AC 2007-272: TEACHING PROGRAMMING AND NUMERICAL METHODS AS CONCURRENT COURSES

David Sawyers, Ohio Northern University

    DAVID R. SAWYERS, JR. is an Assistant Professor of Mechanical Engineering at Ohio Northern University, where he teaches courses in General Engineering and in the Thermal Sciences. He received a BSME degree from Rose-Hulman Institute of Technology and the MS and PhD, both in Mechanical Engineering, from The University of Notre Dame.

John-David Yoder, Ohio Northern University

    JOHN-DAVID YODER is an Associate Professor of Mechanical Engineering at ONU. His Doctorate is from the University of Notre Dame. Research interests include education, controls, robotics, and information processing. Prior to teaching, he ran a small consulting and R&D company and served as proposal engineering supervisor for GROB Systems, Inc.

Teaching Programming and Numerical Methods
as Concurrent Courses

Abstract

This paper discusses the efforts of the authors to coordinate their teaching of Programming and Numerical Methods to third-year mechanical engineering students. Course schedules and assignments have been coordinated to take advantage of the overlapping topics and skills required. In particular, several joint assignments were given to combine the students' understanding of numerical methods with their ability to develop computer programs using the C++ language. These assignments required the numerical solution of problems students had previously encountered in other engineering courses.

Introduction

In the mechanical engineering curriculum of Ohio Northern University, a Junior-level course in Numerical Methods has been offered for many years. This course introduces methods and algorithms for solving a wide range of problems, while presuming that students possess adequate knowledge of a computer language to implement these algorithms. In the past, students who were enrolled in this course had previous experience (usually during the sophomore year) with a structured programming language. However, due to significant changes in the Freshman and Sophomore curriculum, the prerequisite programming course is no longer available.

Due to the removal of programming from the general engineering curriculum at Ohio Northern, two options were considered: ending the requirement for structured programming in upper-level courses (relying solely on application-specific software tools), or offering a Programming course concurrently with Numerical Methods in the Junior year. While commercial software is useful for solving many problems, the mechanical engineering faculty decided that knowledge of a structured programming language was still an important skill for our students to develop[12]. As a result, a C++ programming course was introduced in the fall quarter of 2006. This Junior-level course, consisting of one lecture and one lab per week, was taught concurrently with Numerical Methods.

This paper will provide an overview of the format and topics covered in both courses, will describe how the two courses have been coordinated, and will discuss some of the benefits and difficulties encountered. In particular, the authors will discuss several problems, based on engineering applications, that were assigned jointly in both classes.

Numerical Methods

In the recent past, Numerical Methods has been taught to mechanical engineering undergraduates at Ohio Northern University in the fall of the Junior year. The course consists of four fifty-minute lectures per week. The textbook used is **Numerical Methods for Engineers** by Chapra and Canal[3] and the lectures generally follow the text, although only a fraction of the topics included by the authors can be adequately covered in a first course on numerical methods.

During the ten-week quarter, a typical list of topics includes:

- Roots of equations.
- Systems of algebraic equations (linear and nonlinear).
- Optimization (with applications to design).
- Curve fitting.
- Numerical integration and differentiation.
- Ordinary and partial differential equations.

The outcomes for Numerical Methods are stated in the syllabus as follows:

Upon completion of the course, students will be able to:

1. *select and apply the appropriate numerical method for a variety of common engineering problems.*
2. *effectively use commercially-available software tools (e.g. Matlab, Excel).*
3. *develop specialized computer programs to solve engineering problems (e.g. Matlab, C++).*
4. *validate and document numerical solutions.*

While one obvious goal is to introduce students to various techniques for solving specific problems, the course also provides a unique opportunity to accomplish several other objectives as well. The study of numerical methods forces the students to improve their problem solving skills, particularly since most numerical solution techniques require a systematic approach to a clearly-defined problem. Numerical methods is also by nature a multi-disciplinary subject, since the algorithms developed can be applied to problems from various areas of engineering which share the same underlying mathematics. As a result, this course provides an opportunity to review topics and problems seen in other classes. Particular care has been taken to introduce applications problems (or "case studies") from other courses within the mechanical engineering curriculum. The case studies also provide an opportunity for students to develop the ability to communicate clearly and concisely at a very specialized technical level. Finally, emphasis is placed on the need to validate numerical solutions.

