THE USE OF THE COMPUTER IN MATHEMATICS TEACHING PAST HISTORY — FUTURE PROSPECTS

D. J. Evans

Initially, the digital computer was created to facilitate the solution of problems that required numerical computations which were, for all practical purposes, beyond or impossible under previous technologies. Subsequent developments of MAINFRAME COMPUTING have been similarly motivated by the need to solve increasingly complex scientific problems or by the need to manipulate huge quantities of information.

Digital computers first made their appearance on college and university campuses in the late 1950s. At that time virtually all computing was done in batch mode and programming using a scientific programming language such as FORTRAN or possibly a local dialect. Under these conditions, the time from submission to job return would be not less than 45 minutes. The forbidding nature of FORTRAN syntax and the complexity of its input-output statements required that a more "user friendly" language be created. As a result, alternative languages such as BASIC and WATFOR were created and extensively used by students.

The next major development in computing which affected teaching use was INTERACTIVE COMPUTING. Under this mode of operation, a user could be connected to a mainframe from a remote location, enter a program, and execute it. As a result the "turnaround" time between jobs became less than 5 minutes.

New applications of the computer to instruction and teaching also became possible soon after. Perhaps the major new development of concern to us today was COMPUTER AIDED INSTRUCTION (CAI) which quickly became a focal point of concern for

education researchers. The output from most of the programs at that time was still numerical or perhaps consisted of "computer graphics". Some instructors, however, began to use GRAPHICS terminals and their impact was immediately felt.

With the introduction of the MICROCOMPUTER in the late 1970s, graphics became widely available. More importantly, the microcomputer changed the locus of control from the campus computer center running large mainframes in a multiuser environment to a departmental computer laboratory controlled by the local teaching staff. With the introduction of the IBM personal computer and the Apple Macintosh processor, bit mapped screens soon became available for sophisticated graphics use as the amount of random access memory increased from 48K to 512K or more and the processors operated very much faster. The declining price for computers also made it possible for most high schools to have computers available for their students.

The developments I have outlined are of ever changing technology driven primarily by non-instructional needs. Nevertheless, as technology changes the instructional applications will also change. As we look to the future, the one thing we can be certain of is that the technology will continue to advance, and that computers will continue to decline in price. Thus we have today an environment in which our students arrive at college knowing how to use a computer and able to purchase a microcomputer for approximately £200 with a capability greater than many of the early mainframes. In addition, hand held calculators are now readily available with a programming and graphics capability equivalent to the early microcomputers. There can be no doubt that this technology will affect both the way we teach UNDERGRADUATE MATHEMATICS and the COURSE CONTENT.

As we look to the future, we see an ever changing technology which will continue to provide opportunities for teaching innovation. The advanced workstations of today will become the commonplace equipment of tomorrow. These machines will be linked by networks which will access fileservers, high quality printers, and gateways to external resources such as remote data bases and libraries. So instead of focusing on the use of computers to im-

prove the teaching of current mathematical topics, we can begin to explore simulated learning environments which will enable us to teach subject material previously thought to be too abstract or complex for undergraduates.

Computer Impact on Mathematics

In Mathematics computers first made their impact on *Numerical Analysis* and during the period 1956–75 the subject developed greatly, both in stature and discipline. *Statistics,* also being a numerate topic, quickly followed the same fate. It has only been recently, since the introduction of the more powerful supercomputer and parallel computers that we have seen a similar impact on *Algebra,* to be followed shortly with improved graphics and visualization technology on *Geometry.*

Discrete Mathematics

Discrete mathematics is what computers actually do. Therefore it should be compulsory to all our students if they are to achieve some proficiency and affinity with computers. It is the mathematics of finite and countable sets, and it includes topics taught throughout the standard secondary and college curricula. These topics include logic, set theory, combinatorics, discrete probability, functions and relations on discrete structures, induction, recursion, difference equations, graph theory, trees, algebraic structures, and linear algebra.

