Int. J. Engng Ed. Vol. 26, No. 3, pp. 593–601, 2010
Printed in Great Britain.

0949-149X/91 $3.00+0.00

Programming Skills in Graduate Engineering Classes: Students from Disparate Disciplines and Eras*

S. RATNAJEEVAN H. HOOLE
Department of Engineering and Science, Rensselaer Polytechnic Institute, Hartford, CT 06120, USA.
E-mail: HooleR@RPI.edu

In many graduate engineering classes there are often present students who meet prerequisites in programming but are rusty because it was a long time ago or they learnt an outdated language. In courses such as mathematical methods, finite elements, and power engineering there are students from different engineering disciplines; electrical engineers having programming as an ABET accreditation requirement, others not. Recent graduates are more sophisticated programmers than older ones. The problem is to bring the students up to competency without vitiating the content of the course to be taught so as to enhance it with modern computational methods. This paper describes an initial two-week MATLAB-based module on matrix equation solution that was used in four courses over five semesters at Rensselaer Polytechnic Institute. This module was within the scope of all four courses and, given MATLAB's ease of use and the students' mature standing as graduate students with resources among fellow students, it was used successfully (as a low-level programming language rather than a simulation tool as in much of the literature) to train even those who had never programmed before. Programming was thereupon used to enhance the engineering courses through computational exercises and in the process refine students' new programming skills further. A survey confirms the benefits.

Keywords: programming; prerequisites; programming language; education; matrix computation; MATLAB; accreditation; ABET

1. ABET, PROGRAMMING AND DISPARATE STUDENTS

FOUNDED IN 1824, Rensselaer Polytechnic Institute is the oldest degree-granting technological university in North America. This writer in teaching the four graduate engineering courses Introduction to the Finite Element Method, Advanced Engineering Mathematics I and II, and Power, Generation and Control [1] at RPI, had to keep in mind that a well-taught course has to treat many numerical topics to be useful in modern engineering practice. To this end in these four courses, referred to as Mathematics, Finite Elements and Power hereafter, computer proficiency is indeed of the essence.

Moreover, many overenthusiastic universities (e.g. Drexel University's College of Engineering) routinely ask students in their course evaluations for their knowledge of specific topics in computer technology such as object oriented programming before and after the course even when the course does not allow those topics. Because these survey results are made publicly available they can ruin the reputation of an instructor in say electric machinery in the eyes of the lay-public which would be unaware that it is difficult to teach object oriented programming in that course. In addition to pedagogic imperatives therefore, the use of computer technology in class becomes necessary for an instructor's reputation and survival as well.

Imparting computational content to graduate courses, however, is not always simple. A graduate or senior/graduate course is very unlike an undergraduate course where students have similar academic backgrounds and belong essentially to the same age cohort. In a graduate course, especially when many working professionals are present, the backgrounds can vary widely. In engineering mathematics courses there can be science students with engineering students; and full-time, unmarried graduate students with plenty of flexible hours for coursework, together with working professionals pressed for time. Indeed, in the cross-disciplinary engineering subjects under discussion, the class disparities can be even wider because students are from different undergraduate streams with very different accreditation requirements. Even in electric power engineering courses, it is now common to see mechanical and chemical engineers pursuing a graduate degree in nuclear power engineering, sitting together with electrical engineers.

This problem is common worldwide as curricula are increasingly standardized through America's

* Accepted 13 December 2009.

ABET [2] working with the international Washington Accords [3]. Programming has been a required course in undergraduate engineering accreditation only in recent times; indeed, many master's students in engineering have been out of college for some years and have graduated without any programming background. Even today, the criteria laid down by ABET are not explicit on the requirements in computation nor are they uniform across sub-disciplines. For instance, ABET requires that electrical engineers have knowledge of computer science to analyze and design complex software, and systems containing software components [2, p. 10]. No such description lies in the requirements for mechanical engineers [2, p. 15] and others. If a mechanical engineer did have programming skills as some in the graduate courses mentioned did, it would be despite the ABET guidelines and to the credit of farsighted universities that put their students through programming. Ahlstrom and Christie [4] have also grappled with this disparity in Swedish programs between international and local students.

Reliance on prerequisites is not a solution because many do meet prerequisites but because of age have forgotten the material or learnt programming with languages no longer common. The offer of a short 1-credit course is a solution. Indeed, a full course on MATLAB has been offered in Sweden as a freshman course for the purpose of making students MATLAB-proficient so that they may use it in later courses [5]. However, that was not for addressing the problem of class disparities but as part of the normal curriculum. Moreover, as a solution, MATLAB as a programming course would crowd the curriculum in introducing more structured programming later, as with Java. Besides, at graduate level most students would be reluctant to take it because it would not count for graduate credit and there would be problems with employers paying for it. Another solution, slightly different courses for different students, is not acceptable to university administrations because of costs.

