Teaching Linear Algebra with and to Computers

D.J.Jeffrey and Robert M. Corless
Ontario Research Centre for Computer Algebra
University of Western Ontario

Abstract

Three topics are discussed that relate to the teaching of linear algebra using computers (here the term *computers* includes calculators). The first topic is the variation in notation and terminology both between books and computer systems, and between different computer systems. The second topic is the importance of numerical linear algebra, and how it becomes more difficult to avoid numerical aspects of the subject once computers are used in teaching. The final topic is the Turing factoring of a matrix. This is a factoring approach to row reduction, and if it is taught in courses, students can transfer what they know to computers with a minimum of difficulty.

1 Introduction

Teaching linear algebra with computers requires a co-operative effort between teachers and system developers to match course material and software. Neither computer software *nor* course material can remain static. When computers were first used in teaching, students sat down with calculators or general purpose software and continued to study the same concepts and techniques that existed before calculators. In the future we can expect software to adapt itself more closely to teaching requirements, but we must also expect course content will evolve, whether we will it or no, as it adapts to a "material change of circumstances", to use a legal term. Therefore, many decisions remain to be made by teachers and system developers regarding the shape of software and the shape of courses. This article discusses questions in three general areas, and draws on experience at the University of Western Ontario (UWO), where engineering students have been required to buy and use HP48 calculators in their classes and in their examinations.

Courses called 'linear algebra' cover a number of variations, differing basically in the weight they give to theory, applications, and numerical topics, and therefore any discussion must be clear on what type of course is being addressed. Perhaps the most common first course in linear algebra (especially in North America) combines sections on matrix methods for solving simultaneous linear equations with sections on the study of matrix algebra as a concrete example of a vector space. Applications are given short shrift, because students don't like them. A more abstract treatment of linear algebra—what we might call a vector-space course—is usually left to a later course for mathematics majors, and is sometimes used as a vehicle for an introduction to axiomatic algebra; the other type of advanced course is numerical linear algebra, which may be taught as part of a course on numerical analysis. The students also should influence the course. For engineering students, it seems reasonable to emphasize computations and applications, and for mathematicians, mathematical concepts.

The major use of computers has been to assist students with matrix manipulation. Usually the students will be taking a course, such as the one described above, which includes practical computations (solving systems, finding eigenvalues) and some introductory treatment of vector spaces (the ideas of basis, subspaces, mapping between spaces). Consequently, the focus here is on matrix computations in the service of linear algebra.

In an effort to disguise, somewhat, our own prejudices, we have presented the discussion in the form of questions.

2 Representations and Operations

The questions in this section refer to the basic design decisions upon which a software system is based. We use the following notation. Scalar quantities are denoted by Greek letters; vectors are denoted by lower-case roman letters; matrices are denoted by upper-case roman letters.

2.1 Notation

The first question is

> what standard notation should a computer system use?

Mathematical handwriting recognition software is not yet generally available. This forces compromises on mathematical notation in the presence of computers. Let us ignore for the moment the different notations in use for matrix entry or display, and consider just the transpose of a matrix. In textbooks, the transpose of A is variously denoted as

$$A^T, A^t, A', A^*, A^H$$

(the last being used exclusively for Hermitian or conjugate transpose). The computing languages all choose notation other than A^T, A^t, or A^H because it would be too difficult to distinguish this from raising A to a power (we suspect there is at least one paper somewhere that does use A^t to mean raising A to the power t). MATLAB uses `A'` to mean Hermitian transpose and `A.'` for ordinary (non-conjugated) transpose, while Maple uses `Transpose`(A) for the ordinary transpose, and `HermitianTranspose`(A) otherwise. The HP48 series calculators use `TRN` or `TRAN` for Hermitian transpose and do not appear to have a command for ordinary transpose (other than successively using `TRN` and `CONJ`); conversely Derive and Mathematica have commands for ordinary transpose and not the Hermitian one.

The different notational conventions in use prior to computers have thus only multiplied; and what the computer uses will almost certainly not match the textbook.The resulting burden on student memory may play a useful pedagogical role, but it is also a potential source of confusion, especially when more than one computer system is used.