Finite Differences

The computer age has given new impetus to the method of finite differences, which treats problems of time evolution posed in discrete rather than continuous form. This is an old subject studied by Boole in the 19th century.

A discrete mathematics centered on difference equations is timely, not only because of the increased usage of computers, but because of the known shortcomings in the Infinitesimal Calculus.

The Calculus of Newton and Leibniz was designed to circumvent the difficulties of dealing with the discrete by the passage

to continuous limits. Sums became integrals, differences became differentials, and the computational labour of repeated additions and subtractions avoided by the power of the Infinitesimal Calculus. Another difficulty with traditional calculus is that many modern problems resist continuous methods. Even the student who succeeds with calculus needs to appreciate discrete approximation schemes which can be implemented on the computer. Such schemes are increasingly important for the solution of the analytically intractable differential equations arising in many applications in the real world.

Discrete mathematics also creates a link between ideas and techniques of computer science and important and useful mathematical notions. An elementary course on the subject can even bring students into contact with active research in mathematics, physics, chemistry, and other areas.

Linear Algebra

The topic which has been most affected by present day computers is Linear Algebra. What then may the main topics in a Linear Algebra course of the future become? Surely the elementary theory concerning the concepts of linear independence, span, basis, and dimension will remain fundamental, and properties of the algebra of matrices and linear transformations will not lose their importance. Also the geometry of vectors will continue to provide important insights and examples. The analysis of linear systems of equations and the investigation of eigenvectors and eigenvalues will require added emphasis, since the computer software allows us to ask so many more interesting questions involving these objects.

However, reduction methods and echelon forms will need a different approach, since the software and algorithms which carry out the necessary computation are often quite different from those now taught. Without doubt, reduction to upper triangular form, possibly using partial pivoting (and followed by back substitution if there is an equation to be solved) should be the main hand computation approach, as this is similar to the LU-decomposition that a good Linear Algebra software package uses.

Methods for computing matrix inverses and detailed discus-

sions of determinants probably require less attention. Since applications rarely require the computation of a matrix inverse it is always better to solve the related linear system. Also determinants have decreased in importance since modern algorithms for approximating eigenvalues and eigenvectors make no use of them. Furthermore, two other applications of the determinant, Cramer's Rule and the adjoint formula for the matrix inverse have become even more superfluous since the computation is done by software which makes no use of them.

There should also be a subtle alteration in emphasis throughout the course. Instead of paying close attention to the elements of a matrix, a point of view that is reinforced by hand computation, the properties of the matrix as an entity should be stressed.

Finally, the major addition to the Linear Algebra course should be the study of applications. Interesting applications that lead to linear systems of equations should be studied, i.e. temperature distributions found by approximating values at discrete grid points, input-output models in economics, electrical circuit analysis, least-squares approximation, balancing chemical reactions, and network analysis. Applications that involve locating eigenvalues and eigenvectors include: Markov chains, biological population models, and models of genetic inheritance. If there is time to study first-order linear systems of differential equations, many more applications become within reach.

Numerical Solution of Systems of Equations

Methods of solving systems of *equations* are divided into (i) direct and (ii) indirect, or iterative, methods. For linear equations the direct methods include the well known Gaussian elimination process, the indirect methods include the Gauss-Seidel method.

The direct methods have the advantages (a) that they will always produce the solution provided that it exists, is unique and that sufficient accuracy is retained at each and every stage, and (b) that the solution is found after a known number of operations. They have the disadvantage that very sparse systems of equations, such as those which arise in finite difference/element approximations to differential equations, may become rapidly less

sparse as the elimination process proceeds so raising the storage requirement from a multiple of n (for n equations) to something like n^2.

The iterative methods, on the other hand, may fail to converge to a solution and if they do converge it is not obvious how many operations they will require to produce the desired accuracy. They have however, the very considerable advantage that they are very well suited to computers and preserve the sparsity of the coefficient matrix throughout.