Programming

The new structured programming course has been developed to be taught concurrently with the Numerical Methods course described above. The course is based on Microsoft Visual Studio .NET and uses the textbook **How to Program C++** by Deitel and Dietel[4]. The course consisted of one fifty-minute lecture and one 110-minute laboratory session per week. The laboratory sessions were limited to twenty students in order to improve the ability of the professor to interact with the students. The laboratory sessions were conducted in a computer laboratory equipped with a SMART board and SynchonEyes software to assist in instruction.

Weekly assignments were given to be completed in the laboratory. Some of these were the Joint Assignments discussed in the next session. The remainder were individual assignments chosen to include some aspect of the Numerical Methods course. In some cases, students were asked to implement an algorithm discussed the previous week in Numerical Methods and were pointed to (though not required to use) the pseudo-code in the Numerical Methods textbook.

The course outcomes for the course state that:
Upon completion of this course, the student will be able to:
1. *Edit, run, and debug C++ programs.*
2. *Write C++ programs including control structures (loops and ifs).*
3. *Write C++ programs to input data from files and output data to files.*
4. *Write C++ programs to solve numerical methods problems.*

Joint Assignments

Many of the homework problems assigned in Numerical Methods are relatively straightforward, well-defined mathematical problems. The purpose of the homework is to emphasize the basic concepts of various numerical methods, and to reinforce the material covered in the lectures. However, it is also important for students to solve more complex problems relating to engineering practice. In order to accomplish this, several "case studies" are assigned during the course. These case studies are application problems, requiring significantly more effort than the homework. Students are allowed to work in teams of two, and are required to submit a written report for each assignment. The case studies also require students to develop a computer program using a structured programming language (recently C++). With the introduction of a concurrent programming course, the instructors decided to use these assignments as a way of linking the two courses. While students would learn basic numerical algorithms and programming skills independently, the case studies would demonstrate practical applications of programming, while at the same time providing a way to solve problems too large or complex to be efficiently solved using Matlab or Excel.

A list of topics covered in the two courses, along with the joint assignments, is provided in Table 1. Jointly-assigned case studies from the fall quarter of 2006 include:
- Disassociation of Water – Finding roots of an equation.
- Polytropic Process of an Ideal Gas – Linearization and curve fitting.
- Distance, Speed, and Acceleration – Numerical integration and differentiation.
- The Forced Pendulum – Runge-Kutta solution of ODEs.

Care was taken to ensure that each case study was based on a problem that the students would recognize as being of practical importance. The first two problems relate to Thermodynamics, a course most of the students were taking concurrently with Numerical Methods and Programming. The third case study referred to Dynamics, a course taken the previous year. The fourth case study introduced a problem that students had not previously encountered, but which combined Differential Equations (a prerequisite for Numerical Methods) with an extension of the physically intuitive damped, *unforced* pendulum.

For each case study, students were provided with a brief review of the background of the problem, an explanation of what they must do in order to complete the assignment, and a list of deliverables specifying the format and contents of the report (see Appendix A). Students were required to demonstrate their working code to the Programming instructor, who evaluated their implementation of the algorithm and the validity of their results. Students also submitted a written report to the Numerical Methods instructor, who evaluated their writing, understanding of the problem and algorithm, and their validation of results.

A certain amount of coordination was required between the two instructors to ensure that students had both the numerical methods and programming tools needed to complete each case study when assigned. This was accomplished partly by scheduling topics appropriately, and partly by tailoring the requirements of each assignment to match topics that had been covered.

