# Symbolic Scientific Software Skills for Engineering Students

Pramod Abichandani, Richard Primerano, and Moshe Kam
Electrical and Computer Engineering Department
Drexel University
Philadelphia, USA 19104
{Pramod,Rich,Kam}@minerva.ece.drexel.edu

*Abstract*— The emergence of powerful numeric and symbolic scientific software applications, including MATLAB, Maple, and Mathematica, has revolutionized engineering design. These applications have allowed users to perform computations and calculations at levels of sophistication and depth that were not available to practitioners even one generation ago. They have also given educators the ability to convey advanced mathematical and engineering concepts in new ways and spend more time on analysis of engineering systems and less time on remedial mathematics. This new capability, which has become a fundamental tool for sophisticated designers in industry, is still not fully embraced in many engineering curricula.

To exemplify the potential of scientific software in the engineering classroom, we describe a laboratory exercise conducted by second-year engineering students at Drexel University. It introduces a geosynchronous satellite orbital entry problem, and demonstrates how scientific software can help students understand the behavior of an interesting physical system in a way that would have required much more effort using traditional methods.

We believe that early introduction to symbolic computation tools and scientific software would be very valuable to engineering students. Such tools should become standard instruments in the arsenal of present-day engineers. Moreover, their use should be adopted across the curriculum (not only in introductory mathematics classes) and become part of the design experience in all engineering disciplines.

***Keywords - Scientific Software; Symbolic Computation; Engineering Education; MATLAB; Maple; MATHEMATICA.***

## I. Introduction

The emergence of scientific software, and symbolic computation in particular, was recognized early as a potential important addition to the arsenal of tools in the hands of engineers [1]. Applications of symbolic computation, most commonly used at present through MATLAB [2], Maple [3], and Mathematica [4] were demonstrated in all major branches of engineering, ranging from electromagnetic waves [5] to structural engineering [6]. However, the exposure of engineering students to scientific software in general, and to symbolic computation in particular, is still limited. In many programs they are considered "add on" tools, and often are not used beyond introductory classes in mathematics (e.g., [7]). Among the advantages of exposing engineering students to symbolic computation is the opportunity for students and engineers to spend more time on modeling and interpreting results, and less time on technical aspects of solving algebraic problems [8]. The availability of symbolic computation also allows areas of classical analysis to "re-emerge as attractive computational options" [6], providing "a more balanced view that no longer sees analytical techniques and numerical methods as incompatible opposites but combines their usage in a rational way."

As outlined by Røyrvik [8], the process of solving engineering problems from first principles usually involves four steps: (1) stating the problem in mathematical terms (typically developing a set of algebraic of differential equations that model the problem); (2) solving the problem using symbols for variables; (3) substituting numerical values for symbols in the solution that was obtained in Step (2); and (4) validating the outcome and interpreting the results. Steps (2) and (3) are replaced in many instances by developing numerical solutions for a set of fixed parameter values, using numeric computational tools.

Step (2), the development of a solution using symbols for the key variables, is the main 'beneficiary' of symbolic computation techniques. The use of such techniques increases efficiency and accuracy of the design process, and compensates for user limitations in mathematical abilities and technical experience. Not less importantly, the availability of tools to help with Step (2) may alleviate the tendency of students and engineers to avoid solving problems from first principles altogether. Quite often students and engineers seek instead a general formula in a textbook or a research paper, where problem's values can be plugged-in and an (often irrelevant) 'answer' is obtained [8].

We do not discount the importance of teaching and understanding analytical methods. In fact, understanding solution techniques is critical if results are to be validated by the engineer (Step 4, see [9]). However, once the basic solution methodology has been introduced and internalized, students should be introduced to scientific software to implement this methodology and expand it. For example, once students are taught how to solve first and second order differential equations, instead of focusing on "by hand" solutions of higher order differential equations we can introduce and analyze with them the operation of an Ordinary Differential Equation (ODE) solver like the one used by MATLAB [10]. The operation and abilities of different MATLAB calls of the *odexx* kind (*xx* stands for different solvers such as 23, 45) can be studied – including comparisons of the problem types that the different solvers are suitable for, their accuracy and limitations. Similarly, one can teach and analyze the *eig* function in MATLAB and its variants [11], for finding the eigenvalues of a matrix. One of the most important purposes of such study is to provide students with simultaneous appreciation of both the power and the limitations of developing analytical solutions through the use of scientific software.

There are several additional benefits.