Direct methods for the numerical solution of *non-linear systems* are rarely available; there is, after all, no direct method for solving the general polynomial of even the fifth degree and so iterative methods are generally used. As in the case of linear systems, convergence may not always occur, though conditions sufficient to ensure convergence are usually known; and although in some cases the number of iteration necessary to produce convergence to a specified accuracy may not be easily predicted, it is frequently not a matter of great importance. However accelerating techniques can often be used if time is limited.

The revival of interest in iterative methods brought about by the use of computers has led to significant advances in the study of functions which are iteratively defined, e.g. by a nonlinear relation of the type

$$Z_{n+1} = F(Z_n)$$

where Z_0 is a given complex number and the function $F(Z)$ may contain one or more parameters. Some simple functions of this type are the quadratic equation

$$az^2 + bz + c = 0 \qquad (1)$$

By rearranging terms and a change of variable we can express this in quadratic iteration form, i.e.

$$Z_{n+1} = Z_n^2 + C, \qquad (2)$$

from which there are 3 possibilities:

1. The sequence Z_n converges to a limit α which is the solution of (1).
2. The sequence Z_n does not converge but the points Z_n remain bounded.
3. The points Z_n eventually move outside any bounded region.

In general all 3 cases can occur. Moreover, the complex values of C for which the sequence starting with $Z_0 = 0$ is either of type 1 or of type 2 form the well known Mandelbrot set, which has been the topic of much research recently.

Algorithms

An algorithm is simply a procedure for solving a specific problem or class of problems. The idea of an algorithm has been around for over 2000 years (e.g. the Euclidean Algorithm for finding the highest common factor of two integers) but it has attracted much greater interest in recent years following the introduction of computers and their application not only in mathematics but also to problems arising in technology, automation, business, commerce, economics, social sciences, etc.

Computer algorithms have been developed for many commonly occurring types of problem. In some cases several algorithms have been produced to solve the same problem, e.g. to sort a file of names into alphabetical order or to invert a matrix, and in such cases people who wish to use an algorithm will not only want to be sure that the algorithm will do what it is supposed to do, but also which of the several algorithms available is, in some sense, the "best" for their purposes. An algorithm which economizes on processor time may be extravagant in its use of storage space or vice-versa and the need to find algorithms which are optimal, or at least efficient, with respect to one or more parameters has led to the development of Complexity Theory. For instance the Fast Fourier Transform has reduced the time complexity from order n^2 to order $n \log n$, which is of considerable practical importance for large values of n.

An important aim in algorithm design is to ensure that the algorithm is "robust" i.e is guaranteed to produce the required answer under as wide a variety of conditions as possible.

However, the interest is still (and always has been) on reliable methods which always converge. For instance the quadratic equation (1) when a, b and c are real can be solved in many ways. However from the Theory of Equations we have the roots α_1, α_2 are given by

$$\alpha_1 + \alpha_2 = -\frac{b}{a} \quad \text{and} \quad \alpha_1\alpha_2 = \frac{c}{a}$$

which when written in iterative form

$$\alpha_1^{(n+1)} = \frac{b}{a} - \alpha_2^{(n)} \quad \text{and} \quad \alpha_2^{(n+1)} = \frac{c}{a\alpha_1^{(n+1)}} \tag{3}$$

where $\alpha_2^{(0)}$ is given, give a convergent algorithm and is preferable to the quadratic iteration form (2).

The direct methods too should be reconsidered in the demanding circumstances of present day software requirements. A simple program just using the formula

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

will just not do. A robust software package should check its input for validity as well as all the possible variations of the formula, i.e. when the roots are complex, etc. Also it must generate sufficient error messages so as to leave no doubt in the user's mind. Thus a complete flow diagram for the direct solution of the quadratic equation (1) is shown in Fig. 1.

Another area of great importance is the acceleration of the convergence of iterative processes especially for the large systems of linear equations which occur in scientific problems.