Keeping up standards is a challenge in these circumstances. This paper describes how the four courses mentioned with very disparate students were taught without diluting them and indeed enhancing them through computational assignments by first bringing students to speed in low-level programming. By low-level programming is meant the use only of arithmetic and logical operations (+, -, *, /, <=, &&, ~= etc.), simple functions as available on a scientific calculator (like sin(x), ln (x), x^y) and simple programming commands such as *for*, *if*, *while*, and *function* combined with file reading commands. Higher level functions, such as for examining student-computed results through contour plots and graphing, would be permitted at the beginning and, time permitting, replaced with students' own programs.

2. EARLY DECISIONS: MATLAB AND MATRIX COMPUTATION

At the very outset it was decided that no student would go through these courses without programming proficiency as an outcome. It was further decided to use MATLAB® [6] which is now used so extensively in engineering education that special issues of journals are devoted to the subject [7].

The option of open source software was not considered because cheap student versions of MATLAB are available on Windows—making it "available in almost all the technical institutions" [8]—and open source software often runs on Unix-like environments that would make students grapple with the operating system more than with the subject they need to learn [9]. MATLAB's simulation capabilities are also an asset to students in verifying solutions they arrive at through low-level programming. Moreover, there are many Internet-based resources for students who need help with MATLAB.

MATLAB has been correctly described by its vendor [10] as "a high-level language and interactive environment that enables you to perform computationally intensive tasks faster than with traditional programming languages such as C, C++, and FORTRAN." This accords with the independent findings of Fangohr [11]. Wirth and Kovesi [12] further state that MATLAB can be used "for teaching the fundamental constructs of programming languages to engineering and science students" in 50% of the time as with other languages. Without making any claims for MATLAB as a well-structured programming language, it is used here as a language with which a novice can become a functional programmer quickly.

Commercial simulation software is used by many instructors—for example, COMSOL for treating differential equations. Their pedagogic value is in obtaining the solution of large problems quickly, teaching problem set-up including conditions for uniqueness and well-posedness, and through graphics-based output giving students an intuitive grasp of the system and its behavior. However, such commercial simulation software was not considered for two reasons. First, it is difficult to get sufficiently many instructors to ask the university to acquire a license unlike MATLAB which has a large support base. And secondly, and perhaps more importantly, there is relatively little pedagogic or intellectual value in asking students to use someone else's code, compared to creating their own as they would with low-level programming.

3. A LITERATURE CONTEXT

The constraint was to teach low-level programming quickly using MATLAB, bringing the students to competency and speed without diluting the course at hand and indeed enhancing it

through computational assignments. The literature is rich with contributions on the use of MATLAB in engineering education; for example [4, 5, 7, 8, 11–21]. In fact, while the ISI Web of Knowledge on 29 November 2009 listed 15,452 papers on MATLAB, from the top end of engineering educational journals, it listed 58 papers from the *Int. J. Eng. Educ.*, 39 from the *Int. J. Elect. Eng. Educ.,* 55 from *IEEE Trans. Educ.* and 47 from *Comp. Applics. Eng. Educ.* (but curiously none from *ASEE Eng. Educ.*). The vast majority of papers lie outside these educational journals and use MATLAB for analysis in research; there are only occasional papers outside these journals such as [20] that use it for education. It is therefore appropriate to say how this work is distinguished from that most useful, vast literature in the four educational journals that are of relevance here.

In nearly all the papers from the four education journals where it is treated, MATLAB is used as a simulation tool to demonstrate the behavior of engineering systems; examples are [16–19]. These often enhance the learning experience by providing graphical simulations of results, patching together MATLAB's toolboxes. The programming is not independently by the students as libraries like Simulink and LabView are combined with the simulation capabilities of MATLAB; low-level MATLAB programming plays a minor role in piecing the libraries together. Ref. [21] is distinguished in that the instructors use MATLAB programming at a low level to build their own simulations and then use them with students, rather than encouraging students to develop the tools as is done here.

Papers such as [11] and [12] use MATLAB as a programming language dedicated to teaching a full programming course with MATLAB as the only language of programming. In this paper, however, the ease of using MATLAB is exploited to teach the rudiments of programming to students so as to make them functional programmers who are able to implement computational assignments in other engineering courses. Little attention is paid to formal programming concepts like object orientation, inheritance etc. and the focus is on the steps of computation and getting them done. Another genre of papers such as [19] simply uses the power of MATLAB as a programming tool for research with potential applications in education.

In another segment of the literature such as [13, 14], MATLAB is used to give valuable insight to students by exposing them to projects where they use MATLAB to solve large engineering problems through low-level programming. This experience is not uniform for an entire class and applies only to the very few students who choose a particular project using MATLAB programming because of their interest and skills [13]. In [14], which has aspects similar to this effort, although an entire class is involved, each small group has a different project; little programming is required except for a few lines of MATLAB code given by the instructors because the students are freshmen who have done no programming yet.

This paper builds on [4] and [15], the two papers that have weak conceptual similarities to this work. In [4] the obstacles to teaching because of the disparate MATLAB skills in class among Swedish and foreign students, are overcome by giving prewritten MATLAB code so that students could focus on the results. In [15] MATLAB is used to enhance an engineering mechanics course and programming "mastery is achieved incrementally throughout the semester." But the difference is that programming in that paper is at a much higher level employing the toolboxes of MATLAB such as ODE solvers and visualization tools, any lines of code by students being for calling these tools.

