An Integrated Vibrations and System Simulation Course

George M. Swisher, Corinne M. Darvennes
Tennessee Technological University

Abstract

This paper describes a junior-level, three-credit-hour, one-semester, required course in Mechanical Engineering (ME) at Tennessee Technological University. The authors have integrated the analytical (classical) study of vibrating systems with extensive use of digital simulation of the differential equations of motion. This course is a result of combining a traditional three-credit hour, one-quarter vibrations course with a one-hour, one-quarter digital system simulation course. Simulation, employing a sophisticated computation system, lends reality to the solution process and matches the procedures used by practicing engineers in that ME speciality.

I. Introduction

On the quarter system, the ME faculty taught a classical vibrations course emphasizing one and two degrees of freedom systems and their mathematical solutions. A follow-on, one-credit hour digital simulation laboratory (requiring the vibrations class as a pre-requisite) emphasized the numerical solutions of differential equations using such higher-level programs as SL-1 (developed by Xerox in the late 1960's), CSMP (developed by IBM in the late 1960's), ACSL[1], and now MATLAB®[2]; this evolution followed the introduction of each new package designed to solve differential equations. In 1989, during the university transition to the semester system, the faculty combined these two courses into the course described in this paper. MATLAB on a VAX mainframe system was started in 1992 with the migration to the PC version in 1996. The current prerequisites for the combined course are beginning courses in computer programming (FORTRAN or C), engineering dynamics, and ordinary differential equations. System simulation is accomplished using MATLAB's ordinary differential equation (ODE) solvers, primarily ODE23 and ODE45, in a just-in-time, balanced presentation mode. In the current curriculum, this is the ME students' introduction to MATLAB and also serves as the foundation for its later use in other ME courses.

II. Course Content

The major student outcomes of the course are that the student must be able to a) derive the system differential equations for single (one) and multiple degree-of-freedom vibrating systems {sdf and mdf systems}; b) solve for the analytical solutions for single degree-of-freedom systems for free and forced responses; c) solve analytically for the natural frequencies and mode shapes for multiple degree-of-freedom systems, with primary emphasis on two degree-of-freedom systems; d) demonstrate a fundamental capability of using MATLAB to manipulate matrices and solve coupled ordinary differential equations; e) design a simple vibration absorber system; f) utilize MATLAB's differential equation solvers to solve for time-domain free and forced responses of one and two degree-of-freedom systems; and e) use MATLAB's eigenvalue/eigenvector matrix functions to solve for the natural frequencies and mode shapes for

higher-order systems. In addition, the MATLAB skills developed in this course are used later in required Dynamics of Machines and Automatic Controls courses. SIMULINK®[3] (Dynamic System Simulation Software) and the CONTROLS SYSTEMS toolbox are the MATLAB programs of choice in the Controls class. Simulink is introduced briefly in this class when time allows. A typical semester outline is shown in the Appendix. The references to text sections in the syllabi and in the handouts are for the current text of James, Smith, Wolford and Whaley.[4] There are other excellent texts for this course. Three of them are cited in the references.[5, 6, 7] The authors have used Rao and Inman in past years.

Another very important consideration in an integrated course where classical solution techniques as well as modern numerical techniques are taught is the careful balance of the two concepts throughout the course. If that balance is not maintained, the students may over emphasize the MATLAB part of the course to the detriment of the classical (differential equation) solutions/techniques; the course can easily become a MATLAB programming course rather than using MATLAB to facilitate the mathematical solutions. To that end, all weekly homework assignments have both analytical problems, usually from the text, and also MATLAB ODE solutions to problems appropriate to that text chapter.

III. MATLAB Use

In order to get the student's started with MATLAB, the authors generated handouts on how to use the university's computer labs where MATLAB 5.2.1 is installed, a basic MATLAB primer, and a handout on how to utilize the ODE solvers such as ODE23. In addition, the students are referred to demos that come with the software and also to demos and tutorials at the vendor's website http://www.mathworks.com/demos. With these handouts about the TTU labs, the demos, and the excellent "on-line help" part of the program (including the full manuals available for viewing or downloading at the vendor's web site), we have not found it necessary to have the students purchase a separate MATLAB reference although there are several good ones on the market.[6] In addition, about 30 percent of the students have a student version of MATLAB on their personal computers at home, and a very good user's guide comes with the software. [2]