Notation, though often superficially treated when used in teaching, is one of the more significant topics taught; we often convey more in how we choose to denote things than in our calculations. The isomorphism between a row and a column vector of the same dimensions is rarely explained carefully; but students get used to multiplying matrices by column vectors on the right and row vectors on the left, and can be confused if a computer system does this differently. For example, the HP calculators display vectors horizontally, an obvious saving of screen space. We note that tensor notation for multiplication, namely $a_{ij}b_j$, does not rely so much on student memory of how matrix multiplication works (but does require that they get used to the summation convention, which is useful for higher-order tensors).

2.2 Different types of arrays

The second question is

> should a computer system distinguish between scalars, vectors and matrices?

Two systems that conveniently illustrate different positions on the representation question are MATLAB and the HP48/49 calculators. In MATLAB, the data storage makes no distinction between scalars, vectors and matrices, storing all objects as a single type of array; the HP48 has separate data structures for all three types and enforces operational rules between them.

In the HP48, a scalar is a bare number, for example, 0.67 ; a vector is an array enclosed in single brackets, e.g. $[0.67, 2, 3]$; a matrix is an array of vectors, e.g. $[[0.67, 2, 3], [-1, -2, -3]]$. A one-dimensional row matrix $[[0.67, 2, 3]]$ is different from a vector. In the HP, the following operations all generate error messages.

```
1+[1]   ,   1+[[1]]   ,   [1]+[[1]]   .
```

For the purposes of multiplication, vectors can be combined with matrices. Thus the following are legal operations

$$\left[\begin{array}{c}[1,2,3]\\ [4,5,6]\end{array}\right]\left[\begin{array}{c}[1]\\ [2]\\ [3]\end{array}\right] , \quad \left[\begin{array}{c}[1,2,3]\\ [4,5,6]\end{array}\right][1,2,3] \; .$$

In the first case, a 2×3 matrix is multiplied by a 3×1 matrix and a 2×1 matrix will be returned; in the second case, the matrix is replaced by a vector, and the result will be a vector also. In contrast, the expression

$$\left[\begin{array}{c}[1,2,3]\\ [4,5,6]\end{array}\right][[1,2,3]]$$

is not legal syntax. Derive and Mathematica follow similar conventions, but with some variations.

In MATLAB, a 1×1 matrix can have several interpretations. Thus the following operations generate error messages in an HP48 calculator, but are legal in MATLAB.

$$[1]\left[\begin{array}{c}[1]\\ [2]\\ [3]\end{array}\right] , \quad [1] + \left[\begin{array}{c}[1,2,3]\\ [4,5,6]\end{array}\right] \; .$$

In an abstract vector space, the operations $\alpha + v$, $\alpha + A$, are not defined, but αv and αA are defined. Clearly a teacher who wishes to stress this fact had better keep the students away from the computer during this part of the course, or prepare to explain what MATLAB is doing. The point is that an array is both a data structure and a mathematical object. Since MATLAB uses arrays for many more tasks than just linear algebra, all of its extensions are convenient, and make sense in context. The teacher must remember, however, that MATLAB is now a general purpose computational system, and not just a study aid for linear algebra, or more specifically, matrix analysis, and therefore it does not confine users to the narrow streets of linear algebra.

2.3 Matrix and array operations

The next question is

> should computer systems define operations that are not part of standard linear algebra texts?

Most mathematical software packages have extended the operations that can be performed on matrices beyond the operations defined in linear algebra. The most obvious operation is division. Many teachers of linear algebra or matrix analysis spend some time explaining that division "does not exist" and that the system $Ax = b$ has the exact solution $x = A^{-1}b$ only if A^{-1} exists. Further, the solution cannot be written b/A because this notation does not show that multiplication must be on the left. Having taught this, the teacher is then confronted with the HP48 calculator, which happily accepts b/A. Also if b and A are placed on the stack, then pushing the $\boxed{\div}$ button produces the solution of $Ax = b$. In MATLAB, the statement A/B computes the X that solves the problem $B^T X^T = A^T$ in the sense of least squares. Explaining this to students requires that they be taught least-squares solutions.