The emphasis in this paper, so as to widen the courses undertaken beyond small, unrealistic problems by hand, is to make students program the computational assignments at a low level and learn to generalize the steps of their calculations as formal algorithms. This would be as appropriate for students at a reputationally top-end university from where they would then go on to the world of work building simulation software for others to use, rather than being users of other people's packages. Certainly it would not be intellectually appropriate in a university course, especially at graduate level as here, to be focused on running simulations or feed a matrix into the high-level MATLAB function *eig* and then say these are its eigen values. Such an exercise is bereft of anything cerebral. What is done in the engineering mathematics courses under discussion is to ask students to write low-level code using, for example, the power method to obtain the eigen values. At most the students were permitted to use the function *eig* only to verify their answers. In the finite elements course, they create a mesh, form the corresponding matrix equation and then compute the biggest eigen value and, thereupon, the smallest (i.e. dominant) eigen value. That would be a far more justifiable numerical extension of the traditional analytical treatment that this paper seeks to enhance, than running someone else's simulator. What we give up is the ability to run large and varied problems when we use commercial simulators.

4. MATHEMATICS PROGRAMMING MODULE AND DELIVERY

After some meandering in the first semester, matrix computation was chosen as the common core around which the programming skills are built using a two-week module (going into three weeks sometimes depending on the details of the accompanying lectures as appropriate for the course). The justification is that it is part of all the courses and within the authorized syllabus (directly in mathematics and as a tool in the others).

The required algorithms were discussed in class as pseudo-code. Then the early algorithms were developed on the computer together with the students in a modern lecture-cum-computer classroom. Initially as the instructor wrote out the code on a computer projected on a screen, explaining each line and what it does, the students copied it on their machines as best as they could. This process was helpful in giving them a start (rather than assigning to those who were beginners the daunting task of programming from scratch). They learnt in detail in correcting mistakes and getting their programs to run and in doing further homework. Early elements of code simply involved a Fourier series for a square wave, summing a fixed number of terms and plotting the wave (Fig. 1). This exercise, briefly recalling Fourier series, is used to teach starting MATLAB, the *for* loop and the *while* loop and plotting—a major departure from the traditional teaching approach to programming where students go from the basics. The code with corresponding algorithms used in the lectures, is given in this paper explicitly in Figs. 1-8 to share what the students worked with and for traditional teachers who, though expert in the mathematical methods they teach, fail to introduce them in their powerful numerical form because of not knowing programming or how simple it can be in MATLAB.

Thereafter determinant evaluation is taught with the use of *function* and the simple *if* statement using the code of Fig. 2. This introduces recursion in a simple but elegant way. This is a rapid introduction to recursion by normal standards but the students were graduate students.

Now that determinants can be computed, Cramer's rule is used to solve the matrix equation Ax = B using the code of Fig. 3. With these programs the computation of a matrix inverse is feasible after formal definitions in solving Ax = B following the definition of A, n and B as in Fig. 4. The simple function *SubMatSimple* easily computes the determinant of a matrix after deleting a specified row and column for the Minor and Cofactor using the MATLAB command = [].

From this point onwards only the algorithms are discussed in class without giving away any of the code. For example, a useful homework assignment testing a student's comprehensive grasp of programming, building on the *if* statement, is to replace the function SubMatSimple of Fig. 4 with SubMatDetailed without using the MATLAB command = [].

Fig. 5 gives a sample solution. This broadening is especially important for students to go on

```
% Fourier summation of 1000 terms of
%alternating square
clear all; clc; %Clears screen and memory
for k = 1:500
    FSum500(k)=0.0;%500 Frier terms in 2s
end %for
tee=0.0; %tee is time
DelT = 2/499; %Time intervals. Time in X
Count = 0;
while Count <= 499
    X(Count+1) = Count; % indices >=1
for n = 1:1000 %Play with this number
    FSum500(Count+1)= FSum500(Count+1)...
    + 40/((2*n-1)*pi)* ...
     sin((2*n-1)*pi*tee); %... = Continue
end %end for
tee=tee+DelT; %Next time step
Count=Count+1; %Index for next time step
end %while
plot(X,FSum500,'r');%Plot FSum500 Vs. X
                                  %in Red
```

Fig. 1. Plotting Fourier series for alternating square.

```
clc;clear all;%Clears screen/memory
A=[3, 1, 1, 1; 1, 3, 1, 1; 1, 1, 3, 1;
1, 1, 1, 3];
n=4; %4x4 test matrix A defined.
%Compute value by our function
Value=Determinant(A,n)
%Now confirm by MATLAB function
Value = det(A)

%Define function in file Determinant.m
function DetValue = Determinant(A,n)
if n==1
    DetValue=A(1,1);
else
    DetValue = 0.0;
    sign=-1;
    for j=1:n
       sign=-sign;
       [B,m] = SubMatSimple(A,n,1,j);
       CoFactor= Determinant(B,m);
       DetValue=DetValue+ ...
                    sign*A(1,j)*CoFactor;
    end; %Loop for j
end;%if. End of function

%Define function in file SubmatSimple.m
function [SubA,m]=SubMatSimple(A,n,i,j)
%Get matrix SubA: A without row i,col.j
m=n-1;%Size of SubA from size n of A
SubA=A;
SubA(i,:)=[];%set row i to a null array
SubA(:,j)=[];%set col. j to a null array
```