At least three class periods are devoted to hands-on and instructor supervised student work in the computer labs. Parts of these lab periods are used for instructor demonstrations. A two-lab tutorial is used to "walk them" through the basic MATLAB matrix computational functions and simple plotting routines. In the third lab, they are given instructor-written sample programs to experiment with. The most advanced of these is a program (with a corresponding derivative m-file program) to solve for the position and velocity of the unforced, undamped, one degree-of-freedom problem (spring-mass system). The program also calculates the analytical solutions and compares them to those numerically calculated from ODE23. The students run the program in lab for various initial conditions and system parameters. This handout is intended to not only demonstrate how ODE23 works but also to demonstrate many other facets of MATLAB in one sample program. A copy of the program written in MATLAB version 5.2.1 is item #1 in the sample handout section of the Appendix. Most students use this program as a template for more advanced programs assigned later in the course. A MATLAB vibrations toolbox (33 programs), a companion to Inman's text, is available as an educational tool from Prof. Joseph Slater of Wright State University at ftp://ftp.cs.wright.edu/pub/vtoolbox/. The instructors are not using the

toolbox because a major goal of the course is for the students to develop basic MATLAB programming skills for later use in the ME curriculum; using the toolbox is just "too convenient" and allows them to get results without understanding MATLAB programming. In addition, once the students have the "template program" provided in lab and understand ODE solvers, it is not conceptually difficult to modify the template to yield solutions for many typical free and forced vibrations problems.

A special advantage of simulation is to solve problems that require mathematical techniques (primarily solving nonlinear and higher order than two differential equations) beyond those acquired by undergraduate students. One example of this is solving nonlinear vibrating systems. The following sdf vibrating systems can be easily modeled using MATLAB to demonstrate the advantages of simulation for single degree-of-freedom (sdf) non-linear systems:

a) the simple pendulum (linear and nonlinear models)
b) coulomb damping (dry friction) in sdf systems
c) velocity-squared (fluid) damping in sdf systems
d) Van der Pol's equation (viscous and negative damping)

Some of the nonlinear problems MATLAB solutions are completed by the instructors and demonstrated in class, and others are generated by the students. In analytically solving problem a) above, the single pendulum, we invoke the "small angle approximation" to linearize the differential equation; the assumption is that for $\theta << 1$, $\sin(\theta)$ can be replaced by θ when θ is measured in radians. Students invariably will ask how "big" can θ be before this assumption of smallness is violated. A comparison of the outputs from the two programs will allow them to discover the answer for themselves.

Another excellent example of the use of simulation to help students understand difficult concepts is in solving for the time responses for transient inputs. Generally, students understand the step and ramp responses for one degree-of-freedom systems but have much more difficulty grasping the impulse response mathematical subtleties. Another difficult concept for them is the use of convolution for the response to a general pulse shaped input. In the last part of Handout #3, we try to help them, via MATLAB simulation, "discover" that the shape of a very short duration pulse does not affect the position time history i.e. only the area under the pulse matters, and that if the pulse duration is short enough compared to the system dynamic response, the output will approximate that of an impulse. In that handout, a triangular pulse was used. In some semesters, we use a rectangular pulse instead.

Another example where MATLAB is helpful is when solving for the time responses for multiple degree-of-freedom (mdf) systems, that is with degrees of freedom of two or greater (sets of coupled second-order differential equations), since analytical solutions are possible but very time-consuming and may require using a computer anyway to solve higher order polynomials with complex roots. One example of these problems (for a system with 2 masses, 3 springs, and 2 dampers) is shown in handout #4 in the appendix. Other examples are the double pendulum, building horizontal motion, and vehicle suspension systems.

Usually the last MATLAB assignment possible in a one-semester course is to computerize the calculation of the natural frequencies and mode shapes for multiple degree-of-freedom systems. By utilizing a standard second-order matrix formulation of the equations of motion, we must solve for the eigenvalues of the dynamic matrix, **D**, where $\mathbf{D} = \mathbf{M}^{-1}\mathbf{K}$ where **K** represents the system stiffness matrix and **M** represents the system mass matrix; the eigenvalues are the squares of the system natural frequencies. One must solve also for the eigenvectors of **D** since they are the associated mode shapes. The MATLAB *eig* function easily computes the eigenvalues and eigenvectors of **D**. Students find this technique very convenient/helpful after "cranking out" natural frequencies and mode shapes "by hand" for numerous two and three degrees of freedom problems. Some students master the SYMBOLIC toolbox enough to solve for the eigenvalues (square of the natural frequencies) as a product of k/m.

