ChE's Teaching Introductory Computing to ChE Students -- A Modern Computing Course with Emphasis on Problem Solving and Programming

David E. Clough

Department of Chemical Engineering
University of Colorado
Boulder, CO 80309-0424

Abstract

An easy recipe for fomenting heated debate among ChE faculty is to inject the topic of introductory computing for ChE students into a discussion. Most faculty will have strong opinions that are only muted by the threat of a teaching assignment to such a course. There are many questions: Who should teach introductory computing to our students? Faculty from computer science, a general engineering department, the ChE department, or others? What should be taught? Traditional programming with Fortran, object-oriented programming with C++, problem solving with tools such as Excel and Mathcad, or various mixtures?

In ChE at the University of Colorado, we are no different from many other institutions in that these debates have raged on for decades and continue today. In fact, over the years, our students have taken courses in all the various categories mentioned above. Recently, however, we have settled on a scheme and a course design that is working particularly well and bears consideration by others.

In engineering here, introductory computing is taught under an umbrella course number (GEEN 1300 Introduction to Engineering Computing, 3 credit hours). The various engineering degree programs (chemical, mechanical, civil, architectural, environmental, aerospace) each teach a section of this course, and these sections take on "flavors" according to the preferences of the particular program. Students in electrical engineering, computer engineering, and computer science do not take this course, rather a typical "CS101" course based on C/C++. A significant fraction of the entering students, typically 30%, are "open option," not having declared an engineering major. These students are included in the sections of the GEEN 1300 based on their interest in and leanings toward an engineering major. Also, there has occasionally been an additional section of the course for "open option" students and students not yet in the College of Engineering.

Two years ago, ChE at Colorado initiated a change in this course, taking it away from its traditional Fortran/Excel base. In this transition, two central themes were preserved: scientific/engineering problem solving and structured programming. The new course is divided into four roughly-equal parts. First comes a segment on engineering problem solving using the

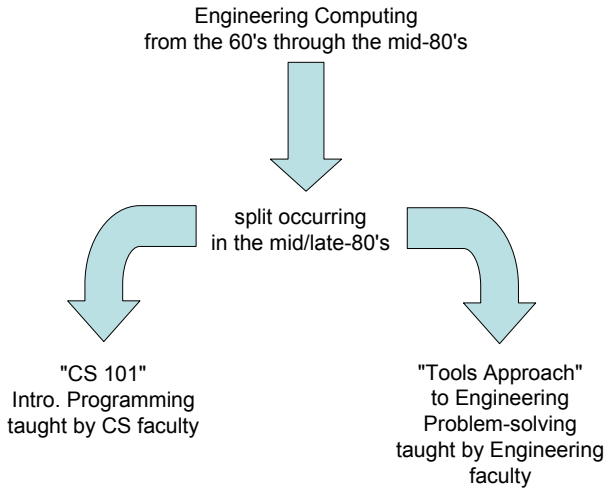
Mathcad package. This is followed by a segment on engineering problem solving and elementary numerical methods with Excel. The third segment expands the use of Excel by introducing structured programming via its Visual Basic for Applications (VBA) language. And the final segment continues the themes with the Matlab package along with an introduction to vector/matrix calculations. Problems from chemistry, physics, engineering and ChE are used throughout.

There are some pedagogical keys to the success of this course. The combined use of Excel and VBA has a strong supporting case. Providing students with knowledge and skills that they can use immediately, during the same semester, in other courses and activities is important to student motivation. Providing a gateway to subsequent use of the software tools and, for some students, to building their computing knowledge in follow-on courses completes the picture. This paper and presentation will provide details of the course design, its evolution, and its evaluation by students and external examiners.

Evolution of engineering computing – from "Slide Rule 100" to "CS 101"

From the 1950's through the 1960's in the US, traditional introductory engineering courses that focused on calculations with slide rules and log tables and graphical representation of information were modified to include scientific numerical computer programming. The vehicle for the latter was Fortran programming on mainframe computers using punched cards. The computing component of these courses grew through the 70's with more attention paid to numerical methods. The minicomputer was a common vehicle, and video terminals gradually replaced the use of punched cards. In the 1980's technology shifted to personal computers with magnetic storage media. Although there were "experiments" or "movements" with different programming languages during the 70's and 80's, such as Pascal, Fortran, in its evolving versions, remained the primary software vehicle.

