*Int. J. Engng Ed.* Vol. 21, No. 5, pp. 885–895, 2005
Printed in Great Britain.

0949-149X/91 $3.00+0.00


# Enhancing an Advanced Engineering Mechanics Course Using MATLAB and Simulink*

JAMES B. DABNEY
*Systems Engineering Program, University of Houston—Clear Lake, Houston, Texas.*
*E-mail: dabney@cl.uh.edu*

FATHI H. GHORBEL
*Department of Mechanical Engineering, Rice University, Houston, Texas*

*An advanced engineering mechanics course teaches students to analyze and model a variety of dynamical systems using Newtonian and Lagrangian mechanics approaches. The modeling task typically produces nonlinear differential equations that are best solved numerically. In order to prepare students to competently solve these systems numerically, they must master a suitable programming environment. This mastery is achieved incrementally throughout the semester. This paper describes a successful approach to developing the necessary programming skills, culminating in a course project in which the students model a complex dynamical system and produce a graphical animation allowing visualization of dynamical behavior. The paper also describes typical course projects that have been successfully completed by advanced undergraduate and beginning graduate students.*

## INTRODUCTION

AN IMPORTANT SKILL in advanced engineering mechanics is the ability to simulate and visualize system dynamical behavior. Traditionally, advanced engineering mechanics courses emphasize analysis of kinematics and dynamics, culminating in formulation of equations of motion in the form of ordinary differential equations (ODE). The advanced engineering mechanics course at Rice University was enhanced via the addition of numerical simulation using MATLAB and Simulink so that the students develop the ability to correlate system behavior with the equations of motion. In order to prepare students for their semester project, which includes a numerical simulation complete with a graphical animation, MATLAB and Simulink were integrated into the course throughout the semester.

The use of MATLAB and Simulink in the engineering curriculum is widespread today. Many engineering textbooks offer MATLAB and Simulink exercises. The MathWorks' website (www.mathworks.com) lists hundreds of engineering texts that feature MATLAB or Simulink examples and exercises. There is also an extensive literature on the use of these tools to augment engineering courses [1–8]. These references provide a representative sampling of the various approaches to exploiting these tools in the curriculum. Two main approaches to the use of MATLAB and Simulink are evident in the literature.

The first approach is to use MATLAB and Simulink to develop custom software to allow the students to experiment with important concepts. For example, Kezunovic et al. [5] developed a set of MATLAB tools to demonstrate issues in power engineering. These tools consist of prebuilt simulations with graphical user interfaces that allow the students to vary system parameters and observe behavior. Gomez et al. [3] employ a similar approach to augmenting an acoustical engineering course. This software allows the students to easily synthesize acoustic arrays and plot their behavior. Johansson and M. Gafvert [7] describe a set of MATLAB programs to assist students in an introductory controls class to experiment with linear systems analysis and design techniques. The success reported for these tools suggests that they are significant enhancements to the curriculum.

A second, more advanced, approach to engineering course enhancement is to require the students to do the programming themselves. Asemi and Yaz [1] describe the use of MATLAB and Simulink in a nonlinear systems analysis course. They observed improved student understanding of the course material and increased student interest in the subject. Asemi and Yaz also note that there are significant challenges to integrating the software tools. Among the challenges are ensuring that students have sufficient access to the software, teaching the students to use the software without interfering with the course

* Accepted 3 July 2005.

material, and making the use of MATLAB and Simulink integral to the course. Similarly, Adawi [6] notes that students will lose motivation to use the software tools unless the MATLAB programming is integrated into the curriculum.

The advanced engineering mechanics course at Rice University is taught to engineering seniors and beginning graduate students. The prerequisites include the sophomore engineering mechanics course, an introductory programming course, and the standard engineering mathematics preparation. The course covers the following topics:

1. Introduction to systems
2. Review of relevant mathematics
3. Three-dimensional kinematics
4. Dynamics of a particle
5. Dynamics of systems of particles
6. Dynamics of rigid bodies (Newtonian mechanics)
7. Elements of analytical dynamics (Lagrangian mechanics)
8. Concepts from systems theory

As can be seen from this list of topics, the skills the students acquire allow them to develop equations of motion for three-dimensional systems of rigid bodies. The use of MATLAB and Simulink is extremely valuable in advanced engineering mechanics in that it allows students to study the system behavior corresponding to the equations of motion. Using graphical animations, the students gain a qualitative understanding of the behavior and can correlate that with the numerical results.

This paper illustrates successful approaches to the two primary challenges to integrating MATLAB and Simulink into an advanced engineering mechanics course: teaching the students to use the software and augmenting the course such that the students build skills and confidence gradually throughout the semester. The students are introduced to MATLAB and Simulink early in the semester via tutorial sessions [9, 10] which include basic MATLAB programming, solving ODEs in MATLAB, MATLAB graphics, Simulink modeling, and graphical animations in MATLAB. The next section describes the MATLAB, Simulink, and graphical animation tutorials that address the first challenge. Throughout the semester, homework sets include MATLAB and Simulink problems of increasing difficulty, allowing the students to gain proficiency and confidence. Example assignment problems for each phase of the course are provided in the following section. Each student in the course is required to complete a course project. The projects are chosen from a set selected to provide a reasonable level of challenge for advanced undergraduates or beginning graduate students. The projects involve modeling a dynamical system present in the dynamic systems laboratory or for which measured data is available for comparison. Three example projects, the Rice SPENDULAP, a human respiration model, and a magnetic levitation experiment, are then described, followed by the conclusions.

## MATLAB/SIMULINK TUTORIAL

Many of the students in this course have some experience with MATLAB. Rice mechanical engineering undergraduates are introduced to MATLAB in an introductory engineering programming course and again in the differential equations course. MATLAB and Simulink are available in the computer labs throughout the campus, providing easy access to the software. However, few of the students are sufficiently proficient to succeed in modeling complex dynamical systems and developing graphical animations. Furthermore, most undergraduates receive their first exposure to block diagram notation and Simulink in this course. Therefore, early in the semester, tutorials are presented to refresh students' basic MATLAB skills, introduce Simulink, and to introduce graphical animations. Both the MATLAB and Simulink tutorials are offered as supplemental evening class sessions to accommodate availability of instructional laboratories and avoid reduction of course content.