Fig. 2. Determinants: recursion.

```
clc;clear all;%Clear Screen/Memory.
%Define eqn. Ax=B
A=[4,-1,0,0;-1,4,-1,0;0,-1,4,-1;0,0,-1,4];
n=4;B=[2; 4;6;13];
Denom=Determinant(A,n);
for i=1:n
    C=A;
    for k=1:n %Replace col. i of C with B
        C(i,k)=B(k);
    end;
    x(i) = Determinant(C,n)/Denom;%Cramer
end;
x
```

Fig. 3. Cramer's rule: equation solution.

```
function [SubA,m] = SubMatSimple(A,n,i,j)
m=n-1;%Returns mXm Matrix SubA after deleting
%row i and column j of the nXn matrix A
SubA=A;
SubA(i,:) = []; %set row i to a null array
SubA(:,j) = []; %set column j to a null array
%End of function

Value= Determinant(A,n);
for i=1:n
    for j=1:n
        sign=(-1)^(i+j);%c.f func.Determinant
        [C,m]=SubMatSimple(A,n,i,j);
        SignedCofactor=sign*Determinant(C,m);
        AInverse(j,i)= SignedCofactor/Value;
    end;%nested for loop j
end;%outer for loop i
AInverse
A*AInverse %Must yield the identity matrix
x=AInverse*B
AInverse= inv(A)%This&next lines double check
x = AInverse*B  %Check: High level MATLAB
```

Fig. 4. Solution by inverse computation

```
function [B, m] = SubMatDetailed(A, n, i, j)
%Computes B = A with row i and col. j deleted

m=n-1;%Size of B
% A(ia,ja) is to go into B(ib,jb)
for ib=1:m
    if i==1 %Then row of B is next row of A
        ia=ib+1;
    elseif i==n %Then row of B=Same row of A
        ia=ib;
    else
        if ib<i  %Up to deleted row i,
                 %Row of B = Same row of A
            ia=ib;
        else %After deleted row i, rows same
            ia=ib+1;
        end;
    end;%Finished deleting row i
    for jb=1:m %Colmns:Same logic as for rows
        if j==1
            ja=jb+1;
        elseif j==n
            ja=jb;
        else
            if jb<j
                ja=jb;
            else
                ja=jb+1;
            end;
        end;
        B(ib,jb)=A(ia,ja);
    end;%Loop jb
end;%Loop ib
```

Fig. 5. Sub-matrix with our code.

independently to other languages. More elaborate solvers are now introduced. Programming niceties are avoided and distinctions between row and column vectors in MATLAB are left to the student to discover.

```
clc; clear all;%Clear screen/memory.Def. eqn.
A= [2,-0.5,0,-1.0,0,0;
 -0.5,1.5,-0.25,0,-0.75,0;0,-0.25,1,0,0,-0.5;
 -1,0,0,2.0,-0.5,0; 0,-0.75,0,-0.5,1.5,-0.25;
 0,0,-0.50,0,-0.25,1.0];
B= [50; 0; 0; 50; 0; 0];%Can "," be in place
%of ";"?
n=6; %Defined equation for testing Ax=B
x=inv(A)*B %Soltn by MATLAB functn (to check)
for i= 1:n
    x(i)=B(i)/A(i,i); %Initial guess
end;
Iter = 1; %Track iteration count for stopping
Error = 1000; %Track largest error for stppng
while (Iter<100*n && Error>0.0001)
  %Above line - 2 condns for stopping.
  %1) Iter >100n => stop because diverging.
  %2) Error>0.0001 => converged
   for i= 1:n %for each row of matrix eqn.
       maxX=0.0;%track max x-component
       maxDel=0.0;%track max. change in xcomp
       sum = B(i);
       %Sum=RHS-LHS terms except A(i,i)*x(i)
       for k= 1:n %All n terms moved right
           sum = sum-A(i,k)*x(k);
       end;
       sum =sum +A(i,i)*x(i);%Put back term i
       old = x(i);
       x(i) = sum/A(i,i)
       Del = x(i) - old;
       x(i) = old + 1.3*Del;
       change = abs(Del);
       absX = abs(x(i));
       if (maxX<absX)
           maxX=absX;
       end;
       if (maxDel<change)
           maxDel=change;
       end;
   end;  %end for loop i.
  Iter=Iter+1; %Ready for next iteration
  Error = maxDel/maxX
end; %while loop
Iter
Error
x
```

