THE POSITION OF THE BALL vs. TIME:

At the start of your program, add this statement to import the graphing module:

```
from visual.graph import *
```

Before the loop, add the following statement, to create a graphing curve object:

```
posgraph = gcurve(color=color.cyan)
```

Inside the loop, after updating the momentum and position of the ball, and the time, add the following statement:

```
posgraph.plot( pos=(t, ball.pos.x) )
```

As you might expect, this plots a point on a graph at a location given by *t* on the horizontal axis, and the x component of the position of the ball on the vertical axis.

You can prevent the two windows from overlapping if you place this statement just after `from visual import *`:

```
scene.y = 400 # move top of animation window down 400 pixels from top of screen
```

### Interpreting the graph

Why doesn't the graph oscillate above and below the x axis? What would you have to change in `posgraph.plot` to make the curve oscillate around the x axis? Make this change. Did you succeed? What are you plotting now? Leave your graph like this.

### Reading the period off the graph

Read the period of the oscillations off of the graph, and compare to your measured period. Is it the same? You may find it easier to read the period off the graph if you change the `while t<3.0:` statement to quit at some other time.

As a comment in your program, state the following results:

Period measured in lab = ?
Period of your computer model = ?
Period predicted from the analytical solution for a spring-mass oscillator = ?

### Numerical integration vs. analytical solution

You have just carried out a "numerical integration" in which you instructed the computer to add up many small impulses to compute the momentum, and many small displacements to compute the position. You saw emerge from this approach a graph that looks suspiciously like a cosine curve, even though you didn't program a cosine into your program. It is in fact a cosine, and one way to tabulate the values of the cosine function is by doing a numerical integration!

The motion of a spring-mass oscillator has an "analytical solution" which you studied in class. By applying the momentum principle we proved that the motion will be a cosine, and we predicted its period. When available, it is very useful to have an analytical solution. Unfortunately, there are not very many situations for which it is possible to derive an analytical solution, whereas numerical solutions are not limited to these very special situations. For example, if the force is proportional to the stretch cubed, there isn't an analytical solution. The two kinds of analysis complement each other. Each has its advantages and disadvantages. It is important that you gain experience with both kinds of approach.

### Varying the parameters

Experiment with your program and write down what you observe. Your instructor will check your results.

- Make the mass 4 times bigger. What is the new period? Does this agree with theory?
  (Afterwards, reset the mass to its original value.)
- Make the spring stiffness 4 times bigger. What is the new period? Does this agree with theory?
  (Afterwards, reset the spring stiffness to its original value.)
- Make the amplitude twice as big. (How?) What is the new period? Does this agree with theory?
  (Afterwards, reset the amplitude to its original value.)
- Change the force to be proportional to the stretch cubed (stretch**3) instead of stretch. Note that we do not have an analytical solution for this case: we need to do a numerical integration. What is the period?
  Double the amplitude. Now what is the period?
  How does this behavior differ from the behavior of an ordinary spring-mass oscillator?
  (Afterwards, reset the force calculation to represent an ordinary spring.)

***Have an instructor check your program and the results of varying the parameters.***

## TURNING IN YOUR PROGRAM TO WEBASSIGN

You will turn a slightly modified version of your program into WebAssign, where you will be asked to respond to various questions and will also be asked to explore what happens if you change the properties of your system in various ways.

## PLAYING AROUND

It is interesting to see what happens if you use an even more extreme force law, such as a force proportional to the stretch to the 7th power (stretch**7). With large amplitude, you can see that much of the graph looks straight (constant speed, small force), with sharp turn-around points where the force suddenly gets very large.

It isn't difficult to model a vertical oscillator. In addition to changing the positions of the spring and mass, you of course need to include the gravitational force of the Earth. What does your computer model show for the period of the vertical oscillator, compared to the period of the horizontal oscillator? (See textbook for a proof of this property.)

You can even model complicated motions in all three dimensions for a mass hanging from a spring. Create a vector r pointing from the support location to the mass, just as you did in your orbit programs, and create a unit vector rhat to use for the direction of the spring force. The magnitude of the spring force is obtained as usual from the stretch, which is the magnitude r minus the relaxed length of the spring. Include the gravitational force of the Earth. Give the mass an initial momentum that has components in all three dimensions, and you'll see complicated motion of the hanging mass. It is interesting to add a trail to the mass, to follow the complicated trajectory of the mass.

When you first start up the IDLE editor, the File open menu option shows you a variety of demo programs. Study the module helix.py. You'll see how you can replace your cylinder with an actual helical spring. Copy helix.py into the folder where your own program is stored, and in your program add a statement at the start "from helix import *". Note that if your spring is represented as a helix you can modify its direction and length by saying

```
spring.modify(axis=rhat, length=L)
```