### *MATLAB tutorial*

The MATLAB tutorial [9] consists of two main parts. First, the basics of MATLAB programming are reviewed, including syntax, M-file structure, and MATLAB data structures. This lecture also covers basic MATLAB graphics. Next, the MATLAB ordinary differential equation (ODE) solvers are introduced. The MATLAB tutorial is scheduled to coincide with the classroom introduction of state–space methods. Thus, the ODE tutorial reinforces the classroom lecture and provides some practical experience converting scalar higher-order and coupled systems of ODEs to state–space form.

MATLAB has grown into a comprehensive programming environment suitable for a wide range of engineering problems. Therefore, it is important when developing a MATLAB tutorial to focus on a subset of MATLAB features and capabilities applicable to the problem class at hand.

The tutorial begins with an introduction to relevant MATLAB data constructs: vectors, matrices, structures. The MATLAB language is rich with additional data constructs, including a variety of data types (fixed point, single precision, etc.) and constructs (cell arrays, multidimensional arrays). We restrict coverage to two-dimensional matrices and structures using the default double-precision floating point data type, because we have found them to be sufficient for the modeling and simulation task.

The tutorial next addresses basic MATLAB plotting capabilities (plot command), including axis labels and titles, but excluding advanced

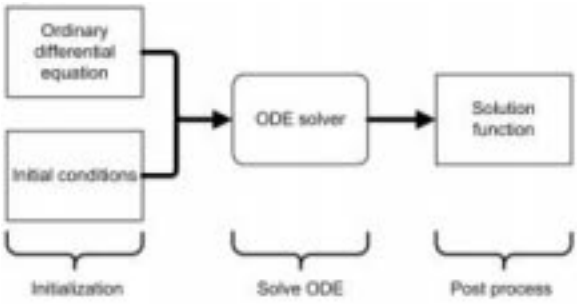


Fig. 1. Solving an ODE.

capabilities such as LaTeX characters, MATLAB Handle Graphics (which is covered in the graphical animation tutorial), and multidimensional plots.

There are many approaches for implementing ODE solutions in MATLAB, and, no doubt, many approaches to teaching the process. We have found that, at this academic level, it is adequate to portray ODE solvers as "black boxes" and defer discussion of the details of numerical ODE solution to other courses. The ODE solvers are presented as black boxes that take as input ODE and initial conditions and produce as output functions that satisfy the ODE and initial conditions. The schematic of solving an ODE is shown in Fig. 1.

The MATLAB ODE solvers can be thought of as functions that replace the exact solution (a function that satisfies the ODE) with a table. The solution structure of Fig. 1 is developed in MATLAB as a pair of M-files. A function M-file with a prescribed set of calling arguments (Fig. 2) comprises the ODE. The remainder of the system is a script M-file with three parts, as illustrated in Fig. 3, corresponding to the three sections shown in Fig. 1. The first part of the script M-file names the ODE function and specifies initial conditions. The second part of the script M-file is the invocation of the solver and consists of a single line of code which returns the solution function as a table. The third part of the script M-file does post-processing, such as plotting the solution table.

```
function x_dot = difeq(t, x, m, c, k)
x_dot = [x(2) ; -(c/m)*x(2) - (k/m)*x(1)] ;
```

Fig. 2. ODE function M-file.

```
dq_fun = @difeq ;
tspan = [0:0.1:50] ;
x0 = [1,0] ;
c = 0.1 ;
k = 2 ;
m = 4 ;

[t,x] = ode45(dq_fun, tspan, x0, [],m, c, k) ;

plot(t,x(:,1)) ;
grid ;
xlabel('Time') ;
ylabel('Displacement') ;
```

Initialization

Solve ODE

Post process

Fig. 3. ODE script M-file structure.

The MATLAB ODE solvers all solve vector first-order ODEs, so it is necessary at this point to introduce the concept of converting higher-order scalar and coupled scalar ODEs into first-order vector ODEs. This concept is presented in two stages. A second-order ODE representing a damped spring mass system (Fig. 4) is presented and used to illustrate the concept. First, the scalar second-order ODE,

$$\ddot{x} = -\frac{k}{m}x - \frac{c}{m}\dot{x}$$

is presented. Next, the concept of state variables is briefly introduced, and the equation of motion is rewritten as a vector first-order ODE after defining $z_1 = x$, $z_2 = \dot{x}$

$$\dot{z} = z_2$$

$$\dot{z}_2 = -\frac{k}{m}z_1 - \frac{c}{m}z_2$$

A pair of M-files implementing the vector ODE (Figs. 2 and 3) is displayed and discussed in some detail.

Finally, a two-mass system as shown in Fig. 5 is used to extend the concept to coupled scalar ODE. First, the two coupled second-order ODE are written,

$$\ddot{z} = -\frac{c_1}{m}(\dot{z} - \dot{y}) - \frac{k_1}{m_1}(z - y)$$

$$\ddot{y} = -\frac{c_1}{m_2}(\dot{y} - \dot{z}) - \frac{c_2}{m_2}\dot{y} - \frac{k_1}{m_2}(y - z) - \frac{k_2}{m_2}y$$

Then, after defining state variables $x_1 = z$, $x_1 = \dot{z}$, $x_3 = y$, $x_4 = \dot{y}$, the equations of motion are written as the vector first-order ODE

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = -\frac{c_1}{m_1}(x_2 - x_4) - \frac{k_1}{m_1}(x_1 - x_3)$$

$$\dot{x}_3 = x_4$$

$$\dot{x}_4 = -\frac{c_1}{m_2}(x_4 - x_2) - \frac{c_2}{m_2}x_4 - \frac{k_1}{m_2}(x_3 - x_1) - \frac{k_2}{m_2}x_3$$

Fig. 6 displays a function M-file that implements the set of first-order ODEs and Fig. 7 the corresponding script M-file that solves the system of ODEs.

