# Oscillations

**Goal:** The goal of this mini-project is to design a suspension system for a car. There are two essential components for such a system: springs and shock absorbers. You’re going to write a design specification for each of them. Essentially, this means choosing a spring constant and a damping constant.

**Strategy:** We obviously are not going to build a car. So first, we’ll write, one piece at a time, a VPython simulation that we can use to test design ideas. It will model one wheel, one spring and one shock. We’ll start with the spring and wheel and add the shock later. Then we’ll add a model to jolt the spring and shock system with an impulsive force, similar to running across a speed bump. Finally, we’ll use this program to test design ideas and find a solution to the design goal.

**What your program must do:** Springs can take a variety of forms (coils, leaf springs, torsion rods, etc.) but they all behave the same way so we’ll take coils as our model. Before you can complete this assignment, you need to have a complete VPython simulation for a simple harmonic oscillator. Your simulation should be able to:

* Animate the oscillator itself (point mass is OK)
* Create one graph window that shows position, velocity and acceleration graphs vs. time
* Create a second graph window that shows Kinetic, Potential and Total Energy vs. time

Your graphs should be color coded so you can tell which curve is which quantity, with an identifying legend. It is handy, where possible, to color code the various forms of energy the same as the kinematic quantity they depend on.

In principle, you could hang your car from springs but that would be weird. The mass of a car sits on top of the spring and the bottom is a fixed position as long as the car is rolling smoothly. Your simulation should behave the same way.

Numerical simulations of oscillatory motion can be unstable *unless* you update velocity first and then update position.

Note that the force will change directions and magnitude during the oscillation. Depending how you have set up your simulation, you may find it useful to express the net force using a stretch unit vector. See chapter 3 for information on unit vectors. It is also possible to set up the animation so that you don’t have to do this but you really have to think about the coordinate system you’re using..

**What your program should accept as inputs:** Your program should take as inputs

* spring constant
* spring equilibrium length
* mass
* starting position
* starting velocity.

These don’t have to be dialog boxes. They can just be a set of lines of code that can be modified easily.

For right now, you don’t have to make it match a car, just get some combination of values that works. Later on, you can tune these values to match those of a car.

For each part, you should have a separate program. When you need to modify an existing program, copy it, rename it, and change the copy.

**How to prepare your program:** Start with a picture of your oscillator situation and coordinate system used by VPython. Use that and a free body diagram to get the net force *in this coordinate system* before trying to code it.

Build your program a bit at a time so you’re not doing too much at once. Just get the animation working before adding graphs, for example.

**A. Simple Harmonic Oscillations**

The first thing to do once you have written a numerical simulation is test it by verifying that it does what it is supposed to do in the areas where you have knowledge of how it should behave. The solution in the textbook for the simple harmonic oscillator is fully worked out, and leads a specific equation for *x(t).* This solution has specific relationships that must be satisfied by frequency, phase constant and amplitude.

and does not depend on A

and does not depend on ω

1. By systematically varying each of your input parameters in turn, determine qualitatively which properties of the motion (frequency, amplitude, phase constant) they have an effect on, and quantitatively whether those effects match the relationships in the text. How does each of your input parameters affect, or not affect, amplitude, frequency, and phase constant? One way to do this is to plot the analytic position solution and the simulated position solution on the same graph so you can compare.

2. Does your simulation agree with the relationship for ω? Explain, showing how you got ω from your simulation.

3. Does your simulation agree with the relationship for total energy? Explain, showing how you computed *E*.

4. For some set of values of the input parameters that you did *not* use in parts 1 through 3, **predict** quantitatively what each of the graphs should look like. Explain your predictions, in particular stating what you think the frequency, phase angle and amplitude for each graph should be. Then test your prediction. If you missed something, figure out why and try a different combination.

**B. Damped Oscillations**

Now add a damping force to your VPython simulation. This will require you to introduce another input, the damping constant.

1. Your textbook also has a solution for this problem. Check your simulation to make sure it agrees with the theoretical predictions. Explain, showing how you computed the relevant quantities.

for *b* small but not for *b* large

2. Explore the behavior of the damped oscillator for various values of the damping constant, from small to large. Be sure to especially note what happens at values less than, greater than, and equal to 2mω where ω is the simple oscillation frequency.

3. We categorize damped oscillations into underdamped, overdamped and critically damped based on the value of *b*. Based on what you’ve seen in part 2, explain what you think the meaning of these terms should be and what range of damping constants does each correspond to?

4. For the following real oscillators, should they be under-, over-, or critically damped? Obviously, any of them can be any of the three possibilities, so the best choice is dictated by design goals rather than physics. For each, **state what your design goal is as well as how that leads you to the type of damping desired.** You can watch them on Youtube if you want.

1. Door closers
2. Bungee cords
3. Automobile suspensions
4. Audio speaker supports (limit vibrations in the room structure)
5. The magnetic thing that makes the triple beam balance stop oscillating. We have some of these if you want to watch what they do.
6. Tuned mass dampers. You’ll probably need to look these up.

**C. Forced Damped Oscillations**

Your book discusses a periodic driving force, but for a car suspension, we’re really more interested in something like a speed bump or a pothole. That is, a sudden, impulsive force that doesn’t last for very long.

Introduce a constant force into your program that plays the role of a bump. Let that force only be active for a short time interval some time after the start of the simulation, and zero otherwise. It needs to hit the tire, not the car.

Following the same strategy as for previous sections, explore the effect of the force on your oscillator, both with and without damping. Try each kind of damping and think about which one you would want to ride in.

**D. Design a car suspension**

Now you need to adapt your program to simulate an actual car. This means you’ll need to estimate plausible values for things like mass, spring constant, damping constant, etc. You know enough about Newtonian analysis and Energy analysis to be able to do this.

1. Estimate (do not look up as you have to justify your estimates) a plausible spring constant for a typical car. This you should be able to do analytically.
2. Estimate a damping constant for your shock absorbers. This is where the VPython will come in handy.
3. Create a driving force to model a plausible real world situation of your choice. You can use any of the techniques we’ve learned this semester – Newton’s Laws, Impulse-momentum, Work-energy – to estimate (a) the size of the force from a bump and (b) the time interval over which it acts, based on the dimensions of the bump.
4. Use the VPython program to handle testing the impulsive force. You’re going to have to modify it a bit since both the wheel and the car body can move. You no longer have a fixed point on either end of the spring. Instead of the position, velocity, acceleration graph, replace it with a graph of the position only for the wheel and only for the car body. Does your car model behave appropriately?
5. Try a couple of other realistic scenarios in your program.