Code Quality
Validation of Mechanical Systems
December, 2nd 2013

Code Quality

Who cares?

For beginners: Code quality is primarily about making code that is readable, debuggable, and less likely to have bugs in the first place.

For intermediate programmers: Code quality is important because other people (and yourself in the future) will have to read your code. Poor coding practice is a red flag for employers.

For professionals: Code quality is one of the most important tools for managing complexity and making large-scale projects that can be maintained by large numbers of programmers over a long period of time.

Fundamentals of code quality

• Document interfaces: Every function should include a top-level comment that describes the input and output variables, and explains what the function does. It should contain the information the caller needs to use the function, and omit details they don’t need to know.

• Document information about the program that is not encoded as part of the program. Examples: the units of a quantity, the preconditions of a function, non-obvious details about how the code works.

• Avoid mystery constants: Numbers should be assigned to variables with meaningful names. They should not be scattered throughout the code, and definitely not repeated.

• Avoid evaluating the same expression more than once. Consider assigning intermediate values to variables with meaningful names.

• Avoid large blocks of repeated code. Copy-and-paste can be an efficient development plan, but once you figure out the pattern, pull the common code into a function.

• Use whitespace to make code readable, including line breaks between stanzas, correct indentation, and spaces around operators and between vector elements. Avoid long lines.
The code on the next pages was written by a ModSim instructor who will remain nameless. In their defense, this code was written quickly and not intended as an example.

Read through it and identify places that break the rules and/or places where you have a hard time understanding the code.
hopperanimation.m

% This script does a simple animation of a reaction mass hopper.
%
% It calls hopper.m, which takes the two masses as input, and returns the
% trajectory information.
close all
m1=1;
m2=1;
figure;
[t,Y]=hopper(m1,m2);
y=Y(1,:)
h1=rectangle('Position', [y(1), y(2), 0.1, 1], 'FaceColor', [0 0 1]) % create rectangles
h2=rectangle('Position', [y(3)-0.5, y(4)+1, 1, 0.1], 'FaceColor', [1 0 0])
axis([-8 8 0 16])

for i=1:length(t);
    set(h1,'Position', [Y(i,1)-0.05, Y(i,2), 0.1, 1]); % change position of rectangles
    set(h2,'Position', [Y(i,3)-0.5, Y(i,4)+1, 1, 0.1]);
    drawnow; % wait a little while so that the animation is visible
end

hopper.m

function [t,Y]=hopper(m1,m2) % calculate the trajectory of the hopper for given masses of the post and the reaction mass

% Define constants for the hopper
k=1000 % spring constant
l=0 % rest length of spring
R=1 % radius of disk (drives angle of spring)

% Set the events to capture end of simulation condition
options=odeset('Events',@events); %Stop when we hit the ground

% Set initial conditions
X=[0;0;0;-1;0;0;0;0]; % [postx posty reactionmassx reactionmassy vpostx vposty etc]

% Calculate the trajectory, with a fixed timestep to facilitate animations.
[t,Y]=ode45(@dHopperdt,[0:0.003:10],X,options);

    function res=dHopperdt(t,X) % Function that passes back derivatives based on position and velocities.
g=10;
r1=X(1:2);
r2=X(3:4);
v1=X(5:6);
v2=X(7:8);

% Deal with the rod normal force possibilities
if r1(2)<=0
    ay1=max(-springforce(r1,r2)/m1 - g,0); % make sure that if post is on the ground, it doesn't move unless pulled up.
else
    ay1=-springforce(r1,r2)/m1 - g;
end

% Deal with the reaction mass
ay2=springforce(r1,r2)/m2 - g;
ax1=0;
ax2=0;

% Pass out the results
res=[v1;v2;ax1;ay1;ax2;ay2]; % note that v1 and v2 are each two elements.

function res=springforce(r1,r2)
    % simple function to calculate the force in the y direction due
    % to the angled springs
    y1=r1(2);
y2=r2(2);
    res=k*(sqrt(R^2+(y1-y2)^2)-l)*(y1-y2)/sqrt(R^2+(y1-y2)^2);
end

end

function [value, isterminal,direction] = events(t,X)
    value=X(2)+0.1; % Check if we hit the ground (actually a little below it to avoid
    isterminal=1;
direction=-1;
end
**Validation: The Systems**

Today we’re going to talk about validating and exploring two different mechanical systems: the reaction mass hopper we discussed in class two weeks ago, and the infamous skateboarder.