*Simulink tutorial*

The students are introduced early in the semester to block diagram notation. The Simulink tutorial [10] begins with a review of block diagram notation. The same example systems used in the MATLAB tutorial are used as the basis for the block diagram discussion. We have found that the students master Simulink fundamentals rapidly via hands-on experience. Therefore, the Simulink tutorial is presented in a computer lab, where each student can implement the examples during the class. Thus, at the completion of the tutorial, the students have already succeeded in building and running models.

The main steps in the Simulink tutorial are

1. Introduce block diagram notation starting with primitive linear blocks: gain, sum, derivative, and integrator. Introduce a first-order scalar ODE and draw a corresponding block diagram on the board.
2. Demonstrate starting Simulink and discuss briefly the structure of the menus and block libraries.
3. Implement the first-order scalar system using the basic blocks. At this stage, it is emphasized that the essence of Simulink modeling is the

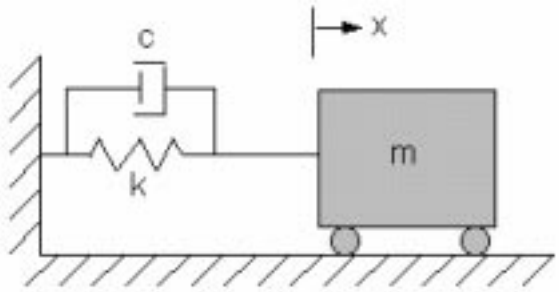


Fig. 4. Spring-mass system.

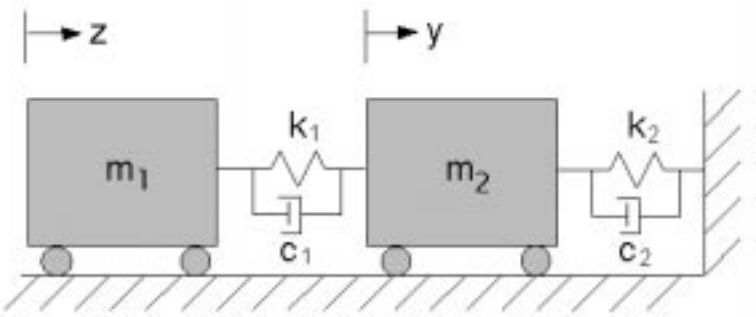


Fig. 5. Two-cart system.

```
function x_dot = difeq_4(t, x, m1, m2, c1, c2, k1, k2)
x_dot = zeros(4,1) ; % Return a column vector
x_dot(1) = x(2) ;
x_dot(2) = -(c1/m1)*(x(2)-x(4)) - (k1/m1)*(x(1)-x(3)) ;
x_dot(3) = x(4) ;
x_dot(4) = -(c1/m2)*(x(4)-x(2)) - (c2/m2)*x(4) - (k1/m2)*(x(3)-x(1)) - (k2/m2)*x(3) ;
```

Fig. 6. ODE function M-file for coupled system.

```
% Solve the differential equation in difeq_4.m
dq_fun = @difeq_4 ;
tspan = [0:0.1:50] ;
x0 = [1,0,0,0] ; % Start with cart 1 deflected 1 ft from equilibrium
c1 = 0.1 ;
c2 = 0.4 ;
k1 = 2 ;
k2 = 3 ;
m1 = 4 ;
m2 = 6 ;
[t,x] = ode45(dq_fun, tspan, x0, [],m1, m2, c1, c2, k1, k2) ;
plot(t,x(:,1)) ;
xlabel('Time') ;
ylabel('Cart 1') ;
```

Fig. 7. Script M-file for coupled system.

same as the MATLAB ODE approach with which the students are already familiar. That is, the block diagram is a graphical representation of the differential system similar to the function ODE M-file used in the MATLAB tutorial, the configuration of the integrator blocks sets initial conditions, and the Simulation menu on the Simulink toolbar implements the solver. Finally, the sinks blocks such as Scope and To Workspace implement the post-processing part of Fig. 1.

4. Once the class is comfortable with the Simulink interface, the tutorial continues with the second-order system, shown in Fig. 4 and used in the MATLAB ODE tutorial. We proceed at this point with a recipe for drawing block diagrams for continuous systems. In the recipe, we start with an integrator for each order of the system. So, for the second-order system, draw two integrators, label, and connect them as shown in Fig. 8. Next, add gain blocks to compute $\ddot{x}$ from $\dot{x}$, $x$, and input (Fig. 9). Finally add the source (input) blocks and output blocks (Fig. 10). The construction of this model is a good place to illustrate additional model-building concepts such as branching from signal lines and flipping and rotating blocks.
5. Next, we discuss development of more complex block diagrams using the two-cart example of Fig. 5. For this example, we start with the two second-order ODE. We construct the block diagram on the board, but do not devote class time to building the Simulink model. Instead, we briefly show the model, which is also included in the class handout [11].

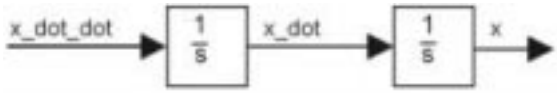


Fig. 8. Draw integrator blocks first.

For this class, we have not found it necessary to cover advanced Simulink concepts such as subsystems and masking, analysis tools (linearization and trim), callbacks, or S-functions.

*Graphical animations*

MATLAB and Simulink provide a rich set of capabilities for producing graphical animations. Among these are primitive graphics using the MATLAB plotting commands, the Animation Toolbox (available for free download from MathWorks' website), Virtual Reality Toolbox, and Dials and Gauges Toolbox. The basic MATLAB plotting commands provide sufficient animation capability for our course, but some students prefer the Animation Toolbox.