Some years ago, questions were set on UWO exams following a model found in many books:

Does the following system of equations for $\{x, y\}$ have a solution?

$$\begin{aligned} 5x + 3y &= 3 \\ -x + y &= 4 \\ 4x + 4y &= 8 \end{aligned}$$

The expected response was a row-reduction of the system followed by the conclusion "no". As it happens, if the last constant is 7 rather than 8, then the system has the exact solution $x = -9/8, y = 23/8$, because in that case the last equation is the sum of the other two, but no solution exists to the problem as printed. Many students, however, entered the system into their HP48 calculators and used the menu item "Solve linear system" to discover, apparently, that a solution existed, and complained when their answers were marked wrong[1]. The same problem is present in MATLAB, where the operation $A\backslash b$ also is defined to return a least-squares solution. The difficulty with general purpose software is that students will sooner or later wander into advanced areas that the teacher is not expecting. Once such an area of misconception is uncovered, there are two responses: first, dictate instructions in the classroom and in the exam that this menu item must not be used, or, second, modify the course so that the students are taught least-squares and then the responsibility for choosing the correct command becomes theirs. If least-squares is included only to explain calculator behaviour, then many teachers might object to the thought that their course material is being decided for them by a calculator company.

Division is not the only problem, consider simple multiplication. In the equation

$$(A\alpha)B = A(\alpha B) ,$$

the multiplications are different, those between α and a matrix being commutative, and those between matrices being non-commutative. This distinction is not recorded in the notation used in any books we know, but it is recorded in some software systems, specifically Maple. In Maple, the equation cannot be written $(A*\alpha)*B = A*(\alpha*B)$, because $A*B$ is illegal notation for the non-commutative matrix product. Instead, one must write $(A*\alpha).B = A.(\alpha*B)$.

In addition, many packages implement *array arithmetic*, also called element-by-element arithmetic. The array product is also called *Hadamard product*, after usage introduced by von Neumann [4]. In MATLAB, the notation for array product and array division is

$$\begin{aligned} [a, b, c, d, \ldots] \;.*\; [w, x, y, z, \ldots] &\equiv [aw, bx, cy, dz, \ldots] \\ [a, b, c, d, \ldots] \;./\; [w, x, y, z, \ldots] &\equiv [a/w, b/x, c/y, d/z, \ldots] \end{aligned}$$

[1] A check of their apparent answer by computing the residual, something explicitly encouraged in the course, would have exposed their error to them; so their marks were not improved by their complaints

From the point of view of linear algebra, these operations are not important, because they are not matrix operations, but for teachers of introductory numerical analysis courses, the notation is the source of endless debugging problems for the students. For professionals, the compactness of this notation is a great convenience.

In Maple, the elementwise operations are available by working with Arrays and not Matrices (these are different from 'arrays' and 'matrices' in Maple, by the way; the case of the initial letter is significant). One can efficiently change the type of a two-dimensional Array to be a Matrix by issuing the command `rtable_options(A,subtype=Matrix)`. This is useful for programming, but not something that most teachers would want to include in a first course on linear algebra.

2.4 The answers

It has not been our intention to sell one particular notation or to recommend one computer system over others. Each teacher will have to select some approach to linear algebra, and then match the software to the course. If the students in the course are comfortable with computers, then usage questions are less important than in the case of students who are not comfortable. If ever the mathematicians can agree on notational issues, then computers can be asked to follow them, but only if the mathematicians agree to change to a notation that can reasonably be implemented. In the meantime, the mathematics teacher is faced with choices both in textbooks and in software. The one point we do want to make is that the decision to adopt computers *forces* such notational choices on the teacher.

3 Numerical or exact linear algebra?

The main question we ask in this section is

> should teachers include any material on numerical linear algebra (i.e. computation in the presence of rounding errors caused by non-exact numerical data) in a first course?

The subject of numerical linear algebra is a large and important one, both for industry and science. However, most teachers of matrix analysis would prefer to avoid any mention of the subject in a first course. There are two reasons, however, why they might not be allowed to.

3.1 Different (sometimes better) procedures

The first reason is that general purpose software, such as we have been describing, must perform correctly for professional users as well as students. Therefore many commands that a student might use will give results in the form that is required by advanced users.

For example, LU factoring is often taught without pivoting[2], or at most with pivoting to avoid exact zeros. Both the HP48 and MATLAB, however, always use partial pivoting in this operation, because professional practice requires it. Therefore any student using either of these products to check homework problems may face the difficulty that the computer answers problems differently from the textbook. Similar remarks apply to QR factoring: the HP48 factors $AP = QR$, where the P matrix again is required by numerical linear algebra.

[2]We might call this a useful "lie-to-children"[6]. The pedagogical point is to avoid unnecessary burdens on the student's first encounter with the concept.

It can also be argued that we should look to the future. If the students are in a program such as engineering, where we can hope that some of them will actually see large linear problems in their future employment, then it is a good thing to start introducing correct ideas as early as possible.