IV. Student Learning Problems/Challenges Encountered

The authors discovered some re-occurring MATLAB student learning issues especially when using ODE solvers. They were:

- A difficulty in thinking primarily in matrix functions/operations instead of scalar operations and understanding which MATLAB operations are scalar in nature versus array (matrix) ones. This issue was compounded by the fact that the current ME students are not required to take a formal matrix algebra course; the curriculum is being revised for next year to incorporate matrix techniques in earlier courses. This dimension difficulty is particularly true when they try to compare analytical solutions to numerically-solved ones. For the analytical solution (say x(t) versus t), a row vector t is usually generated by an implied loop of the type t=0.0:tmax/200:tmax, which generates a t row vector with 201 evenly-spaced numbers from 0 to tmax and then a row vector solution for x(t) can be computed using the t row vector. On the other hand, the ODE solvers, while in the iterative calculation mode, use scalar values of the independent variable, t. But when finished calculating, the ODE solvers return to the main program a column of t values and a matrix of the state variable values (a column for the values of each state variable) at each time increment; one of these columns is the set of numerical values for x(t). Unfortunately, the returned t and state space values are not at the same time spacings as the analytical solutions since the ODE solvers choose their own calculation step-sizes. All of above concepts become very confusing to junior-level students, and these differences must be reviewed for much of the course. The new Editor/Debugger in version 5.2 helps them see the dimensionality of variables much better than the techniques used with version 4.2.

- An unfamiliarity with/inexperience in formulating the higher-order differential equations (n-th order) into the n first-order (state space) equations required by the ODE solvers. For example, a two degree-of-freedom vibrations problem has two second-order differential equations and must be formulated into four first-order differential equations for the ODE solvers. It takes much of the semester for students to become comfortable with doing this.

V. Student Skills Developed

By combining vibration theory with MATLAB simulation, the students develop considerable skill in numerically solving coupled differential equations, which is the primary skill developed to complement the theoretical applications in this course. This requires them to think about whether the systems are "stiff" and select the proper ODE algorithm. However, there are many side benefits that they encounter in the process of solving differential equations via MATLAB. For example, they become comfortable thinking in a "complex matrix world" instead of always in a "real scalar world" by mastering many matrix manipulation skills; these include evaluating complex-valued matrices functions and finding matrix inverses, eigenvalues, and eigenvectors. Their skill in developing 2-D plots increases tremendously also. Some students develop considerable symbolic computational skills by doing some of their homework with the SYMBOLIC toolbox functions. We have found that the graduates of this course are ready to work with many of the other MATLAB toolboxes in their later undergraduate courses. For example, they use SIMULINK and the CONTROL SYSTEMS toolbox in the automatic controls course. In later kinematics courses, they use MATLAB skills to develop personal "toolboxes" to solve commonly-encountered problems, especially those in which matrix calculations are helpful. In machine design classes, one use is for the simplification of tedious fatigue factor calculations where previously empirically-derived chart lookup skills were needed; this is no longer necessary when they use 2-D curve fitting functions. In addition, when the students are given programming language choices in thermal science courses, they are now choosing MATLAB to solve fluid mechanics and heat transfer problems.

VI. Conclusions

The authors have been successfully teaching an integrated vibrations/digital simulation course for nine years; MATLAB has been the simulation language of choice since 1992. Balancing the classical solution of vibration systems with the application of a simulation package such as MATLAB continues to be a challenge for the instructors as well as for the students.

Bibliography

1. Mitchell and Gauthier Associates (MGA) Inc. *Advanced Continuous Simulation Language (ACSL) Reference Manual*, Concord MA (1991).
2. The Mathworks Inc., *The Student Edition of MATLAB® Version 5 User's Guide*, Prentice Hall, Englewood Cliffs, NJ, 1997.
3. The Mathworks Inc.(Dabney, J., B. and Harmon, T., L)., *The Student Edition of SIMULINK® Version 2 User's Guide*, Prentice Hall, Englewood Cliffs, NJ, 1998.
4. James, M. L., Smith, G. M., Wolford, J. C., and Whaley, P. W., *Vibration of Mechanical and Structural Systems with Microcomputer Applications, Second Edition*, Harper Collins, New York, NY, 1994.
5. Inman, Daniel J., *Engineering Vibration*, Prentice Hall, Englewood Cliffs, NJ, 1996.
6. Thomson, W. T. and Dahleh, M. D., *Theory of Vibration with Applications, Fifth Edition*, Prentice Hall, Englewood Cliffs, NJ, 1998

7. Rao, S. S., *Mechanical Vibrations, Third Edition,* Addison Wesley Longman, Reading, MA, 1995
8. Palm, William J., *Introduction to MATLAB for Engineers,* WCB/McGraw Hill Co., New York, NY, 1998.