MATLAB Graphics Animations are also taught in the computer lab, to allow the students to implement the example programs during the

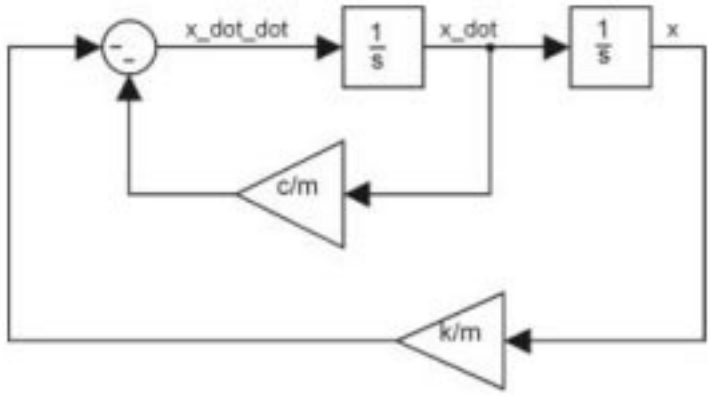


Fig. 9. Add gain and sum blocks.

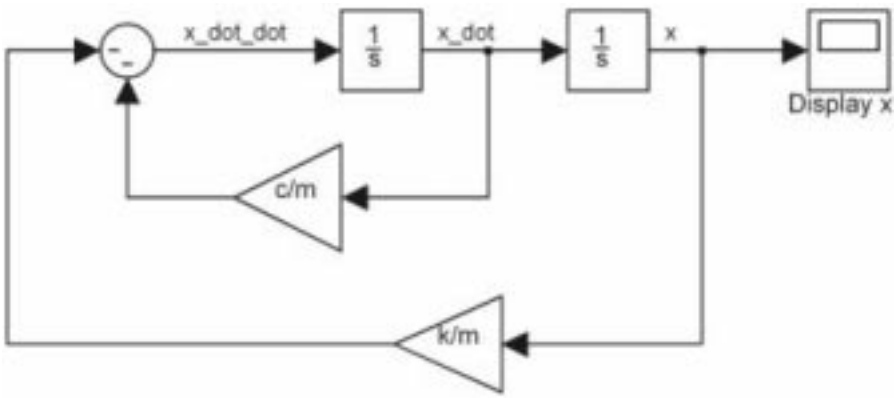


Fig. 10. Complete Simulink model.

```
figure                  % Draw the figure to be animated
axis([0,1,0,1])
hold on ;
xd = [0.3,0.4,0.4,0.3,0.3] ;   % Corners of a square
yd = [0.3,0.3,0.4,0.4,0.3] ;
hd = fill(xd, yd, 'r') ;       % Draw the square and save handle
for t = 0:0.1:10               % Animation loop
   set(hd,'Xdata',xd+0.3*sin(t),'Ydata',yd+0.3*cos(t)) ;
   pause(0.1) ;
end
```

Fig. 11. Simple MATLAB animation.

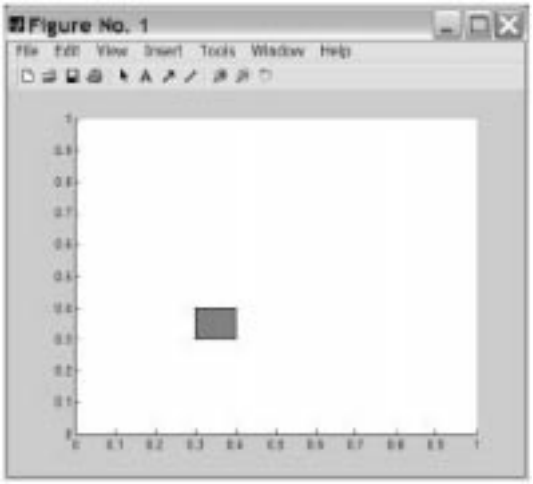


Fig. 12. Simple animation window.

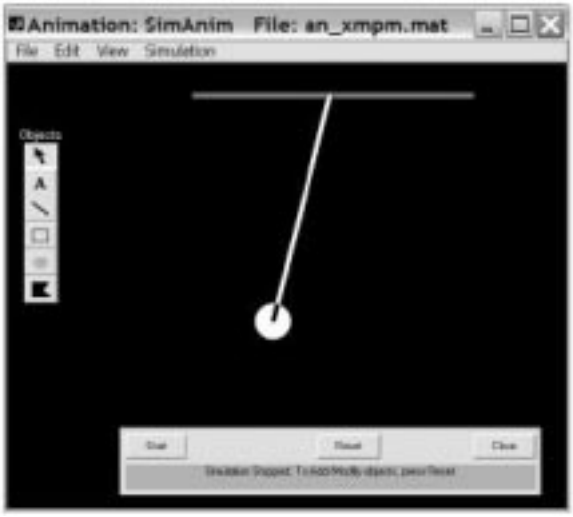


Fig. 13. Animation toolbox pendulum figure.

lecture. Although there are many alternatives for developing graphical animations, we present a method that is simple to implement and consistent with the ODE simulation structure of Fig. 1 in which the animation is a logical part of the post-processing segment. The steps are as follows, starting with the $[t, x]$ output of the solver where $t$ is the time vector and $x$ the computed state vector history corresponding to time points $t$:

1. Initialize the graphical animation, saving handles to the graphical objects
2. Loop through the time points:
   a. Compute new object parameters based on state
   b. Set object characteristics
   c. Pause briefly ($\sim 0.1$ sec) for the image to update to the new configuration

The animation process is illustrated with a simple example in which a small square moves in a circle using the MATLAB code in Fig. 11.

The animation window is shown in Fig. 12. As the animation loop executes, the small square follows a circular path. The students enter the example code during the tutorial and experiment with changing parameters. Although extremely simple, this example contains all the elements of a graphical animation and students typically find it straightforward to extend the example to the more complex animations of their course projects.

