# An Interdisciplinary Approach to Scientific Modeling and Simulation

**Mike O'Leary**
**Department of Mathematics**
**Towson University**
**moleary@towson.edu**

**Shiva Azadegan**
**Department of Computer and Information Sciences**
**Towson University**
**azadegan@towson.edu**

## Abstract

We describe a model for an interdisciplinary course in scientific modeling and simulation. We discuss the course structure and content, as well as the results of our evaluation process.

**Keywords:** Education, Simulation, Software engineering

## 1 Introduction

This paper describes a new upper-level undergraduate course in scientific modeling and simulation. The course is centered on a sequence of projects, each based on a realistic scientific problem. We teach the students the mathematics necessary to model the problem, and the numerical and computational methods for its solution. We then teach the students how to implement these methods using object-oriented programming. We use Microsoft Visual C++ 6.0 and its associated Microsoft Foundation Classes (MFC). In particular, the students develop dialog-based windows programs that take full advantage of our computers' graphical capabilities. The scientific and programming aspects of the course are integrated with one another and taught concurrently. The course is inherently interdisciplinary, and has been team-taught by one computer scientist and one applied mathematician.

In this paper, we present the course model, including a description of the course's topics. We shall also describe some of the projects that have been developed especially for this course, together with references. Finally, we shall present the results of our evaluation of the effectiveness of the course.

We have two primary audiences for our course. First are computer science majors who are looking for additional experience with mathematics and scientific applications. Second are majors in mathematics and the sciences who wish to take a second course in computing that is tailored to their discipline.

This course is valuable to our computer science majors because it exposes them to realistic scientific problems and the mathematics necessary for their solution. The Steelman draft of *Computing Curricula 2001 Computer Science* [11, Chap. 9] recommends that computer science majors take additional mathematics in areas like numerical methods. They also recommend that computer science majors engage in an in-depth study of some subject that uses computing in some substantive way. This course meets these goals.

The course is also valuable to majors in the sciences. Most of these students take only one general introductory course in computer science. Our course expands on the topics that they learned in their introductory course and drives home the fundamental importance of object-oriented programming in a setting that is tailored for science and mathematics students. The Steelman draft [11, Chap. 12] emphasizes the need for computer science departments to teach computing across the curriculum, and holds up an example of a computational science course as an example of an area-wide or multidisciplinary course that our departments can offer.

## 2 Audience

As we noted in our introduction, the primary audience for our course consists of computer science majors with an interest in science and mathematics, as well as science and math majors who wish to take a second course in computer science. Another group of students that have taken the course are computer science majors who are more interested in learning MFC and windows programming than the scientific content of the course. In part this is because we currently do not have a course that covers MFC.

More interestingly, we have been approached by the director of the graduate program in mathematics education about the possibility of offering a graduate level version of this course to in-service high school teachers working towards their Master's degrees.

The prerequisites for the course include our Introduction to Computer Science I course, which covers C++ programming up through classes. We also require students to take Calculus 1 and Calculus 2. Despite this minimal list of prerequisites, there are many interesting and real scientific problems that can be studied in the course.

## 3 Format

Our course is inherently interdisciplinary, and is team-taught by one computer scientist and one applied mathematician.

The course is organized around a series of projects. Each project is a self-contained scientific question. As an example, consider the following: At what angle should a real baseball be thrown to maximize the horizontal distance it will travel, taking into account air resistance?

We begin each project by learning the mathematics necessary to model the problem, and constructing that model. We then teach the numerical analysis necessary to construct an algorithm that can compute the solution.

Next, we teach students the programming techniques that are needed to implement the numerical method. Our emphasis is on object-oriented programming techniques. In particular, we use Visual C++ 6.0 and MFC as our implementation language. This choice was made because this is the language used in our Introduction to Computer Science Programming 1 course. We focus our attention on dialog-based windows programming that take full use of the graphical capabilities of MFC.

Our course model is especially suited to teaching object-oriented programming because the programs break down naturally into routines for input, output and computation. The programs are sufficiently complex that the students see immediately the benefits of this approach. The complexity of the programs also drives home to the students the importance of good software design; we spend class time discussing this issue.

The programs that students write are not the end of the course; they are used as tools to solve scientific problems. Students find that, even if their program is running correctly, it may not be giving them the answers to the scientific questions that they need to know. Because students *use* their programs as well as write them, we find that students revise their programs as they discover more about the scientific problem.

The course itself is a mixture of traditional lecture and computer laboratory. Our class met twice a week for 75 minutes, and we had a lecture component and a laboratory component to nearly every class meeting.

Student progress is assessed primarily by project reports. At the conclusion of each project, students hand in a complete report of their investigations. The report contains the mathematical model for the problem, including an outline of its derivation. It also describes the mathematics of the algorithm that is used in the solution of the problem. Students discuss the code for the program that they have written to solve the problem. Finally, students describe the results of their investigations, and draw conclusions.