GEORGE M. SWISHER
George M. Swisher, Professor of Mechanical Engineering, received the Ph.D. in M.E. (1969) from Ohio State University. He served as TTU's Dean of Engineering from 1989 until October of 1997. His areas of interest are linear systems, control systems, computer simulation, vibrations, measurement systems, and engineering education.

CORINNE M. DARVENNES
Corinne M. Darvennes, Associate Professor of Mechanical Engineering, received the Ph.D. in M.E. in 1989 from The University of Texas at Austin. Her professional activities include chairing the Nashville Section of ASME (1995-96) and being a member of several national professional societies. She regularly teaches undergraduate and graduate courses in vibrations, acoustics, noise control, and fluid mechanics.

Appendix (Typical Syllabi and MATLAB Handouts)

Typical Course Syllabus

Class Material
Vibration and Simulation
ME 305 - Fall 1998

Text: *Vibration of Mechanical and Structural Systems with Microcomputer Applications* by James, Smith, Wolford, and Whaley (2d edition)

Tentative Topics (in order)*	MATLAB Usage+	Material to Study/Use
Introduction		Book 1-1
Degrees of Freedom		Class Notes
Math (differential equations and matrices) Review		Handouts and Book 1-3
Dynamic review		Handout
Introduction to MATLAB	2 lab/HW	Handouts
Harmonic Motion		1-2
Single Degree-of-Freedom Systems		
Writing Differential Eq. of Motion (1dof)		2-2
Solving Differential Eq. of Motion (1dof)	1 demo/HW	2-3
Measurement of System Damping	HW	2-4
Coulomb Damping of 1dof	1 demo	2-5
Equivalent Springs		2-7
Rayleighs's Energy Method		2-8
Test One		
Harmonic Excitation of 1dof Systems	HW	3-1
Rotating Unbalance		3-2
Critical Speed of Rotating Shafts		3-3
Support Excitation	HW	3-6
Simulation of Forced Response of 1dof Systems	SIMULINK demo/HW	Handout
General Periodic Inputs/Steady State Response	1 demo	3-7
Vibration Isolation		3-8
Test Two		

Transient Vibration		4-1
Nonharmonic Force Excitation (step, ramp, impulse, and pulses)	1 demo/HW	4-2
Free Vibration of Multiple dof Systems	SIMULINK demo/HW	5-1 Handout
Differential Eq. Of Motion		5-2
Coordinate Coupling		5-4
Test Three		
Natural Frequencies and Mode Shapes	1 demo/HW	5-5
Forced Vibration of Multiple dof Systems	1 demo/HW	6-1
Equations of Motion		6-2
Simulation of Forced 2dof Systems		6-3
Vibration Absorber	HW	6-4
Final Exam		

+ MATLAB usage (lab is a supervised lab session, demo indicates done by instructor in class or handed out in class, and HW indicates an outside-of-class MATLAB assignment)* Topics do not necessarily represent individual class periods