The Simulink Animation Toolbox contains a single animation block and supporting MATLAB functions. The animation toolbox allows students to develop relatively sophisticated animations using a simple drawing scheme. The animation block receives as input a single vector signal. Using a drawing palette, the figure to be animated is drawn in the animation workspace. Next, the properties of the drawing objects are associated with elements of the block input vector. A simple example of a pendulum simulation is illustrated in class (Fig. 13).

## BUILDING MATLAB/SIMULINK SKILLS

Many of the homework assignments throughout the semester include MATLAB or Simulink problems of increasing difficulty. This section provides examples of problems used throughout the course.

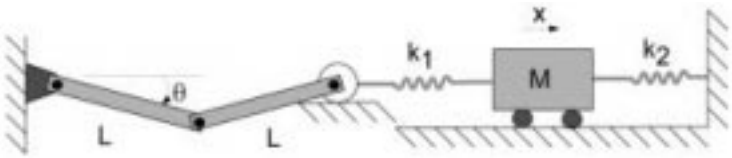


Fig. 14. Two-degree-of-freedom system.

*Basic MATLAB skills*

The first few lectures provide a review of the mathematics concepts that are important in advanced dynamics, including a review of linear algebra. The lectures are supported by a series of homework assignments that reinforce analytical solutions with basic MATLAB exercises.

*Example 1: Dot Products*

Given vectors $\overset{1}{a}$, $\overset{1}{b}$, $\overset{1}{c}$, show that $\overset{r}{a} \times \overset{1}{b} = \overset{1}{b} \times \overset{r}{a}$. Verify the result for $\overset{1}{a} = [1, 3, 5]^T$, $\overset{1}{b} = [2, 1, 3]^T$ using MATLAB.

*Example 2: Matrix Transpose*

Given matrices **A**, **B**, **C**, show that $(\mathbf{ABC})^T = \mathbf{C}^T\mathbf{B}^T\mathbf{A}^T$. Verify the results for

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 1 & 3 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1.5 & 2 \\ 2.5 & 1 \\ 3 & 2 \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} 1.2 \\ 3.1 \end{bmatrix}$$

*Example 3: Positive Definiteness*

A square matrix **A** is said to be positive definite if the scalar quantity $\overrightarrow{x}^T\mathbf{A}\overrightarrow{x} \geq 0$ for all vectors $\overset{1}{x}$ of the same order as **A**, and $\overrightarrow{x}^T\mathbf{A}\overrightarrow{x} = 0$ only if $\overset{1}{x}$ is a zero vector. Determine by direct computation whether the matrix

$$\mathbf{A} = \begin{bmatrix} 1.5 & 3 \\ 1 & 4.5 \end{bmatrix}$$

is positive definite. It can be proven that all eigenvalues of a square matrix have positive real parts if and only if the matrix is positive definite. Use MATLAB function `eig` to verify your analytical result.

*Example 4: Matrix Rank*

The rank of a matrix is the largest non-singular submatrix, where we say a matrix is singular if its determinant is zero. Determine by hand calculation the rank of the matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 2.5 & 2.5 & 4 \\ 2 & 4 & 6 \end{bmatrix}$$

and verify your result using MATLAB command `rank`.

*Solving ODEs with MATLAB*

An important component of an advanced dynamics course is solving the derived equations of motion analytically and numerically. This skill is the primary goal of the MATLAB component of the course. Homework problems proceed from simple second-order systems to more complex higher-order and coupled systems. Examples of introductory problems and more advanced homework problems are presented below.

*Example 5*

Given the linear second-order time-invariant system $x + x + x = e^{-1}$, $x(0) = 5$, $x(0) = 0$, solve for $x(t)$ analytically and numerically. Plot the results of both solutions using MATLAB. Note that this is a minor extension of the example used in the MATLAB and Simulink tutorials.

*Example 6*

The system shown in Fig. 14 (based on problem 6.16 from [12]) consists of two homogeneous links of mass $m$, two springs, and a cart of mass $M$. Assuming all components other than the cart and links are massless, derive the equations of motion using both Newtonian and Lagrangian approaches. Model the system dynamics using MATLAB and plot the block location. Dimensions, mass properties, spring constants, and initial conditions are supplied.

*Simulink and animation skills*

Simulink exercises are introduced from the beginning of the course. The first exercises are simple block diagrams of systems similar to those described above for MATLAB. As the semester proceeds, Simulink exercises are included in the homework assignments and require graphical animations as well.

*Example 7*

Consider a disk of radius $r$ rolling in a circular trough of radius $R$ as shown in Fig. 15 (based on problem 6.10 from [12]). Derive the equation of motion of the disk, model the dynamics in Simulink, and produce a graphical animation that resembles Fig. 15. Dimensions, mass properties, and initial conditions are specified.

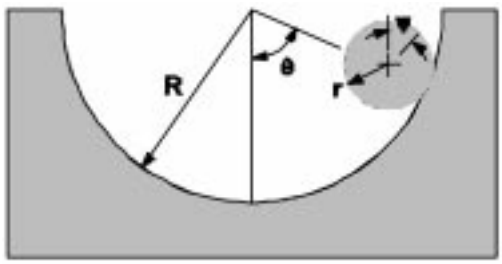


Fig. 15. Disk rolling in a trough.

## EXAMPLE PROJECTS

Each student is required to complete a course project that entails modeling a moderately complex system, deriving equations of motion, building a simulation in MATLAB or Simulink, and developing a graphical animation driven by the simulation. In previous semesters, we experimented with providing broad guidelines for projects but allowing the students to conceive the projects on their own. Frequently, the students selected projects that were either too simple or too difficult. We have observed consistently better results and student feedback if they are provided with a set of candidate systems to model with clearly specified tasks. The systems from which the students may choose are either laboratory apparatuses in the robotics lab or biological systems for which experimental data is available. Among the systems offered are the Rice SPENDULAP spherical pendulum system, a human respiration model, industrial emulator [13], a control moment gyroscope apparatus [14], and a magnetic levitation apparatus [15]. To illustrate the projects, the Rice SPENDULAP, human respiration model, and magnetic levitation system are described next.