Fig. 6. Gauss Seidel iterations.

```
clc; clear all;
A=[4 0 -1 0 0 -1;0 4 0 0 -1 0;-1 0 4 -1  0 0;
0 0 -1 4 -1  -1;0 -1 0 -1 4 0; -1 0 0 -1 0 4]
n=6; x = [1;2;3;3;2;1] %The sltn s.t. RHS B:
B= A*x; %We test if given A,n,B, we can get x
%Begin Upper Triangulation of A
for i=1:n-1 %for every row i except the last
     %Make: A(i,i)=1 by scaling row eqn.
    for k=i+1:n %Note: Upp.Trngltd to row i-1
        A(i,k)=A(i,k)/A(i,i);
    end;
    B(i)= B(i)/A(i,i);
    A(i,i)=1.0; %Scaling of eqn. i done
    for ii=i+1:n %for every row ii after i
          % subtract a mltple of row i to
          % make column i of row ii 0.0
        for jj=i+1:n
            A(ii,jj)=A(ii,jj)- ...
              A(i,jj)*A(ii,i);
        end;
        B(ii)=B(ii)-B(i)*A(ii,i);
        A(ii,i)=0.0;
    end;
end;
A
B
%Done Upper Triangulation. Back-substitution:
x(n)=B(n)/A(n,n);
for i=1:n-1
    r=n-i; %Row dealt with: n-1 down to 1
    x(r)=B(r);
    for c=r+1:n %Column
        x(r)=x(r)-A(r,c)*x(c);%Note A(r,r)=1
    end;
end;
x
```

Fig. 7. Gaussian triangulation.

The Gauss-Seidel iterative method for solving Ax = B, very much a part of Mathematics I and Power, is treated as pseudo-code leading to a model solution as in Fig. 6. As necessary—certainly in Mathematics I and Finite Elements—Gaussian elimination for positive definite matrices which require no row swapping, is taught using the algorithms of Fig. 7 as pseudo-code.

Matrix solution by LU decomposition is today important in many spheres of engineering including Finite Elements and Power. For a symmetric coefficient matrix A with A = LU, U being the transpose of the lower triangular L for which case Cholesky solvers become competitive, the algorithms reflected in Fig. 8 are taught and their development as code left for homework.

5. ADJUSTMENTS AND EXPERIENCE

The module has been taught for four semesters now, almost finishing the fifth in multiple offerings over time, the courses sharing the common outcome of producing graduates competent in programming concepts and using that competency to teach the course at hand in a more useful way by incorporating computational methods. The two mathematics courses had about 28–30 students at a time and the two engineering courses 10–20. Yet the differences in courses and time required some adjustments. Mathematics I and II initially were taught alike in the first two weeks. Now in the ongoing semester (Fall 2009) there are students in Mathematics II and Finite Elements who were in Mathematics I or II taught as in this paper. So some severe adjustment were made curtailing the

```
clc;clear all;%Clear screen/memory. Def. eqn.
A=[4 0 -1 0 0 -1;0 4 0 0 -1 0;-1 0 4 -1  0 0;
 0 0 -1 4 -1  -1;0 -1 0 -1 4 0;-1 0 0 -1 0 4]
B = [0;6;8;6;3;0]; n=6;
%Factoring A = L*Transpose(L) begins
L(1,1)= sqrt(A(1,1));
for i=2:n %For row i compute L up to diagonl
    for j=1:i-1
        sum=A(i,j);
        if j>1
            for k=1:j-1
                sum=sum-L(j,k)*L(i,k);
            end;
        end;
        L(i,j)=sum/L(j,j);
    end;%Loop for j
    %Then compute L at the diagonal
    sum=A(i,i);
    for k=1:i-1
        sum=sum-L(i,k)*L(i,k);
    end;%Loop for k
    L(i,i)=sqrt(sum);
end;%Loop for i and check if L is right
L
U=L';%U is the transpose of L; U for checking
P=L*U %P must be A if properly factored-L*Ltranspose
Z(1)=B(1)/L(1,1);%Solve L*Z=B -forward elmntn
for i=2:n
    sum=B(i);
    for k=1:i-1
        sum=sum-L(i,k)*Z(k);
    end;
    Z(i)=sum/L(i,i);
end;
Q=L*Z' %Q for checking. Must match B. Why Z'?
x(n)=Z(n)/L(n,n); %Solving U*x=Z:back-subst
for i=1:n-1
    ir=n-i;
    sum=Z(ir);
    for ic=ir+1:n
        sum=sum-L(ic,ir)*x(ic);
            %L(ic,ir)=U(ir,ic)
    end;
    x(ir)=sum/L(ir,ir);
end;
x
```

Fig. 8. Solution by LU decomposition for symmetric positive definite A.

module and dealing with those few who missed this handling of mathematical programming. The Power course had added emphasis on Gauss-Seidel iterations and LU decomposition with complex arithmetic. Finite Elements emphasized LU decomposition, use of text files and graphics.