Sample MATLAB Assignments

1. One Degree-of-freedom Lab Assignment (Undamped Free Response)

Main Program m-file (this line not part of numdf98 m-file)

```
% Free vibration of an undamped 1 dof system (Matlab version 5.2.1)
% Analytical and numerical solutions
% File called numds99.m
% input the data values from the keyboard
%
x0=input('enter the initial displacement in ft  ');
v0=input('enter the initial velocity in ft/second          ');
k=input('enter the spring stiffness in lbf/ft     ');
m=input('enter the mass in slugs      ');
tmax=input('enter the time range in seconds  ');
%
% Calculate the analytical solution
%
t=0:tmax/200:tmax;                % sets the time vector t for 201 spaces
omega_n=sqrt(k/m);                % calculates the natural frequency
A=sqrt(x0^2+v0^2/omega_n^2);      % calculates the amplitude
phi=atan2(x0,v0/omega_n);         % calculates the phase angle
x=A* sin(omega_n*t+phi);          % calculates the amplitude time function row vector
%
% numerical solution follows using ode23
%
save omega omega_n                % saves omega_n in a file omega.mat
                                  % so that we can read omega_n in
                                  % the function dif1s99
tspan=[0,tmax];                   % sets the time span on integration
initial=[x0;v0];                  % sets an initial conditions column vector
                                  % in ascending order of derivatives
[t_num,x_num]=ode23('dif1s99',tspan,initial);
                                  % evaluates x and t numerically.
                                  % The differential equation is defined
                                  % in file dif1s99.m
                                  % Note that now t_num and x_num are vectors
```

```
                                % plot the two solutions on one graph
                                % then plot the velocity found numerically on another graph
%
figure(1)                       % assigns a number to the first figure
xn=x_num(:,1);                  %extracts the position vector from x_num
                                % it's the 1st column of x_num
plot(t,x,t_num,xn,'*')          % plots the analytical solution x vs. t
                                % then the numerical displacement
                                % which is in the 1st column of x_num
title('Free vibration of an Undamped 1 Dof System')
xlabel('Time, s'),   ylabel('Displacement both x and x-num*')
gtext(['The Peak Value (A) is  ',num2str(A),' ft'])      % adds text with the cursor on a graph
                                % num2str changes data to printable characters
grid                            % puts a grid on figure 1
disp ('Hit any key to continue ');  % displays text on the screen
                                % as a reminder to get out of pause
pause;                          % gives us time to see the first plot
figure(2)                       % sets up the second figure for velocity
vn=x_num(:,2);                  %extracts the velocity from x_num
                                % it's the 2nd column of x_num
plot(t_num,vn);                 % plots the velocity evaluated numerically
title('Velocity of a Free, Undamped, 1 dof System');
xlabel('Time, s');  ylabel('Numerical Velocity, ft/second');
grid                            % puts a grid on figure 2

figure(3)                       % sets up the third figure for phase plane plot
plot(xn,vn);                    % plots the position versus velocity (phase plane)
title('Phase Plane Plot of a Free, Undamped, 1 dof system');
xlabel('Displacement, ft');  ylabel('Velocity, ft/second');
grid                            % puts a grid on figure 2
```

Derivative Program m-file (line not part of dif1f98 m-file)

```
function u_prime=dif1f98(t,u)
%
% Defines the differential equations for a free,
% Undamped,  one degree-of-freedom system
% x_dot_dot+omega_n^2 x=0
% That is solved numerically in numdf98.m
%
load omega                      % omega_n is defined in numdf98.m
                                % and saved in the omega.mat file
% Define the rows of u as u(1) = x and u(2) = x_dot, the ascending order assignment
% And u_prime(1)=x_dot=u(2) by definition
% Then, u_prime(2) = x_dot_dot = -omega_n^2 x = -omega_n^2 u(1)
u_prime(1)=u(2);                % the velocity definition
u_prime(2)=-omega_n^2*u(1);     %the differential equation
u_prime=[u_prime(1);u_prime(2)]; %the derivative vector to be solved by ode23
```

2. One Degree-of-freedom Handout (Damped Free Response)

Homework on MATLAB (Fall 1998)
Second Order Differential Equation Solutions using MATLAB

In this homework set, we will generate/use a MATLAB program using the ODE23 function to solve a general second-order differential equation of the type:

$$m\,\ddot{x} + c\,\dot{x} + k\,x = F_o$$

allowing the constants m, c, k, and F_o to vary. This equation can be rewritten into a common form as shown above.

$$\ddot{x} + 2\xi\,\omega_n\dot{x} + \omega_n{}^2 x = F(t)/m$$

In addition, in your program, allow the initial condition on x (x_o) and dx/dt (v_o) to be inputs. This is the typical output equation for a generalized input to any second-order system such as a spring-mass-damper system.

Part a: Generate a main program (m-file) and a derivative m-file program to solve this general problem (use ODE23) for the first equation shown above. Input m, c, k, F_o, and t_{max} from the MATLAB command window (CW). The t_{max} value can be adjusted to get the full response. Since we will be solving all three possible cases (underdamped, critically damped, and overdamped) using Ode23, you will not need to program all three analytical solutions. There would be three different ones depending on the value of zeta, ζ. Just program the one for the underdamped cases as the analytical solution, understanding that for the other two cases, you will get incorrect answers. For all these cases, enter F_o as zero. Your programs should be well-commented and should generate plots of x and dx/dt versus time. Compute the values of zeta (ζ) and ω_n for each case. The plots should be well-labeled, titled, and provide sufficient information to know which case you are plotting. From these plots, also pick off the peak x values and the damped period ($\tau_{d\ measured}$) if they make sense. For example, in the overdamped and critically damped case, τ_d is meaningless since there are no oscillations. Compare the measured value of the damped period with the calculated value from the equation in the text for the applicable cases.