*Rice SPENDULAP*

The Rice SPENDULAP [11, 16] (Fig. 16) is a spherical pendulum apparatus that is useful for illustrating a variety of important topics in kinematics, dynamics, simulation, and control. The pendulum is mounted in a rotating frame that is supported at the top and bottom by bearings mounted in a cylindrical housing. The pendulum consists of a steel tube that is threaded on both ends and attached to a cylindrical aluminum bob at the bottom. At the top, the pendulum is attached to a tee that swings on a stainless steel pivot pin. The pivot pin is mounted in bearings to the rotating frame. The rotating frame is driven by a direct-drive direct current motor (in the $\phi$ coordinate direction) and the pendulum swings freely on the pivot pin (in the $\theta$ coordinate direction) when the clutch and lift brake are released.

The SPENDULAP is a two-degree-of-freedom under-actuated system. Furthermore, the kinematics are three-dimensional. The highly nonlinear dynamics make it a rich source of examples for topics such as ignorable coordinates, bifurcation, and limit cycles.

The student tasks are:

1. Model the system dynamics using Newtonian and Lagrangian approaches.
2. Simulate the system dynamic response using MATLAB or Simulink for the following cases:
   a. For a constant frame angular velocity consistent with an equilibrium pendulum deflection of 30 degrees, plot pendulum deflection as a function of time using an initial pendulum deflection of 30 degrees.
   b. Repeat using an initial deflection of 45 degrees and explain the differences.
   c. Analytically and experimentally, study the equilibrium at zero pendulum deflection.
3. Design an input torque rule to cause the pendulum to stabilize at a deflection of 30 degrees and verify the controller using the simulation and initial deflection of 45 degrees.
4. Develop a graphical animation of the pendulum using the simulated trajectory. (A typical animation is shown in Fig. 17.)
5. Compare the numerical results with trajectory data from the lab apparatus and explain the differences.

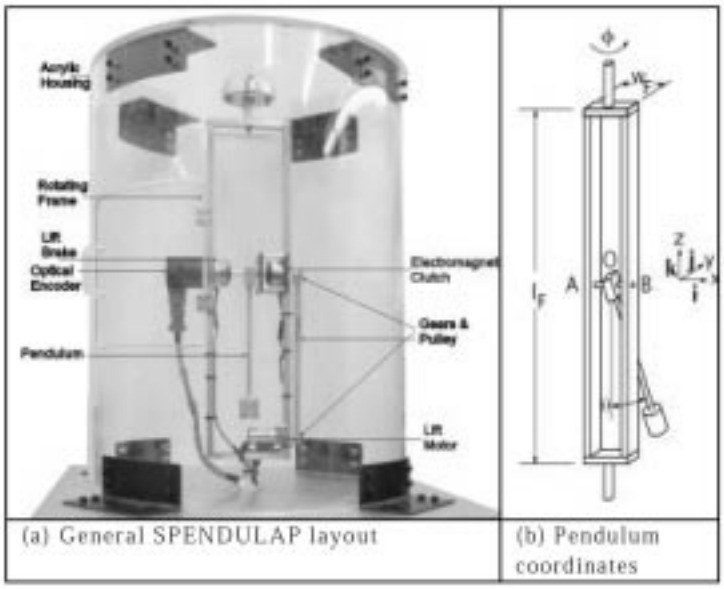


Fig. 16. Rice SPENDLAP.

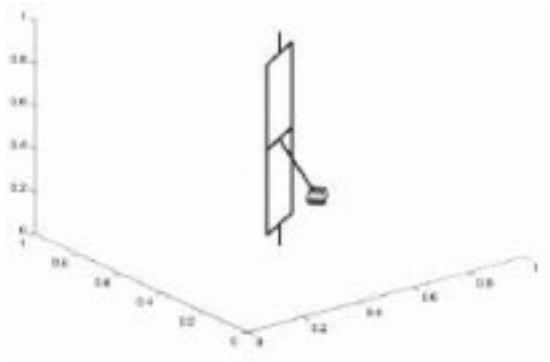

Fig. 17. SPENDULAP animation example.

Table 1. Respiratory model characteristics and state variables

| Region | Characteristics | State variable |
|---|---|---|
| Upper supported airway | Resistance $R_{uaw}$ | $Q$ (mass flow rate) |
| Collapsible airway | Resistance $R_C$ and recoil pressure $P_{TM}$ | $V_C$ (local volume) |
| Small airways | Resistance $R_s$ | $V_L$ (lung volume) |
| Lung (alveolar) | Recoil pressure $P_{EL}$ and constant resistance $R_{LT}$ | $V_L$ |

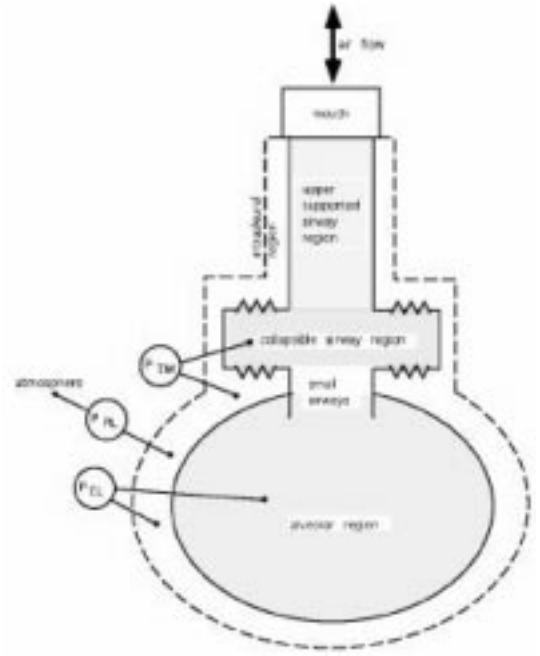

Fig. 18. Model representation of airway mechanics.