A particular student who had never programmed, came up with finite element code that though inefficient, worked. For example, instead of loops he repeated lines of similar code for each instance of the loop. In time his coding improved in efficiency and elegance. He after initially wishing to drop-out, stayed on, and became a good programmer. An important way in which the teaching here is distinguished from formal programming courses is captured by the comment of a student in the survey at the end: "Writing the code for the homework problems was not especially onerous, as the sub-routines did not necessarily have to be optimized for speed or memory usage, and thus could be very crudely written."

It was important to be flexible. When a class complained that they could not keep to the homework deadlines, extensions were readily given. Without the extensions, the students would have given up. But with the extensions, they worked hard and the outcome was as desired—they learnt the subject and became proficient programmers—judging, say, by the complexity of the programs they wrote, students, including the few who had never programmed, reached that stage where they built up a complete finite element program—reading input data from a text file using MATLAB's *textread* command, assembling the corresponding matrix equation, and then plotting the solution. Others in the power course performed steepest descent optimization for the scheduling of power generation units at optimal cost.

Even though the course emphasized low-level programming, MATLAB's tool box is useful to students in verifying the results of finite element code and ODEs. Fig. 9 (top) shows the steady-state solution by a student of the differential equation

$$\frac{d^2V}{dt^2} + 2\frac{dV}{dt} + 17V = E(t);\ \text{with } \dot{V}(0) = 4.0 \text{ and } V(0) = 0.0$$

where $E(t)$ is a periodic, half-rectified square wave, obtained by breaking up the input into Fourier components. After obtaining the solution $V(t)$ themselves, the students verified it using MATLAB's function *ode23* from the ODE toolbox (Fig. 9 Bottom). The similarity is clear after the MATLAB solution's transients die out. Similarly,

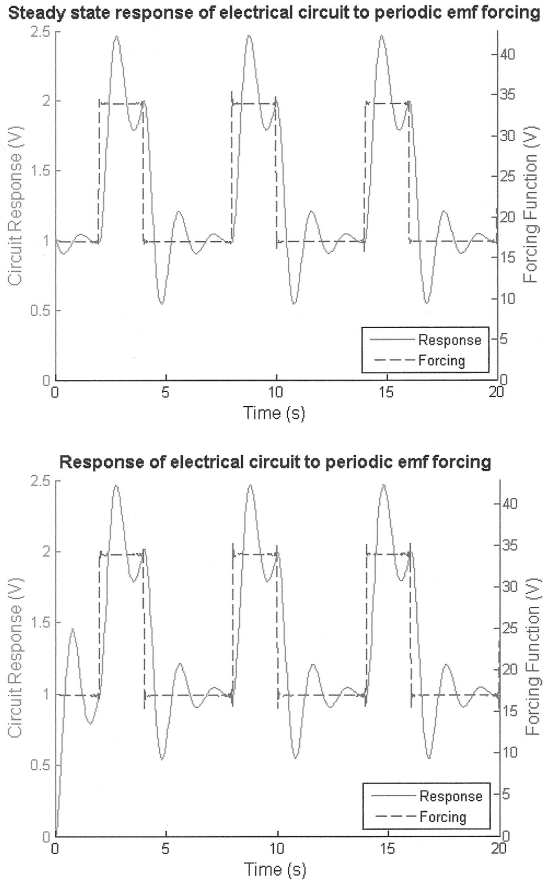

Fig. 9. Student's steady state solution to ODE with half-square wave (top) verified by full solution by MATLAB's toolbox (bottom).

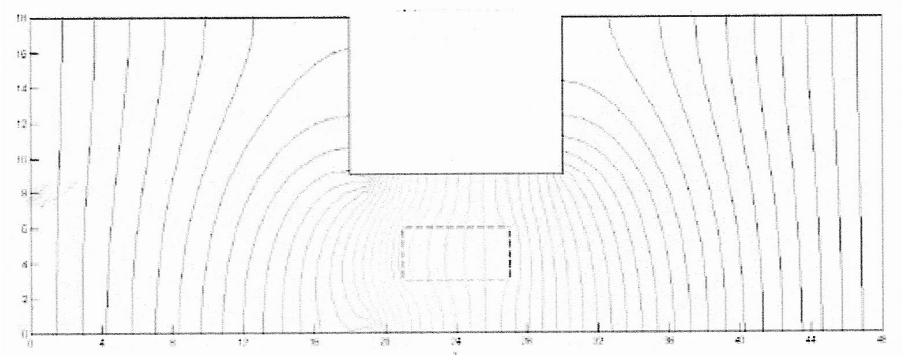

Fig. 10. Flow in a pipe with a source at a constriction.

other MATLAB toolboxes are of assistance in students' verifying their own solutions—for example, poisolv and poicalc of MATLAB's PDE toolbox when dealing with the Poisson equation in the finite elements course. Fig. 10 gives a student's finite difference solution to a Poissonian flow problem with inhomogeneities. Figs. 9 and 10 emphasize that MATLAB was used as a low-level programming tool and not as a simulator. The assignment for Fig. 9 was early in Mathematics II and for Fig. 10 at the end of that course showing the building up of program complexity as the course progressed.

