Constructing the convex hull of a set of points in the plane

P. J. Green* and B. W. Silverman†

A highly efficient algorithm for computing the vertices of the convex hull of a set of points in the plane is described and compared with existing methods. Empirical timings of a FORTRAN implementation of the algorithm show it to be faster than previous methods for most configurations of points. Careful consideration is given throughout to ensure that degenerate configurations, containing collinear or near-collinear points, are handled correctly.

(Received June 1978)

1. Introduction

The construction of the convex hull of a finite set of points in the plane seems initially to be fairly trivial. However, it is by no means obvious how to devise an algorithm which will be efficient for this task, one of many more easily done by human eye and brain than by a computer program. The method we shall describe is an improvement of the algorithm described by Eddy (1977) and was developed independently. Our implementation was found to be appreciably faster than that of Eddy, particularly for small or moderate samples. In addition our method makes some attempt to deal with 'degeneracies' caused by rounding errors. In Section 5 we review some previous methods, all of which were found to be considerably slower than ours for almost all configurations of points.

A related requirement, of particular interest in statistics, is to 'peel' the set of points by finding the points on the convex hull, removing them, finding the points on the convex hull of the remaining set, removing them, and so on until there are no more points left. Points which are removed at an early stage are near the edge of the configuration and are possible outliers; the last point (or the mean of the last points) to be removed is analogous to the median in the one-dimensional case. Indeed it is possible to use the stage at which a point is removed as the analogue of the rank of a univariate observation; these are the 'c-ranks' of Barnett (1976). Our convex hull algorithm can easily be adapted to peel a data set in this way. Two planar configurations, with the successive peels marked, are illustrated in **Figs. 1 and 2.** Note that our convention is that successive collinear points at the periphery of a configuration are *all* counted as hull vertices.

2. The algorithm

Suppose we are given n points in the plane. These points will be denoted by their index numbers $1, 2, \ldots, n$. The representation we shall require of the hull is a list of the index numbers of the vertices of the hull, in anticlockwise order round the hull, but starting from no particular point. Given this list, it is a trivial matter to plot the hull and almost as simple, for example, to discover whether or not a test point is inside the hull.

We shall first define some geometrical terms; these are illustrated in **Fig. 3.** Given an ordered pair of points ij, a further point k_1 is said to be *outside ij* if the triangle ijk_1 is oriented negatively (clockwise). If ijk_2 is anticlockwise then k_2 will be said to be *inside ij*; if i, j and k_3 are collinear then k_3 is *on ij*, and if in addition ik_3j occur in that order on the line, k_3 is *between ij*.

The orientation of the triangle ijk is most easily determined by the sign of its area; note that, for fixed ij, the area is proportional to the directed length of the perpendicular from k to the line ij. Any point k of the configuration yielding the most negative value of the area ijk, and therefore whose perpendicular distance outside ij is greatest, is called *furthest outside ij*. A *test list* $[ij]$ is a pair ij, together with the set of points, if any, between or outside ij, a point furthest outside ij being marked. This test list is an *edge ij* of the hull if there are no points between or outside ij, that is, if it consists only of the pair ij. The algorithm proceeds by scanning the test lists; each test list is either deleted and replaced by other test lists, or is confirmed as an edge, as in the following procedure.

Step 1
Find any two points i and j certain to be on the hull, for example those with largest and smallest x-coordinates. Form the test lists $[ij]$ and $[ji]$.

Step 2
Consider any test list $[ij]$. If there are no points outside ij, confirm ij as an edge. Otherwise suppose k is furthest outside ij; then replace the test list $[ij]$ by forming new test lists $[ik]$ and $[kj]$.

Step 3
If there are any remaining test lists not confirmed as edges or deleted, repeat step 2. Otherwise, all the edges are now found, and it is simple to list them in cyclic order.

The procedure is illustrated in **Fig. 4**; test lists subsequently deleted are indicated as pairs linked by broken lines, while those confirmed as edges are joined by solid lines.