From the mid-80's forward through the 90's, a split occurred in the direction taken toward introductory computing for engineers. In part, this was catalyzed by the growth of computer science programs and the computer science faculty taking over this instructional role. This is illustrated in the figure below.



Both branches are prevalent today. The "CS 101" courses have migrated through several programming languages: Pascal, C, C++, and Java. In the engineering branch, software vehicles such as spreadsheets (first Lotus 123, then Quattro Pro, and now Excel), TK Solver, Mathcad, and Matlab have gradually pushed out programming languages, primarily Fortran, which are becoming endangered species in these courses.

The "CS 101" branch would claim a number of reasons for existence:

- engineers should learn fundamental concepts of programming and computer science
- computing should be taught by computer scientists, not engineers
- engineering faculty are not interested in teaching computing to their students
- these courses provide a significant chunk of student credit hours (SCH) and budgetary resources

as would the "Tools Approach" branch:

- engineering students need a solid grounding in problem solving with modern computing tools
- engineering students need the knowledge and tools required in their professions
- engineering computing and problem-solving are best taught by engineers
- these courses provide a significant chunk of student credit hours (SCH) and budgetary resources

A real assessment reveals that the two branches are complementary and that many/most engineering students can benefit from both courses, although most curricula are found to be too congested to make room for both. That raises the question: *Is each branch at fault more for what it leaves out than what it includes?*

Student needs – a balancing act

Having participated in discussions on introductory computing with engineering faculty over the past 35 years, I am concerned with arguments put forth over and over again. When it comes to computing needs, faculty often confuse what is important for their students with what is important for themselves. Faculty needs, more often than not, align with their research interests and activities, and these are distant, if not disconnected, from the needs of their undergraduate students. Also, faculty often carry an impression of what the needs of professionals are that is off base, by being either out of date or out of touch. Few discussions proceed on the basis of evidence from alumni and employer surveys. Finally, computing is not part of the daily professional existence of most faculty and is not expected to be. Their computing skills are oxidized, and most of their computing is carried out in the trenches by their students. This is, perhaps, part of the more general dilemma of faculty preparing students for a profession that many/most faculty themselves have not practiced.

Even given this troubling backdrop of faculty perspective, an encompassing view of student needs includes several areas that compete for their slice of the instructional pie:

- fundamental knowledge of computing, programming and computers
- awareness of and preparation in emerging aspects of computing
- computing requirements in the other courses of their curriculum
- knowledge and skills required by engineers in their day-to-day professional lives
- opening the door for further study and specialization in computing and computer science

The argument for fundamental knowledge is sound. Such knowledge in computing will transcend the skills and tools of the day. Fundamentals provide the foundation. A criticism is that fundamentals tend to be abstract and difficult for students to grasp and appreciate. Most students learn better inductively, generalizing fundamentals from specific and practical exercises and examples.

Given the rapid rate of change in computing, teaching only the state of the art in the profession will leave students out of date by the time they get there. Therefore, educating them in emerging trends is important. A problem with this is judging which trends will stick and which will be flashes in the pan.

Students will appreciate and be motivated by the acquisition of skills that they can put to immediate use, even during the semester in which they are taking the introductory computing course. Of course, focusing too much on immediate needs may miss the mark when it comes to professional needs. Many computing tools used in the curriculum satisfy learning objectives but are of little use in professional life.

Teaching students in the context of computing vehicles used by practicing professionals has attractive payouts down the road. Focusing on the day-to-day problem solving activities of engineering professionals has high relevance and importance. However, there are difficulties. The learning curve for some software packages is far too steep, and some packages have a knowledge prerequisite, especially in mathematics, that far outstrips the abilities of first-year students.

There is also a significant problem with using tools that automate tasks, where the user has little view of the internal operations of the task – the *black-box* syndrome. This works fine and is greatly appreciated by the professional that already has knowledge of and experience with the task being performed. But there is a great temptation for mindless button-pushing (or mouse-clicking) by students who should be learning what is behind the button. Also, there is the danger of becoming trapped by the built-in capabilities of one's software, in other words, being incapable of extending the software capabilities through programming.