What made learning low-level programming as described here possible was:

(1) the students being mature as graduate students;
(2) the mathematics of the matrix module having been already covered in all the students' undergraduate work, the module's not taking away valuable class time from the subject to be taught;
(3) the class having some students who were very familiar with programming and therefore able to help those who were behind.

These latter students who were sufficiently experienced programmers, on their own initiative, had asked for advanced assignments and engaged with the advanced work with the promise that they will be guaranteed the grade of A upon completion of the advanced work.

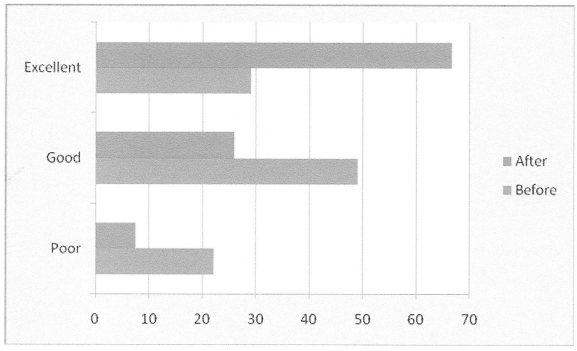

Fig. 11. Improvement in skills before and after the course.
Key: Before: Lower Bar. After: Upper Bar
Poor: 0–4/10. Good: 5–6/10; Excellent: 7–10/10.

6. STUDENT SURVEY

The students were surveyed for their perceptions (Fig. 11). A strong measure of the success achieved is that the 29% who saw themselves as excellent in programming at the beginning of the courses, by the end of the courses had risen to 67% of the classes. Twenty two percent of the students in these courses perceived themselves as poorly skilled in programming (at or below 4 on a scale of 10) at the beginning of their course. Their improvement went up from a mean of 2.62 by 3.14 to 5.76 on a scale of 10—an improvement of 120%, a most impressive gain given the shortness of the module, reinforced through course-relevant assignments. In contrast, the overall skill improvement for all students was about 27%.

When asked if they would have preferred another language or environment to MATLAB on Windows, the mean response was 2.48 on a scale of 10, confirming the correctness of the decision to go with MATLAB on Windows.

In the mathematics courses where usually the treatment is seen as drab by students, the reported value added to the course by MATLAB was 8.2/10. In the engineering courses also it was affirmative but less resounding at 6.25/10 for Finite Elements and 5.53/10 for Power. The lowest mean beginning programming skill level was in Power at 5.0/10.0 as also the highest class disparities with a standard deviation of 2.3 in this number because of the mixture of electrical, mechanical and chemical engineers in class from very varying age groups. Subsequent coursework using programming was also the most intense in this course (on load-flow, various optimization methods for scheduling generators, load-flow and contingency studies etc.). This may explain the muted enthusiasm, albeit positive.

Overall, on the scale of 10, the student recommendation for using MATLAB in future offerings was 6.93, with the math course students recommending it more strongly by about 1.0 than the engineering course students. This may be because

as engineers, they were able to appreciate the application of mathematics more through the treatment of large real problems using computation than in a traditional treatment of mathematics courses. However, the 22% who saw themselves as poor programmers answered 5.42 for recommending MATLAB for future offerings of the courses saying that it increased their workload for the course. That is, ironically, those who benefited the most, while marginally positive, were not as enthusiastic as the others because of the extra time they spent grappling with the subsequent programming assignments.

About 29% of the students saw themselves as 7 or better on a scale of 10 in programming at the beginning. These naturally reported only a small improvement (about 0.6) in this figure at the end of course. We may then say that the initial two-week module took away valuable class time from them. Yet, they recommended 8.13/10 that the module be retained in future offerings—1.2 more than the rest of the students. That is, those who were already proficient programmers, despite not having benefited through the initial two weeks, realized most the benefits of the subsequent realistic treatment of larger problems through numerical methods.

7. CONCLUSIONS

This paper has addressed the problem of teaching computational methods in advanced graduate courses with students who have different skills in programming; some refined programmers, some with no skills at all. A simple short module based on matrix algebra within the syllabus has been developed to teach programming using MATLAB as a user-friendly, but powerful, programming tool rather than as a high-level simulator. This initial module has then been used to enhance the course at hand through methods drawing on their new programming skills. Students affirm the benefit of this approach in a survey and strongly recommend the use of this module with MATLAB in future offerings of these courses.

Given the shortness of the module, the achievement is not to be compared with that from a full course on programming. The 120% skill improvement realized with the less skilled programmers in class, though important, left them as still-learning programmers. These students showed the greatest improvement; but positive though not as strong an appreciation of the programming module as the others.

The strongest programmers at the beginning appreciated most the numerical content of the courses although they did not benefit much as programmers. These suggest that the approach taken is good but that ways must be found to address the difficulties and overloading of the initially poor programmers in coping with programming assignments that were used to enhance the courses. Similarly work is required to make those already proficient as programmers at the beginning, gain something from the initial two weeks. This may be by a formalized different track for these students as was done for some students on their request.