Week	Numerical Methods	Programming
1	Error Analysis and Validation	Introduction to C++
2	Roots of Equations	Loops
3	Linear and Nonlinear Algebraic Equations	Case Study #1
Case Study #1: Disassociation of Water – Finding roots of an equation.		
4	Optimization	Arrays
5	Interpolation and Curve Fitting	Searching and Sorting
Case Study #2: Polytropic Process of an Ideal Gas – Curve fitting.		
6	Numerical Differentiation and Integration	In-class exam
7	ODEs: Initial Value Problems	File I/O
Case Study #3: Distance, Speed, Acceleration – Integration and differentiation.		
8	ODEs: Boundary Value Problems	Functions
9	Partial Differential Equations	Case Study #4
Case Study #4: The Forced Pendulum – Runge-Kutta solution of ODEs.		
10	Partial Differential Equations	In-laboratory exam.

Table 1: Course content and joint assignments.

Assessment

Each case study was graded in two parts. The Programming instructor evaluated the performance of the program, including the technical results produced. The Numerical Methods instructor evaluated the written report, which included a description of the method used, presentation of results (graphs, tables, etc.), and an explanation of how the results were validated. Table 2a shows the resulting grades for each program, while Table 2b shows the grades for each report. For each case study, Table 2 shows the average grade, as well as the distribution of grades in the form of an EAMU vector, commonly used in assessment for accreditation purposes.[5]

Case Study	Average (%)	Excellent ($\geq 90\%$)	Acceptable ($\geq 70\%$)	Marginal ($\geq 60\%$)	Unacceptable ($< 60\%$)
1	96	30	2	1	0
2	95	32	0	0	1
3	84	12	20	1	0
4	99	30	3	0	0

Table 2a: Grade distribution for case study C++ programs.

Case Study	Average (%)	Excellent ($\geq 90\%$)	Acceptable ($\geq 70\%$)	Marginal ($\geq 60\%$)	Unacceptable ($< 60\%$)
1	80	3	28	1	2
2	89	16	16	2	0
3	84	13	16	4	1
4	85	16	18	0	0

Table 2b: Grade distribution for case study reports.

A survey was also administered in Programming at the end of the quarter, to evaluate student opinions of the case studies. Students were asked to respond to the questions shown in Table 3 using a scale from 1 (Strongly Disagree) to 4 (Strongly Agree). It should be noted that one student in each course was not enrolled in the other, haven taken either Programming or Numerical Methods previously.

Question	average	agree	disagree
1. The programming assignments helped my understanding of numerical methods.	2.9	24	10
2. I liked having one case study count as an assignment in two classes.	2.9	21	12
3. I was able to complete the case studies without using information from courses other than Numerical Methods and Programming.	3.1	28	6
4. Developing computer programs related to numerical methods helped me understand the need for programming.	3.2	28	6
5. As engineering students, we should have more projects that combine topics from more than one class.	3.1	28	6
6. I learned more from the individual assignments than from the team assignments.	2.3	11	22
7. I can now write a program to implement an algorithm from numerical methods.	3.0	29	5
8. It is easier for me to write a program from scratch than by following the pseudocode provided in the book.	3.1	25	9
9. Completing case studies based on engineering problems gave me confidence that I could validate the solutions.	3.0	30	4

Table 3: Student responses to end-of-course survey.

The responses to questions 1, 4, 5, and 7 indicate that students generally found the joint assignments to be beneficial. The response to question 3 suggests that it might be possible to provide somewhat less background information in future, requiring students to refer to their notes and textbooks from other engineering courses for the required theory. The response to question 8 is somewhat surprising; the instructors had assumed that allowing the students to begin with pseudocode provided in the Numerical Methods text would ease the workload somewhat. However, this does not appear to be the perception of the students. Finally, the response to question 9 is encouraging, since a significant emphasis was placed on validation of solutions in the Numerical Methods lectures.

Conclusions

While the addition of a Junior-level programming course was necessitated by changes in the general education curriculum, it provided a beneficial learning experience for the authors. On one hand, offering Programming as a separate course (with a separate instructor) ensured that C++ was not seen by the students only as a tool for completing Numerical Methods assignments. On the other hand, including joint assignments provided a rare opportunity for coordination between different instructors across course boundaries. The inclusion of problems from other engineering courses was also successfully accomplished, and the feedback provided by the student reports and the end-of-course survey should allow these problems to be further refined for next year.