3.2 Incorrect results

> Should system programmers design their products so as to expose or to disguise the effects of rounding errors?

Some years ago, the following problem was set on a UWO examination:

> Find the rank of the matrix $\begin{pmatrix} 1 & 2/3 \\ 3 & 2 \end{pmatrix}$.

Of course the expected solution was for the students to see that multiplying the first row by 3 gave the second row, and hence the rank is 1. However, many students typed the matrix into their calculators and selected the menu item "rank". The calculator returned 2, because it worked with floating point numbers, and rounding errors had corrupted the solution. Students find a discussion of incorrect results from calculators (such as in [1]) to be shocking. Thus, this simple problem created a head-on collision between the students and numerical linear algebra. In this circumstance the teacher has several options: the obvious ones are not to use a calculator, or to ensure that the students will get correct results by setting only those problems that have been pre-tested on the calculator. This is awkward, now that there are so many menus and methods to try; it is difficult to anticipate just which menu item a student will press into service.

More options exist. The designers of software can give in to the temptation to disguise the problem by forcing tiny numbers to zero. The row reduction routines in MATLAB have such a default behaviour. It should be noted, though, that examples can be constructed showing that any fixed choice of "tiny" leads to incorrect answers. In the short term, such a quick fix saves the teacher much talking, but at the cost of failing to alert students to the dangers that will face them in the world of very large matrix problems. Instead of disguising the problem, one might incorporate some aspects of numerical analysis into the course. In this particular example, a numerical analyst would comment that rank is known to be a quantity that cannot be reliably computed, because it is ill-conditioned.

Yet another option is to switch to exact computation using a computer algebra (CA) package, or a CA enabled calculator. This raises the other spectre of linear algebra: *complexity*, in the computer science sense of the 'cost' of a computation. Computations using exact numbers grow in cost (with the size of the matrix) more quickly than floating-point computations, because the individual matrix elements grow in size. For example, if one tries to invert a matrix of integers, the numerator and denominator of each element will increase in length regardless of the magnitudes of the numbers being represented. As another example, if one tries to compute the eigenvectors of a general matrix, then for matrices up to 4×4, one gets nested radicals, while for larger systems the computation will be plagued by general algebraic numbers.

Once we start to compute, both stability and complexity of computation will force themselves on the attention of the student and teacher sooner or later. The only solution is never to venture outside the domain of 2×2 and 3×3 integer matrices. If, however, one is going to do that, then one of the most exciting possibilities that computers offer is lost, namely the chance for students to explore for themselves.

3.3 Never learning the best method

A book on canoeing [5] recommends that the first paddling stroke to teach students is the *back*stroke,

> because we note a tendency to revert to the most familiar stroke when flustered.

Many readers will have heard stories of students graduating and going to work in industry, and then applying mathematics from their undergraduate textbooks. Perhaps they try solving 100 equations in 100 unknowns using Cramer's rule, or searching for the eigenvalue of a large matrix by trying to solve its characteristic polynomial. With today's arbitrary precision software, they might even get the correct answer after a long wait.

We can ask

> how far into the future does your course project?

We happily teach children that "you cannot take 3 from 2" because we are confident that someone will later introduce them to negative numbers. No one suggests we should change this[3]. Should we insist that students learn Gaussian elimination with partial pivoting (insist on less lies-to-children); should software vendors program and sell systems offering Gaussian elimination without pivoting (computers enshrine lies-to-children)? In fact, in many introductory books, every computational procedure they teach requires later modification for the purposes of numerical linear algebra. Should we worry about this?

3.4 The answers

At UWO, we have modified the linear algebra that is taught to engineers; it now exhibits a more numerical slant. Thus we teach partial pivoting and least-squares solutions. Gram–Schmidt is linked to the QR factoring on the calculator. We demonstrate, to the students' horror, the possibility that computers will return a wrong answer; we do our best to persuade students to check their results using their calculators. In contrast, the linear algebra taught to our mathematicians remains more traditional. Ultimately, the discussion above combines with the personal tastes of the instructor to decide what action is taken. Again, the principle is that the presence of the computer will force these issues into the area of discussion.

4 Row reduction and symbolic systems

This section addresses the question

> should symbolic computation systems be allowed to ask teachers to modify time-honoured material in their courses?