Accreditation agencies need to take note of the mismatched criteria between the different engineering sub-disciplines and take appropriate action.

REFERENCES

1. Rensselaer Polytechnic Institute, *Rensselaer Catalog: 2009–2010*, Rensselaer Polytechnic Institute, Troy, NY. 2009. Also http://catalog.rpi.edu/ (Accessed 28 November 2009).
2. Accreditation Board for Engineering and Technology, *Criteria for Accrediting Engineering Programs*, Baltimore, MD, 3 November 2007.
3. Washington Accord Secretariat, *The Washington Accord*, Washington Accord Secretariat, Baltimore MD. 2000. http://www.engc.org.uk/international/international_agreements/washington_accord.aspx (Accessed 29 May 2009).
4. J. Ahlstrom and M. Christie, Using a MATLAB exercise to improve the teaching and learning of heat conduction during welding, *Int. J. Eng. Educ.,* **21**(5), 2005, pp. 769–777.
5. H. P. Wallin, U. Carlsson, U. Ross and K. el Gaidi, Learning MATLAB: evaluation of methods and materials for first-year engineering students, *Int. J. Eng. Educ.* **21**(4), 2005, pp. 692–701.
6. Eva Part-Enander, Anders Sjoberg (eds), Eva Pärt-Enander and Anders Sjöberg, *The Matlab V Handbook*, 2nd edn. Addison Wesley, 2000.
7. Ahmed Ibrahim, Guest Editorial: Special MATLAB and Simulink in engineering education, *Int. J. Eng. Educ.* **21**(4), 2005, p. 768 and (5), pp. 558–559.
8. M. Varadarajan and S. P. Valsan, MatPECS—A MATLAB-based power electronic circuit simulation package with GUI for effective classroom teaching, *Int. J. Eng. Educ.* **21**(4), 2005, pp. 606–611.
9. J. F. Hoburg, Can Computers Really Help Students Understand Electromagnetics?, *IEEE Trans. Educ.* **36**(1), 1993, pp. 119–122.
10. http://www.mathworks.com/products/matlab/ (Accessed 31 December 2008).
11. H. Fangohr, A comparison of C, MATLAB, and python as teaching languages in engineering, in M. Bubak, G. DickVanAlbada, P. M. A. Sloot and J. J. Dongarra, (eds), Proc. *ICCS 2004, Lecture Notes in Computer Science*, Vol. 3039, Part 4, Springer-Verlag, Berlin, 2004, pp. 1210–1217.
12. M. A. Wirth and P. Kovesi, MATLAB as an introductory programming language, *Comp. Applics. Eng. Educ.* **14**(1), 2006, pp. 20–30.

13. M. K. Haldar, Introducing the Finite Element Method in electromagnetics to undergraduates using MATLAB, *Int. J. Elect. Eng. Educ*. **43**(3), 2006, pp. 232–244.
14. A. Horwitz and A. Ebrahimpour, Engineering applications in differential and integral calculus, *Int. J. Eng. Educ.*, **18**(1), 2002, pp. 78–88.
15. J. B. Dabney and F. H. Ghorbel, Enhancing an advanced engineering mechanics course using MATLAB and Simulink, *Int. J. Eng. Educ.* **21**(5), 2005, pp. 885–895.
16. J. Sanchez, F. Esquembre, C. Martin, S. Dormindo, S. Dormindo-Canto, R. D. Canto, R. Pastor and A. Urquia, Easy Java simulations: an open-source tool to develop interactive virtual laboratories using MATLAB/Simulink, *Int. J. Eng. Educ*. **21**(5), 2005, pp. 798–813.
17. P. Kar and J. W. Evans, A MATLAB-based teaching approach to dilute and concentrated solution theories of electrochemical cells, *Int. J. Eng. Educ.* **25**(1), 2009, pp. 17–23.
18. M. S. Habib, Enhancing mechanical engineering deep learning approach by integrating MATLAB/ Simulink, *Int. J. Eng. Educ.* **21**(5), 2005, pp. 906–914.
19 R. A. Jabr, M. Hamad and Y. M. Mohanna, Newton-Raphson solution of Poisson's equation in a pn diode, *Int. J. Elect. Eng. Educ.* **44**(1), 2007, pp. 22–33.
20. M. Kezunovic, A. Abur, G. Huang, A. Bose and T. Tomsovic, The role of digital modeling and simulation in power engineering education, *IEEE Trans. Power Systems,* **19**(1), 2004, pp. 64–72.
21. P. Krysl and A. Trivedi, Instructional use of MATLAB software components for computational structural engineering, *Int. J. Eng. Educ.* **21**(5), 2005, pp. 778–783.

S. Ratnajeevan H. Hoole D.Sc. (Eng.) London, Ph.D. Carnegie Mellon University, MSc Distinction London, B.Sc. Eng. Hons. Cey., IEEE Fellow, C.Eng. (Sri Lanka). He is Professor of Engineering and Science at Rensselaer Polytechnic Institute. See *Int. J. Eng. Educ.* **25**(6), p. 1235 for a detailed biography.