Given the trivial (2×2) system $Ax = b$ where A is symmetric and positive definite, i.e.

$$A = \begin{pmatrix} 1 & -a \\ -a & 1 \end{pmatrix}$$

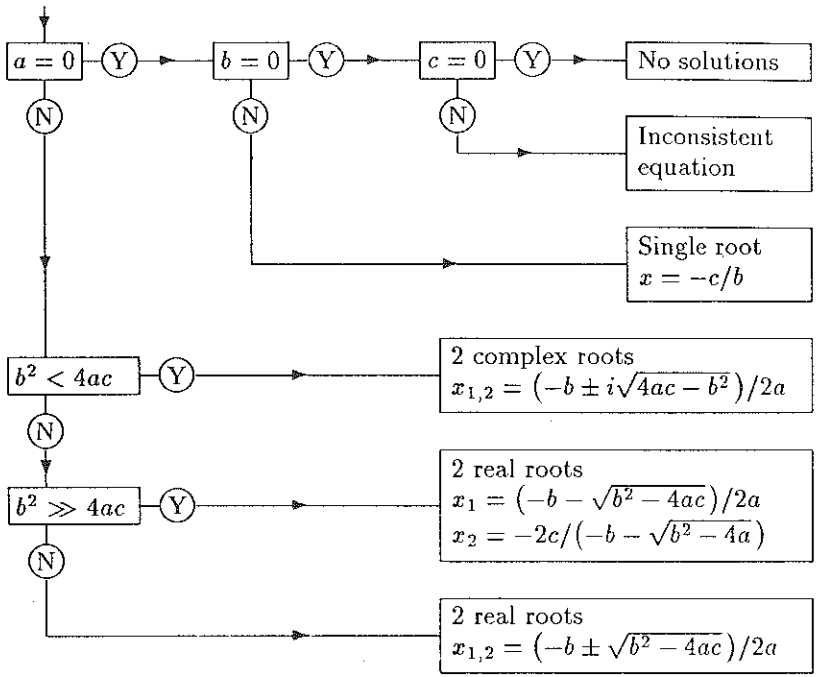

FIGURE 1

then the well known Gauss-Seidel method

$$x_1^{(k+1)} = ax_2^{(k)} + b_1$$
$$x_2^{(k+1)} = ax_1^{(k+1)} + b_2$$

is known to give acceptable convergence unless $a = 1$. It is usual to apply the successive overrelaxation (SOR) method, i.e.

$$x_1^{(k+1)} = x_1^{(k)} + \omega_1 \left(ax_2^{(k)} + b_1 - x_1^{(k)}\right)$$
$$x_2^{(k+1)} = x_2^{(k)} + \omega_1 \left(ax_1^{(k+1)} + b_2 - x_2^{(k)}\right)$$

where the overrelaxation parameter ω_1 is chosen from the formula

$$\omega_1 = \frac{2}{1 + \sqrt{1 - a^2}}$$

to obtain an extremely rapid convergence when a is very close to 1.

An even more rapid convergence is obtained when A is skew symmetric, i.e.

$$A = \begin{pmatrix} 1 & a \\ -a & 1 \end{pmatrix}.$$

Now the successive underrelaxation (SUR) method becomes

$$x_1^{(k+1)} = x_1^{(k)} + \omega_2 \left(-ax_2^{(k)} + b_1 - x_1^{(k)}\right)$$
$$x_2^{(k+1)} = x_2^{(k)} + \omega_2 \left(ax_1^{(k+1)} + b_2 - x_2^{(k)}\right)$$

where the optimal acceleration parameter ω_2 is now given by

$$\omega_2 = \frac{2}{1 + \sqrt{1 + a^2}}.$$

A comparison of the results for the 3 methods is given in Table 1, where N is the number of iterations required to achieve an accuracy of 10^{-6}.

a	Gauss-Seidel		SOR		SUR	
	$\omega = 1$	N	ω_1	N	ω_2	N
0.8090	1	33	1.2596	11	0.8748	7
0.9969	1	2,238	1.8545	88	0.8292	8
0.9988	1	5,732	1.9065	141	0.8288	8

TABLE 1: A comparison of the convergence rates of the Gauss-Seidel, SOR and SUR methods.

