Coding and GUI Use in the Teaching of Undergraduate Numerical Analysis

Paul W. von Dohlen
vondohlenp@wpunj.edu
Department of Mathematics
William Paterson University
Wayne, NJ 07470 USA

Abstract

In this paper we will investigate different approaches to using coding and graphical user interfaces (GUIs) in the teaching of an undergraduate numerical analysis class. When learning about numerical methods, it is essential that students experience applications of the methods to problems for which hand calculations would be excessively tedious. In order to do so, an instructor can use various levels of coding based on, amongst other factors, the computing proficiency of the intended students. Thus, the instructor can use approaches ranging from pre-programmed graphical user interfaces (GUIs) to complete code creation and implementation. Here, we will explore the possible methods while noting observations based on the author's personal experience. Ultimately, the instructor must decide what level of coding is appropriate for the course so as to use the computing experiences most effectively.

1 Introduction

The teaching of numerical analysis underwent a major reformation beginning in the 1960s and accelerating during the 1980s and 1990s, first because of the advent of computing machines and then because of the increased power and availability of personal computers. Much of the early change in the approach to numerical analysis can be credited to George E. Forsythe and were examined in his 1959 paper "The Role of Numerical Analysis in an Undergraduate Program" appearing in The American Mathematical Monthly [5]. Forsythe detailed the need for computational mathematics as part of the undergraduate mathematics curriculum and listed numerous examples of where such investigations could take place within a variety of mathematical areas. The study of numerical analysis is no longer limited to mathematics as it can be found in computer science, engineering and the sciences. Forsythe's influence and accomplishments were so noteworthy that Donald Knuth [10] wrote: "It is generally agreed that he more than any other man, is responsible for the rapid development of computer science in the world's colleges and universities." Of relevance here is that the issues and concerns posed by Forsythe are still pertinent to the modern day study of numerical analysis. In fact, Forsythe even responds to a common mathematical criticism of the area of numerical computation in

stating that the student "should learn that collaboration with an automatic computer compels precise formulation of the problem and 100 percent accuracy in the preparation of the code. In this respect the automatic computer really forces the precision of thinking which is alleged to be a product of any study of mathematics." [5]

Over three decades later, in 1992, Carroll [4] examined specifically how computer software can and has been used in the teaching of numerical analysis. Carroll offers a quite comprehensive study of the various approaches (at that time) to integrating computer explorations into the numerical analysis curriculum. The paper is of particular note because it characterizes the shift towards incorporation of actual computer code and programs into the teaching and textbooks of numerical analysis. Even more recently we have seen textbooks which use the actual numerical computing as the foundation for teaching the concepts of numerical analysis; for example, Cleve Moler has authored a text called "Numerical Computing with MATLAB" [12] in which the topics of numerical analysis are introduced directly via computing explorations in MATLAB. It should also be noted that in the 1980s the CIText group, founded by members of five British universities, created a number of computer illustrated texts (CITs), including one for numerical analysis, designed to pair the use of traditional textbook material with computer explorations. Further information concerning the computer illustrated text project can be found in an article by Harding and Quinney [7]. More recently, numerical analysis coding with Python has been the focus of a number of studies including those detailed in articles by Michal Kaukic [8] and David I. Ketcheson [9].

In this paper we will examine the particular question of how the use of GUIs and computer coding can be employed in an undergraduate numerical analysis course to varying degrees. Because of issues such as goals of the course, levels of student computer proficiency and availability of computing resources, it is necessary to examine various approaches for incorporating the use of computers in the numerical analysis course. We will investigate approaches ranging from the use of widely available graphical user interfaces (GUIs) to more extensive computer code generation and implementation. Some resources will also be provided.

2 Various Approaches

Before surveying the different approaches to incorporating software/coding into a numerical analysis course, it is important to consider the reasons for employing such different approaches. Based on the author's personal experience, the three primary driving forces behind the choice of an approach are the intended goal of the course, the coding proficiency of the students and the availability of computing resources (hardware and software). These conditions will obviously shape the nature of course and therefore must be taken into account.

While most would agree on the importance of seeing the numerical methods "in action" as part of a numerical analysis course, the extent to which time and effort should be devoted to this end is debatable. The process of coding, debugging and producing graphical output is time-consuming and has the potential to dominate the students' energy and focus thus limiting the amount of material which can be covered. The overall goal of the course within a particular program must be considered. If the course is intended to be an introduction to numerical methods with the intent of surveying the numerous computational topics within mathematics, then it may be appropriate to limit the extent of coding. Whereas if the intent of the course is to prepare students for further use of these methods,

as might be applicable for future engineers, it may be quite beneficial to devote significant time and effort to the intricacies of the coding and implementation.