1. After debugging your programs, run the four test cases shown below:

Case	1 Undamped	2	3	4
x_o	1	1	1	2
v_o	0	0	2	-1
m	1	1	1	4
c	0	6	2	5
k	9	9	9	4
$\tau_{d\ measured}$				
$\tau_{d\ calculated}$				
$x_{max\ meas.}$				

Turn in your programs and the plots for the four cases shown.

Part b: In this part, you are to generate a very lightly damped case, say about ζ = 0.06, for initial conditions of your choice using your program developed in Part a. From your output, verify that Equation (2-31) on page 68 in the text is correct. That is compute the zeta from your measured data for two peaks. Again set F_o equal to zero. Use at least two damped cycles in your calculations.

Part c: In this part, we will generate what is referred to as a step response for some typical cases. This is accomplished by entering F_o as 8. We will obtain the analytical solutions in Chapter 4. If there is a steady-state solution (which will occur if there is some damping present), it is of value F_o/k (see Equation 4-12, page 234). Only measure τ_d if there are oscillations. The $x_{max\ meas}$ value is the peak value from your plot.

Case	1 Undamped	2	3	4
x_o	1	1	1	2
v_o	0	0	2	-1
m	1	1	4	4
c	0	4	1	10

k	4	4	4	4
$\tau_{d\ meas.}$				
$x_{max\ meas}$				

Turn in the program, data and plots for the four cases.

3. Transient Analysis of One-Degree-of-Freedom Systems (midway in the semester)

Homework on MATLAB (Fall 1998)
Second Order Differential Equation Solutions for Some Common Transients

In this homework set, we will generate/use a MATLAB program using the ODE23 function to solve the spring-mass-damper systems for some common transient inputs (step, ramp, and rectangular pulses):

$$m\ \ddot{x} + c\ \dot{x} + k\ x = F\ (t)$$

allowing the constants m, c, k, to vary. This equation can be rewritten as shown:

$$\ddot{x} + 2\xi\ \omega_n \dot{x} + \omega_n{}^2 x = F(t)/m$$

In addition, allow the initial condition on x (x_o) and dx/dt (v_o) to also be inputs. Generate main program (m-file) and a derivative file programs to solve the following transient input problems (use ODE23) for the first equation shown above. Input x_o, v_o, m, c, k, $t_{max,}$ and other needed inputs from the command window. We are going to concentrate on underdamped systems at rest (both initial conditions are zero) for this homework set. Attached are the analytical solutions for all three cases and non-zero initial conditions. Your programs should be well commented and should generate plots of x (t) versus time unless otherwise specified.

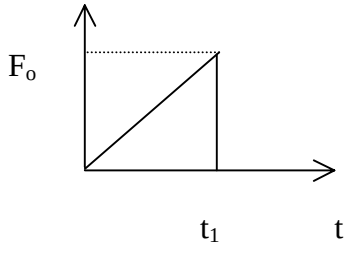
Case I (Step response): Generate a MATLAB program to solve the equation for a step input of size F_o (as shown in Figure 4-1b) ; the analytical solution for x_o=v_o=0 is Eq. 4-13 in the text. Use m=16,c=0.6, k=16, and F_o =16. You can use the data in Figure 4-4 to check your result. Compare your MATLAB result to that generated by Eq. 4-13 (since it is underdamped) on the same graph or use the equations in the attachment. When generating the analytical solution in the main program, you will probably need to use the dot commands (.*t, for example) since the t variable will be a vector.

Case II (Ramp response): Generate a MATLAB program to solve the equation for a ramp response {F(t) = F_o t}; there is not an analytical solution for x_o = v_o= 0 in the text (see the attached handout). Use m=2, c=0.2, k=2, and F_o =4. The displacement x(t) will have a transient term (which will die out), and a steady-state linear term of

$$x_{ss}\ (t)\ =\ \frac{F_o}{k}\left[\ t\ -\ \frac{2\ \varsigma}{\omega_n}\right]$$

which will survive. Plot your x (t) and the steady state response on the same graph.