Recursive Algorithms

Algorithm design is an important topic which we should teach to our students — especially how to construct a recursive algorithm and in the following we show the details of a recursive parallel algorithm design.

In the numerical solution of partial differential equations by the implicit methods there occurs the problem of repeatedly solving linear systems involving tridiagonal matrices possessing diagonal dominance. Current algorithmic solution methods involve a Gaussian elimination of the matrix equation to upper triangular form with unit diagonal entries, from which the solution vector can be easily obtained by a back substitution process. In algorithmic form, we calculate the quantities

$$\begin{aligned} g_1 &= \frac{c_1}{b_1}, \quad g_i = \frac{c_i}{b_i - a_i g_{i-1}}, \quad i = 2, 3, \ldots, n-1 \\ h_1 &= \frac{d_1}{b_1}, \quad h_i = \frac{d_i - a_i h_{i-1}}{b_i - a_i g_{i-1}}, \quad i = 2, 3, \ldots, n, \end{aligned} \tag{4a}$$

and the solution is given by

$$x_n = h_n, \quad x_i = h_i - g_i x_{i+1}, \quad i = n-1, n-2, \ldots, 2, 1. \tag{4b}$$

However, it is well known that such back substitution processes (4b) are more ideally suited for serial computers and nowadays with the ever increasing usage of parallelism in algorithms it is necessary to investigate whether a more efficient parallel algorithm based on the Gauss-Jordan method can be formulated.

We now consider the $(n \times n)$ tridiagonal system given by

$$\begin{pmatrix} b_1 & c_1 & & & \\ a_2 & b_2 & c_2 & & 0 \\ & \ddots & \ddots & \ddots & \\ 0 & & \ddots & \ddots & c_{n-1} \\ & & & a_n & b_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ \vdots \\ d_n \end{pmatrix} \tag{5}$$

and since $b_i > a_i + c_i$, $i = 1, 2, \ldots, n$ then we are assured that no pivoting is required in any ensuing elimination process and hence

the tridiagonal matrix structure will be maintained in successive eliminations.

Initially the first equation is normalized by setting

$$g_1 = \frac{c_1}{b_1}, \quad h_{1,0} = \frac{d_1}{b_1}. \tag{6}$$

Then, the coefficient of x_1 in the second equation is eliminated by multiplying the first equation by $-a_2$ and adding to the second equation, i.e.

$$\begin{pmatrix} 1 & g_1 & & & \\ 0 & b_2 - a_2 g_1 & c_2 & & 0 \\ & a_3 & b_3 & c_3 & \\ & 0 & \ddots & \ddots & \ddots \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \end{pmatrix} = \begin{pmatrix} h_{1,0} \\ d_2 - a_2 h_{1,0} \\ \vdots \\ \vdots \end{pmatrix} \tag{7}$$

which is then normalized by setting

$$g_2 = \frac{c_2}{b_2 - a_2 g_1}, \quad h_{2,0} = \frac{d_2 - a_2 h_{1,0}}{b_2 - a_2 g_1}.$$

From now on the Gauss-Jordan elimination proceeds differently and eliminates coefficients both below and above the diagonal as follows:

The coefficient of x_2 in the third equation is eliminated by multiplying the second equation by $-a_3$ and adding to the third equation, and the coefficient of x_2 in the first equation is eliminated by multiplying the second equation by $-g_1$ and adding to the first equation, i.e.