*Human respiration model*

The human respiration model is based on recent research results [17] that account for mechanical and chemical dynamics involved in the exchange of carbon dioxide and oxygen during breathing. The students are presented with a simplified candidate model structure [18] which represents only the mechanics of air transport into the lungs during inspiration and from the lungs during expiration. The physical model is shown in Fig. 18. In this

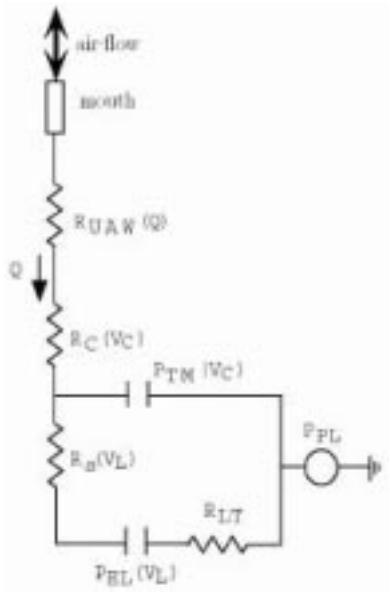


Fig. 19. Pneumatic-electrical model equivalent.

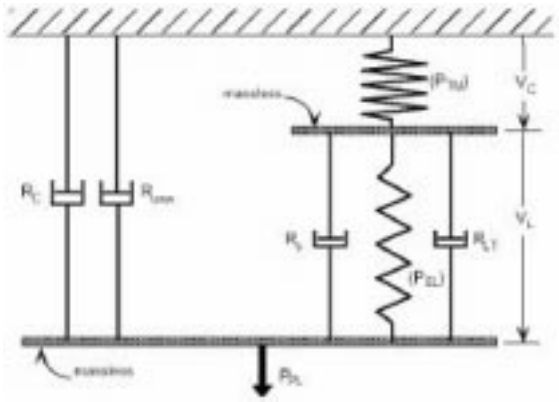

Fig. 20. Mechanical system analogous to respiration dynamics model.

model, the lungs are characterized by four regions with characteristics which are dependent on three state variables, as defined in Table 1.

The four regions are connected, as shown in Fig. 18. They all lie within a single enclosure representing the thoracic cage. Air transport is driven by the pressure difference between the intrapleural region and the atmosphere.

The students are also provided with schematic diagrams of an electrical equivalent circuit (Fig. 19) and a mechanical equivalent system (Fig. 20). All parameters of the equivalent systems are provided as well.

The student tasks are:

1. Using the mechanical equivalent system, develop equations of motion using the Newtonian and Lagrangian approaches.
2. Draw a block diagram of the system.
3. Given a set of MATLAB functions for parameters $R_{uaw}$, $R_C$, $P_{TM}$, $R_s$, and $P_{EL}$ and a MATLAB file containing measured values of $P_{PL}$, compute the trajectories of $Q$, $V_L$, and $V_C$.
4. Develop a graphical animation of Fig. 18.

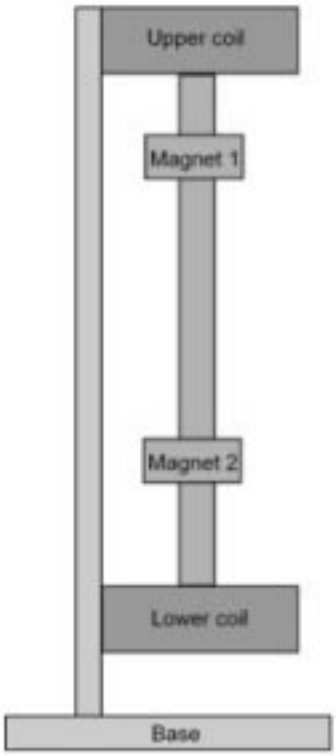


Fig. 21. Maglev apparatus layout.

*Magnetic levitation model*

A third course project is based on a commercial laboratory apparatus produced by Educational Control Products, Inc. (ECP) employing the principle of magnetic levitation. This project is somewhat more challenging, although the graphical animation task is less difficult. The apparatus used is the ECP Model 730 MagLev apparatus [15]. The apparatus can be configured in a variety of ways. The most general configuration is a multiple input—multiple output (MIMO) system in which two electromagnets (upper and lower) control the position of a pair of rare earth permanent magnets, as illustrated in Fig. 21.

In the configuration shown in Fig. 21, the system has two degrees of freedom, which can be represented as the elevations of the two magnets, $y_1$ and $y_2$, relative to some datum. There are two inputs, the control efforts for the lower coil ($u_1$) and upper coil ($u_2$). The equations of motion are linear with respect to the control effort terms and nonlinear with respect to the state variables. The students are provided with a set of candidate functions relating the control effort inputs and state variables to corresponding forces on each magnet. For example, the force exerted on the lower magnet by the lower coil is approximately

$$F_{u_{11}} = \frac{u_1}{a(y_1 + b)^4}$$

where $a$ and $b$ are coefficients to be determined experimentally.

1. Develop equations of motion of the full MIMO system using both the Newtonian and Lagrangian approaches.
2. Given a set of measurements of steady-state values of control effort and magnet positions, compute the force coefficients ($a$, $b$, etc.).
3. Linearize the equations of motion about an equilibrium.
4. Design a feedback control to stabilize the linear system and demonstrate it using a simulation of the nonlinear system.
5. Implement the feedback control on the laboratory apparatus controller, test it, and explain the differences between simulation and experimental results.
6. Develop a graphical animation of the magnetic levitation system resembling Fig. 21.