One issue raised on student course evaluations was the workload involved in completing the Programming assignments. The instructors had expected students to see the joint assignments as a reduction in overall workload, since the same work would be counted in both courses. However, as seen in the response to question 2 of the end-of-course survey (Table 3) as well as in course evaluation comments made by several students, this was not the case. In future, it will be made clear to students that the work done in developing programs to solve the case studies is part of the workload for both courses. This can probably be accomplished by spending more lecture time in Numerical Methods dealing with the case study programming assignments.

References

[1] Urban-Lurain, M. and D. J. Weinshank, "Do Non-Computer Science Students Need to Program?" Journal of Engineering Education, October 2001.

[2] Hodge, B. K. and W. G. Steele, "A Survey of Computational Paradigms in Undergraduate Mechanical Engineering Education," Journal of Engineering Education, October 2002.

[3] Chapra, S. C. and R. C. Canale, **Numerical Methods for Engineers**, 5th ed., McGraw-Hill, 2006.

[4] Dietel, H. M. and P. J. Dietel, **How to Program C++**, 4th ed., Prentice-Hall, 2002.

[5] Estell, J. K., "The Faculty Course Assessment Report," Proceedings of the Best Assessment Practices VII Symposium, Rose-Hulman Institute of Technology, Terre Haute, IN, April 2005.

Appendix A – Sample Handout

ME371 Numerical Methods, ME372 Programming
Case Study #4 – The Forced Pendulum

Background

The following differential equation governs the angular position θ(t) of a simple forced pendulum with damping (friction):

$$\frac{d^2\theta}{dt^2} + c\frac{d\theta}{dt} + \frac{g}{L}\sin\theta = f(t) = a\sin(bt) \tag{1}$$

where c is the damping coefficient, g is the gravitational constant, L is the length of the pendulum, and f(t) is the forcing function.
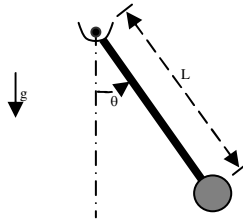

Figure 1: Geometry of a simple pendulum.

Assignment

You will work in teams of two to complete the following tasks:

1. Develop a program (using C++) which will implement the 4th-order Runge-Kutta algorithm to solve a 2nd-order initial value problem. The algorithm should be implemented as a function as shown in the pseudocode. The program should allow the user to type in the values for *a*, *b*, and *c* in equation 1. The program should also let you type in the name of the file in which you want to store the results (this will be covered in class and lab in week 9).

2. Use the program to solve the forced pendulum problem (eqn 1) for the following cases:
    a) c = 0, f(t) = 0.
    b) c = 0.1 r/s, f(t) = 0.
    c) c = 0.1 r/s , f(t) = 5sin(t)
    d) c= 0.1 r/s, f(t) = sin(4.4 t)
    e) c= 0.1 r/s, f(t) = sin(26.4 t)
    f) c= 0.1 r/s, f(t) = 100*sin(26.4 t)

   For each case, use g = 9.81 m/s^2, L = 0.5m and initial conditions of θ(0) = π/4, θ'(0) = 0.

Deliverables

Submit your results in the form of a technical memo[1]:
- Begin with the standard memo heading: *Date*, *To*, *From*, and *Subject*.
- Include section headings as needed (in bold, underlined text). Your memo must address the topics listed in the "Solution Format" handout previously provided.
- The *Results* section should include the graphs of θ(t) and of the phase space (dθ/dt vs θ) for cases a-f listed above for 0<t<20 s.
- Validate your program by
    a) applying your engineering judgment to the solution of cases a and b.
    b) repeat the solution of case c, using a smaller time step. Compare the two solutions graphically.
- Include a *commented* copy of your program separately at the end of the memo report (as an "Attachment").
- If bibliographic references are required, you may use the footnote method as applied in this handout.

Deadlines

You will demonstrate your working program to Dr. XXX in ME372 lab during the 10th week of the quarter. Your memo report is due to Dr. YYY in class on Tuesday of 10th week.

---

[1] *A Guide to Writing as an Engineer, 2nd ed.*, D. Beer and D. McMurrey, Wiley, 2005. See pp. 82-84