3. Numerical considerations

The only arithmetic operations required by the algorithm are the computing and comparing of signed areas of triangles specified by the coordinates of their vertices. These can lead to problems of numerical accuracy, particularly in the presence of degeneracy or near-degeneracy; a typical degeneracy is the occurrence of three or more collinear points. It is possible, for example, in the case of four nearly collinear points, for the inconsistency illustrated in **Fig. 5** to appear. Suppose that, on a previous application of Step 2, k was found to be furthest outside ij, and that l was the only other outside point. When $[ik]$ is considered as a test list, it is possible that l may be found to be outside it, leading to confirmed edges il, lk and kj as shown in Fig. 5. This is of course geometrically impossible, but can occur numerically because of rounding errors.

To avoid difficulties of this kind, a tolerance parameter ε is introduced. Areas numerically less than ε are considered to be zero, and so points k such that the area of the triangle ijk has magnitude less than ε are taken to be on ij.

*University of Bath: now at Department of Mathematics, University of Durham, South Road, Durham, DH1 3LE
†University of Oxford: now at School of Mathematics, University of Bath, Claverton Down, Bath, BA2 7AY

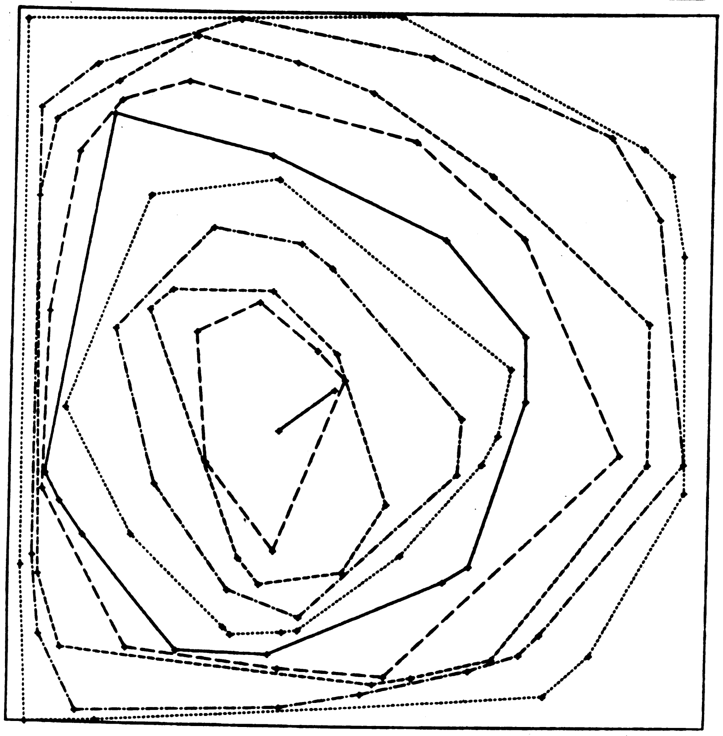
Fig. 1 A set of 100 random points, showing the convex hull peels

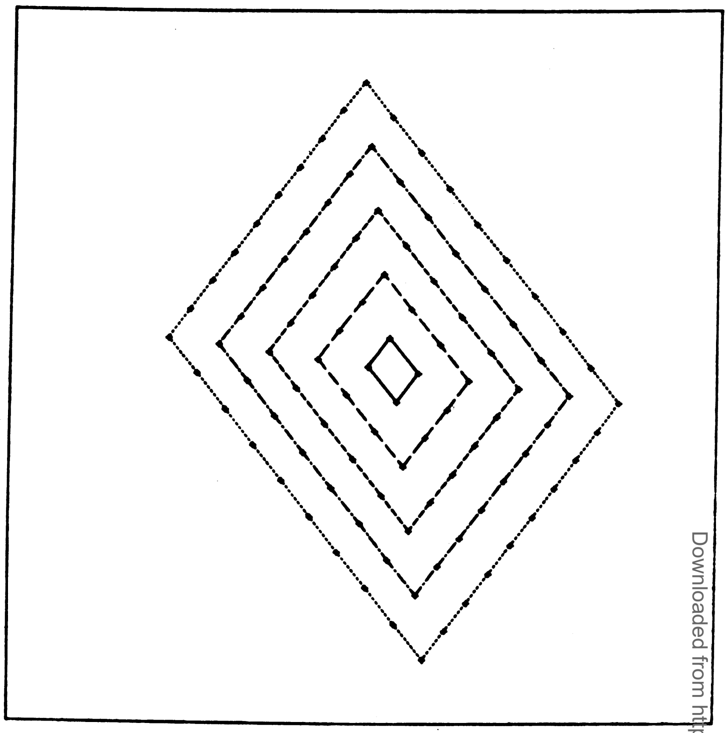
Fig. 2 An oblique lattice of 100 points, showing the convex hull peels