Possibly the most important factor in determining the level of coding involved in a numerical analysis course is the level of computing proficiency of the students. Even though the use of computers and technology has exploded over the past few decades, many students lack the requisite programming skills and experience necessary to make coding a significant component of a numerical analysis course. Thus one must consider the amount of programming experience of the students or familiarity with a specific software or computing environment before deciding on an appropriate use of software/coding. If a significant amount to coding is to be required, it may be necessary to have a programming course as a prerequisite.

Many instructors of a numerical analysis course will choose to use a computing environment such as MATLAB or a computer algebra system such as Mathematica or Maple. While commercially available software can simplify the amount and level of coding as well as provide robust graphical output capabilities, there are associated drawbacks. Kaukic [8] analyzes many of the advantages and disadvantages of using commercial software and also considers open-source alternatives, namely Python. One of the most significant drawbacks to the commercially available software is the cost which may be prohibitive for some numerical analysis courses. Furthermore, the availability of computer labs and other physical resources may limit the ability to incorporate software/coding into the numerical analysis course. Yet, if available, these software packages can provide powerful environments for computer explorations of the numerical methods.

Thus there are pedagogical and practical reasons for deciding on a level of software/GUI/coding inclusion in a numerical analysis course. Note that each of these approaches aims to address the analyze-evaluate-create levels of Bloom's taxonomy with the more active coding involvement leaning more towards the create level. We will now examine four different approaches, with varying levels of coding, any one of which could be appropriate given the factors mentioned above and others.

2.1 Approach I: GUIs

In recent years, the availability of graphical user interfaces (GUIs) dealing with numerical methods has expanded greatly. Allowing students to explore the implementation of numerical methods through mouse-clicks, drop-down windows and input boxes provides the opportunity for instructors to quickly and effectively introduce the implementation of the numerical methods without the overhead of programming. Numerous examples can be investigated in a short time and the students can easily play with the GUI to explore standard cases as well as any "pitfall" examples which show the possible limitations of the method.

As an example, consider the interpolation GUI provided by Atkinson [1] and shown in figure 1. This GUI works well because of its simplicity and ability to provide useful information both visually and numerically. Users of the GUI can select from a list of functions, vary the order of interpolation, change the interval and also visualize the error. A sample assignment using this GUI could be:

> Using the Polynomial Interpolation GUI, which uses evenly-spaced nodes to create the polynomial interpolant,
>
> **a.** Create plots of the function $f(x) = e^x$ along with its n^{th}-order polynomial interpolant for $n = 1, 2, 3, 5, 10$.