*A.* Introduction to scientific software can be used to introduce programming concepts in addition to, and sometimes even instead of, a formal programming course. Computational software packages provide powerful scientific functions that require just a few lines of code to add significant functionality. In this respect, scientific software libraries complement and often surpass capabilities of standard programming languages (like C/C++, Java, and Python). The latter offer only minimal functionality for scientific computing. An example of the power of scientific comptation as a programming tool is the use of a logical vectorization approach to reference elements of matrices. It provides exceptionally rapid code development capabilities [13]. With Integrated Developed Environments (IDEs) that provide intelligent feedback to the user in real time, extensive documentation, and interactive help features, scientific software tools are a perfect starting point for the young engineering student who is new to writing code.

*B.* Introducing powerful computational tools to first and second year engineering students sets the stage for more advanced classes in the future. Concepts that were once reserved only for the upper class students can now be introduced more effectively to undergraduate students. For example, using the *ranksum* function in MATLAB, students can easily perform a statistical test on data like the Wilcoxon rank sum test [14] with only limited introduction to statistical testing.

*C.* Symbolic computation has entered the profession and is increasingly used by practitioners for modeling and simulating engineering systems of various complexities. An introduction to this functionaliy during the early stages of the engineering curriculum would help students develop a skill set that wouild be valuable "for life."

## II. CONTEXT: INTRODUCING SYMBOLIC COMPUTATION AT THE DREXEL UNIVERSITY ENGINEERING CURRICULUM

Recognition of the importance and benefits of using scientific computation as a learning tool led Drexel University's College of Engineering to integrate scientific software tools from MATLAB and MAPLE into the core engineering curriculum (offered to all first and second year engineering students). In this paper we describe an exercise incorporating such elements (from MATLAB) in sophomore-level lab on mathematical modeling of engineering systems. MATLAB is used for illustrating concepts in engineering analysis and design within a sequence of two courses that introduce linear algebra and ordinary differential equations[1]. The two courses are:

1) ENGR 231: Linear Engineering Systems – the course introduces modeling, simulation, and analysis of linear systems using concepts from linear algebra ([15] is used as textbook). Some of the specific topics that are taught focus on various analytical and numerical techniques for classifying and solving systems of algebraic equations. In the first two weeks of ENGR 231, students are taught how to solve up to 3x3 systems "by hand." Beyond that point they are directed to MATLAB's *rref* and similar functions to develop solutions. This approach allows the instructors to concentrate on modeling problems with a larger number of variables, without having to spend time on many technical methodological aspects that are "taken care of" by the software.

2) ENGR 232: Dynamic Engineering Systems – the course introduces modeling, simulation, and analysis of dynamic systems using concepts from theory of ordinary differential equations (ODEs) ([16] is used as textbook). Topics like solving first and second order linear differential equations, finding critical points of autonomous differential equations, converting high order differential equations into systems of systems of first order differential equations and using techniques from linear algebra to solve such systems are introduced in the first few weeks. Later, students are referred to MATLAB's ODE solver functions to solve and analyze the solutions of dynamic system problems of higher complexity.

Both courses provide two one-hour lectures per week (for duration of ten weeks for each course). In addition, a two-hour laboratory session is conducted every week. In the laboratory session, students are presented with modeling problems that build on the concepts presented during lectures. The laboratory manual [17] provides thorough descriptions of several physical systems, along with sample MATLAB code that can be used to model them. During each laboratory session, students are presented with the basic mathematical models of a physical system. They then use MATLAB to answer questions about the system that lead to a solution. A verification procedure is used "to make sense" of results. The students plot selected trajectories and perform sensitivity analysis.

[1] These classes were developed in the last few years by a group of Drexel University professors, including Bruce Char, Thomas Chmielewski, Leonid Hrebien, Jeremy Johnson, Karkal Prabhu, and Oleh Tretiak.

## III. An Example Laboratory Exercise: Geosynchronous Satellite Orbital Entry[17]

To illustrate the type of exercises that are presented to students, we describe a challenge presented during a laboratory session of ENGR 232 that deals with geosynchronous satellite orbital entry [17]. The aim is to understand satellite navigation and guidance into a geosynchronous orbit.

Students are provided with preliminary description of the problem, and are presented with mathematical models that make use of orbital dynamics of satellites to describe their motion. The model is developed based on the scenrio, the basic physics and a list of assumptiuons, and leads to a system of coupled non-linear second-order differential equation system of the form,

$$\frac{d^2x}{dt^2} = -k\frac{x}{(x^2+y^2)^{3/2}}, \frac{d^2y}{dt^2} = -k\frac{y}{(x^2+y^2)^{3/2}}, \quad (1)$$

where $(x,y)$ describes the location of the satellite in the Cartesian coordinate system, and $k = GM$, $G$ being the gravitational constant and $M$ being the mass of the earth.

