2006-434: ANIMATION AS THE FINAL STEP IN THE DYNAMICS EXPERIENCE

Thomas Nordenholz, California Maritime Academy

Thomas Nordenholz is an Associate Professor of Mechanical Engineering at the California Maritime Academy. He received his Ph.D. from the University of California at Berkeley in 1998. His present interests include the improvement of undergraduate engineering science instruction, and the development of laboratory experiments and software for undergraduate courses.

Animation as the Final Step in the Dynamics Experience

Abstract

A method of incorporating animation into the student experience in the analysis of dynamics (especially vibrations) problems is presented. After a student models the problem, draws free-body diagrams, and derives equations of motion, he/she then obtains the solution for the position coordinates as functions of time. The student generates and plots the solution within a simple MATLAB program in which all parameters, such as mass, stiffness, damping, lengths, initial conditions, etc. can be easily changed. The solution can be generated using either a closed form solution or a numerical differential equation solver. In either case, at the end of the program, the student can animate *his/her own solution* by running an animation function file provided by the instructor. The function file requires a simple one-line command to run it. The function file does *not* solve for the motion of the system; it merely provides the animation graphics. Specifically, it displays the system in motion in real time (according to the student's solution) while simultaneously redrawing the student's plots. The animation function files are problem-specific. Several have been created by the author and are available for download.

The advantages of this approach to animation are that: i) it is simple, requiring only an elementary knowledge of MATLAB, and no additional software, ii) it can be used with *either* closed-form or numerical solutions to the problem, iii) it provides a physical interpretation of a student's mathematical solution (even if his/her solution is wrong!), and iv) it easily facilitates the investigation of how the parameters of the problem affect the motion.

Four examples will be presented to illustrate the scope of this method: i) a basic free spring/mass/damper, ii) a multi-degree of freedom system, iii) a suspension system subject to shock (requiring numerical solution), and iv) a dynamics problem (the rolling/slipping wheel).

The author's overall goal in using this approach is to provide students with a cumulative experience in dynamics, understanding how the complicated motion of systems results from the basic laws of mechanics.

This method of using animations has been used in the author's vibration course. Some feedback from the students on its effectiveness will be presented.

Finally, there will be a short section describing the basic techniques used by the author to program the animation files

I. **Introduction**.

Several engineering educators [1-5] have written on the use of animation in dynamics, vibrations, and controls courses. Certainly, the theory behind the motion of mechanical systems is mathematical and difficult for many students to grasp, and the animation of these systems provides enhanced understanding and motivation.

One common approach to animation involves the use of commercial software such as Working Model that simulates motion from objects drawn by the user without any mathematical formulation of the problem required by the user.[5] Animations of this type can be used alongside the theoretical solution developed in class, but there is no direct connection between the two.

Another method has been developed for use with the MATLAB Control Toolbox[3], and later, with Simulink [1, 2] simulations . Here, an animation function provided by the instructor can be incorporated with a numerical solution. In the case of Simulink, it is incorporated into a block diagram model of the mechanical system. Here, the user (for example, a student) would develop the block diagram, which represents the equations of motion of the system. The solution, however, is obtained numerically.

My ultimate (and elusive) goal in teaching dynamics and vibrations is for my students to understand a dynamics problem seamlessly from start to finish - from the derivation of equations of motion (applying the laws of motion and kinematical relationships), to the solution of those equations of motion (using techniques from differential equations), to the interpretation of the solution (through the use of plots and animations).

This presentation is an effort in moving towards that goal. It describes a method of using animation that is designed to *support* (rather than to *avoid*) the mathematical formulation studied in class. Essentially, this is an extension of the work of [1, 2, 3] to animations of closed form solutions studied in vibrations and dynamics and a simplification in the procedure, requiring elementary MATLAB programming only.

This method has been used in a vibrations course. Students are required to derive equations of motion for simple systems. Most of the problems encountered have closed form solutions; others can be integrated numerically. In either case, the student is required to write a short, simple MATLAB program which generates this solution (in the form of time and position coordinate arrays). All parameters, such as mass, stiffness, damping, lengths, initial conditions, etc., are left as variables that can be easily changed. At the end of the program, the student types a one-line command that runs an animation function. The animation function, provided by the instructor, animates the student's solution (whether it is correct or not) to the problem, while simultaneously plotting it in real time.

Following are four examples that illustrate this method, three in vibrations and one in dynamics.

II Free Vibration of the Spring/Mass/Damper

The free vibration of the mass, spring, damper, shown in figure 1, is one of the first systems encountered in a vibrations course.[6]

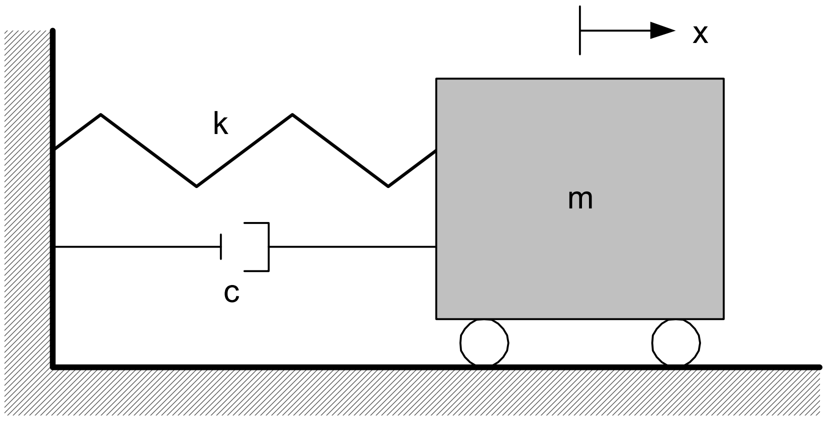

Figure 1

The equations of motion for displacement x as a function of time t are derived in class from Newton's Second Law:

$$\ddot{x} + 2\zeta\omega_n \dot{x} + \omega_n^2 x = 0, \qquad (1)$$

with the undamped natural frequency $\omega_n = \sqrt{\frac{k}{m}}$ and damping ratio $\zeta = \frac{c}{2\sqrt{km}}$,

and with initial conditions defined by $x(0) = x_0$, $\dot{x}(0) = v_0$. The analytical solution is derived in class from standard methods of differential equations. The form of the solution depends on the value of the damping ratio ζ. [6]