We prefer this method of assessment because it helps develop student's communication skills. For some projects, we allow the students to work in teams; other projects are solved singly. Communication skills and teamwork have been identified in the Steelman draft of *Computing Curricula 2001 Computer Science* [11, §§ 9.1.4, 9.1.5] as an important part of a good computer science curriculum.

Our experience using this method of assessment has been positive. However, we find that when there is one deadline for the entire project, students work on their programs right up to the deadline, and consequently do not spend sufficient time analyzing the results and drawing conclusions. We recommend that adopters of this method have two deadlines, one for the program and a later date for the project report.

## 4 Topics

We teach the course by examining a sequence of projects. As a part of each project, we teach the students some new mathematics and some new programming techniques.

Mathematically, we begin by reviewing the usual methods for numerical integration; in particular, we discuss the Trapezoidal Rule and Simpson's Rule. These topics are now a standard part of most calculus sequences [20 §7.7]. Next, we teach students how to model problems using ordinary differential equations.

We continue with methods for solving initial value problems for systems of ordinary differential equations. In particular, we start with a discussion of Euler's method, describing its derivation and error estimates [2 §8.1, 3 §5.2, 20 §7.7]. We discuss issues of machine precision and round-off error. We quickly move to Runge-Kutta methods [2 §8.3, 3 §5.4]. Our coverage of Runge-Kutta methods is applications based, so we study implementation and error analysis; but we do not present the details of the derivation.

Next, we introduce students to the notion of a partial derivative. This is a standard part of the Calculus 3 course; however many of our computer science students do not take Calculus 3. Rather than add to the prerequisites of the course, we introduce only what we actually need from the theory of partial derivatives. In particular, we discuss their definition and computation, as well as the chain rule. We omit many traditional topics, like their application to finding tangent planes, solely because they are not needed in the applications we have selected, and because our time is limited.

Once students are familiar with the elementary properties of partial derivatives, we introduce them to modeling scientific problems with them. One successful approach has been to introduce Lagrangian dynamics [22]. This approach uses partial derivatives, but the result is a system of ordinary differential equations that can be solved using the Runge-Kutta techniques that have already been taught. We have also studied classical problems that result in partial differential equations proper, like the diffusion equation, and the wave equation. [6, 12] We then taught students finite difference methods for the solution of these equations [3 Chap. 12, 21]. We chose finite difference methods because of their conceptual similarity to Euler's method and the Runge-Kutta methods that students learned in their study of ordinary differential equations. We have been successful introducing students to the concepts of consistency, stability and convergence of finite difference schemes, including a brief discussion of the Lax-Richtmyer Equivalence Theorem and the Courant-Friedrichs-Lewy Condition. [21, Chap. 1]

Our emphasis on programming from the beginning is on object-oriented programming. We begin the course with a brief review of classes; we then introduce Visual C++ 6.0, and discuss its features. Students begin by learning how to construct dialog-based windows programs. Consequently, students are introduced to event-driven programming. At the outset, the programs are simple, with a few edit boxes and a few buttons, which enables us to introduce this concept gradually.

After students have assimilated these topics, we introduce the graphical routines provided in MFC; in particular how to use the device contexts provided by MFC. We teach students how to create and initialize multiple dialog windows, and how to draw in each. Emphasis is placed on animating the results. The importance of good graphics and animation cannot be underestimated in scientific computing.

We do not try to introduce the entire set of graphics classes available in MFC. Moreover, the portions of MFC that are taught are introduced one at a time, on an as needed basis. The sheer size of the MFC library prevents us from offering a comprehensive introduction even to just the graphical classes. Instead, students are introduced only to those classes and member functions that are required.

As students progress through the course, we build on this foundation, introducing additional concepts. Of particular value have been the routines to take input from the mouse. Other routines that can be introduced handle saving and loading files, and printing.

Our experience has not been confined to solely teaching dialog-based programs; we have also taught single-document-interface (SDI) programming. We found that SDI programs were more difficult for the students to understand. In part this is because the standard framework constructed by the MFC AppWizard includes a large number of features; so many that inexperienced programmers become overwhelmed. We recommend that instructors not begin the course with SDI programs, but they are probably suitable at the end of a course taught to strong programmers.

There are a number of texts suitable as supplements that discuss programming and MFC; we specifically mention [1, 5, 23].

## 5 Projects

At the center of our course are the projects, four per semester. The first project is elementary. In it, we introduce MFC and dialog-based programs; we also review numerical integration and the numerical solution of ordinary differential equations. The scientific and modeling content is small, as we focus primarily on the programming and mathematical techniques that we will be using for the remainder of the course.