## CONCLUSIONS

In this paper, we have described a successful approach to enhancing an advanced engineering mechanics course via the addition of numerical simulation using MATLAB and Simulink. First, we described tutorial sessions that introduce the students to systematic approaches for solving ODEs using MATLAB and Simulink and techniques for developing graphical animations. Next, we discussed augmenting homework assignments with MATLAB and Simulink exercises to gradually improve student proficiency with the software. Finally, we described three typical course projects that involve a laboratory apparatus or experimental data, increasing the students' confidence in the techniques they have learned and exposing them to the reality of modeling uncertainty. The approach we have described in this paper can be tailored to a variety of advanced engineering courses involving dynamical systems.

## REFERENCES

1. A Azemi and E. Yaz, Utilizing Simulink and MATLAB in a graduate nonlinear systems analysis course, 26th Frontier in Education Conference, Salt Lake, Utah (1996), pp. 595–599.
2. A. R. Miller, Applying advanced computer tools to engineering education, International Conference on Engineering Education 1999, Progue, Czech Republic (1999).
3. J. J. Gomez-Alfageme, M. Recuero-Lopez and J. L. Sanchez-Bote, The computer in acoustical engineering education: An experience, International Conference on Engineering Education 1999, Progue, Czech Republic (1999).

4. P. Masson and P. Micheau, Teaching mechatronics to mechanical engineers: Five years' experience at the Université de Sherbrooke, Forum on Mechatronics Education in Canada: Past Experiences and Future Directions, Waterloo, Canada (2001), pp. 10–13.
5. M. Kezunovic, A. Abur, G. Huang, A. Bose and K. Tomsovic, The role of digital modeling and simulation in power engineering education, *IEEE Transactions on Power Systems*, **19**(1) (2004), pp. 64–72.
6. T. W. Adawi, *Innovation and Integration in Curriculum and Learning: Results and Experiences from C-SELT Projects*, Center for Digital Media and Higher Education, Chalmers University of Technology (2003).
7. M. Johansson and M. Gafvert, ICTools: Interactive learning tools for control, Lund Institute of Technology, Lund, Sweden (1997) (http://www.control.lth.se/~ictools).
8. E. Yaz and Azemi, *Utilizing MATLAB in Two Graduate Electrical Engineering Courses*, Proceedings of the 25th Frontiers in Education Conference (1995).
9. J. B. Dabney, Course notes for introduction to MATLAB, Rice University, Houston, Texas (http://nas.cl.uh.edu/dabney/Mech501RefsPage/MatlabTutorial.pdf).
10. J. B. Dabney, Course notes for Simulink tutorial, Rice University, Houston, Texas (http://nas.cl.uh.edu/dabney/Mech501RefsPage/smlkntro.pdf).
11. F. H. Ghorbel and J. B. Dabney, A spherical pendulum system to teach key concepts in kinematics, dynamics, control, and simulation, *IEEE Transactions on Education* (HTML format-special CD-ROM issue), **42**(4) (1999).
12. L. Meirovitch, *Introduction to Dynamics and Control*, John Wiley and Sons (1985).
13. *Industrial Emulator/Servo Trainer Instructor's Manual*, Educational Control Products Inc., Woodland Hills, California (1995).
14. *Control Moment Gyroscope Instructor's Manual*, Educational Control Products Inc., Woodland Hills, California (1999).
15. *Magnetic Levitation System Instructor's Manual*, Educational Control Products Inc., Woodland Hills, California (1999).
16. J. B. Dabney, F. H. Ghorbel and J. McCune, Web-based control of the Rice SPENDULAP, *International Journal of Engineering Education*, **19**(3) (2003), pp. 478–486.
17. A. Athanasiades, F. H. Ghorbel, J. W. Clark Jr., S. C. Niranjan, J. Olansen, J. B. Zwischenberger and A. Bidani, Energy analysis of a nonlinear human lung model, *Journal of Biological Systems*, **8**(2) (2000), pp. 115–139.
18. F. H. Ghorbel, Respiratory dynamics project handbook, Rice University, Houston, Texas (http://nas.cl.uh.edu/dabney/Mech501RefsPage/Respiration.pdf).

**James B. Dabney** is an Assistant Professor of Systems Engineering at University of Houston—Clear Lake, Houston, Texas, where he directs the Systems Engineering Laboratory. His research interests include mechatronics, dynamics, and control, robotics, and aircraft trajectory optimization. Dr. Dabney received the B.Sc. degree in Mechanical Engineering from Virginia Tech in 1974, the M.Sc. degree in Process Monitoring and Control from University of Houston—Clear Lake, and the Ph.D. degree in Mechanical Engineering from Rice University in 1998. He is an Associate Member of IEEE, ASME, and AIAA and a Member of INCOSE. He is past editor of the ASME Dynamic Systems and Control Newsletter and Associate Editor for the IEEE Control Systems Society Conference Editorial Board.

**Fathi H. Ghorbel** is an Associate Professor at the Department of Mechanical Engineering and the Department of Bioengineering at Rice University, Houston, Texas, where he is also the Director of the Dynamic Systems & Control Laboratory, and the Robotics Laboratory, and Co-Director of the Biomedical Systems and Instrumentation Lab. His research is in the areas of systems and control theory, robotics, and biomedical engineering systems. Dr. Ghorbel received a B.Sc. degree with honors from the Pennsylvania State University in 1985, an M.Sc. from Carnegie-Mellon University in 1987, and a Ph.D. from the University of Illinois at Urbana-Champaign in 1991, all in Mechanical Engineering. Dr. Ghorbel is a Member of the ASME, a Senior Member of the IEEE, a member of Sigma Xi, IFAC, SIAM, IASTED, ASEE, and past President of the Tunisian Scientific Society (TSS). He is a Founding Member and Member of the Board of Directors of the Arab Science and Technology Foundation (ASTF). He is a recipient of the medal of "Order of National Merit in the field of education and sciences" by order of the President of Tunisia. He is an Associate Editor for the IEEE Transactions on Control Systems Technology, an Associate Editor for the ASME Journal of Dynamic Systems, Measurement, and Control, an Associate Editor for the IEEE Control Systems Society Conference Editorial Board, and a past Associate Editor for the International Journal of Robotics and Automation.