$$\begin{pmatrix} 1 & 0 & -g_1 g_2 & & & \\ 0 & 1 & g_2 & & & 0 \\ & & b_3 - a_3 g_2 & c_3 & & \\ & & a_4 & b_4 & c_4 & \\ & 0 & & \ddots & \ddots & \ddots \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ \vdots \end{pmatrix} = \begin{pmatrix} h_{1,0} - g_1 h_{2,0} \\ h_{2,0} \\ d_3 - a_3 h_{2,0} \\ \vdots \\ \vdots \end{pmatrix} \tag{8}$$

which is similarly normalized by setting

$$g_3 = \frac{c_3}{b_3 - a_3 g_2}, \quad h_{3,0} = \frac{d_3 - a_3 h_{2,0}}{b_3 - a_3 g_2}, \quad \text{and} \quad h_{1,1} = h_{1,0} - g_1 h_{2,0}.$$

A further step of elimination illustrates further the pattern of the algorithm. The coefficient of x_3 in the fourth equation is eliminated by multiplying the third equation by $-a_4$ and adding to the fourth equation, the coefficient of x_3 in the second equation is eliminated by multiplying the third equation by $-g_2$ and adding to the second equation, the coefficient of x_3 in the first equation is eliminated by multiplying the third equation by g_1g_2 and adding to the first equation, i.e.

$$\begin{pmatrix} 1 & 0 & 0 & g_1g_2g_3 & & & \\ & 1 & 0 & -g_2g_3 & & 0 & \\ & & 1 & g_3 & & & \\ & & & b_4 - a_4g_3 & c_4 & & \\ & 0 & & a_5 & b_5 & c_5 & \\ & & & & \ddots & \ddots & \ddots \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ \vdots \\ \vdots \end{pmatrix} = \begin{pmatrix} h_{1,1} + g_1g_2h_{3,0} \\ h_{2,0} - g_2h_{3,0} \\ h_{3,0} \\ d_4 - a_4h_{3,0} \\ \vdots \\ \vdots \\ \vdots \end{pmatrix} \tag{9}$$

which is similarly normalized by setting

$$g_4 = \frac{c_4}{b_4 - a_4g_3}, \quad h_{4,0} = \frac{d_4 - a_4h_{3,0}}{b_4 - a_4g_3},$$
$$h_{1,2} = h_{1,1} + g_1g_2h_{2,0}, \quad \text{and} \quad h_{2,1} = h_{2,0} - g_2h_{3,0} \tag{10}$$

By continuing in a similar manner for the rows $3, 4, \ldots, n$, it can be verified that ultimately the system (5) is transformed to the form

$$\begin{pmatrix} 1 & & & \\ & 1 & & 0 \\ & & \ddots & \\ & 0 & & \ddots \\ & & & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} h_{1,n-1} \\ h_{2,n-2} \\ \vdots \\ h_{n-1,1} \\ h_{n,0} \end{pmatrix} \tag{11}$$

from which we can see that the original tridiagonal system is now completely decoupled and the solution is immediately available.

Thus, to summarize the algorithmic process, we calculate the quantities

$$g_1 = \frac{c_1}{b_1}, \quad h_{1,0} = \frac{d_1}{b_1},$$

$$g_i = \frac{c_i}{b_i - a_ig_{i-1}}, \quad i = 2, 3, \ldots, n-1 \tag{12}$$
$$h_{i,0} = \frac{d_i - a_ih_{i-1,0}}{b_i - a_ig_{i-1}}, \quad i = 2, 3, \ldots, n.$$

Then, for $i = 3, 4, \ldots, n-1$, $k = i-2, i-3, \ldots, 2, 1$,

$$h_{k,i-k-1} = h_{k,i-k-2} + (-1)^{i-k-1} \prod_{j=k}^{j=i-2} g_j h_{i-1,0}.$$

Finally, the solution vector x is given by

$$x_i = h_{i,n-i}, \quad i = 1, 2, \ldots, n.$$

Thus, by using column sweep techniques which can be completed in parallel as the algorithm proceeds we are able to eliminate the recursive back substitution process completely from the computation.