A few students will want to specialize in computing. For example, some students will become software developers in the context of engineering applications. These individuals will require more education in computing, perhaps a minor or even a double degree. The introductory computing course in engineering should not attempt to redirect these students away from computer science; rather, it should open the door.
Since valid arguments can be made for the five areas of need listed above, it becomes a challenge to design an introductory computing experience that balances and, at least in part, satisfies the needs. We have attempted to meet that challenge at the University of Colorado.

Course objectives & outline – narrowing the focus

In the mid-80's, at the University of Colorado, the split in courses between "introductory engineering computing" and "introductory computer science" took place. Since then, two courses have predominated

- GEEN 1300 Introduction to Engineering Computing (3 semester credit hours)
- CSCI 1300 Computer Science 1: Programming (4 semester credit hours)

The engineering computing course is taken by all chemical, civil, environmental and mechanical engineering students. The computer programming course is taken by all electrical engineering and computer science students. Many students take both courses – the courses are considered to be complementary. Computer Science offers a second introductory course

- CSCI 2270 Computer Science 2: Data Structures (4 semester credit hours)

This course is also taken by many engineering students wishing to specialize in computing.

The introductory engineering computing course has been established with a set of objectives that seek to address the needs described above. These are:

1. Problem Solving
   - Apply the "engineering method" to the solution of quantitative problems
   - Evaluate engineering formulas, carrying units and appropriate precision through calculations
   - Practice working in groups to tackle larger-scale engineering problems

2. Symbolic Computing
   - Enter and edit symbolic expressions in computer software
   - Manipulate and solve algebraic expressions
   - Carry out symbolic manipulations for calculus

3. Spreadsheet Techniques
   - Develop efficient spreadsheet skills
   - Set up and interpret "what-if" and case study scenarios
   - Organize and layout spreadsheet solutions to engineering problems

4. Programming Fundamentals
   - Learn how information is represented by different data types
   - Learn program-flow algorithm structure and modularity
   - Program with object-oriented features

5. Elementary Numerical and Statistical Methods
   - Develop the ability to solve single nonlinear algebraic equations using elementary numerical methods, such as bisection, false position or Newton's method
   - Solve sets of linear and nonlinear algebraic equations
   - Carry out regression calculations

6. Software Tools
   - Develop skills with and knowledge of the following software tools:
     Mathcad 2001
     Excel 2000 & Visual Basic for Applications (VBA)
     Matlab 6

The "Problem Solving" objective is a carryover from the old "slide rule" courses. Most entering students lack practice and abilities in numeric problem solving. Many of the lessons from the old courses still have much value and prepare students for the activities in their other courses, in particular, the ChE material & energy balances course. Achieving this objective has an obvious beneficial long-term impact.

"Symbolic Computing" is of immediate utility in the students' math courses. They also make use of this in their science and engineering courses. It is common for students to check their manual work using the computer. A byproduct lesson learned is that not all equations or systems of equations have analytical solutions [not obvious to many freshmen].

"Spreadsheet Techniques" provides a problem-solving methodology that has the broadest and longest impact of any objective in the course. Excel is the day-to-day problem solving tool of most practicing ChE's, and it is the software tool used most frequently by ChE students. Spreadsheet methods are becoming recognized in their own right along with the need to teach them separately to ChE students. The author's AIChE short course in spreadsheet problem-solving has been one of the most frequently offered courses (approaching 100 offerings) over the past dozen years.

"Programming Fundamentals" represents the *lost objective* in many engineering computing courses. It has been retained in this course in a creative way. The fundamentals of structured, algorithmic programming and data structure are introduced via the VBA[1] language within Excel. This provides a natural setting for students to "elevate" prototypes developed on the spreadsheet into more elegant and efficient VBA macros (Subs) and user-defined functions. The portability of the programming concepts is further emphasized by learning the m-script language in Matlab. Achieving this objective opens the door for students in two important ways:

- students who move on to take the computer science courses have a leg up on students with no background in programming – our students enjoy greater success in these follow-on courses

[1] VBA: Visual Basic for Applications, a full-featured, object-oriented programming language with structure superior to C/C++ and Fortran 95.