To jog your memory, the hopper consists of a rod, with a “reaction mass” which is attached via a spring or latex tube to the rod. The hopper is loaded by stretching the tubing; when the hopper is triggered, the reaction mass accelerates up, and the hopper leaps (hopefully to incredible heights). We built a two particle model of this system: the hopper consists of two point masses, connected by a special spring that captures some of the behaviors of a latex tube at an angle:

![Diagram of the reaction mass hopper](image)

The skateboard system, on the other hand, consisted of a skateboarder on a pivoting ramp; we abstracted this to a point mass (the skateboarder) moving along the rotating ramp:

![Diagram of the skateboard system](image)

**Strategies for Validation**

A potentially useful framework for both generation and organization of different validation strategies, is to place validation strategies on a two axis plot with one axis being qualitative versus quantitative validation and the other being experimental versus first-principles validation.
Quantitative vs. Qualitative

*Quantitative:* the validation strategy examines the model in a numerical fashion. Typically, the end-result will be some sort of number that quantifies how well the model matches expectations.

*Qualitative:* the validation strategy examines the model to make sure it has the correct properties. The result of this type of validation might be an appraisal of your model behavior along the lines of, “yeah, that seems kind of sensible.”

Experimental vs. First-Principles

*Experimental:* the validation strategy compares the model’s behavior to data. For instance, you might compare the model’s behavior to measurements from the actual physical system.

*First-Principles:* the validation strategy verifies that the model obeys the laws of physics (e.g. conservation of energy, momentum, etc.).
Big Ideas

What are different strategies you could use to validate a model of a mechanical system? Try to identify at least three different concrete strategies and graphs (or other evidence) that you could use to convince an audience that your model and was sensible, for both the hopper and for the skateboard. For each strategy try to place it on the axes of Quantitative vs. Qualitative and Experimental vs. First-Principles.
Quantitative, First-Principles

Where’s the energy?

One approach in validating mechanical systems is to check if the system obeys the laws of physics. In cases where the system is conservative – i.e., there are no loss mechanisms, such as friction, drag, inelastic collisions, etc – the total energy in the system is conserved. Of course, keeping track of energy requires that you are able to identify where the energy is stored, and also that you can calculate that energy. This type of validation is quantitative since we are calculating a numerical value for energy and first-principles since we are comparing the model behavior to a well-known law of physics (conservation of energy).

Is the hopper, as we modeled it last time, conservative? Why or why not? Where is energy stored in the hopper system? Make a list.

Think back to the skateboarder. Is this system conservative? Why or why not? Where is energy stored in the skateboarder system? Make a list.
Calculate the energy

If you’re going to validate your model using energy conservation, you of course need to be able to calculate the relevant energies. Sometimes this is relatively easy to do, sometimes it takes a little more thought.

In the case of the hopper, ode45 passes back $y_{rod}$, $y_{ring}$, $\dot{y}_{rod}$, and $\dot{y}_{ring}$. We’ll derive expressions for the energy stored different locations in terms of these variables, as well as any appropriate constants.

In the case of the skateboard, ode45 passes back $r$, $\theta$, $\dot{r}$, and $\dot{\theta}$. We’ll derive expressions for the energy stored different locations in terms of these variables, as well as any appropriate constants.
Qualitative, First Principles - Limiting case validation

Examining limiting cases can be a very effective way to validate that a model and the implementation thereof are behaving as you’d expect them to. Often complicated dynamics become very simple when you manipulate parameters in your model to make one effect dominate over all the others. This type of validation strategy is qualitative since we are verifying that the behavior of the system has the right kind of behavior, based on a first principles model of a simpler limiting case. For example, in the case of a falling body, if you take the limiting case of turning off drag, you can immediately predict that the position versus time should be quadratic (good old $\frac{1}{2}at^2$).