Specifically, the time-honoured material is row reduction. The teaching of the Gauss-Jordan reduction of a matrix to reduced row-echelon form (RREF) is thoroughly engrained in North American linear algebra courses. A source of frustration, for students and teachers alike, is the fact that a lot of arithmetic is required and subtraction and division errors are common. The automation of this procedure, therefore, seems to be a natural application of computer technology. Unfortunately the spread of symbolic systems has created a problem for designers and teachers.

[3]It is interesting to wonder how many people there are in the world who are living their lives happily, not knowing anything about negative numbers.

Consider the plight of some adventurous students at UWO who decided to use a computer to solve some eigenvalue problems by an original method. Instead of computing the characteristic polynomial using $\det(A - \lambda I)$, they argued that the real problem was to find non-trivial solutions to $(A - \lambda I)x = 0$. Surely, they argued, we can simply row-reduce the matrix and see when it is singular. The idea is not so unusual, because many textbooks include problems in which they present a system of equations with one or more parameters, and the students must decide when the system fails to have a unique solution. Using their favorite CA system, the students typed in

$$\texttt{Find-Reduced-Row-Echelon-Form}\begin{pmatrix} 1-\lambda & 3 \\ -2 & 7-\lambda \end{pmatrix} .$$

Of course, all they got back was the identity matrix and were none the wiser.

There are several solutions to this problem. The first one is to say to the students "That's what you get for being adventurous. In future, never try to solve any problem in a way different from what I showed you in class". No one would want a teacher to say that. A second solution is to go to the writer of the CA system and demand that the system be rewritten using provisos, or special case analysis [2]. Thus, in the example, the system would return an identity matrix together with a statement that $\lambda^2 - 8\lambda + 13 \neq 0$. A third solution, however, is to modify the definition in the textbook, and the way that row reduction is taught, so that the computer systems can obey the definition and yet not lose information. Such an approach uses the Turing factors of a matrix.

4.1 Turing factors of a matrix

The definition of Reduced Row Echelon Form forces us to divide out pivots. In the example above, this forced us to discard the characteristic polynomial. The nub of the problem is that row reduction is a *transformation* of one matrix into another matrix, and the transformation forces us to throw away information. What if we define a new format that retains all information? The key observation is that everywhere else in modern linear algebra, people work by *factoring* a matrix. English speakers also use the term *decomposition* for a matrix factoring. Thus, in linear algebra, we find LU factors[4], QR factors and singular-value factors $U\Sigma V^T$ (usually called the SVD, instead of the SVF).

The new format for RREF is an extension of LU factoring. The LU factoring of a matrix has been generalized to a rectangular, symbolic matrix[3]; we have called this the 'Turing factoring' in honour of the remarkable paper published in 1948 by Alan Turing [7] in which he proved that row-reduction is equivalent to the following 'resolution into the product of matrices'

$$PA = LDU ,$$

where P is a permutation matrix, L is unit lower triangular, D is diagonal with nonzero entries, and U is unit upper triangular. Modern textbook writers prefer to write this as $PA = LU$, and either combine the D with the L (sometimes called Crout factoring) or with the U (Doolittle factoring). The choice is made in order to reproduce what the writers were taught when they were students.

Many textbooks and software packages consider LU factoring to be applicable only to invertible matrices. Thus in MATLAB version 5.3 (release 11), applying the function `lu` to a

[4]or factorizations if you are being paid by the letter.

non-square matrix produced an error message; in MATLAB 6 (release 12), this is no longer the case, but Mathematica and the HP still insist on an invertible matrix.

In fact, any rectangular matrix A has the Turing factors

$$PA = LDUR \ .$$

Here, the R matrix is the unique reduced row-echelon form of A. The matrix P is a preconditioning matrix; it is usually a permutation matrix, but it can be a more general matrix. For example, the standard software package LAPACK uses row and column equilibration (routines xGESVX) and this can be described using preconditioning matrices.

4.2 What can we do with this?

We return to the eigenvalue problem above and use Turing factoring:

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1-\lambda & 3 \\ -2 & 7-\lambda \end{pmatrix} =$$

$$\begin{pmatrix} 1 & 0 \\ \frac{1}{2}(\lambda-1) & 1 \end{pmatrix} \begin{pmatrix} -2 & 0 \\ 0 & \frac{1}{2}(\lambda^2-8\lambda+13) \end{pmatrix} \begin{pmatrix} 1 & \frac{1}{2}(\lambda-7) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Notice the rather useless row-echelon form at the end, and notice that the characteristic polynomial has appeared naturally in the D matrix. Students could be taught the simple rule "always check the cases $\det D = 0$ separately", but it is rewarding to understand where this rule comes from.