For ζ > 1 (the overdamped case)[i],

$$x(t) = a_1 e^{\lambda_1 t} + a_2 e^{\lambda_2 t} \quad where \quad \lambda_{1,2} = -\zeta\omega_n \pm \omega_n\sqrt{\zeta^2 - 1}, \quad a_{1,2} = \frac{\lambda_{2,1} x_0 - v_0}{\lambda_{2,1} - \lambda_{1,2}}. \qquad (2a)$$

For ζ = 1 (the critically damped case),

$$x(t) = (a_1 + a_2 t)e^{-\omega_n t}, \quad where \quad a_1 = x_0, \quad a_2 = v_0 + \omega_n x_0 \qquad (2b)$$

For ζ < 1 (the underdamped case),

$$x(t) = Ae^{-\zeta\omega_n t}\sin(\omega_d t + \phi), \qquad (2c)$$

where

$$\omega_d = \omega_n \sqrt{1-\zeta^2}, \quad A = \sqrt{x_0^2 + \left(\frac{v_0 + \zeta\omega_n x_0}{\omega_d}\right)^2}, \quad \phi = \tan^{-1}\left(\frac{\omega_d x_0}{v_0 + \zeta\omega_n x_0}\right).$$

The method for studying this problem now proceeds as follows. Students are asked to write a MATLAB program to compute x(t) for set values of the parameters m, k, c, x_0, and v_0. An example is shown below:

```
% free sping/mass/damper
clear,clc,close all
% set parameters
% all dimensions in m, kg, s
k=100;m=4;c=4;
x0=.2;v0=0;
% calculate wn(natural frequency)and z(damping ratio)
wn=sqrt(k/m);
z=c/2/sqrt(k*m);
% define time array
t=0:.02:5;
% generate x array (depends on z)
if z>1
    %overdamped case
    lam1=-z*wn+wn*sqrt(z^2-1);
    lam2=-z*wn-wn*sqrt(z^2-1);
    a1=(lam2*x0-v0)/(lam2-lam1);
    a2=(lam1*x0-v0)/(lam1-lam2);
    x=a1*exp(lam1*t)+a2*exp(lam2*t);
elseif z==1
    %critically damped case
    a1=x0;
    a2=v0+wn*x0;
    x=(a1+a2*t).*exp(-wn*t);
else
    %underdamped case
    wd=wn*sqrt(1-z^2);
    A=sqrt(x0^2+((v0+z*wn*x0)/wd)^2);
    phi=atan2(x0,(v0+z*wn*x0)/wd);
    x=A*exp(-z*wn*t).*sin(wd*t+phi);
end
%plot and animate!
freesmd_sim(t,x);
```

In the last line of the program, the student runs a function file which is provided by the instructor. The specific one used here, **freesmd_sim**, is displayed in the Appendix and is available for download. (See the Appendix.) The inputs to the function are the t and x arrays which are generated in the program. The function animates the motion of the mass while simultaneously plotting x(t) in real time[ii], as shown in figure 2.

The student can run several simulations of the system for different parameters m, k, c, and different initial conditions, and in so doing, can investigate how the parameters affect the motion.

If the student makes an error in obtaining the solution (or program), the incorrect solution is animated. This may tip the alert student off that something is wrong. A good example of this occurs when the constants of integration are incorrectly calculated, so that the initial conditions are not satisfied.

Of course, the instructor can write the program above and run simulations as in-class demonstrations.