The next two projects are at an intermediate level. In these projects, we introduce the graphical elements of MFC, as well as the modeling and solution of scientific problems resulting in systems of ordinary differential equations.

We end the course with an advanced project that requires students model and solve a problem that results in a partial differential equation. Concurrently, we introduce additional elements of MFC, like the use of the mouse.

The hardest part of teaching a course like this one is finding appropriate project topics. Here are some of the projects that either have been used for our class in the past, or will be used when the course is next offered.

**Motion of a baseball with air resistance.** The motion of a baseball without air resistance is a simple classical problem that can be solved symbolically. When air resistance is included, the problem becomes much more difficult to solve. The general structure of the force due to air resistance can be found by dimensional analysis, and the precise form from experimental data; see [9, 14]. The problem results in a simple system of ordinary differential equations.

In the project, we asked our students to find the angle at which to hit a baseball at a fixed velocity so that it would travel the farthest horizontal distance.

**The three-body problem.** Newton's law tells us that the force of gravity of one body, say a planet, on another is proportional to their masses and inversely proportional to the square of the distance between them [20, §13.4]. The three-body problem is to determine the motion of three bodies where the only force acting between them is the force of gravity. Even if we restrict our motion to the plane, in the nineteenth century Poincaré showed that, in effect, it is impossible to find an explicit formula that describes in general the solution to the three-body problem. It is however, possible to solve the resulting system of differential equations numerically.

Our students wrote a program that simulated the motion of three bodies under gravity. Their program animates the numerical solution of the problem, and presents the motion graphically. A good project question would be to determine how many types of stable orbits that the students can find using their simulation. This question has more than a pedagogic interest, as mathematicians are still studying the question; see [4, 15, 18] for readable accounts of the current state of the field.

**Filter Circuits.** A circuit consisting of a resistor, capacitor and inductor can be used as a filter circuit [19 §4.5], allowing some signal frequencies to pass while limiting others. This is an example of a phenomenon called resonance in differential equations [2, §3.9].

The project consists of two parts. First students write a program that simulates the behavior of an LRC resonant filter. Then we give the students a signal composed of two sine waves of different, unknown frequencies and have them use their program to determine the component frequencies.

**Double pendulum.** A double pendulum is a pendulum attached to the end of a second pendulum, the whole thing being constrained to move in a single plane. The model for a single pendulum is relatively easy to derive; however the double pendulum is much more complex. The best approach to modeling the double pendulum is to use Lagrangian dynamics [10 pp. 355-357, 22]. This requires the instructor to introduce partial derivatives and some elements of the calculus of variations [7, IV §§1,10]. However, the resulting model is only a system of ordinary differential equations.

For the project, we asked the students to determine if the system had sensitive dependence on initial conditions; namely would small changes in the initial positions of the pendulum cause large variations in the subsequent motion of the double pendulum.

**Spread of HIV.** An area of cutting-edge concern is the modeling of the spread of the HIV virus through the human body. Perelson and Nelson in [17] describe a model for the dynamics of HIV infection that is suitable for student investigation. This work is based on the seminal paper of Perelson, Kirschner, and De Boer [16]. In addition to describing a model for the dynamics of infection, it also presents models for the effect of various treatment strategies, including the use of RT inhibitors and the use of protease inhibitors. Both of these models are also systems of ordinary differential equations.

One project would be to simulate the effect of RT inhibitors. It can be shown analytically that if the RT inhibitor is 100% effective then the number of virus particles in the blood will decay to zero. We ask the students to use the simulation to show that there is an effectiveness threshold; if the inhibitor is sufficiently effective then the virus will be gradually eliminated from the bloodstream, but below that point, the virus load will decay to a nonzero constant.

**Wave Motion.** Wave motion occurs in many different contexts, for example, compression waves arise when a solid object is struck, while transverse waves arise when a taut string is plucked. Both of these physical situations can be described by the same partial differential equation, the wave equation. A nice derivation of the wave equation in each of these contexts can be found in [6, Chap. 1] while a good general description of wave motion can be found in [12].

We asked students to create simulations of the plucked taut string, and to investigate how the speed of the wave is related to the physical parameters of the string.

**Traffic Flow.** Traffic flow down a single road can also be modeled using partial differential equations; a nice derivation appropriate for our students is in [12, Chap. 16]. We can numerically solve the partial differential equation that governs traffic flow using the Lax-Friedrichs finite difference method; see [13, Chap. 10].

For the project, we asked students first to show that this model allows for the formation of traffic jams. These show up as a shock waves in the mathematical model and for this reason care must be used in the selection of the finite difference scheme used to solve the problem. We then asked the students to determine how the traffic jam would evolve; in particular does the point at which the traffic jam begins move, and if so does it move with or against the traffic.

## 6 Evaluation

Pilot versions of this course ran in the Fall 2000 semester, and again in the Spring 2001 semester.