- programming opens the door to extension of many software packages – without the ability to program, users are forever limited to the conventional, built-in capabilities of the packages

There are certain numerical and statistical methods that represent the bread-and-butter of applications in engineering, both in the academic curriculum and in practice. Several of these are within reach of entering students, and these are represented in the "Elementary Numerical and Statistical Methods" objective. Apart from the practical relevance of equation-solving and regression methods, the lessons learned through a disciplined approach to Gaussian elimination are of great general value.

Our students are exposed to a great variety of software tools during their undergraduate ChE curriculum [Word, Powerpoint, Excel, Mathcad, Matlab, Mathematica, Simulink, Polymath, EZ-Solve, HYSYS, Aspen+, Minitab, Control Station, LabView, LadSim, AutoCAD, to name a few]. To achieve the "Software Tools" objective, we choose to expose the students to several software tools that will be of general utility, teach portable concepts, be accessible and easily acquired, be relevant to their other courses and/or to professional practice. Additionally, we need to prepare the students for the onslaught of the other software packages they will encounter. They obviously need to become skilled at picking up new tools quickly.

The course objectives are embodied in a course outline as follows:

Topic	No. of Lectures	No. of Labs
Introduction, problem solving, MathCAD	6	3
Spreadsheet problem solving, Excel	7	4
Introduction to programming, VBA	7	4
Numerical methods, Matlab	7	4
	27	15

The deliverables of the course are summarized in the grading policy table below.

	Points	*x*	*Weight*	=	*Total*	*Pct.*
Project Assignments [9, 1 double]	100		5		500	36.5
Homeworks [4]	40		2		80	5.8
Pop Quizzes [9]	90		1		90	6.6
Lab Sessions [15]	150		2		300	21.9
Midterm Examinations [2]	200		1		200	14.6
Final Examination	100		2		200	14.6
					1300	100.0

Students prefer to have multiple measures of their performance in the course, as opposed to putting all their eggs in a couple baskets. This allows some compromise for students who perform well in an exam setting to those who work better on project assignments.

Course design – practical considerations

The introductory engineering computing course is taught in several sections that align with the engineering disciplines and the instructors are faculty from those disciplines. Consequently, there is a limited emphasis on examples and problem solving within the particular discipline of the instructor and section. However, the course sections are similar enough that students can cross over sections. This is also important for engineering students who have not declared a specific major yet.

Most believe that a "learn by doing" approach must be an integral part to an introductory computing course. The GEEN 1300 course incorporates a lab component, replacing a 1-hour lecture meeting with a 2-hour workshop in a computer laboratory. The workshops are tutorial in nature with some open-ended, exploratory content. Lab sections are mentored by upper-class undergraduate students who were successful in the course as freshmen. Experience over the years has taught us that this works better than instruction by graduate student TAs, many of whom come to us with varied and limited computing experience.

Outside work is dominated by weekly computing projects that require deliverables of computer files and written reports. There are frequent in-class demonstrations in lieu of conventional lecture. All lecture materials are available to students before class time as Powerpoint files. There are frequent, in-class, "pop" quizzes, two 1-hour midterm examinations, and a 2-1/2-hour final examination.

Nearly all students have their own computers in their dorm rooms, and, although University computer labs are available in many locations for their use around the clock, students prefer to do their work on their own computers. For about 50% of the course, this need is easily answered by Excel, a standard package on their computers. Many students elect to acquire Mathcad for a student price of about $125. Fewer choose to buy the student edition of Matlab, although many do this later in their academic careers when the software package comes into more frequent use.

From our alumni and employer surveys, we find that Mathcad and Matlab are not generally available to practicing ChE's. Of course, Excel is available to all. So, the former packages answer mainly educational and academic needs.