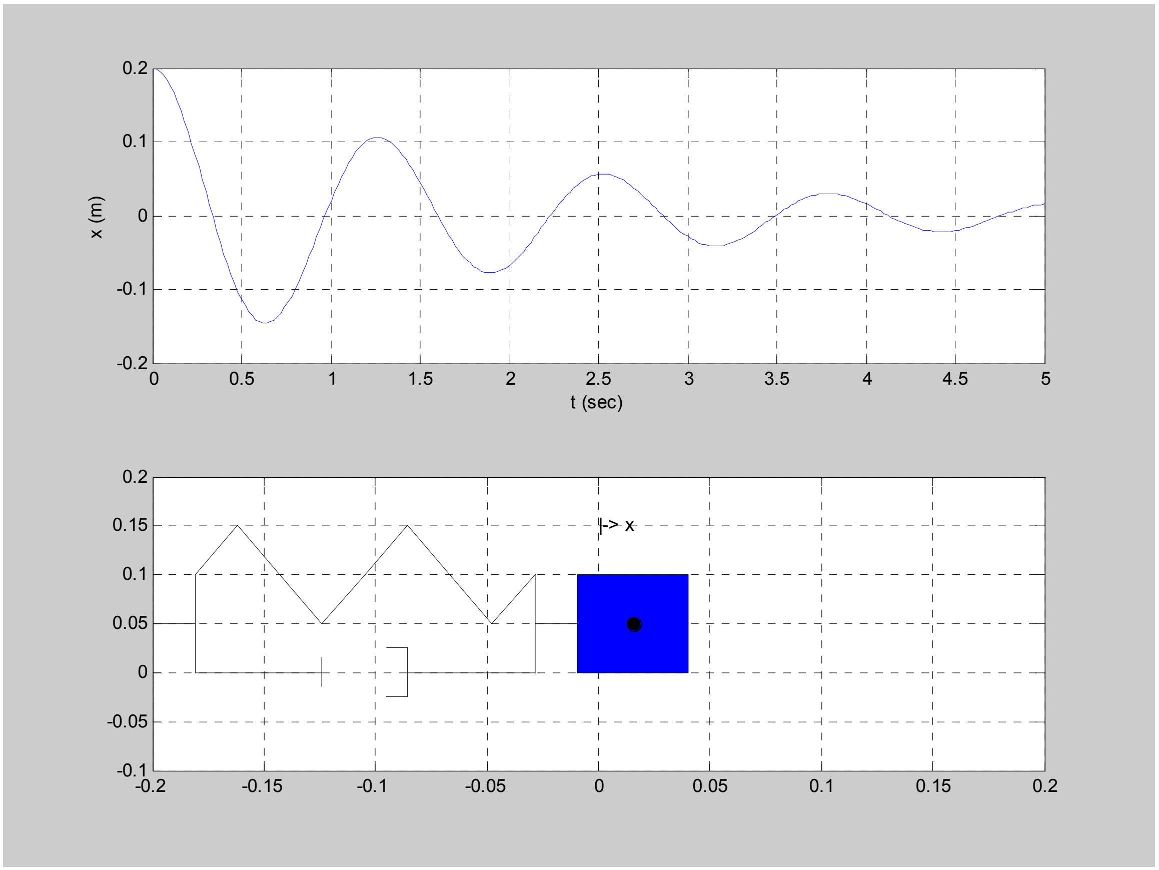

Figure 2

III A Multi-degree of Freedom System

This method is particularly suitable for multi-degree of freedom systems, where the computer is an essential tool for both the calculation of the coordinate responses (calculations which involve matrices), as well as the visualization of the complicated motions that occur. Figure 3 shows a classic three degree of freedom structure, modeled with the lumped masses (the floors) connected by massless springs (the walls).

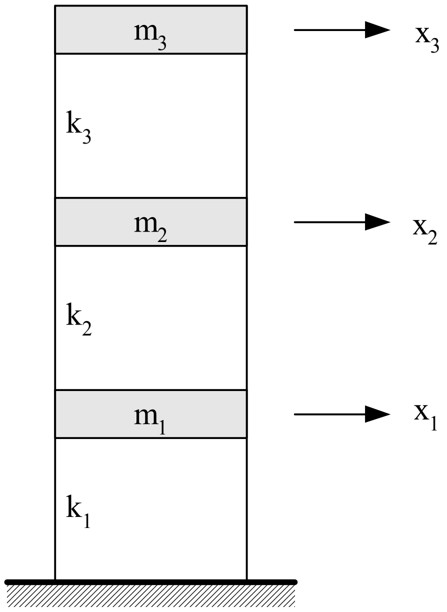

Figure 3

The equations of motion, derived from Newton's Laws, are

$$[M][\ddot{x}]+[K][x]=[0], \qquad (3)$$

where

$$[M]=\begin{bmatrix} m_1 & 0 & 0 \\ 0 & m_2 & 0 \\ 0 & 0 & m_3 \end{bmatrix}, \quad [K]=\begin{bmatrix} k_1+k_2 & -k_2 & 0 \\ -k_2 & k_2+k_3 & -k_3 \\ 0 & -k_3 & k_3 \end{bmatrix}, \quad [x]=\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}.$$

To solve this system of equations, Inman's [6] version[iii] of modal analysis, a procedure which decouples the equations of motion, is applied. The eigenvalues and eigenvectors of a symmetric *mass normalized stiffness matrix* $[\widetilde{K}]=[M]^{-\frac{1}{2}}[K][M]^{-\frac{1}{2}}$ are found. The eigenvalues are the squares of the natural frequencies. The (normalized) eigenvectors are arranged in columns in a matrix which is then premultiplied by the matrix $[M]^{-\frac{1}{2}}$ to form the *matrix of mode shapes* [S] (the columns of which are the mode shape vectors arranged in the same order as the corresponding eigenvalues) . The coordinate transformation [x]=[S][r] transforms the equations from *physical* coordinates [x] to a *decoupled* set of equations in the *modal* coordinates [r], which can be easily solved as functions of time (using simple single degree of freedom methods) and transformed back to physical coordinates.

After deriving the equations of motion expressed in (3), students write a simple program that performs modal analysis and calculates displacement responses $x_1(t)$, $x_2(t)$, and $x_3(t)$ for set values of the mass and stiffness parameters and initial conditions. For example,

```
%threedofbuilding
clear,clc,close all
%set parameters (units in kg, m, s)
m1=4000; m2=2000; m3=2000; k1=2e5; k2=1e5; k3=1e5;
%set M and K matrices
M=[m1,0,0;0,m2,0;0,0,m3];
K=[k1+k2,-k2,0;-k2,k2+k3,-k3;0,-k3,k3];
%get Ktilde(mass normalized stiffness),
Ktilde=M^(-.5)*K*M^(-.5);
%get eigenvectors eigenvalues, matrix of mode shapes
[P,L]=eig(Ktilde);
S=M^(-.5)*P;
%extract natural frequencies(wi)and mode shapes(ui)
w1=sqrt(L(1,1)),u1=S(:,1)%mode 1
w2=sqrt(L(2,2));u2=S(:,2)%mode 2
w3=sqrt(L(3,3));u3=S(:,3)%mode 3
% set initial conditions in physical coordinates [x]
x0=[0;0;.1]%initial positions
xdot0=[0;0;0] %initial velocities
% convert to modal coordinates [r]
r0=S^-1*x0
rdot0=S^-1*xdot0
%solve for [r(t)]
w=[w1;w2;w3];
A=sqrt(r0.^2+rdot0.^2./w.^2);
phi=atan2(r0,rdot0./w);
%set time array
t=0:.04:5;
%modal coordinate solutions
r1=A(1)*sin(w(1)*t+phi(1));
r2=A(2)*sin(w(2)*t+phi(2));
r3=A(3)*sin(w(3)*t+phi(3));
%transform back to [x(t)]  [x]=[S][r]
x1=S(1,1)*r1+S(1,2)*r2+S(1,3)*r3;
x2=S(2,1)*r1+S(2,2)*r2+S(2,3)*r3;
```

```
x3=S(3,1)*r1+S(3,2)*r2+S(3,3)*r3;
%plot and animate!
threedofbuilding_sim(t,x1,x2,x3);
```

The last line of the program runs a function file which is provided by the instructor. The specific one used here, **threedofbuilding_sim**, is available for download. (See the Appendix.) The inputs to the function are the t and x_1, x_2, and x_3 arrays which are generated in the program. The function animates the motion of the building while simultaneously plotting $x_1(t)$, $x_2(t)$, and $x_3(t)$ in real time, as shown in figure 4.