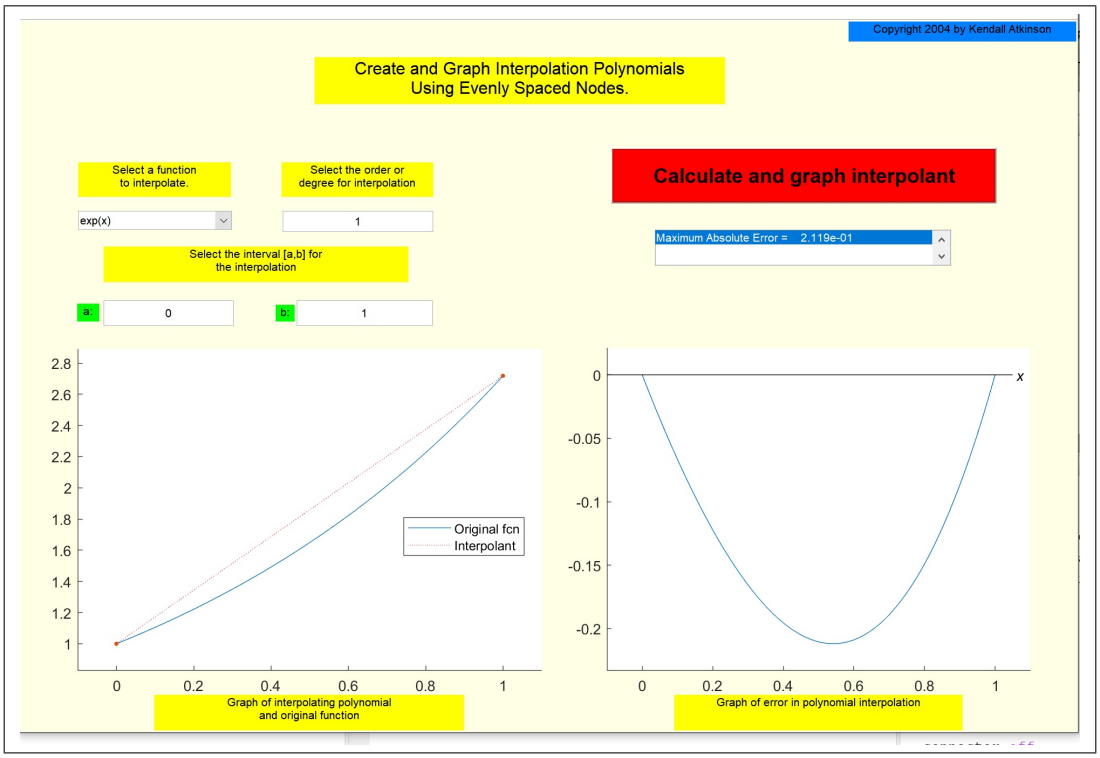

Figure 1: Polynomial Interpolation GUI

b. Create plots of the error in the polynomial interpolant for each case in part (a).

c. How does the error in the linear and quadratic case compare with calculated error bounds?

d. Try using $f(x) = 1/(1+x^2)$. What happens in this case as n increases? Can you offer an explanation?

Atkinson has created a number of these GUIs which can be run under MATLAB [11]. Moler's textbook [12], through its online companion site, provides a number of MATLAB GUIs to correspond with the material coverage in the book. Others have written similar GUIs which are not MATLAB-based; for example, another popular numerical analysis text by Burden and Faires [3] offers (via website) numerous java programs (amongst other materials) designed to offer quick and informative visualizations of numerical methods.

The use of such GUIs provides a solid option for instructors teaching a numerical analysis course to students with no or very limited programming experience and for whom it is simply not possible to spend the time and/or resources overcoming those deficits. Based on the experience of the author, the GUIs provide valuable quick insight opportunities and can be added easily to a course. Admittedly, there is a great deal of understanding accomplished through the programming of these methods (and the authors of the GUIs often suggest using them in conjunction with some programming) but the use of these GUIs should provide valuable illustrations of the implementation of the numerical methods, especially to students who would otherwise not do any computer explorations. Furthermore, GUIs

can be used alongside more involved coding explorations (such as those described below) as they provide visualizations which would require additional coding dealing with plotting features which may extend beyond the scope of relevance to the methods.

2.2 Approach II: Code Modification

Even in the case where the students have had some programming experience prior to the numerical analysis course, it may be beyond the scope of the course for the students to do their own coding of the methods. As stated previously, coding (especially from scratch) can be a very time-consuming venture, and an instructor may choose to limit the time and energy expended on coding by employing code modification. Using the code modification approach the instructor would provide the students with the basic code for a method and let the students modify the code in order to explore a variety of examples or make changes to the particular method.

Even when using this approach, the instructor has significant flexibility in which the code is used. The code can be distributed to the class electronically, thus requiring the student to simply run the code from within MATLAB, for example, by properly formatting function calls in the command window. Another approach (often used by the author of this article) is to provide the students with a hard-copy of the code and have them rewrite the code in MATLAB. The benefit of this approach is that, even though the students are not creating the code, they do experience the process of code creation and debugging (as they will usually make an error in the transcription). Using either approach, the student can then be asked to modify the code to use different functions or even change the code to produce code for another method, such as the secant method in the provided example. Doing the latter would force the students to consider precisely how the methods differ in what is needed in each case and how many calculations are being performed.

Consider both the MATLAB code for Newton's Method shown in figure 2 and the output of the MATLAB code shown in figure 3. The code is a slightly altered version of the code provided by Atkinson and Han in their numerical analysis text [2]. Using the code requires only a brief introduction to the MATLAB environment. A sample assignment using the supplied code could be:

> Given the printed copy of the MATLAB code for Newton's Method,
>
> **a.** Retype the code, creating the function newtonmeth.m in MATLAB.
>
> **b.** Use the code to investigate the roots of $f(x) = x^3 - 4x^2 - 5$.
>
> **c.** Modify the code to include the function $f(x) = x^4 + 4x^3 - 18x^2 - 108x - 135$.
>
> **d.** Investigate the roots of $f(x) = x^4 + 4x^3 - 18x^2 - 108x - 135$. Be sure to consider the effect of a multiple root.
>
> **e.** Modify the code to create a function secantmeth.m which uses the Secant Method. Investigate the roots of all the above functions using this code.

