A Contribution to Using MATLAB in Teaching Numerical Methods at Technical Universities

Michal Novák
Brno University of Technology,
Faculty of Electrical Engineering and Communication,
Department of Mathematics
e-mail: novakm@feec.vutbr.cz

Abstract

The text makes a contribution to the issue of reasons for and ways of using computers in teaching mathematics. I focus on the subject of using MATLAB in teaching numerical methods. I also comment on some typical misconceptions and prejudices in (students') attitude towards algorithm development in practical classes. The contribution is set in the background of FEEC BUT.

1 Introduction

Numerical methods in mathematics are an integral part of training prospective engineers. Due to its often lengthy computations students usually label it as uninspiring and mechanical part of mathematics where no creative abilities are needed. They also point out that no true practice can be performed as curricula are tight. Yet this prejudice can be turned into an advantage as the nature of numerical methods makes it ideally suitable for computer-aided teaching.

In the bachelor student programmes at the Faculty of Electrical Engineering and Communication, Brno University of Technology, numerical methods are the subject matter of *Mathematics 3* (abbreviated as BMA3 for attended form of study, KMA3 for combined form of study). It is taught in the winter term of the second year of study and is compulsory for all students. The subject matter[1] is evenly distributed between probability theory and statistics, and numerical

[1] The *Mathematics 3* numerical methods block includes:

1^{st} **lecture:** Banach theorem, Iteration methods of systems of linear equations

2^{nd} **lecture:** Interpolation polynomials, Spline functions

3^{rd} **lecture:** Least-square method, numerical differentiation

4^{th} **lecture:** Numerical integration – trapezoid and Simpson methods

5^{th} **lecture:** Solution of partial differential equations: Euler method and its modifications, Runge-Kutta methods

6^{th} **lecture:** Solution of partial differential equations: Euler method for system of differential equations, Finite-difference method

1^{st} **practical:** Locating roots, Bisection method, Regula falsi method (not lectured)

2^{nd} **practical:** Newton and Iteration methods (not lectured)

3^{rd} **practical:** Systems of non-linear equations, Interpolation polynomials (Lagrange and Newton polynomials)

4^{th} **practical:** Spline functions, Least-square method

5^{th} **practical:** Numerical differentiation and integration

6^{th} **practical:** Euler method and its modifications, Runge-Kutta methods, Finite-difference method

methods. Given the 2/2 scheme of lectures / practical classes, numerical methods are dealt with in six or seven 100 minutes long lectures and six or seven practical classes of the same length.

Final exams in *Mathematics 3* are written ones and do not contain practical tasks solved with the aid of computers or writing scripts.

2 Nature of practical classes

All numerical methods practical classes of *Mathematics 3* are computer-aided ones in attended form of study; MATLAB software is used. The obvious question, which arises in this respect, is how to make use of computer classes when students naturally point out that the final exam does not involve computer performance at all. The wide range of options includes using MATLAB as a "well-equipped" calculator, setting tasks of writing a MATLAB script for each particular method, using MATLAB to show the importance of assuptions and conditions for using a given method, pitfalls and risks of applying and following algorithms without considerations, etc.

The format of practical classes I am going to discuss follows from the belief that by that once students themselves design an algorithm regarding a particular numerical method, the efficiency in practising it increases.[2] It seems to be a time-saving format as well, as once students are in possession of an algorithm, have access to the computer software, they need only to consider the necessary assumptions and considerations of the particular method and can (to a great extent) make the tasks on their own or compute tasks found elsewhere. Such a script then works as a verification of the often lengthy hand computations, which cannot be performed in class for time reasons.

3 Examples of scripts

Let us consider two examples: the task of finding a Newton interpolation polynomial and the task of interpolation by natural cubic spline functions. For both cases, let us follow a typical student's encounter with the subject matter and let us discuss the problems he or she faces.

3.1 Finding a Newton interpolation polynomial

The electronic text for *BMA3* (or rather *KMA3*) is available – [4] is used. The respective quotes are demonstrations only, as the actual form of formulas is similar in all texts. In [4], chapter 6.1.3, the divided differences are mentioned and the formulas for finding them are given:

> For a given function f and nodes x_i, $i = 0, \ldots, n$ the ratios
>
> $$f[x_i, x_{i+1}] = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}, \quad i = 0, 1, \ldots n-1$$
>
> are called *divided differences of the first order*. Using them we define the *divided diferences of the second order* as
>
> $$f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i}, \quad i = 0, 1, \ldots, n-2;$$
>
> and in a general way for $k \leq n$ *divided differences of the* k^{th} *order*
>
> $$f[x_i, x_{i+1}, x_{i+2}, \ldots x_{i+k}] = \frac{f[x_{i+1}, x_{i+2}, \ldots, x_{i+k}] - f[x_i, x_{i+1}, \ldots, x_{i+k-1}]}{x_{i+k} - x_i},$$

[2] This does not mean that the question *Can this particular task be solved by this particular method?* should be ignored or omitted; verifying conditions of use is equally important.

$$i = 0, \ldots, n-k.$$

> Substituing these values [into a previously stated form of interpolation polynomial] we get *Newton interpolation polynomial*
>
> $$N_n(x) = f(x_0) + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \ldots$$
> $$\ldots f[x_0, x_1, \ldots, x_n](x - x_0)(x - x_1)\ldots(x - x_{n-1})$$

A well commented example with the divided-difference table follows. The actual computation of the divided diferences, however, relies on the above mentioned formulas and naturally does not include a down-to-earth explanation of how each respective value was obtained.