In the case of the hopper, the obvious parameters are the masses, the spring constant, and gravity. In the space below, identify at least two different limiting cases for which the behavior of the system is really easy to predict – and predict what that behavior is!

In the case of the skateboard, there are quite a few parameters: the length of the ramp, the angle of the ramp, the initial velocity of the skateboarder, the mass of the ramp, the mass of the skateboarder, gravity... In the space below, identify at least two different limiting cases, and make a prediction of the behavior you should see.
Qualitative, Experimental - Make a Movie

Going beyond the first principles version of things, you can exploit your own innate sense of how the world works to see if the behavior of the model is qualitatively correct. This amounts to (1) making the movie in your head of what should happen, and (2) making a movie of your model to see if the model actually behaves in accord with your intuition.

This approach to validation is a bit dangerous, for couple of reasons. First, it’s easy to convince yourself that what you see in the animation is right, simply because you’re seeing it on your screen. Often people are good at coming up with complicated physical explanations for highly non-physical behavior. So it’s important to make sure that you play the movie in your head first, rather than looking at the animation first and then justifying it.

By the same token, it’s also possible to convince yourself that the model is wrong when in fact your intuition is wrong. For example, in creating the skateboard example, we initially believed that the “launch left” condition was not possible. We were wrong, and we only discovered this when we ran the animation, saw the behavior, and then did some careful thinking to convince ourselves that the result was credible.

Think back on your projects thus far. Have there been times when qualitative experimental validation has gone wrong for you? Gone right?
Quantitative, Experimental - Digitizing Physical Motion from a Movie

One strategy for validation is to perform a quantitative comparison of your model’s behavior and some experimental data. For instance, suppose we wanted to validate our model of a basketball free throw. If we could somehow gather some experimental data of the basketball’s flight path (position and velocity over time), then we could directly compare this data to our model’s predictions to get a sense of just how closely our model reflects the physical system.

Below are four key frames from a youtube video of a free throw. (http://www.youtube.com/watch?v=b3M2HJQolag). I used the site http://www.clipconverter.cc/ to download the clip to my computer. This video along with one of an unsuccessful freethrow are available on the course website.

A useful tool for taking a video like this one and producing data that can be imported into MATLAB is the Tracker: Video Analysis and Modeling Tool (http://www.cabrillo.edu/~dbrown/tracker/).
Writing code to validate and to explore

It is often helpful to spend some time explicitly creating code that helps you convince yourself that your model works as you expect it to. Download and unzip the MechanicalValidation package on the website. This package contains a variety of matlab scripts (not all elegantly written, and many that are poorly documented – sorry!) that illustrate approaches to validation.

Example Hopper Validation

Play with each of the following codes. What do they tell you? What do they NOT tell you? Which are useful for a particular validation strategy (e.g. limiting cases), and why?

1. HopperRawResultsPlot. This script creates a plot of the positions of the ring and the rod versus time.

2. HopperAnimationValidation. This script uses the same data and ode solver as HopperRawResultsPlot.

3. HopperEnergyConservationCheck. Again, this uses the same information as the previous two simulations. What does this approach tell you?

4. Use the provided simulations to investigate limiting cases, and to explore the behavior of the system. Does the model behave the way it “should”? Why or why not?

Example Skateboard Validation

Play with each of the following codes. What do they tell you? What do they NOT tell you? Which are useful for a particular validation strategy (e.g. limiting cases), and why?

1. SkateboardRawResultsPlot. This script creates a simple plot of the two position variables ($r$ and $\theta$) as a function of time.

2. SkateboardAnimationValidation. This script uses the same information as above, but creates a simple animation of the results.

3. SkateboardEnergyPositionValidation. This script uses the same information as the above two scripts, but instead plots both energy and shows key frames at different points during the motion.

4. SkateboardPlaneExploration. This script loads a pcolor of the time on the ramp as a function of the ratio of ramp mass to rider mass and the rider velocity, and then allows you to click on
different parts of the plane to see what the behavior of the system is for different combinations of mass ratio and initial velocity.

5. Using the provided codes, investigate limiting cases, and explore the behavior of the system.