Case III (Pulse-type responses): The objective of this case is to generate a MATLAB program to solve for triangular pulse type inputs (as shown in Problem 4-4 and below) and ultimately approximate the impulse response.

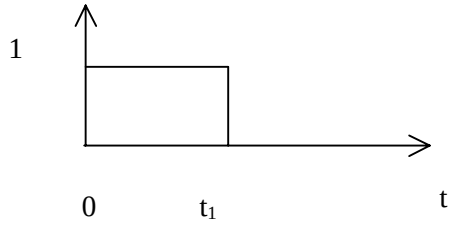


The analytical impulse response is for a relaxed system($x_o=v_o=0$) and is Eq. 4-5 in the text with C (the area of the pulse) equal to 1. Use m=6, c=0.5, k=6, for all test runs. Compare your MATLAB results to that generated by Eq 4-5. Only the fastest pulses (cases 4 and 5) will give outputs that look close to that of Eq. 4-5. All the triangular pulses are of height F_o and length t_1 and have an area of 1 (that's why C=1). See the hint below as one way to generate the triangular pulse in your derivative m-file program.

Case	1 (long pulse)	2	3	4	5 (short pulse)
F_o	0.25	0.8	2	10	20
t_1	8	2.5	1	0.2	0.1

Turn in your programs and the plots for all cases.

Hint: Here's a controls/system modeling hint (trick) that makes the simulation for case III easier (and more sophisticated). Of course, you could use IF statements in your derivative m-file program and build a function which is the triangular pulse, but it is usually better to make use of the mathematical and systems functions already available in MATLAB. A rectangular pulse of height 1.0 that goes from t=0 to $t=t_1$ only (see figure below).



This function can be modeled using a MATLAB *sign* (signum in mathematics) function where:

$$sign\ (x) = \begin{cases} 1 \ if & x > 0 \\ -1 \ if & x < 0 \end{cases}$$

Use MATLAB *help sign* to read more about this function. Then a step function (part I) can be defined as Step $=F_o$*sign(t) since we don't care about negative time (that is, our t will always be greater than 0). Then the rectangular pulse of height 1.0 and duration t_1, F, as shown above can be defined after working with the two sign functions a little as:

$$F = [\text{sign}(t) - \text{sign}(t - t_1)] / 2$$

Check it out. When $0<t<t1$, F is [1-(-1)]/2=1 and when $t>t_1$ F is [1-1]/2=0 - just the property that we want. Then the triangular pulse can be modelled by multiplying a ramp function by the rectangular pulse F as

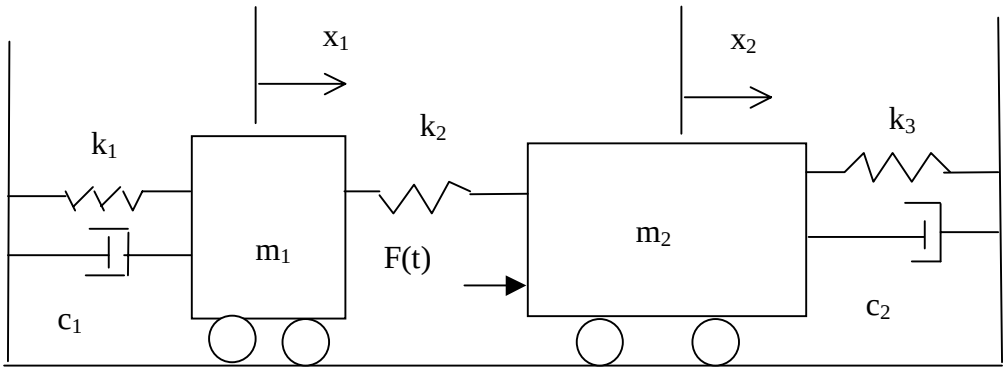
$$\text{Triangular Pulse} = F_o\ (t / t_1)\ F$$

4. Handout on Two Degree-of-freedom Systems (assigned late in the semester)

Homework on MATLAB (Fall 1998)
Higher-Order Differential Equation Solutions
Free, Step, and Sinusoidal Responses

In this homework set, we will generate a MATLAB program to solve a two-degree-of-freedom spring-mass-damper system, specifically Problem 5-1 in the text with a damper between each body and the wall and a potential force on

body two, for the free response, for a common transient, the step input, and the steady-state response for some typical sinusoidal inputs.