Yet it is the down-to-earth explanations that cause the greatest amount of problems as students surprisingly are not able to handle the general formulas and obtain the values in the table. With respect to the have-the-result-don't-need-more attitude it seems not to matter how many numerical tasks are solved in the class as every extra task is likely to cause the same problems. The algorithm development is therefore needed, even though students usually percieve it as wasting precious time. They do not seem to realize that when developing the algorithm they are forced to think over the problem and that they can only accomplish the task when they have fully understood what exactly the general notation means.

One of the possible forms of the resulting MATLAB script may be as follows[3]:

```
x=input('Nodes (in the form [a b c etc.]): ');
gx=input('Function values in nodes (in the form [a b c etc.]): ');

for i=1:size(gx,2)
    deltag(i,1)=gx(i);
end

for i=1:size(x,2)-1
    for j=1:size(x,2)-1
        if (i+j)<size(x,2)+1
            deltag(j,i+1)=(deltag(j+1,i)-deltag(j,i))/(x(j+i)-x(j));
        end
    end
end

deltag
```

A common comment on using algorithm development techniques for students other than those of information technologies is that students do not need to be familiar with "programming". It has also been often pointed out that syntax rules of respective softwares differ considerably and that students are not meant to learn what they are not going to apply once they are employed. Yet the above mentioned script only requires the knowledge of the following: `for n = a to b` loop and its difference from `do while` loop, the `if then` concept and `input` and `size` commands. The first two concepts being notorious while the last one obvious (even if not used before) and the command predictable. Syntax (as well as the `size` command can be found in help once the user knows what to look for.) Furthermore – naturally – the task assumes that the general notation has been comprehended and different ways of handling the problem considered. In other words – writing such a script should be an easy task for a second year student who attended the respective lecture and has revised before the practical class.

[3]The script finishes when variable *deltag* storing the divided-difference table is printed on the screen; constructing the polynomial may be another task. Entering correct values is assumed.

3.2 Interpolation by natural cubic spline functions

[4], chapter 6.2, states that:

> In intervals $< x_i, x_{i+1} >$, $i = 0, 1, \ldots, n-1$ we are going to look for spline functions
>
> $$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$
>
> We have that $a_i = f(x_i)$, $i = 0, 1, \ldots, n-1$. Therefore [considering also further assumptions] after some effort we get a system of equations with unknown coefficients c_i, $i = 0, \ldots, n$
>
> $$\begin{aligned} h_{i-1}c_{i-1} + 2(h_{i-1} + h_i)c_i + h_i c_{i+1} &= 3(\frac{\Delta f_i}{h_i} - \frac{\Delta f_{i-1}}{h_{i-1}}), \\ & \quad i = 1, \ldots, n-1 \\ c_0 = c_n &= 0, \end{aligned}$$
>
> where $h_i = x_{i+1} - x_i$ and $\Delta f_i = f(x_{i+1}) - f(x_i)$, $i = 0, \ldots, n-1$.
>
> [The first, second and last equation of the general system is then given.]
>
> Coefficients b_i, d_i can be found with the help of c_i from formulas [which follow from others stated in the text]
>
> $$\begin{aligned} b_i &= \frac{f(x_{i+1}) - f(x_i)}{h_i} - \frac{c_{i+1} + 2c_i}{3}h_i, \qquad i = 0, \ldots, n-1 \\ d_i &= \frac{c_{i+1} - c_i}{3h_i}, \qquad i = 0, \ldots, n-1 \end{aligned}$$

A well commented example is again given. A table summarises i, x_i, $f(x_i)$, h_i, Δf_i values, the system of three equations for c_i is included, and formulas for b_i, d_i are mentioned. All coefficients a_i, b_i, c_i, d_i are summarised in a table and the set of spline functions is then given. An example value is computed in the end.

The task of interpolation by spline functions is that of lengthy computations. If learning is supposed to be effective, they must be performed, though. Yet regarding the time allowed for the topic (even though 50 minutes for this particular method in *BMA3* at FEEC BUT may not be typical), and number of students in the class, solving one or two tasks in the class is all but effective, even if students are required to do the computations themselves and not to "follow the board".

The issue of efficiency

More practice is needed. Students can refer to books – yet not knowing how the method works (since there was no time to catch this during the one or two examples) or its pitfalls, they are not likely to make much use of this option.

