Teaching MATLAB in First-year Engineering: A GUI Tool Directed Approach

Craig S. Lent, Jay Brockman, Victoria Goodrich, Kerry Meyers
University of Notre Dame, lent@nd.edu, jbb@nd.edu, vfroude@nd.edu, kmeyers1@nd.edu

***Abstract* - We describe an approach to teaching MATLAB that focuses on student-written computational models with a graphical user interface (GUI). The curriculum teaches the basics of programming but emphasizes getting as soon as possible to GUI tool development. Students learn a straightforward process for constructing a computational model of a physical system, and then attaching it to a GUI.**

Index Terms – Programming, MATLAB, graphical user interface, introductory engineering.

MATLAB GUI TOOLS

The MATLAB programming language has evolved to include many powerful and convenient graphical and analysis tools. It has become an important platform for engineering and science education, as well as research. MATLAB is a very valuable first programming language, for beginning engineering students, and, for many, will be the preferred language for most, if not all, of the computational work they do.

An often-overlooked feature of MATLAB is the GUIDE program (*Graphical User Interface Development Environment)*. Using GUIDE makes construction of GUI-based tools straightforward. (GUI is pronounced "gooey".) Interface elements such as buttons, menus, editable text objects, sliders, and plotting areas, can be arranged on a panel using familiar drag-and-drop operations. GUIDE automatically generates the code for the user interface; the programmer has merely to fill in functionality at key points to connect his or her code to the GUI tool. By templating this process, it's possible to take novice users quickly into the realm of GUI tool development. This makes programming both more appealing to students and more useful in learning engineering. Interactive graphics allow students to explore behavior that results from the underlying physical model.

While computer programs can be used in many ways, the emphasis here is on building computational models, primarily of physical phenomena. [1] A physical system is modeled first conceptually, using ideas such as momentum, force, energy, reactions, fields, etc. These concepts are expressed mathematically and applied to a particular class of problem. Such a class might be, for example, projectile motion, fluid flow in a pipe, tunneling between quantum dots, signals in chained transmission lines, an electrical oscillator circuit, or beams under load. Typically, the model involves a set of parameters which describe the physical system and a set of mathematical relations—systems of equations, integrals, differential equations, eigensystems, etc. The computational solution of the mathematical model for a given set of parameters must be realized through a computational algorithm—a step-by-step procedure for calculating the desired quantities. The behavior of the model is then usually visualized graphically, *e.g.,* one or more plots, bar graphs, or animations.

A GUI-tool consists of a (1) computational model of the system, and (2) a graphical user interface that provides a graphical representation of the model performance and lets the user easily and naturally adjust the parameters of the model and see the resulting change in behavior. Depcik and Assanis reported that "the most important benefit of a GUI is that it can post-process the results of the simulation providing the user with instant feedback. This is especially important when performing parametric studies where variables are changed over a certain range. A visual illustration of how results change as a function of various variables, or parametric sweeps, reinforces the lessons learned in the classroom." [2]

The GUI tool approach was originally developed for high school students at the 11th-grade level. It has been used for more than six years at the three campuses of Trinity School. All students at the school take a one-semester MATLAB programming course and use the GUI tool techniques in their subsequent mathematics and physics courses.

Student learning is enhanced if the students themselves build the GUI tools: construct the computational model, implement the visualization of results, design and program the GUI. Building the GUI tool is valuable for student learning for several reasons.

(a) *Insight into system behavior.* Exploring model behavior, by manipulating sliders, buttons, checkboxes etc., helps to develop an intuitive insight into the model behavior. Insight is the primary goal. By running the computational model many times with different inputs, the user, here the student, can begin to see and understand the characteristic behavior of physical system represented by the model.

(b) *Increased realism of models.* With appropriate algorithms, students can construct computational models which are more realistic than the idealized text-book behavior accessible with closed-form algebraic solutions. For example, models of ballistic motion can include aerodynamic drag,