The two equations of motion are shown below.

$$m_1\ddot{x}_1 + c_1\dot{x}_1 + k_1 x_1 + k_2(x_1 - x_2) = 0$$

$$m_2\ddot{x}_2 + c_2\dot{x}_2 + k_3 x_2 + k_2(x_2 - x_1) = F(t)$$

Note that the two second order differential equations are coupled. Allow the system constants k_1, k_2, k_3, c_1, c_2, m_1, and m_2, to vary. In addition, allow the initial conditions on x_1, {(x_{10}) and dx_1/dt (v_{10})} and on x_2 {$x_{20,}$ and dx_2/dt (v_{20})} to be inputs. Generate a main program (m-file) and a derivative file program to solve this system of equations for a general input, F(t). Your derivative program m-file derivative vector will now be of fourth order since we are solving two coupled second-order differential equations. Remember that ode23 requires a column vector first order differential equation. Input x_{10}, v_{10}, x_{20}, v_{20}, the system parameters c_1, c_2, k_1, k_2, k_3, m_1, m_2, $t_{max,}$ and any other needed inputs from the command window. We are going to concentrate on underdamped systems for this homework set. Your programs should be well commented and should generate plots of x_1 (t) and x_2 (t) versus time unless otherwise specified. Hint: the most common method of specifying the four state variables (to allow us to generate the four first-order differential equations for MATLAB simulation) is to let $y_1=x_1$, $y_2=dx_1/dt$, $y_3=x_2$, and $y_4=dx_2/dt$.

Then the four first order differential equations to program in your derivative m-file are:

$$\frac{d}{dt}\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} y_2 \\ \text{From the first equation,} \quad \ddot{x}_1 = \dot{y}_2 = f_1(y_1, y_2, y_3, y_4) \\ y_4 \\ \text{From the second equation,} \quad \ddot{x}_2 = \dot{y}_4 = f_2(y_1, y_2, y_3, y_4) \end{bmatrix}$$

Case I (Free response): Generate a MATLAB program to solve the equations for initial conditions only {F(t)=0}. Use m_1=4 slugs, m_2=8 slugs, c_1= c_2=1.5 lbf-sec/ft, k_1=4 lbf/ft, and k_2=2 lbf/ft, k_3=4 lbf/ft. Generate a solution for x_1(t) and x_2 (t) for initial conditions of x_{10}=1 ft, v_{10}= -1 ft/sec, x_{20} = 2 ft, and v_{20} = 2 ft/sec.

Case II (Step response) Modify your program above to solve the equation for a step input in F(t) of variable size. We will use F_o = 16 lbf with the system at rest (all initial conditions equal to zero). Use the same system parameters from Case I. Plot your x_1(t) and x_2(t) on different graphs. As a check on your simulation, the steady-state constant values for x_1 and x_2 should be as defined as below:

$$x_{2ss} = \frac{F_0(k_1 + k_2)}{k_1(k_2 + k_3) + k_2 k_3}$$

$$x_{1ss} = \frac{x_{2ss}\, k_2}{k_1 + k_2}$$

These steady-state results were obtained from the differential equations by setting all derivatives equal to zero and solving for x_{1ss} and x_{2ss}.

Case III (Steady-state sine response) Modify your program above to solve the equation for a sine wave input in F(t) of variable size and frequency. We will use $F_o = 16 \sin(\omega t)$ for $\omega = 0.5$, 1, and 3 radians/second with the system at rest (all initial conditions equal to zero). Use the same system parameters. Plot your $x_1(t)$ and $x_2(t)$ on different graphs. As a check on your simulation, the steady-state amplitude values for x_1 and x_2 should be as defined below:

$$|X_1| = \left| \frac{F_o\, k_2}{(k_1 + k_2 - m_1\omega^2 + c_1 j\omega)(k_2 + k_3 - m_2\omega^2 + c_2 j\omega) - k_2^2} \right|$$

$$|X_2| = \left| \frac{F_o(k_1 + k_2 - m_1\omega^2 + c_1 j\omega)}{(k_1 + k_2 - m_1\omega^2 + c_1 j\omega)(k_2 + k_3 - m_2\omega^2 + c_2 j\omega) - k_2^2} \right|$$

The above equations can be derived from the concepts/techniques in Chapter 6. The undamped natural frequencies of this system can be shown to be about 0.78 radians/second and 1.28 radians/second. Turn in your programs and the plots for all three cases.