Students are typically asked to run this program for several different sets of initial conditions, including independent excitations of each of the modes, as well as complicated multi-mode motions (like the one shown in figure 4).

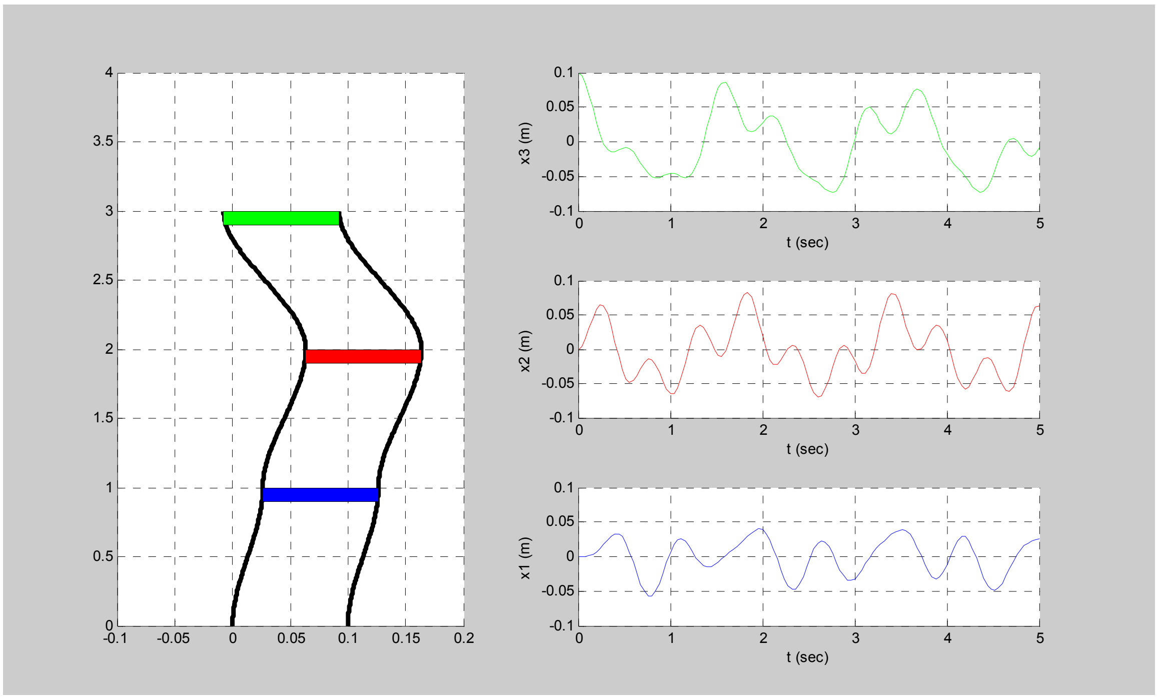

Figure 4

Damping (modal) and forcing can be easily incorporated. In the case of forcing, the animation file can be modified to provide applied force vector arrows acting on the floors.

IV Numerical Solutions

This method can also be used with numerical solutions. Consider the example of a 1 degree of freedom vehicle traveling over a sinusoidally shaped speed bump as shown in figure 5.[iv]

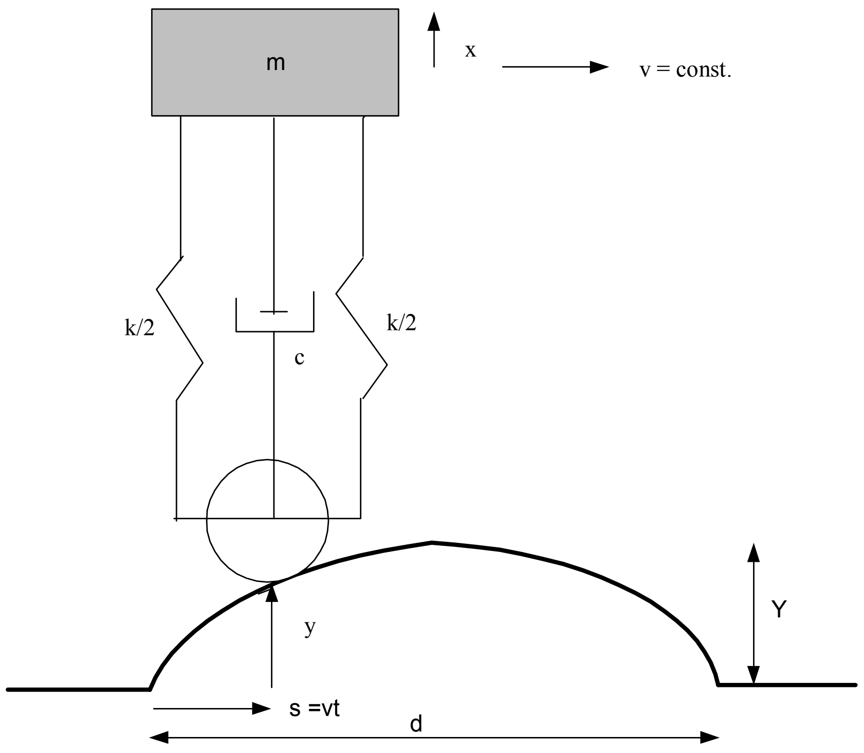

Figure 5

The equations of motion are

$$m\ddot{x} + c\dot{x} + kx = c\dot{y} + ky, \qquad (4a)$$

where x(t) represents the vertical displacement of the sprung mass of the vehicle, while y(t) represents the given base motion y(t):

$$y(t) = \begin{cases} 0, & t < 0, t > d/v \\ Y \sin \pi vt/d & 0 \le t \le d/v \end{cases}$$