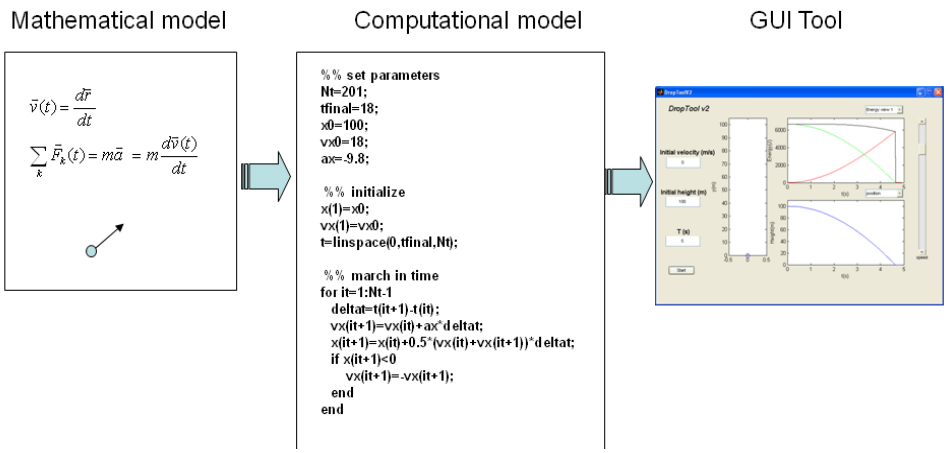

FIGURE 1
STUDENTS EXPRESS A MATHEMATICAL MODEL AS A COMPUTATIONAL MODEL, WHICH IS THEN CONNECTED TO A GRAPHICAL USER INTERFACE (GUI) TO FORM A GUI TOOL.

wind, and other complications. This results in a better match with reality. The gap between text-book problems (baseballs hit into a vacuum) and reality can otherwise leave students with an unwarranted suspicion that physics never really describes the physical world.

(c) *Design experience.* Particularly for engineering students, the discipline of wrapping the model in a form that someone *else* could use encourages a design-oriented focus. For longer programs there is a significant design element in how the computational model behind the GUI is decomposed into functions, and how objects and data interact. Yet, even for short programs, determining the layout and function of the user interface presents an early opportunity for students to engage in user-centered design.

(d) *Familiarity.* Students are accustomed to operating computers through graphically driven tools. A command line, text-based interface seems crude and retrograde.

(e) *Fun.* Building and delivering a sophisticated computational model that is operated through a GUI interface is rewarding and makes the student's first experience with programming more enjoyable.

There is considerable literature that recognizes the benefits of using GUI tools to enhance student learning in engineering educational settings, for example in controls [3] robotics[4], and numerical methods [5] to name a few. However, there seems to be a gap in the literature relating to the educational benefits for students to learn how to design and implement a GUI.

VISUALIZATION AND ANIMATION

Visualization, particularly through animation of motion in space, provides a first check of the computational model against the student's experience of the physical world. If the ball in the program is seen to fall up, the student immediately knows something is wrong (probably the sign of the acceleration due to gravity) because experientially balls don't fall up.

Other visual feedback is more iterative and increasingly sophisticated. For example, by plotting the kinetic and potential energies of a bouncing ball synchronously with an animation of the ball, a student might first notice that in the drag-less model the total energy, kinetic plus potential, is constant in time. The student can then ask why and when do real balls loose energy, which they certainly do. Does more energy loss occur in a bounce or in aerodynamic drag? How could we include these in the computational model? The process of seeing the energy visually as a function of time makes an otherwise unseen quantity an item of inquiry and iterative reflection. The student's mental model, paralleling the computational model, is refined.

Rapid and nearly continuous visual feedback reveals how the computational model responds to the variation of parameters. For example: how does the impact distance change when the spring constant is varied? The GUI interface makes it especially easy and revealing. Parameters can be altered easily by filling in text boxes or moving sliders. Sliders are helpful because they enable the feel of continuous variations. Tuning or designing for particular outcomes (hit the target) is both educational productive and rewarding.

COURSE STRUCTURE

This approach has been implemented for the past three years in a first-year engineering course at the College of Engineering of the University of Notre Dame. Entering students who intend to major in engineering normally take a two-semester sequence called *Introduction to Engineering Systems* I and II. Here we describe only the MATLAB programming component of this course.

In the first semester, students learn to use MATLAB as a calculator and to write simple scripts with no flow control structures. They use some of the statistical analysis

functions of MATLAB to analyze course projects, and learn to make simple plots.

The first ten weeks of the second semester focuses primarily on learning MATLAB with the GUI tools approach, although there is a simultaneous semester-project which is begun in the third week. The basics of programming are covered: variables, vectors, strings, conditionals, loops, plotting, and functions. Animation using loops is introduces as early. A few advanced features, data structures and cell arrays, are covered in order to be able to understand GUIs. In weeks five and six, GUI construction with GUIDE is introduced. Some special topics then follow: working with bitmapped images, the Euler and Verlet methods for solving systems of ordinary differential equations, numerical integration, and differentiation. These are chosen in part to satisfy the requests of the students for capabilities they need for their semester projects.