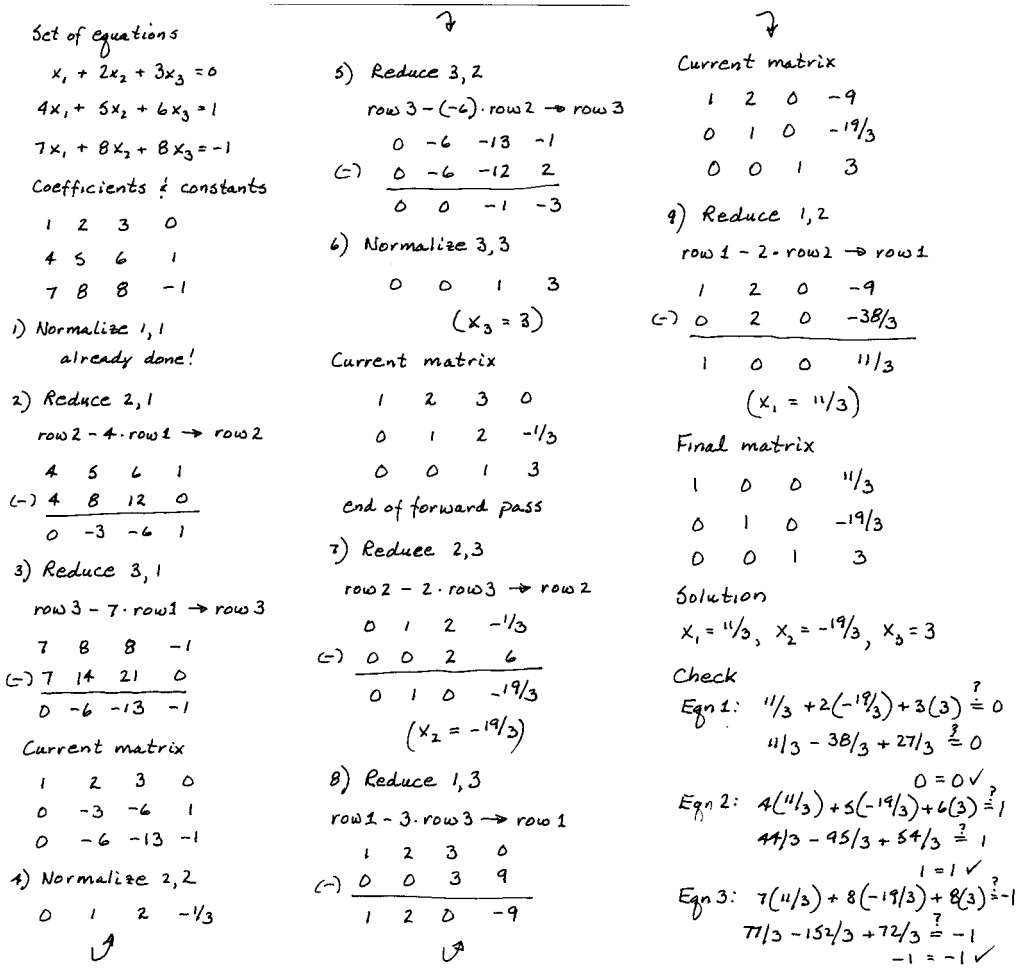
An example of the pedagogical approach used in the course

The engineering computing course in ChE at Colorado introduces students to various bread-and-butter numerical methods. This is done in a "crawl-then-walk-then-run" fashion where students first solve problems with pencil and paper ("crawl"), then use the spreadsheet environment ("walk"), then program the method in VBA and later in Matlab ("run"), and finally use the "black box" capabilities of the software packages (Mathcad, Excel & Matlab). This approach has two benefits:

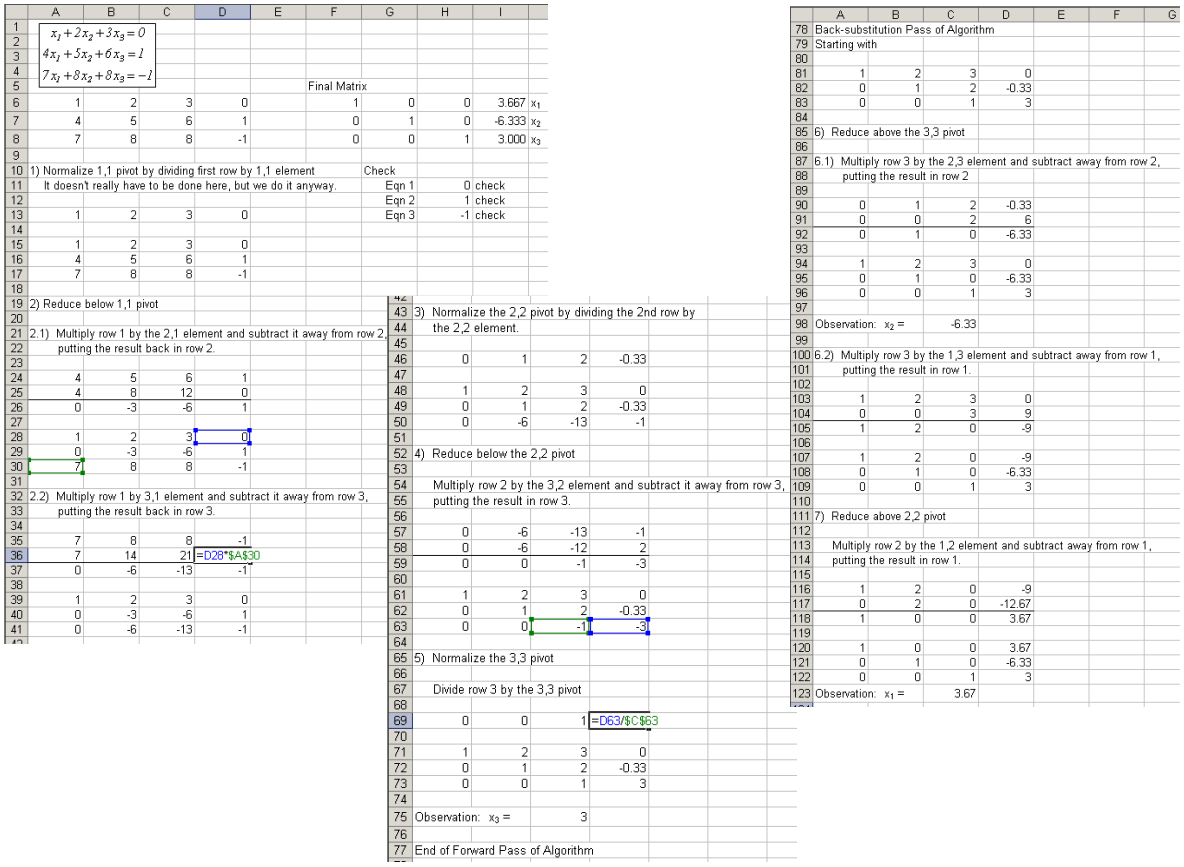
- students gain an appreciation and understanding of the method and its limitations that is not possible by merely pushing the buttons of the "black box" software features, and
- students learn the discipline required to understand and carry out an algorithmic numerical method.

Methods taught include equation solving (single nonlinear equations, methods such as bisection, false position and Newton's), solving sets of linear algebraic equations via Gaussian elimination, and linear regression. We will illustrate the approach here with the Gaussian elimination method.

For a simple set of three linear algebraic equations in three unknowns, students are taught the naïve Gaussian elimination method. They then must practice the method by hand and complete an in-class "pop" quiz to test their knowledge. See the figure below.

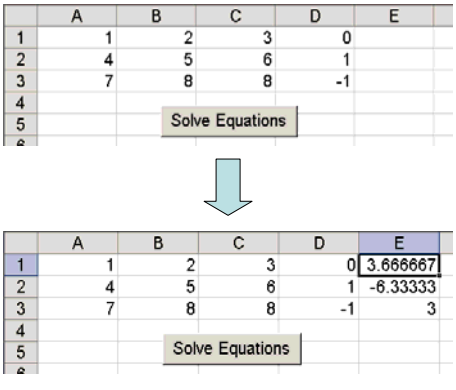


Following the manual practice, students implement the procedure on an Excel spreadsheet with formulas as shown below.



Later, students "elevate" their spreadsheet solution into VBA within Excel in the form of a macro (Sub). This includes flow-charting the Gaussian elimination method first. The VBA code is shown in the figure below.

```
Option Explicit
Option Base 1
Sub Gauss()
  Dim Amat() As Double, bvec() As Double
  Dim i As Integer, j As Integer, k As Integer
  Dim n As Integer
  n = Range("A").Rows.Count
  ReDim Amat(n, n) As Double
  ReDim bvec(n) As Double
  For i = 1 To n
    For j = 1 To n
      Amat(i, j) = Application.WorksheetFunction.Index(Range("A"), i, j)
    Next j
  bvec(i) = Application.WorksheetFunction.Index(Range("b"), i, 1)
  Next i
  For i = 1 To n
    For j = i + 1 To n
      Amat(i, j) = Amat(i, j) / Amat(i, i)
    Next j
    bvec(i) = bvec(i) / Amat(i, i)
    For k = i + 1 To n
      For j = i + 1 To n
        Amat(k, j) = Amat(k, j) - Amat(i, j) * Amat(k, i)
      Next j
      bvec(k) = bvec(k) - bvec(i) * Amat(k, i)
    Next k
  Next i
  For i = n To 2 Step -1
    For j = i - 1 To 1 Step -1
      bvec(j) = bvec(j) - Amat(j, i) * bvec(i)
    Next j
  Next i
  Range("b").Select
  ActiveCell.Offset(0, 1).Select
  For i = 1 To n
    ActiveCell.Offset(i - 1, 0).Value = bvec(i)
  Next i
End Sub
```