Let F(t) = $c\dot{y} + ky$, the right hand side of (4a), since it is specified by the base motion above and acts as a forcing function to (4a). Defining $z_1 = x$ and $z_2 = \dot{x}$, the 2nd order differential equation (4a) can be rewritten as a system of two 1st order differential equations:

$$\begin{aligned} \dot{z}_1 &= z_2 \\ \dot{z}_2 &= -\frac{c}{m} z_2 - \frac{k}{m} z_1 + \frac{F(t)}{m} \end{aligned} \qquad (4b)$$

This system of differential equations can be solved in MATLAB using the ode45 numerical solver and a user-written function defining the system. This is shown below. The first program is the "driver" program which sets parameters, solves, and animates. The second program is the function **speedbump** which defines (4b).

```
%speedbumpdriver
% driver program to numerically solve and animate
% the motion of a cart over a speed bump
clear,clc,close all
% make parameters global for use by function
global v m k c d Y
%set parameters
m=1000;k=4e5;c=2e4;d=.8;Y=.1;
v=10;
% set initial conditions and time span
y0=[0;0];
tspan=d/v*linspace(-1,5,100);
% Use numerical solver ode45 to solve the differential
% equations represented in speedbump
[t,z]=ode45('speedbump',tspan,y0);
x=z(:,1); %ode45 returns z in 2 columns, z1 & z2
% x = z1
%animate and plot results
speedbump_sim(v,t',x');
```

```
function zdot=speedbump(t,z)
%function for use with speedbumpdriver
global m k c v d Y
if t<=0|t>(d/v)
    y=0;
    ydot=0;
else
    y=Y*sin(pi/d*v*t);
    ydot=Y*pi*v/d*cos(pi/d*v*t);
end
F=k*y+c*ydot;
zdot(1,1)=z(2);
zdot(2,1)=-k/m*z(1)-c/m*z(2)+F/m;
```

The last line of the driver program animates and (simultaneously) plots the motion x(t) using the animation function file **speedbump_sim** (available for download - see the Appendix). The inputs to **speedbump.sim** are the speed v of the car and the t and x arrays.[v] This particular animation is shown in figure 6.

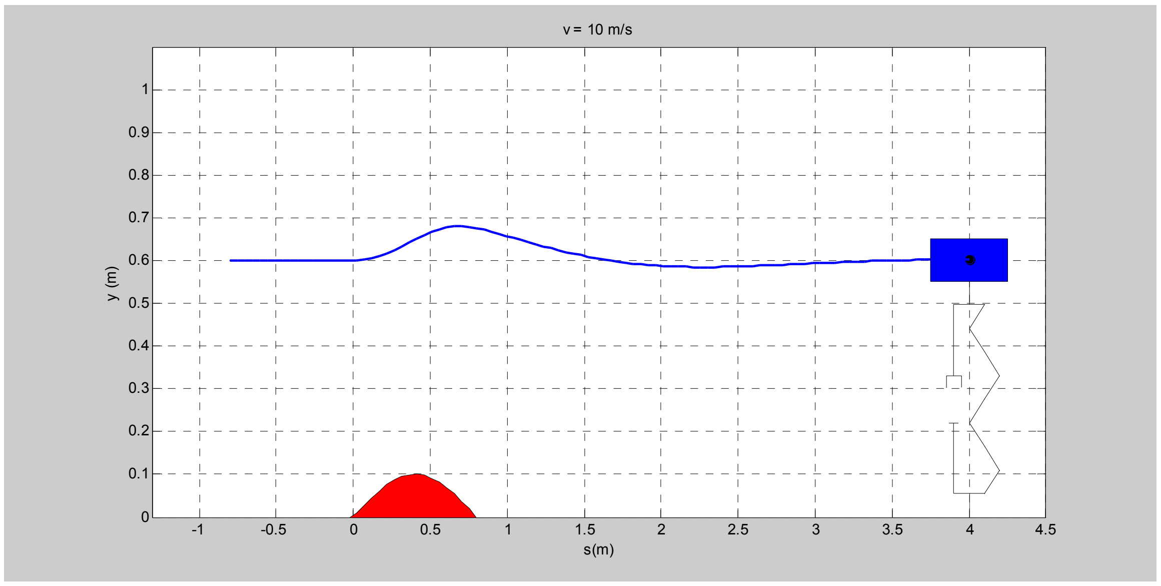

Figure 6

Students are required to derive the equations of motion and write the driver program and system function. The instructor provides the animation function. Students are typically required to run several animations at different speeds, in order to investigate the effect of speed on the vertical motion of the car.

Instead of using one of the **ode** functions solve the differential equations, Simulink can be used. A driver program sets the parameters and runs a Simulink Model (using the **sim** command). The solution array is returned to the workspace, and the driver program runs the same animation function (**speedbump_sim)** as used above.

V. A Dynamics Example

This method can also be employed in a dynamics course. Consider the example of a wheel of mass m, moment of inertia (about the center of mass) I_G and radius R, released from rest on an incline of angle β with a coefficient of friction μ (figure 7). Let x and θ denote translational and rotational displacement as shown.