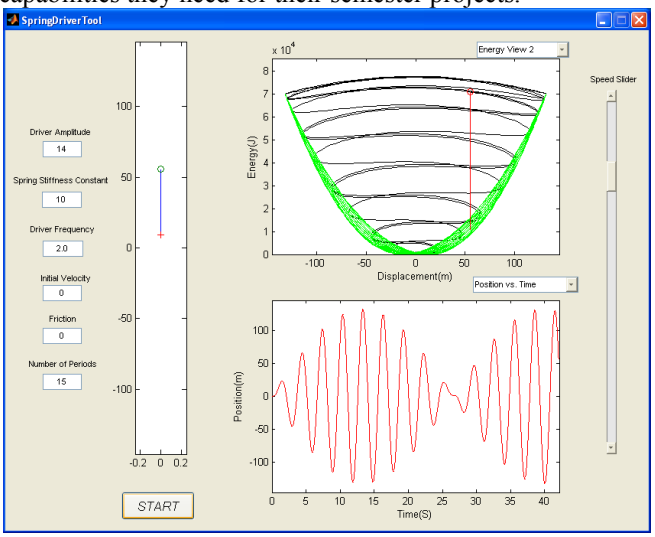

FIGURE 2
STUDENT-WRITTEN GUI TOOL MODELING A DAMPED, DRIVEN, HARMONIC OSCILLATOR.

Lectures are twice a week in sections of more than 200 students. The primary format of the lecture is real-time programming by the instructor, although brief individual or small group exercises help maintain student engagement. Some of the students program on laptops along with the instructor (perhaps 20%). Each week the material introduced by the lectures is the subject of Learning Center (LC) activities in smaller sections (about 25 students per section). The 75 minute LC activities are highly scaffolded and result in deliverable code turned in by each student. They are designed to be a guided introduction to the new material that can be worked out with readily available help from the LC instructor. The weekly homework builds on the capabilities introduced in the LC.

The course project, created in teams of four to five students, is normally a physical device or demonstration and must have a MATLAB GUI tool model accompanying it. The model should be refined enough to demonstrate predictive power, although some calibration of model parameters is allowed. Students can a elect to do a software-only project, in which case a more sophisticated model is required. If the project is game-based, there must be a substantial artificial intelligence component.

ASSESSMENT

The homework sets entail online submission of programs and GUI tools. The final project GUI tool is demonstrated in the LC section and submitted with a final written report.

In addition a midterm and final exam assess the student's understanding of MATLAB programming, including GUI tool creation techniques. This is largely accomplished through multiple choice questions of the form: what will happen when this code is executed? Well-written questions like this assess an interpretive/predictive knowledge of programming.

Constructive programming ability is assessed by two in-class Programming Challenges, performed during a weekly LC time-slot. Each is an individual test of the student's ability to write a MATLAB program. The first Challenge probes the student's ability to employ conditionals, loops, MATLAB vector manipulations, and plotting commands to solve problems. The second Challenge requires the student to construct a complete GUI tool. Both result in online submission of code.

REFERENCES

[1] Lent, C. S., Learning to Program with MATLAB: Building GUI Tools (Wiley, 2013).
[2] Depcik, C. and Assanis, D. Graphical User Interface in an Engineering Educational Environment in Graphical User Interfaces, pg. 48, Wiley Periodicals (2005).
[3] Andreatos, A. and Zagorianos, A. MATLAB GUI Application for Teaching Control Systems. 6th WSEAS International Conference on Engineering Education.
[4] Aliane, N. A MATLAB/Simulink-Based Interactive Module for Servo Systems Learning. IEEE Transactions in Education, vol. 53, No.2, May 2010.
[5] Avitabile,P., McKelliget, J., and Van Zandt, T. Interweaving Numerical Processing Techniques in Multisemester Projects. American Society for Engineering Education Annual Conference (2005).

AUTHOR INFORMATION

Craig S. Lent, Freimann Professor of Electrical Engineering, University of Notre Dame
Jay Brockman, Associate Dean of Engineering for Educational Programs, Associate Professor in the Department of Computer Science and Engineering, and in the Department of Electrical Engineering at the University of Notre Dame.
Kerry Meyers, co-Director, First-Year Engineering Program at the University of Notre Dame, and the Material Science & Engineering ABET Coordinator and Teaching Professor of the First-Year Engineering Program, University of Pittsburgh.
Victoria Goodrich, co-Director, First-Year Engineering Program at the University of Notre Dame.