Employing the code modification approach can be a very effective way of using a limited amount of coding to achieve much greater understanding of the methods of numerical analysis and also perform some actual calculations utilizing the methods. Modifications of varying extent can be used

```
function root =
newtonmeth(x0,error_bd,max_iterate,index_f)
%
% function
% newtonmeth(x0,error_bd,max_iterate,index_f)
%
% Newton's method for solving f(x) = 0.
%
% The functions f(x) and deriv_f(x) are given below.
% The parameter error_bd is used in the error test
% for the accuracy of each iterate.  The parameter
% max_iterate is an upper limit on the number of
% iterates to be computed.  An initial guess x0
% must also be given.
%
% For the given function f(x), an example of a
% calling sequence might be the following:
%    root = newtonmeth(3,1.0E-3,10,1)
% The parameter index_f specifies the function to
% be used.
%
% The program prints the iteration values
%      iterate_number, x, f(x), deriv_f(x),
%                                 error, lambda
% The value of x is the most current initial guess,
% called previous_iterate here, and it is updated
% with each iteration.
% The value of error is
%   error = newly_comp_iterate - previous_iterate
% and it is an estimated error for previous_iterate.
% Lambda is a ratio indicator used to identify
% multiple roots.
% Tap the carriage return to continue with the
iteration.
%
% This is modified code from the text Elementary
% Numerical Analysis by Atkinson and Han, Third
% Edition, John Wiley & Sons, Inc, 2004.

format short e
error = 1;
it_count = 0;
while abs(error) > error_bd & it_count <=
max_iterate
    fx = f(x0,index_f);
    dfx = deriv_f(x0,index_f);
    if dfx == 0
        fprintf(1,'The derivative is zero.\n');
        fprintf(1,'Stop.\n')
        return
    end
    x1 = x0 - fx/dfx;
    error = x1 - x0;

    if it_count > 1
        lambda = (x1 - x0)/(x0 - xneg1);
    else
        lambda = 0;
    end

%   Tap the carriage return key to continue
%    iteration = [it_count x0 fx dfx error lambda]

    if it_count == 0
    fprintf(1, '\nIter        xn            ');
    fprintf(1, 'f(xn)         df(xn)       ');
    fprintf(1, 'err_est      lambda_n\n');
    end

    fprintf(1, '%4d    %10.6f    ', it_count, x0);
    fprintf(1, '%10.6f    %10.6f    ', fx, dfx);
    fprintf(1, '%10.6f    %10.6f\n', error, lambda);

    pause
    xneg1 = x0;
    x0 = x1;
    it_count = it_count + 1;
end

    fprintf(1, '%4d    %10.6f\n',it_count, x0);

if it_count > max_iterate
    fprintf(1,'The number of iterates
calculated\n');
    fprintf(1,'exceeded max_iterate.  An
accurate\n');
    fprintf(1,'root was not calculated.\n');
else
    root = x1;
    fprintf(1,'\nRESULTS:\n');
    fprintf(1,'root = %10.6f\n',root);
    fprintf(1,'error bound = %10.6e\n',error);
    fprintf(1,'iteration count = %d\n',it_count);

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function value = f(x,index)

% function to define equation for rootfinding
% problem.

switch index
case 1
    value = x^3 - 4*x^2 - 5;
case 2
    value = x^4 - 3*x^3 - 6*x^2 + 28*x - 24;
case 3
    value = x^4-3.2*x^3+0.96*x^2+4.608*x-3.456;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function value = deriv_f(x,index)

% Derivative of function defining equation for
% rootfinding problem.

switch index
case 1
    value = 3*x^2 - 8*x ;
case 2
    value = 4*x^3 - 9*x^2 - 12*x +28;
case 3
    value = 4*x^3-3*3.2*x^2+2*0.96*x+4.608;
end
```