Subsequently, by testing another set of equations, students "discover" the pivoting limitation with the naïve Gaussian method and they implement a partial pivoting feature. This introduces them to iterative refinement of numerical methods. During the Matlab segment of the course, they translate their VBA code to a Matlab m-script, which is shown below.

```
function soln = GaussElim(Amat,bvec)
n = length(bvec);
eps = 1.e-6;
sing = 0;
for i = 1:n
   [Amat bvec] = Pivot(Amat,bvec,n,i);
   if abs(Amat(i,i))<eps
      sing = 1;
      break
   end
   Amat(i,i+1:n)=Amat(i,i+1:n)/Amat(i,i);
   bvec(i) = bvec(i)/Amat(i,i);
   for k = i+1:n
      Amat(k,i+1:n) = Amat(k,i+1:n) - Amat(i,i+1:n) .* Amat(k,i);
      bvec(k) = bvec(k)-bvec(i)*Amat(k,i);
   end
end
if sing == 0
   for i = n:-1:2
      bvec(1:i-1) = bvec(1:i-1) - bvec(i) * Amat(1:i-1,i);
   end
   soln = bvec;
else
   disp('system of equations is singular')
end
```

```
function [x,y] = Swap(a,b)
x=b;
y=a;
```

Finally, students use Mathcad, Excel, and Matlab to solve linear equations in a more streamlined, "black box" fashion. Examples of these solutions are shown below.

Mathcad solution of linear algebraic equations

$$A := \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 8 \end{pmatrix} \qquad b := \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix}$$

$$x := \text{lsolve}(A, b) \qquad x = \begin{pmatrix} 3.667 \\ -6.333 \\ 3 \end{pmatrix}$$

	A	B	C	D	E
1	1	2	3		0
2	4	5	6		1
3	7	8	8		-1
4			A		b
5	3.667				
6	-6.333	<== mmult(minverse(A),b)			
7	3				
8					
9	Excel solution of linear equations using array functions				

Matlab solution of linear algebraic equations using matrix "left division"

```
>> A=[1 2 3; 4 5 6; 7 8 8]

A =

     1     2     3
     4     5     6
     7     8     8

>> b=[0 1 -1]'

b =

     0
     1
    -1

>> x=A\b

x =

    3.6667
   -6.3333
    3.0000
```

In a subsequent lab project, students apply these techniques to the solution of a problem in a ChE setting.

Progress report – student and external assessment

The Introduction to Engineering Computing course (GEEN 1300) was revised to its current design for the Fall 2000 semester. Student evaluations via the standard University survey have provided the following information:

Semester	No. of Responses	Course Rating	Instructor Rating	Workload Rating
Fall 2000	63	B	B+	6.2
Fall 2001	52	B	B+	6.8

Compared to the history of ratings for this and similar courses, the course and instructor ratings are above average. The average course rating is C and instructor rating is C+. Workload is rated on a 0-to-10 scale with 5 being "about right" in student minds. These ratings are the norm, more than students would like, but not cruel and inhumane.