The importance of $\det D = 0$ comes because we are interested in special cases, and special cases are often points of discontinuity; indeed, this is why they are interesting and considered special. So we ask about the continuity properties of Turing factoring.

Consider the RREF of the matrix $A(k) = \begin{pmatrix} 1 & 0 \\ 0 & k \end{pmatrix}$ as the parameter k passes through 0. The RREF is the identity I, except for a sudden, discontinuous change at $k = 0$. Linear algebra courses do not usually discuss the limit of a matrix, but under any reasonable definition, the limit of RREF(A) as $k \to 0$ must be the unit matrix. So we have $\lim_{k\to 0} RREF(A) = I \neq A(0)$. Under any definition of continuity, the RREF of A is discontinuous at $k = 0$.

Definition. A matrix $A(x)$ is continuous at $x = a$ if each of its elements is continuous at $x = a$.

Once we become accustomed to thinking of (interesting) special cases as discontinuities, we can frame the following theorem.

Theorem: Let $A(x)$ be a matrix depending upon one or more variables or parameters x, and let A be continuous at a point $x = a$. For any fixed x, let $A(x)$ have the Turing factoring given by $P(x)A(x) = L(x)D(x)U(x)R(x)$. If $\det D(x) \neq 0$ in some neighbourhood of $x = a$, then $R(x)$, $L(x)$, $D(x)$, $U(x)$ are all continuous at $x = a$ and moreover $P(x)$ may be taken constant in a neighbourhood of $x = a$.

This theorem is proved in [3] and means two things.

- A CAS can give an RREF which contains **visible failure** built in. Places where an RREF might fail are no longer invisible because of the definition.
- The discontinuity information is collected in a single place, namely, along the diagonal of D.

To return to the eigenvalue problem, the Turing factors of a matrix $A - \lambda I$, will always lead to a diagonal matrix D in which the diagonal entries are

$$p_1(\lambda), p_2(\lambda)/p_1(\lambda), \ldots, p_n(\lambda)/p_{n-1}(\lambda)$$

where $p_k(\lambda)$ is a polynomial of degree k, and in the last entry, $p_n(\lambda)$ is the characteristic polynomial. Thus, $\det D = p_n(\lambda)$ and only the roots of the characteristic polynomial are special cases. For some matrices, a fraction p_k/p_{k-1} might simplify; this would simply mean that a preliminary splitting of the characteristic polynomial had been found during the computation.

4.3 The benefits

The immediate benefit to the teacher of Turing factoring is the combining together of row reduction and LU factoring. If LU factoring was not previously in the course material, then it comes along at no extra cost to the student. A common objection to Turing factoring is that it is "a bit rich" for beginning students. Its computation also threatens a great deal of computation. However, the point of computers is exactly to take over the burden of computation. Provided students know what Turing factors are, computer algebra systems can easily obtain them for the student.

The immediate benefit to the system designer is that a mechanism becomes available for returning special case information back to the user. This obviates the need to develop new user interfaces that allow the passing back to the user of proviso information. The benefit to the student is a gentle introduction to one of the most powerful ideas of modern linear algebra: factoring.

References

[1] Robert M. Corless. Six, lies, and calculators. *The American Mathematical Monthly*, 100(4):344–350, 1993.

[2] Robert M. Corless and David J. Jeffrey. Well, it isn't quite that simple. *SIGSAM Bulletin*, 26(3):2–6, 1992.

[3] Robert M. Corless and David J. Jeffrey. The Turing factorization of a rectangular matrix. *SIGSAM Bulletin*, 31(3):20–28, 1997.

[4] Roger A. Horn. *The Hadamard Product, In Proc. Sympos. Applied Maths (Charles R. Johnson, Ed.).* AMS Volume 40, 1990.

[5] Robert E. McNair. *Basic River Canoeing.* American Camping Association, Martinsville, Indiana, USA, 1972.

[6] Terry Pratchett, Ian Stewart, and Jack Cohen. *The Science of Discworld.* Ebury Press, 1999.

[7] Alan M. Turing. Rounding-off errors in matrix processes. *Quart. J. Mech. Appl. Math.*, 1:287–308, 1948.