Figure 2: MATLAB Code

```
>> root = newtonmeth(3,1.0E-3,10,1);

Iter        xn           f(xn)         df(xn)       err_est      lambda_n
   0      3.000000    -14.000000      3.000000      4.666667      0.000000
   1      7.666667    210.518519    115.000000     -1.830596      0.000000
   2      5.836071     57.536065     55.490602     -1.036861      0.566407
   3      4.799209     13.407720     30.703558     -0.436683      0.421158
   4      4.362526      1.899473     22.194700     -0.085582      0.195983      RESULTS:
   5      4.276944      0.065934     20.661202     -0.003191      0.037288      root =   4.273749
   6      4.273753      0.000090     20.604871     -0.000004      0.001367      error bound = -4.362911e-006
   7      4.273749                                                             iteration count = 7
```

Figure 3: Output Generated by MATLAB Code

depending on the programming capabilities of the students. Thus, this code modification approach can effectively be used for a rather wide range of student programming experience. Furthermore, the approach can be coupled with the use of GUIs to provide an even stronger visualization of the methods. It should be noted that the author currently employs this approach because of the programming experiences of the students involved as well as time constraints.

2.3 Approach III: Pseudocode Translation

If an instructor desires to put greater emphasis on the actual coding of the methods, he/she may choose to employ pseudocode translation. In doing such the instructor will provide pseudocode for a given method and have the students create the actual code in a programming language or environment of his/her preference. The use of pseudocode offers a great deal of flexibility in that the instructor could conceivably allow students to use whatever language or environment with which they are familiar and even make observations about the advantages or disadvantages of each.

Many numerical analysis texts, such as those by Burden and Faires [3] and Gerald and Wheatley [6] provide excellent pseudocode with the description of each numerical method. For example, the text by Burden and Faires [3] offers the following for a Composite Simpson's Rule approximation to the integral $I = \int_a^b f(x)\,dx$:

INPUT	endpoints a, b; even positive integer n
OUTPUT	approximation XI to I
Step 1	Set $h = (b-a)/n$
Step 2	Set $XI0 = f(a) + f(b)$;
	$XI1 = 0$; (Summation of $f(x_{2i-1})$)
	$XI2 = 0$; (Summation of $f(x_{2i})$)
Step 3	For $i = 1, \ldots, n-1$ do Steps 4 and 5
Step 4	Set $X = a + ih$
Step 5	If i is even then set $XI2 = XI2 + f(X)$
	else set $XI1 = XI1 + f(X)$
Step 6	Set $XI = h(XI0 + 2 \cdot XI2 + 4 \cdot XI1)/3$
Step 7	OUTPUT (XI)
	STOP

Thus, a sample assignment could be:

> Using the provided pseudocode for the Simpson's Rule approximation,
>
> **a.** Create a program which, when given an interval $[a, b]$, a value n for the number of subintervals and a selected function $f(x)$, computes the Simpson's Rule approximation to the integral $\int_a^b f(x)\,dx$.

b. Use the code to approximate $\int_{-1}^{2} f(x)\,dx$ for $f(x) = xe^x$ using $n = 4, 8, 16, 32, 64$ subintervals.

c. Given that the error in the Simpson's Rule approximation can be estimated by $E \approx -\frac{h^4}{180}\left[f'''(b) - f'''(a)\right]$ where h is the length of each subinterval, find an error estimate for each case in part (b) and compare with the results from part (b).

d. Make an observation about the change in the error compared to the change in the subinterval length from the results in part (c).

With the given pseudocode the students would have to create working code for the numerical method. They key aspect here is that they would need to be fairly well-versed in the syntax of the language or environment which has been chosen. But in actually writing the code, the students should experience a deeper interaction with the method; they would see more of the details and intricacies of the method and be allowed greater flexibility in the implementation of the method.

2.4 Approach IV: Code Creation

It is possible that a main goal of a numerical analysis course might be on accurate and efficient code creation. In such a case, code creation stemming from only a description or derivation of the numerical method could be employed. For example, a instructor could simply give a derivation of the general form of a Taylor polynomial of degree n and then ask the students to create code which would generate any degree Taylor polynomial (based on user input) to approximate a given function. Furthermore, the instructor could ask for error calculations and plots showing the agreement between the function and the Taylor polynomials. Thus, a sample assignment could be:

Given the general form of a Taylor polynomial of degree n (centered at the point a) as

$$p_n(x) = \sum_{j=0}^{n} \frac{(x-a)^j}{j!} f^{(j)}(a)$$

a. Create a program which will calculate Taylor polynomial approximations centered at a given point a, for the function $f(x) = \sin x$ and a given degree n.

b. Give the program the capability to produce graphs of $f(x) = \sin x$ along with its Taylor polynomials.

c. Use the code to produce graphs of the Taylor polynomials (centered at $a = 0$) of degrees $n = 1, 3, 5, 11, 15$ for $f(x) = \sin x$.

d. Using the point $x = \pi$, calculate the error in each Taylor polynomial of part (c).