One internal and one external evaluator examined the pilot course. In addition to class visitations, surveys were conducted at the beginning, in the middle and at the end of each semester. In addition, a follow up survey was conducted with the students in Fall 2000 class. Because the number of students in each class was small, the findings from the evaluation activities should be taken as preliminary.

There were more computer science majors in Fall 2000 class than in Spring 2001 class, although the spring class included students who were double-majors in Computer Science and Mathematics. While the majority of the Fall 2000 students were sophomores, all students in Spring 2001 were juniors or seniors. Thus, these two classes were somewhat different in their backgrounds.

In spite of these differences, these students' overall sentiment to the course can be summarized as, "It was challenging but I liked it." Several reasons for the "challenging" nature of the course were suggested by the students. They include deficiencies in their background in mathematics, programming and/or science, the complexity of mathematics involved, and the difficulty of the course projects. The survey results suggested that learning something practical made the course interesting. Several students indicated that they wished the course discussed in more depth the programming aspect of the course.

The course has been well received by the students in general. They enjoyed the course and many of them felt they learned much.

As a consequence, this course has been approved by both the Computer and Information Sciences Department and the Mathematics Department as a regular course that will now become part of the curriculum.

## 7 Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. DUE 9952625.

## References

[1] Bates, J. and Tompkins, T. *Practical Visual C++ 6*, Que Corporation, 1999.

[2] Boyce, W. and DiPrima, R.C. *Elementary Differential Equations*, sixth edition, John Wiley & Sons Inc., 1997.

[3] Burden, R.L., and Faires, J.D. *Numerical Analysis*, sixth edition, Brooks/Cole Publishing, 1997.

[4] Casselman, B. A New Solution to the Three Body Problem - and More. Online. Internet. Available WWW: http://www.ams.org/new-in-math/cover/orbits1.html

[5] Chapman, D. and Heaton, J. *Teach Yourself Visual C++ 6 in 21 Days*, Sams Publishing, 1999.

[6] Churchill, R.V. and Brown, J.W. *Fourier Series and Boundary Value Problems*, fourth edition, McGraw Hill, 1987.

[7] Courant, R and Hilbert, D. *Methods of Mathematical Physics, volume 1*, John Wiley & Sons, 1953.

[8] Feldman, J. *The Animated Telegraph Equation*, Online. Internet. WWW: http://www.math.ubc.ca/~feldman/apps/telegrph.ps and http://www.math.ubc.ca/~feldman/demos/demo8.html

[9] Frohlich, C. Aerodynamic Drag Crisis and its Possible Effect of the Flight of Baseballs. *American Journal of Physics, 52:4* (1984), 325-334.

[10] Hestenes, D. *New Foundations for Classical Mechanics*, D. Riedel Publishing, 1987.

[11] The Joint Task Force on Computing Curricula, IEEE Computer Society and the Association for Computing Machinery, *Computing Curricula 2001, Steelman Draft*, *August 1, 2001*. Online. Internet. WWW: http://www.acm.org/sigs/sigcse/cc2001/steelman/index.html

[12] Knobel, R. *An Introduction to the Mathematical Theory of Waves*, American Mathematical Society, 2000.

[13] LeVeque, R.J. *Numerical Methods for Conservation Laws*, Birkhäuser Verlag, 1992.

[14] Long, L. N and Weiss, H. The Velocity Dependence of Aerodynamic Drag: A Primer for Mathematicians. *The American Mathematical Monthly, 106:2* (1999), 127-135.

[15] Montgomery, R. A New Solution to the Three-Body Problem. *Notices of the American Mathematical Society 48:5* (2001), 471-481.

[16] Perelson, A.S., Kirschner, D.E., and De Boer, R. Dynamics of HIV Infection of CD4$^+$ T Cells. *Mathematical Biosciences, 114* (1993), 81-125.

[17] Perelson, A.S. and Nelson, P.W. Mathematical Analysis of HIV-I in vivo. *SIAM Review, 41:1* (1999), 3-44.

[18] Simó, C. Three Body Choreopgraphies. Online. Internet. Available WWW: http://www.maia.ub.es/dsg/3body.html

[19] Sprott, J.C. *Introduction to Modern Electronics*, John Wiles & Sons Inc., 1981.

[20] Stewart, J. *Calculus. Early Transcendentals*, fourth edition, Brooks/Cole 1999.

[21] Strikwerda, J.C., *Finite Difference Schemes and Partial Differential Equations*, Wadsworth & Brooks/Cole, 1989.

[22] Wells, D.A. *Theory and Problems of Lagrangian Dynamics*, Schaum's Outline Series, McGraw Hill, 1967.

[23] Yang, D. *C++ and Object-Oriented Numeric Computing*, Springer-Verlag, 2001.