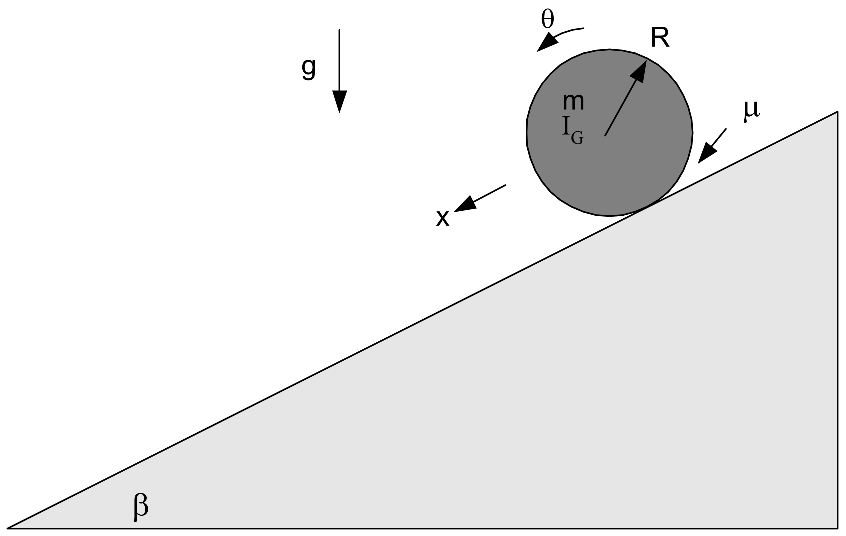

Figure 7

The wheel will either roll or slip, depending on the value of μ. After applying Newton's and Euler's Laws for translational and rotational motion and the necessary roll or slip conditions, one can conclude the following:

If $\mu \geq \frac{I_G}{I_G + mR^2}\tan\beta$, the wheel is *rolling without slipping* and the angular and linear acceleration of the wheel are

$$\ddot{\theta} = \frac{mg\sin\beta}{mR + \frac{I_G}{R}}, \quad \ddot{x} = R\ddot{\theta}. \qquad (5a)$$

If $\mu \leq \frac{I_G}{I_G + mR^2}\tan\beta$, the wheel is *slipping* and

$$\ddot{\theta} = \frac{\mu mgR\cos\beta}{I_G}, \quad \ddot{x} = g\sin\beta - \mu g\cos\beta. \qquad (5b)$$

In either case, it is easily demonstrated that linear and angular accelerations are constant and so can be easily integrated.

The student (or instructor, in the case of a class demo) prepares a MATLAB program to generate x(t) and θ(t) for a specified value of μ and a length of time sufficient for the wheel to travel a set distance d. For example,

```
%rollslipwheel
clear,clc,close all
%all dimensions in kg, m, s, rad
beta=30*pi/180; %incline angle
m=10;R=.1;
IG=m*R^2/2; %for a homogeneous cylinder
g=9.81;
mu=.3; %set mu here
d=2*pi*R;  %distance traveled by wheel
mucrit=IG/(IG+m*R^2)*tan(beta);%critical value of mu
% for no slip
if mu>mucrit
    %wheel is rolling
    alpha=m*g*sin(beta)/(m*R+IG/R); %ang accel
    a=R*alpha; %linear accel
else
    %wheel is slipping
    alpha=mu*m*g*R*cos(beta)/IG; %ang accel
    a=g*sin(beta)-mu*g*cos(beta); %linear accel
end
t=linspace(0,sqrt(2*d/a),100); %generate time array
x=a*t.^2/2; %translation x
theta=alpha*t.^2/2; %rotation theta
rollslipwheel_sim(t,x,theta); %animate!
```

The last line of the program animates the motion of the wheel using the animation function file **rollslipwheel_sim** (available for download – see the Appendix). The inputs to **rollslipwheel_sim** are the t, x, and θ arrays generated in the program. In this example, (m = 10 kg, R = .1 m, $I_G = \frac{1}{2}mR^2$, β= 30°), and the value of μ is set to .3, which is large enough for rolling without slipping. Furthermore, the distance d is set to one circumference of the wheel, so that during the simulation the wheel turns through exactly one revolution. (The wheel was released with its center positioned adjacent to the upper end of the line segment, and with its line marker parallel to and pointing up the incline) The results of this animation are shown below in Figure 8a.

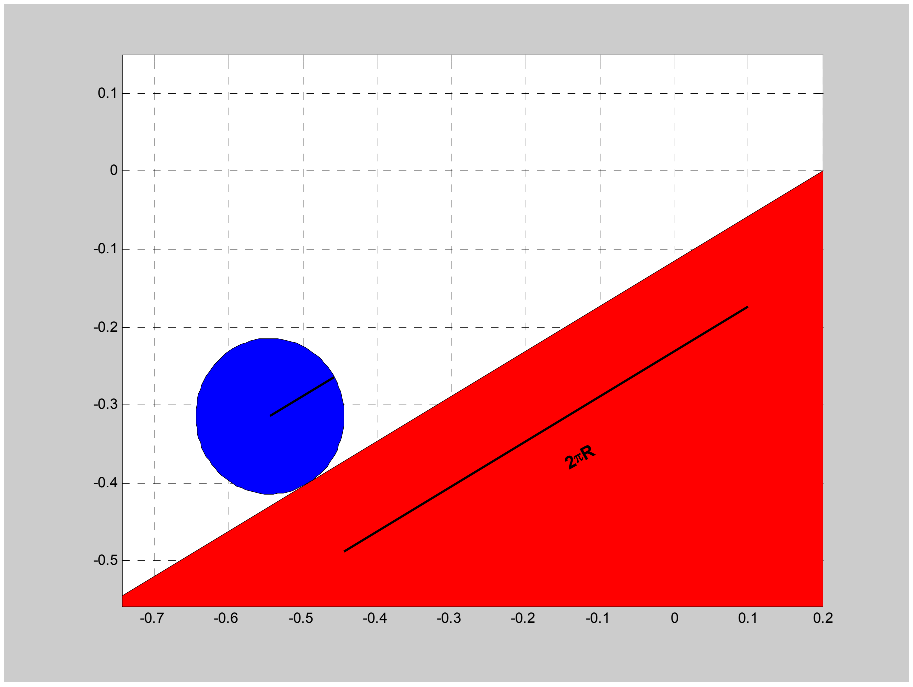

Figure 8a

If the value of μ (in line 8) is changed to .1, which is less than the critical value for no slip, and the program is rerun, the wheel is seen to slip as it rolls down the incline. This time, the wheel rotates less than once around as its center travels one circumference down the incline. The result is shown in Figure 8b.