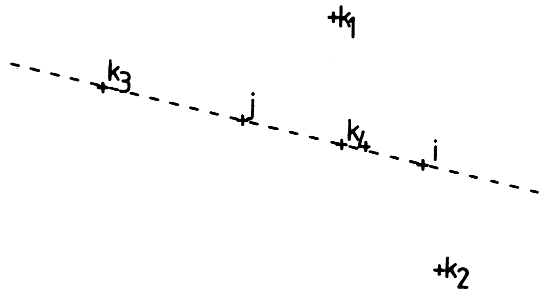

Fig. 3 Illustrating the geometrical definitions: K_1, K_2, K_3, K_4 are respectively outside, inside, on and between ij

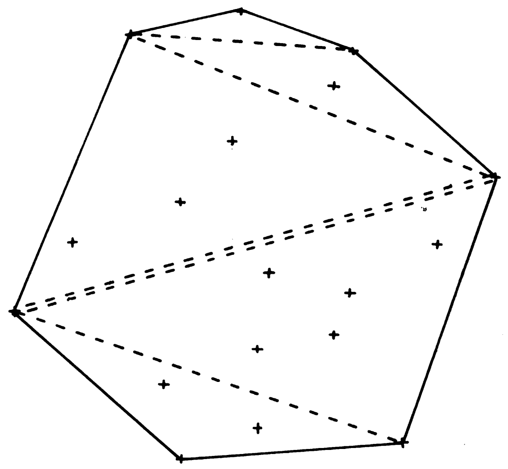
Fig. 4 Illustrating the algorithm of Method GS

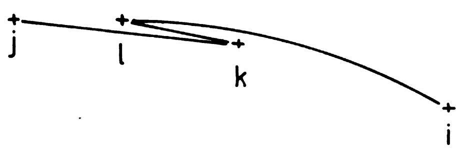

Fig. 5 Inconsistency arising from a near degeneracy

Thus all the 'doubtful' points are included in the new test list. When the test list is complete, it is flagged as degenerate if it consists only of doubtful points—in this case they are treated as collinear, and their order along the line determined, if necessary, by a standard sorting method. Increasing the value of ε makes the algorithm less sensitive—that is, more likely to find collinearities not actually present in the data—but less likely to produce inconsistencies of the kind described. The effect of varying ε is illustrated in **Table 1**, where the two data sets considered are those of Figs. 1 and 2. The random data consists of points drawn pseudo-randomly from a uniform distribution on the unit square, while the lattice data is constructed to lie on a lattice as shown. The table shows which values of ε give the 'correct' hull in each case, on two computers of different wordlength. Eddy's method does not give the correct hull for the lattice data.

In cases where degeneracy is likely to occur, we recommend, as a final check, testing that the edges of the hull are directed in a constant anticlockwise direction around the configuration, so that the hull is plausible. For data arising from random phenomena, degeneracy is unlikely to occur and setting ε to zero should provide satisfactory results.

4. Implementation and data structures

The authors have prepared a FORTRAN implementation of the algorithm described above. In designing this, it transpired that the execution time is highly dependent on the details of the implementation, particularly for configurations of realistic size. This point is further illustrated by the comparisons described in Section 6. We shall therefore describe our implementation in some detail. We found that the cost in execution time and storage space of enabling subsequent calls to the routine to compute successive peels of the data, as described in Section 1, was negligible, and so this feature has been included.