Observations from student comments and additional surveys are summarized below:

- Students preferred Excel/VBA over Mathcad and Matlab. Mathcad finished a distant 2nd with Matlab close behind.
- In Mathcad, students liked the presentation of equations, symbolic operations, and explicit handling of units. They did not like the finicky editing and the graphics.
- In Excel/VBA, students liked the logical layout of the spreadsheet with everything organized and the results visible. Reviews on programming with VBA were mixed but tilted to the positive. Some thought that VBA could get more complicated than they would like. Students wished that Excel had symbolic capabilities.
- As for Matlab, students liked the vector/matrix capabilities, although they realized that much of this power was beyond their appreciation. Students disliked the Matlab syntax that includes "dots" for array operations. They strongly disliked the Matlab command window interface, considering it primitive when compared to Mathcad and Excel. They really liked Matlab's 3D plotting capabilities. They disliked Matlab's C-like loop structures, seeing them as inferior to those of VBA.
- Over 80% of the students used Excel/VBA in one or more other courses during the same semester of the computing course. About 40% of the students used Mathcad in another course. Only about 10% of the students used Matlab outside of the computing course. Such use was generally spontaneous and not required in these other courses. Students appreciated the immediate impact of their learning.
- In comparison to their other freshman-level courses (calculus, chemistry, physics, etc.), students felt strongly that they learned significantly more in the computing course.

As part of the ABET 2000 process, the course was evaluated by the external Advisory Committee of the Department of Chemical Engineering. Their findings are summarized in the Department's annual ABET report for academic year 2000-2001:

GEEN-1300 – Introduction to Computing
(ABET Score 3.63/4.00; Learning Goals 4.00/4.00))
Reviewers: Ann Butchello, Dynegy Midstream Services, Inc.
Vern Norviel, Affymetrix
Dan Schwartz & Dhinakar Kompala, CU Faculty
Hiwot Molla & Matt Zimmerman, CU Students

1. Good introduction to Excel, which is used extensively after that.
2. Homework requires explanation of approach – not just an answer (Great!)
3. Excellent introduction to concept of engineering and how to think like an engineer.
4. Not clear how much communication or teamwork is used in this course (consider removing this ABET Program Element from the ABET matrix)

The 4th comment above is being addressed in adjustments to the course.

A longer-term evaluation of the course awaits the passage of time.

Should ChE's teach computing to ChE students?

In short, we would claim a resounding "YES!" The benefits are both direct and indirect.

Direct benefits:

- students learn computing skills that are on target for subsequent ChE courses and professional practice,
- students are motivated to learn by exercises and projects with a ChE (and, more generally, engineering) flavor, and
- we can challenge our students more than might be done in other courses that are watered down.

Indirect benefits:

- ChE faculty establish contact with ChE students early on in the curriculum (higher retention in the short term – stronger relations with alumni in the long term),
- ChE faculty are in touch with the academic needs of their entering students, and
- ChE faculty teaching (and assessing) the course provides a tighter and more effective feedback loop vis-à-vis the ABET 2000 continuous improvement process.

ChE faculty from other institutions may express skepticism, claiming that they have no control over the introductory computing course provided to their students. We would suggest that they take control over it. Ultimately, the curriculum offered to ChE students should be controlled by ChE faculty. *Where there is a will, there is a way!* Others may question whether they have faculty who are qualified to teach introductory computing. Of course they have them. It is a matter of one or more faculty with the abilities walking up to the challenge. Bringing new students into the world of engineering and ChE via a course like this is a rewarding experience for a faculty member.

Bibliography

1. Clough, David E., Steven C. Chapra, and Gary S. Huvard, "A Change in Approach to Engineering Computing for Freshmen – Similar Directions at Three Dissimilar Institutions", 2001 ASEE Annual Conference, Albuquerque, NM.

2. Clough, David E., "In with the new, but not quite out with the old – Introductory Computing for Engineers at the University of Colorado", 2001 ASEE Rocky Mountain Section Annual Meeting, Salt Lake City, UT.

3. Weimer, Alan W., ABET Annual Report – Academic Year 2000-2001, Department of Chemical Engineering, University of Colorado, Boulder, CO.

4. Student Course Evaluation results, GEEN 1300 Introduction to Engineering Computing, University of Colorado, Boulder, CO, http://www.colorado.edu/pba/fcq/by_coll/en/geen.html.

Author Information

DAVID E. CLOUGH

Dave Clough is Professor of Chemical Engineering at the University of Colorado and has been on the faculty there since 1975. His research interests are centered on the optimization and control of chemical processes. He teaches process control, applied statistics, and introductory computing. He lives in a log home in the Rocky Mountains. Email: David.Clough@Colorado.edu