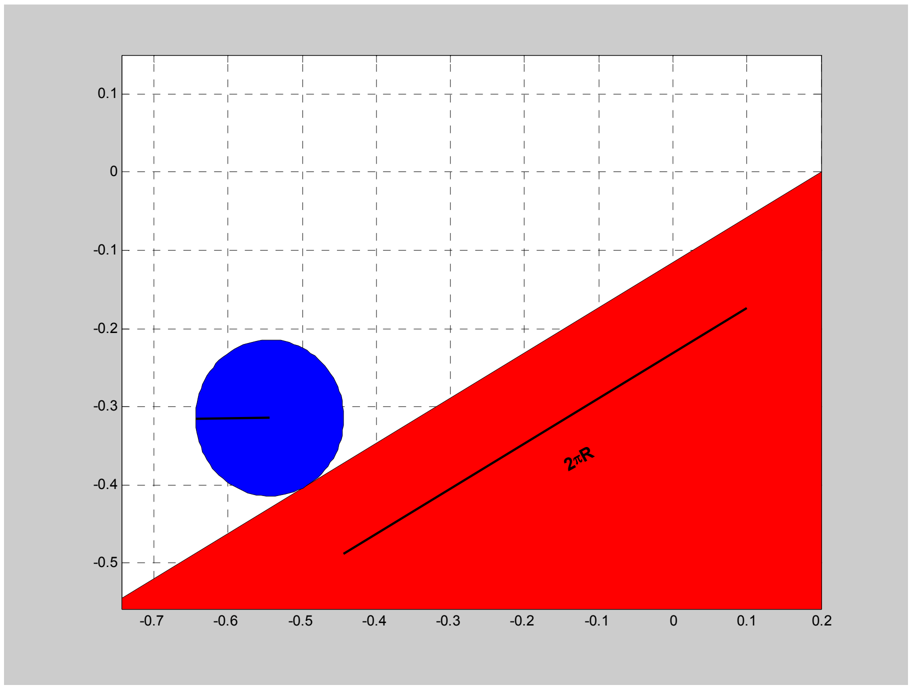

Figure 8b

VI Conclusion

All four cases above illustrate the basic approach to this method. The student formulates the equations of motion, and then writes a MATLAB program to generate the solution arrays for the position coordinates, either utilizing a closed form solution derived by the student (as in examples of Sections II, III, and V) or by numerical integration (as in example IV) .The program is written in such a way that the parameters can easily be changed. Then, instead of just plotting the position coordinates as functions of time, the solution is animated (and simultaneously plotted, in many cases) by a function file provided by the instructor. The function files are available for download, but are easy to create as well. (See the Appendix.) Once the program is written, the student can run several simulations for different values of the parameters, thus enhancing his/her understanding of how the parameters affect the motion.

Of course, the instructor can also play the role of the student described above for the purposes of doing in class demonstrations.

This approach was used in 21 assignments and in-class demonstrations (including the 3 examples of sections II, III, and IV) in a fall 2005 vibrations course at the California Maritime Academy. The class was very small (6 students), so that it is premature to draw conclusions about the effectiveness of this approach. However, the response from the class was overwhelmingly enthusiastic. A survey was issued at the end of the course to all 6 students. Students we asked to express their agreement or disagreement with three statements, on a scale of 1 (strongly disagree) to 5 (strongly agree). The results are tabulated below:

Statement	Average Student Response (1: strongly disagree, 2: disagree, 3:neutral, 4: agree, 5: strongly agree)
"I ran and carefully studied the animations whenever they were provided"	4.83
"The animations helped me to understand the problems"	5.00
"The animations helped motivate me in this course"	4.83

In addition, a space for comments was provided. Some of the comments were:

"Loved them, able to see what the math does"

"Very good learning tool!"

"The sim files helped to see effects of changing variables"

"It really motivated me to finish my m-file to see if it worked out properly. It also helped me fix my m-files if there was a problem"

"The sims helped visualize the actual problem and helped determine if the analysis is correct"

With the dynamics of such a small class, it is hard to tell whether the animations have significantly improved student performance, especially on exams. The use of animations in this course will be continued and their effect on student learning and motivation will be continually assessed.

Appendix: Animation Files

The animation files referred in the sections above, along with many more, are available for free download at the MathWorks File Exchange:

http://www.MathWorks.com/matlabcentral/fileexchange/loadCategory.do

Search for the author upon arriving at this site. The files can also be accessed from the author's website at: www.csum.edu/faculty/n/tnordenholz.

However, the animation files, which are problem specific, are generally easy to write. The following is a brief outline of the general procedure used. For illustration, the animation file of section II (the spring/mass/damper) is included.

The procedure is based on the techniques established in [2] and [3], with some modifications and enhancements. MATLAB Handle Graphics is used[vi]. There are three basic steps:

1. A function statement, followed by initialization of geometric parameters.

2. The creation, formatting, and initialization of plots. Subplots are used to create axes for both the position coordinate vs. time plots and the animations. The initial point of the plot and configuration of the system are drawn. Lines are drawn using the **line** or **plot** commands, and areas are drawn using the **patch**, **fill**, or **area** commands. Object handles are assigned to variable names for later access. Objects can be grouped (using the **hgtransform** command and setting the '**Parent**' property of all objects within the group to the group handle), allowing the entire group to be translated, scaled, or rotated (by setting values of the group property '**Matrix**'). All formatting of objects, like setting color and line styles, is done here. After all plots and objects have been created, the **drawnow** commands updates the figure, and a while loop with timing features **tic** and **toc** is used to hold the plots for one second.