The application of this direct method to the numerical solution of matrix equations arising from finite difference approximations to elliptic partial differential equations in two and higher dimensions can be made in the following manner. For two dimensional problems, these finite difference approximations produce matrix equations of the form $AX = D$, where the matrix A has the form

$$A = \begin{pmatrix} B_1 & C_1 & & & \\ A_2 & B_2 & C_2 & & 0 \\ & \ddots & \ddots & \ddots & \\ 0 & & \ddots & \ddots & C_{N-1} \\ & & & A_N & B_N \end{pmatrix}. \tag{13}$$

Here, the square submatrices B_j are of order n_j, where n_j corresponds to the number of mesh points on the jth horizontal mesh line of the discrete problem. The direct inversion method (12) can be immediately generalized so as to apply to $AX = D$. Indeed, if the vectors X and D are partitioned relative to the matrix A of

(13), then we define

$$\begin{aligned} G_1 &= B_1^{-1}C_1, \quad H_{1,0} = B_1^{-1}D_1, \\ G_i &= (B_i - A_iG_{i-1})^{-1}C_i, \quad i = 2,3,\ldots,N-1 \\ H_{i,0} &= (B_i - A_iG_{i-1})^{-1}(D_i - A_iH_{i-1,0}), \quad i = 2,3,\ldots,N. \end{aligned} \tag{14}$$

Then, for $i = 3,4,\ldots,n-1$, $k = i-2, i-3,\ldots,1$,

$$H_{k,i-k-1} = H_{k,i-k-2} + (-1)^{i-k-1}\prod_{j=k}^{j=i-2} G_jH_{i-1,0},$$

and the vector components X_i of the solution are given by

$$X_i = H_{i,n-i}, \quad i = 1,2,3,\ldots,n.$$

Parallel Algorithms

More recently the problem of designing algorithms which can be efficiently run on several processors working in parallel has attracted considerable interest. Algorithms which are ideal on a single processor may be highly inefficient, or even fail entirely on parallel processors and the design of suitable parallel algorithms for even the commonest problems is a matter for present day research.

Conclusions

The power of computers has given us the following opportunities:

i) to make new discoveries in Mathematics;
ii) in the teaching of Mathematics itself;
iii) to develop new methods (algorithms) which are efficient on computers for the solution of a wide range of problems and particularly so on parallel computers.

D. J. Evans,
Parallel Algorithms Research Centre,
University of Technology,
Loughborough,
Leicestershire LE11 3TU,
England.

CAUCHY'S MATRIX, THE VANDERMONDE MATRIX AND POLYNOMIAL INTERPRETATION

R. Gow

Let K be a field and let $\alpha_1,\ldots,\alpha_n$ be elements of K. The $n \times n$ matrix $V = V(\alpha_1,\ldots,\alpha_n)$, where

$$V = \begin{pmatrix} 1 & 1 & \ldots & 1 \\ \alpha_1 & \alpha_2 & \ldots & \alpha_n \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{n-1} & \alpha_2^{n-1} & \ldots & \alpha_n^{n-1} \end{pmatrix},$$

is called a Vandermonde matrix. It is an example of a type of matrix known as an *alternant*. See, for example, Chapter XI of [5]. The Vandermonde matrix plays an important role in problems concerning polynomials, symmetric polynomials in particular. The determinant of V is well known to be the difference product

$$\prod_{i>j}(\alpha_i - \alpha_j)$$

and thus V is invertible precisely when the α_i are all different. A proof that $\det V$ has the form stated above may be given as follows. Row operations show that $\det V$ equals the determinant of the $n \times n$ matrix obtained from V by replacing its last row by the row

$$(\, f(\alpha_1) \quad f(\alpha_2) \quad \ldots \quad f(\alpha_n)\,),$$

where f is any monic polynomial in $K[x]$ of degree $n-1$. We choose f to equal

$$(x-\alpha_1)\ldots(x-\alpha_{n-1}).$$