The storage requirement for n points, in addition to the $2n$ REAL locations for the point coordinates, is of two INTEGER arrays of length n and an INTEGER stack of dimensions $(3, m)$. The value of m required is difficult to predict in advance, but it is sufficient if it equals the number of peels required plus

Table 1

Data set	*Machine*	
Random (Fig. 1)	IBM 370	False collinearities found if $\varepsilon \geqslant 10^{-4}$
	CDC 7600	False collinearities found if $\varepsilon \geqslant 10^{-5}$
Lattice (Fig. 2)	IBM 370	Method fails if $\varepsilon \leqslant 10^{-8}$
	CDC 7600	Method fails if $\varepsilon \leqslant 10^{-10}$

Notes
Only powers of 10 were considered as values for ε.
All computations were performed in single precision FORTRAN.

the maximum number of points found to lie on any one of these peels; thus m will almost always be of smaller order than n.

At an intermediate stage in the computation there is a need to store two types of information; records of peels already constructed and records of test lists for the current peel. Both types of record consist of a fixed-length label, a triple of integers stored in the stack M, and a variable length list addressed by the label. In the case of a peel record, the list is held in one of the integer arrays LIST, and consists of the indices of the points on the relevant peel, in anticlockwise order. The label for such a record is (address in LIST, length in LIST, degeneracy flag). The segments of LIST thus referenced occupy consecutive elements at the beginning of LIST, the remainder of the array containing the indices of points not yet encountered, that is those interior to the latest peel found.

The labels occupy consecutive rows at the beginning of the stack M, and the remaining space in the stack is used to label test lists for the current peel. For test list $[ij]$ the label is of the form (i, j, address) where 'address' is a pointer to a data item in the second integer array NWK. For a test list confirmed as an edge, this address is set to zero. A data item in NWK is of variable length, beginning at the element addressed by the label. This element contains the length of the remainder of the item, which consists of a list of the points found to be outside the line ij, beginning with a furthest outside point. If the test list is degenerate, in the sense described in Section 3, and has length at least two, the length is flagged with a minus sign. The program is organised as follows; it is entered once for each peel required.

Step 1
If there are fewer than three points left, a trivial computation is needed; then branch to step 6.

Step 2
Scan the coordinates of the points listed at the tail end of LIST to find those with minimum and maximum x-coordinate. If these extreme coordinates are the same, the points are all collinear; sort the tail of LIST by y coordinate and branch to step 6.

Step 3
Denoting the extreme points i and j, form the test lists $[ij]$ and $[ji]$ by copying point indices from the tail of list LIST into two lists at either end of NWK. Set the lengths and 'furthest outside' locations, and set up test lists record labels in the stack. Note that all of NWK is occupied at this point for the first peel; two length items and $n - 2$ point indices are stored.

Step 4
For each remaining unconfirmed test list $[ij]$ the list is either

(*a*) of length one: in this case it is trivial to form new test lists $[ik]$ and $[kj]$ say: both are immediately confirmed as edges

(*b*) degenerate, with length greater than one: sort the immediate points by an appropriate coordinate, and form new test lists $[i\,k_1]$, $[k_1\,k_2]$, . . . , $[k_m\,j]$ say, all immediately confirmed as edges, or

(*c*) non-degenerate with length greater than one: suppose the point furthest outside is k. Form new test lists $[ik]$ and $[kj]$ by splitting the record in NWK into those points (i) outside ik (these overwrite the beginning of the item in NWK) (ii) outside kj (these overwrite the end) and (iii) outside neither (these are written back into the tail of LIST.) Then complete the test list records by setting the length and 'furthest outside' locations and writing new test list labels in the stack. Note that in the worst possible case (all points outside either ik or kj) the whole of the old list in NWK will be overwritten; there is one fewer point (k) to list, but one more length location.