The students are guided to break the system (1) into a system of four coupled first-order equations using the transformation:

$$u_1 = x\,,\ u_2 = y\,, u_3 = x'\,, \text{ and } u_4 = y'\,, \quad (2)$$

to get the system,

$$\begin{aligned} u_1' &= u_3 \\ u_2' &= u_4 \\ u_3' &= -ku_1(u_1^2+u_2^2)^{-3/2} \\ u_4' &= -ku_2(u_1^2+u_2^2)^{-3/2} \quad . \end{aligned} \quad (3)$$

Students are also provided MATLAB code to implement and solve this system (3) using MATLAB's *ode45* solver. They then plot the path of the satellite for different initial conditions. Upon running this code, students are able to observe different path shapes like circular, elliptical, or hyperbolic (shown in Fig. 1). These are obtained from different initial velocities and demonstrate the implications of Kepler's laws and Newton's laws of gravitation.

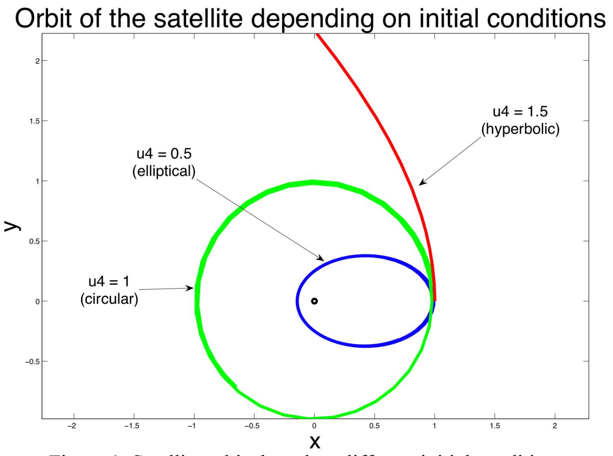

Figure 1. Satellite orbits based on different initial conditions

Finally, as a part of this laboratory exercise, students are required to 'navigate' a vehicle carrying a satellite from a near-earth starting trajectory to a synchronous orbit, using the following guidelines.

(a) The vehicle is initially orbiting in a circular orbit at an altitude of 200 km.

(b) The velocity of vehicle is given a boost so that it moves in an elliptical orbit with apogee at the radius of the synchronous orbit.

(c) Once it arrives at the apogee, an additional velocity boost is given to insert the satellite into the synchronous orbit.

A possible solution is shown in Figure 2.

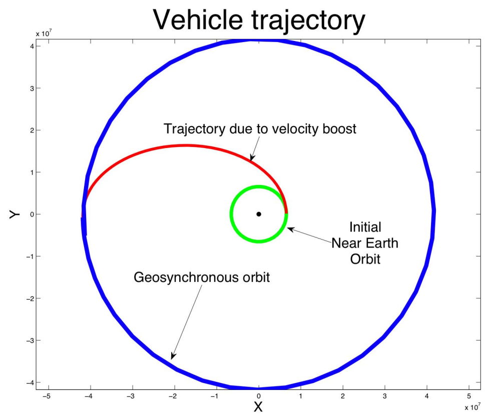

Figure 2. Solution plot of the different phases of a vehicle trajectory

Using this exercise, students are able to solve and plot the solution of a set of nonlinear coupled differential equations, visualize the results, assess the effects of initial conditions and problem parameters, and relate the results to earlier studies in physics. The emphasis is on the problem, its modeling and interpretation of the results, not on the technical aspects of solving the equation set (3).

## IV. OUTCOMES

Towards the end of the course sequence, we observe that for approximately twenty (20) consecutive weeks,

a) students are presented with concepts from linear algebra and theory of ordinary differential equations;

b) students use these concepts to develop mathematical models for various physical systems;

c) students use MATLAB to solve and analyze the mathematical models they see in classes, study in labs and develop in homework; and

d) students use different types of plot functions to develop effective data visualization that helps verify solutions, assess sensitivity, and and communicate ideas to others.

Students often use the knowledge acquired in these class for undergraduate research projects. In Figure 3 we show snapshots of animations of robot motion along various paths (the locations of the robots are represented by circular markings on the paths). These robot paths were studied by a second year student who participated in a project on robot control along pre-specified paths under wireless communication constraints between the robots [18].