Obviously, this approach would require the highest level of programming proficiency on the part

of the students. It would also require the greatest amount of time and energy spent on code creation and refinement. But the students would experience the most layers of interaction with the numerical method. Without the outline from provided pseudocode, the students would need to carefully consider what information is available or necessary as input and also consider what would be the desired output. They would have to design an algorithm for the calculation, noting any error checks or possible complications, and in doing so they would have to consider the efficiencies of different algorithms for the numerical method. For example, considering the Taylor polynomial exercise above, the students would want to consider the use of nested multiplication when computing the Taylor polynomial as an efficient computational approach. Finally they would have to design a particular form of output, possibly including automatically generated plots. As with the pseudocode approach, there is a great deal of flexibility here as to what programming language or environment is to be used.

3 Conclusion

Allowing students to interact with the computational methods of numerical analysis is a necessary and informative aspect of a numerical analysis course. But the extent to which such computing explorations and the coding involved serve as a focus of the course can be varied based on certain determining factors. The goals of the course, the computing proficiency of the students and the availability of computing resources all play roles in determining what level of coding will be most effective in such computer explorations. Ideally, an instructor would likely have the students experience a great deal of coding so as to gain the most insight into the intricacies of the computational methods. Yet realistically, such an approach may not be possible, thus we have outlined four different approaches for incorporating computer explorations into the numerical analysis course.

The use of pre-programmed GUIs (graphical user interfaces) offers an excellent opportunity for students to get immediate hands-on experience with the numerical methods. This approach can be used when students lack any programming knowledge or, even more effectively, when paired with some coding. In order to introduce students with limited programming experience to computational code, an instructor can use code modification which allows those students to experiment with provided code. For students capable of writing their own code, an instructor can choose to provide pseudocode or have the students design their own algorithms. Either of these approaches will offer valuable coding and debugging experiences in addition to the explorations of the impacts of the numerical methods.

To date, the author been able to gain insight into the effectiveness of these methods. Future plans involve linking these approaches to educational theories such as those based on Russian psychologist Lev Vygotsky's Zone of Proximal Development. Such theories have been explored extensively in childhood learning but should prove to be informative in a study seeking to measure the effectiveness of these methods in an undergraduate setting.

Regardless of the approach, students should benefit greatly from experiencing the numerical methods through the computer explorations. Each individual instructor must decide what level of coding is appropriate so as to make the explorations as insightful and efficient as possible.

References

[1] K. Atkinson. Teaching Numerical Analysis Using Elementary Numerical Analysis by K. Atkinson and W. Han. Website. http://www.math.uiowa.edu/ atkinson/ena_master.html.

[2] K. Atkinson and W. Han. *Elementary Numerical Analysis*. John Wiley and Sons, Hoboken, NJ, third edition, 2004.

[3] R. L. Burden and J. D. Faires. *Numerical Analysis*. Thomson Brooks/Cole, Belmont, CA, eighth edition, 2005.

[4] J. Carroll. The Role of Computer Software in Numerical Analysis Teaching. *ACM SIGNUM Newsletter*, 27(2), April 1992.

[5] G. E. Forsythe. The Role of Numerical Analysis in an Undergraduate Program. *The American Mathematical Monthly*, 66(8):651–662, October 1959.

[6] C. F. Gerald and P. O. Wheatley. *Applied Numerical Analysis*. Pearson Education, Inc., New York, seventh edition, 2004.

[7] R. Harding and D. Quinney. Computer Illustrated Texts (CITs) for Teaching Numerical Analysis. *Computers & Education*, 15(1-3), 1990.

[8] M. Kaukic. Open Source Software Resources for Numerical Analysis Teaching. *International Journal for Mathematics Teaching and Learning*, October 2005.

[9] D. Ketcheson. Teaching numerical methods with IPython notebooks and inquiry-based learning. *Conference: Python in Science Conference*, 2014.

[10] D. E. Knuth. George Forsythe and the Development of Computer Science. *Communications of the ACM*, 15(8), August 1972.

[11] MATLAB. version 9.4.0.813654 (R2018a), 2018. The MathWorks Inc.

[12] C. B. Moler. *Numerical Computing with MATLAB*. Society for Industrial and Applied Mathematics, Philadelphia, 2004.