Step 5
When all test lists remaining are confirmed as edges, scan the stack to form an anticlockwise list in LIST of the peel vertex indices.

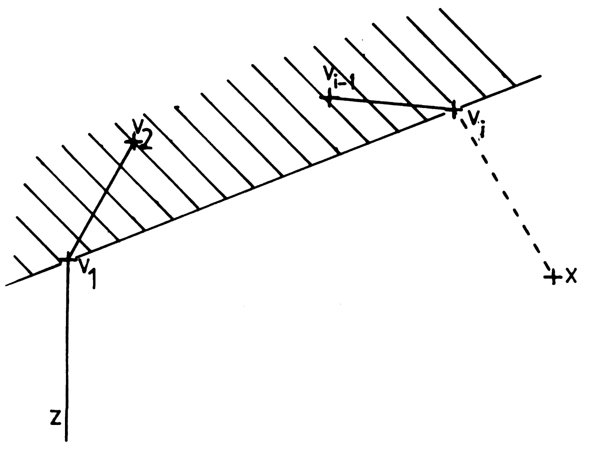

Fig. 6 **Illustrating Methods J1 and J2**

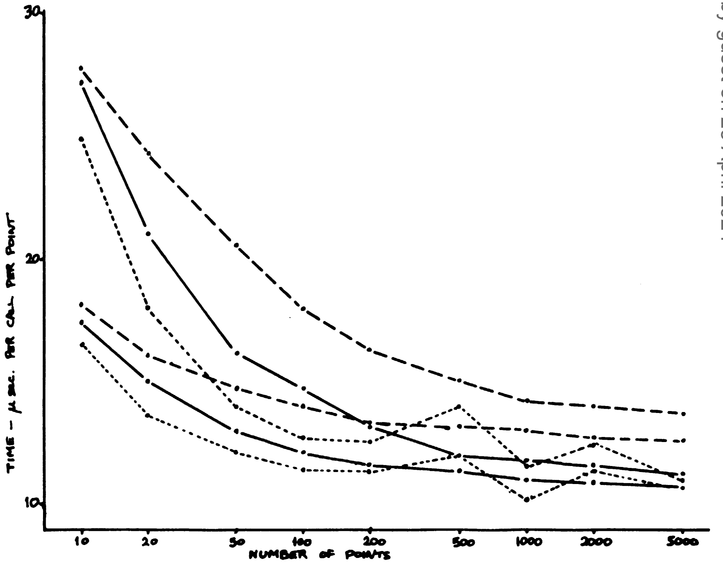

Fig. 7 **The relationship between execution time and number of points in the configuration, for Methods GS (lower curves) and E (upper curves). The distributions are those of Table 2: Uniform (broken lines), Normal (solid lines) and Cauchy (dotted lines)**

Table 2

		Uniform		*Normal*		*Cauchy*	
Number of points		100	1,000	100	1,000	100	1,000
Number of realisations		10	5	10	5	10	5
Number of repetitions		10	1	10	1	10	1
Average number of vertices		15·1	31·0	9·5	12·8	4·7	5·0
Average time in milliseconds for one call to method	GS	1·40	13·0	1·21	11·0	1·14	10·2
	E	1·80	14·2	1·47	11·8	1·27	11·6
	G	2·70	46·2	2·70	45·4	2·68	45·0
	J1	31·73	649·8	19·95	266·6	10·03	106·8
	J2	11·55	204·8	8·17	115·0	5·64	67·8
	J3	4·84	86·2	4·42	72·8	4·10	67·2
	R	7·88	174·2	6·02	80·4	3·51	42·2

Notes

1. The three distributions used were the uniform distribution on a disc, the circular normal distribution, and the bivariate Cauchy distribution.
2. Several realisations were taken from each model, and, in the case of 100-point configurations, several repetitions were made of each call, in order to increase the precision of the timings.
3. All computations were performed in single precision FORTRAN, run under the optimising compiler on the CDC 7600 of the University of London Computer Centre.