3. The Animation. The animation proceeds by looping through the time and coordinate arrays, and updating the location of the objects within the plots at each step. Plots of position coordinates vs. time are always drawn from the initial value through the current value (so that a trace is seen). To simulate in real time, a while loop is inserted to wait until the current value of simulated time has elapsed in real time before plotting. (Commands **tic** and **toc** start a clock and read the elapsed time, respectively)

```
function freesmd_sim(t,x)
%animation function for a horizontal spring/mass/damper
%written by T. Nordenholz, Fall 05

% set geometric parameters
W=.05; %width of mass
H=.1; %height of mass
L0=.2;%unstretched spring length
Wsd=.5*H; %spring width
%plotting coordinates of mass
xrect=[-W/2,-W/2,W/2,W/2,-W/2];
yrect=[0,H,H,0,0];

%set up and initialize plots
%x vs t plot
Hf=figure('Units','normalized','Position',[.1,.1,.8,.8]);
Ha_f2a1=subplot(2,1,1);
Hls_f2plot1=plot(t(1),x(1));axis([0,t(end),-L0,L0]),...
grid on,xlabel('t (sec)'),ylabel('x (m)')

% animation plot
Ha_f2a2=subplot(2,1,2);
% create mass
Hp_f2rect=fill(xrect+x(1),yrect,'b');
axis([-L0,L0,-H,2*H]),grid on
Hl_cm=line(x(1),H/2,'Marker','O','MarkerSize',8,'MarkerFaceColor
','k');
% create spring/damper
Hgt_springdamp=hgtransform;
Hl_Lend=line([0,.1],[0,0],'Color','k','Parent',Hgt_springdamp);
Hl_Rend=line([.9,1],[0,0],'Color','k','Parent',Hgt_springdamp);
Hl_Lbar=line([.1,.1],Wsd*[-
1,1],'Color','k','Parent',Hgt_springdamp);
Hl_Rbar=line([.9,.9],Wsd*[-
1,1],'Color','k','Parent',Hgt_springdamp);
Hl_spring=line(linspace(.1,.9,9),Wsd*[1,2,1,0,1,2,1,0,1],'Color'
,'k','Parent',Hgt_springdamp);
Hl_dampL=line([.1,.4],Wsd*[-1,-
1],'Color','k','Parent',Hgt_springdamp);
Hl_dampLpist=line([.4,.4],Wsd*[-1.3,-
.7],'Color','k','Parent',Hgt_springdamp);
Hl_dampR=line([.6,.9],Wsd*[-1,-
1],'Color','k','Parent',Hgt_springdamp);
Hl_dampRcyl=line([.55,.6,.6,.55],Wsd*[-.5,-.5,-1.5,-
1.5],'Color','k','Parent',Hgt_springdamp);
% set initial length
```

```
L=L0+x(1)-W/2;
set(Hgt_springdamp,'Matrix',[L,0,0,-
L0;0,1,0,H/2;0,0,1,0;0,0,0,1]);
text(0,1.5*H,'|-> x');
% draw and hold for 1 second
drawnow
tic;while toc<1,end
tic

% Animate by looping through time and x arrays
% and redrawing at each value
for n=1:length(t)
   L=L0+x(n)-W/2;
   set(Hls_f2plot1,'XData',t(1:n),'YData',x(1:n));
   set(Hp_f2rect,'XData',xrect+x(n));
   set(Hl_cm,'XData',x(n));
   set(Hgt_springdamp,'Matrix',[L,0,0,-L0;0,1,0,H/2;
    0,0,1,0;0,0,0,1]);
   while toc<t(n),end;
   drawnow;
end
```

Bibliography

1. Chang, Timothy and Chang, Daphne, *Enhancing Learning Experience with Dynamic Animation,* Proceedings of the 2002 Annual ASEE Conference.

2. Zivi, Edwin, and Piepmeier, Jenelle A., *Dynamic System Animation Within a Simulink Laboratory Environment*, Proceedings of the 2001 Annual ASEE Conference.

3. Watkins, John, Piper, George, Wedeward, Kevin, Mitchell, E. Eugene, *Computer Animation: A Visualization Tool for Dynamic System Simulations*, Proceedings of the 1997 Annual ASEE Conference.

4. Brooking, Cole J, and Smith, Donald A., *Simulation and Animation of Kinematic and Dynamic Machinery Systems with MATLAB*, Proceedings of the 1998 Annual ASEE Conference.

5. Rezaei, Amir G. and Davari, Asad, *Teaching Vibration and Control Courses Using Animation, Simulation, and Experimentation*, Proceedings of the 2005 Annual ASEE Conference.

6. Inman, D., Engineering Vibration, 2nd Ed., Prentice Hall, 2001

7. Rao, S., Mechanical Vibrations, 4th Ed., Prentice Hall, 2004.

8. Hanselman, D. and Littlefield, B., Mastering MATLAB 7, Prentice Hall, 2005

[i] The second of (2a) represents two equations, one for λ_1 and one for λ_2. Similarly, the last of (2a) represents two equations, one for the subscripts of a and λ to the left of the comma, and one for the subscripts to the right of the comma.

[ii] Animations are performed in real time provided that the time increment in the time array is greater than the time required to render each step of the animation.

[iii] There are several ways of performing modal analysis. Inman's approach is used here.

[iv] This example is taken from Rao [7].

[v] This particular animation function is not written to receive geometrical parameters d and Y as inputs or global parameters, but it could easily be modified to do so.

[vi] See [8] for a guide of how to use MATLAB Handle Graphics.