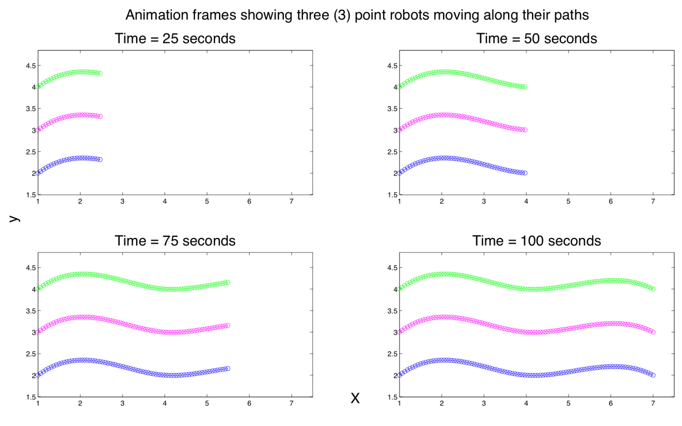

Figure 3. Frame capture of the motion of three robots under path and communication constraints

## V. CONCLUDING REMARKS

We provided general observations on the role of scientific software and symbolic computation in the engineering curriculum, and showed an example from our practice at Drexel University on how to incorporate such tools into the curriculum. The bottom line of our experience is that, owing to the intensive use of numerical and symbolic computation tools, we are able now to work with first- and second-year students on models and systems that were beyond the capabilities of such students just a few years ago.

In this context it is regrettable that, rather than being at the core of the engineering curriculum, scientific computation is still considered in many programs as an addition or an external assistance tool. In the absence of an authoritative study of the matter, one can only speculate on the reasons for the slow acceptance of scientific computation in the engineering curriculum. Possible explanations include the background of many present-day engineering professors and instructors who have developed their own arsenal of analysis and design tools before the emergence of symbolic computation. Some evidence also exists of engineering programs that tried to introduce symbolic computation software into their curricula, but encountered a learning curve which was considered too steep, and resistance from students.

The reality is however that many traditional analytical techniques that were at the core of many engineering curricula for decades are being replaced by the new tools. Tools that are based on computational software and on symbolic computation should therefore enrich all stages of the engineering curriculum, especially upper class design projects where they can show their potential to speed up analysis and improve selection of alternatives. Such use would provide engineering students with experience that is likely to prove highly valuable in their future engineering practice.

## REFERENCES

[1] A.I. Beltzer and A.L. Shenkman: "Use of symbolic computation in engineering education," *IEEE Transactions on Education,*, vol. 38, no. 2, pp.177-184, May 1995.

[2] MATLAB, http://www.mathworks.com/products/matlab/

[3] MAPLE, http://www.maplesoft.com/Products/Maple/

[4] MATHEMATICA, http://www.wolfram.com/products/mathematica/index.html

[5] D. T. Alves, J. V. Amaral, J. F. Medeiros Neto, and E. S. Cheb-Terrab: "Learning Electromagnetism Via Programming and Symbolic Computation," *Revista Brasileira de Ensino de Física*, Vol. 24, No. 2, São Paulo, June 2002.

[6] M. N. Pavlovi, "Symbolic computation in structural engineering," *Computers & Structures*, Vol. 81, nos. 22-23, pp. 2121-2136, September 2003.

[7] S. Noinang, B. Wiwatanapataphee and Y. H. Wu: "Teaching-learning Tool for Integral Calculs," Far East Journal of Mathematical Education, Vol. 3, No. 3, pp. 203 – 212, October 2009.

[8] Ola Røyrvik: "Teacing Electrical Engineering using Maple," International Journal of Electrical Engineering Education, Vol. 39, No. 4, October 2002.

[9] Mordechai Shacham, Neima Brauner, W. Robert Ashurst, and Michael B. Cutlip: "Can I Trust This Software Package? An Exercise In Validation of Computational Results, Chem. Eng. Educ., Vol. 42, No. 1, pp. 53-59, Winter 2008 (available on-line: http://www.polymath-software.com/papers/CEE_42_53_01.pdf).

[10] Mathwork's MATLAB ODE solver documentation, available at http://www.mathworks.com/support/tech-notes/1500/1510.html

[11] Mathwork's MATLAB documentation for eig function that calcuates eigenvalues and eigenvectors, available at http://www.mathworks.com/access/helpdesk/help/techdoc/ref/eig.html

[12] Mathwork's MATLAB page on the plotting functions, available at http://www.mathworks.com/access/helpdesk/help/techdoc/ref/f16-8602.html

[13] Mathwork's MATLAB page on logical indexing, available at http://www.mathworks.com/access/helpdesk/help/techdoc/math/f1-85462.html

[14] Mathwork's MATLAB page on Wilcoxon rank sum test, available at http://www.mathworks.com/access/helpdesk/help/toolbox/stats/ranksum.html

[15] D. Lay, Linear Algebra and its applications , 3rd edition, Addison Wesley.

[16] J. Brannan, W. Boyce, Differential Equations: An Introduction to Modern Methods and Applications [With ODE Architect Website Support], John Wiley & Sons , 2006.

[17] O. Tretiak et al., Student Manual for ENGR 232, College of Engineering, Drexel University, 2007.

[18] P. Abichandani, H. Benson, and M. Kam, "Multi-vehicle path coordination under communication constraints," in Proc. AmericanControl Conference (ACC'08), Seattle,WA, June 2008.