The teacher may prepare the script, let students download it and have it for practice. Once they come across an example, the script may be a quick verification tool of hand computations. This option seems to be clearly favoured by students (cf. [5]).

However, the script may also be prepared by students themselves in the actual class. The advantages are obvious – by actually writing the script themselves, students learn much more than by downloading the work of somebody else. Furthermore, they can write the script in such

a way with such notation as is convenient for them. Obviously, a great number of students are not happy with this approach as it requires more work on their part than clicking "Save as...".

Producing the whole scripts is not necessary – parts of scripts may be asked for, modifications required, etc. A sample task in this respect may be: *The following script works only for five nodes. Change it in such a way that the number of nodes is not relevant.* (The script produces natural cubic spline functions.)

```
clear all
x=sym('x');
% entering the task
funkce=input('Enter function f(x): ');
uzly=input('Enter nodes in the form [a b c etc.]: ');
n=size(uzly,2);
% computing function values in nodes
for i=1:size(uzly,2)
    hodnoty(i)=subs(funkce,uzly(i));
end
% computing length of the intervals
for i=1:size(uzly,2)-1
    h(i)=uzly(i+1)-uzly(i);
end
% computing deltaf(i)
for i=1:size(hodnoty,2)-1
    deltaf(i)=hodnoty(i+1)-hodnoty(i);
end
% summarising results for clarity reasons (same form as in teaching text)
uzly
hodnoty
h
deltaf
% computing coefficients c
% MAKE THIS PART WORK REGARDLESS OF THE NUMBER OF NODES
matice=[2*(h(1)+h(2)) h(2) 0; h(2) 2*(h(2)+h(3)) h(3); 0 h(3) 2*(h(3)+h(4))]
prava_strana=[3*(deltaf(2)/h(2)-deltaf(1)/h(1));
3*(deltaf(3)/h(3)-deltaf(2)/h(2));
3*(deltaf(4)/h(4)-deltaf(3)/h(3))]
pom=pinv(matice)*prava_strana;
% c(0)=0 a c(n)=0 added to the above results
c(1)=0;
for i=1:size(pom)
    c(i+1)=pom(i);
end
c(n)=0;
% computing remaining coefficients
for i=1:size(hodnoty,2)-1
    b(i)=(hodnoty(i+1)-hodnoty(i))/h(i)-(c(i+1)+2*c(i))/3*h(i);
end
for i=1:size(hodnoty,2)-1
    d(i)=(c(i+1)-c(i))/(3*h(i));
end
% results printed in a convenient form (not included)
```

Remark: This particular task has been chosen as an example because it helps students to grasp the general notation of the system of equations for computing c_i values, which seems to be the most problematic part in the process of actual computing the tasks as students are often at a loss with the number of different indices and their ranges. Also, to accomplish the task students have to comprehend the method itself as well as the rest of the script. For time saving reasons they are not required to produce the whole of it, though. It is also to be noticed that from the IT point of view writing or adjusting this particular script again does not require more skills than any student of technology should possess.

4 Conclusions

It is the issue of efficiency of numerical methods practical classes that has been discussed. It is obvious that once curricula are tight, the number of lessons decreasing and the amount of subject matter increasing, teacher — student cooperation in the class is the key to efficiency. This, however, assumes that students understand the rational, are willing to co-operate, and work on their own outside the class, which are conditions not always satisfied. The general attitude of (a representative sample of FEEC BUT) students towards computer-aided classes has been discussed in [5]. It shows the lack of interest in any non-immediatelly-applicable and non-immediatelly-result-giving techniques. Algorithm development as a tool of (deeper) understanding of the concept is clearly not favoured; students prefer a set of particular tasks to a general solution. Yet I believe that a suitable form of practical classes can help to overcome this attitude.

References

[1] B. Bačová. MATLAB and Mathematica in teaching numerical mathematics. In: *The 1st international conference on applied mathematics and informatics at universities 2001*, 294–299 Gabčíkovo, 2001.

[2] J. Baštinec. Matematika pro sériové bakaláře na FEKT VUT. In: *3. konference o matematice a fyzice na vysokých školách technických s mezinárodní účastí. Sborník příspěvků*, 31–36 Brno, 2005.

[3] J. Baštinec. O schopnosti studovat u studentů FEI VUT. In: *XVII International Colloqium on the Acquisition Process Management. Proceedings contributions*, 10–13 VVŠ PV, FEM, Vyškov, 1999.

[4] B. Fajmon, I. Růžicková. *Matematika 3.* VUT, Brno, 2005. (elektronický text na `http:\\www.umat.feec.vutbr.cz/~fajmon/bma3/matematika3.pdf`)

[5] M. Novák. On Problems of Computer Aided Teaching of Mathematics at Technical Universities. In: *XXIII International Colloqium on the Acquisition Process Management. Proceedings of electronic versions of contributions.* University of Defence, Faculty of Economics and Management, Brno, 2005.