Step 6
Enter the peel record label in M to address the appropriate segment of LIST, and set the degeneracy flag if any degeneracy was encountered during the computation of the peel. Exit.

5. Some other methods

In this section we discuss some other methods which have been suggested for finding convex hulls. Our own method will be referred to as method GS. Our algorithm and implementation are similar in spirit to the method considered by Eddy (1977). Eddy's algorithm, which we shall refer to as method E, does not, however, make any attempt to deal with degeneracies. His suggestion that higher precision arithmetic should be used unfortunately does not always work; it is quite possible for a degeneracy which is found to be exact by single precision arithmetic to be only a near degeneracy in double precision. It is easy to deal with exact degeneracies properly while near degeneracies can easily lead to contradictory results. In addition Eddy's method is somewhat laborious for small numbers of points; this is illustrated in **Fig. 7**.

An earlier method, referred to as method G, was described by Graham (1972). Graham's method works by expressing each point in polar coordinates relative to some interior point, sorting the points in order of increasing polar argument and then successively confirming points as hull vertices or else deleting them by considering the orientation of each triple of consecutive remaining points. For details see Graham's paper. The number of operations required by Method G is shown by Graham to be 0 ($n \log n$).

Jarvis (1973) considered two methods which we shall refer to as J1 and J2 respectively. The first works as follows. Find the leftmost point; this is the first vertex v_1. (Jarvis uses a slightly slower procedure for this step.) The second vertex v_2 is the point which maximises the angle zv_1v_2, as in **Fig. 6.** Now suppose $v_1, v_2, \ldots, v_i$ have already been found to be consecutive vertices of the hull. By searching through all the points, find the point x which maximises the angle $v_{i-1}\, v_i\, x$ and set v_{i+1} to be x. If v_{i+1} is v_1 then the hull has been found.

Jarvis's second method, a refinement of his first, works by eliminating points which are inside the hull of the vertices already found. After v_1 has been found, sort all the points in order of increasing angle zv_1x. By making use of this ordering, the subsequent search for v_{i+1} at each stage is restricted to points x lying on the same side of v_1v_i as z. Points in the area shaded in Fig. 6 are not considered.

A method similar to those considered by Jarvis operates as follows: sort the points in order of increasing x coordinate and hence find the leftmost point, which must be a vertex. Look at the lines to all the other points, and find the one with the (algebraically) smallest slope. The corresponding point is the next vertex. Now look at the lines from this point to all points on its right, making use of the ordering of the points, and choose the line with the smallest slope. Continue in this way until the rightmost point is reached, and then return in a similar fashion around the upper half of the hull. We call this method J3.

Recursive methods depend on the fact that the hull of the union of sets is the hull of the union of the hulls of those sets. The recursive method we shall consider (method R) adds points one at a time. The first three points form their own hull. As each remaining point is added the list of hull vertices is modified by checking whether the new point lies outside any of the old hull edges; if it does, all such edges are deleted and replaced by two new edges to the new point. If the new point lies inside all the existing hull edges, then no modification is necessary. Another recursive method was described by Preparata and Hong (1977) and modified in principle by Bentley and Shamos (1978).

It should be emphasised that all these methods are susceptible to numerical errors of the kind described in relation to method GS, though some of them are fairly robust. However, it will be shown below that method GS is easily the fastest method, even when it has been modified to deal with numerical errors.

6. Performance of the algorithms

The various methods described above were compared on several different sets of data. Each set of points is a pseudo-random sample from a particular probability distribution in the plane. Timings for the various methods are summarised in **Table 2.** Note that the three distributions used yield greatly differing numbers of vertices. It can be seen that Method GS was fastest for all the configurations considered. It can also be seen from Fig. 7 that the time required by Method GS seems to be proportional to the number of points considered, thus giving it an advantage over the other methods. The two best methods, GS and E, are further compared in Fig. 7. It should be pointed out that all the times refer to our own implementations of the various algorithms, except for method E where the program CONVEX described by Eddy (1977) was used.

Acknowledgements
We are grateful to W. F. Eddy for some helpful correspondence, to A. Bowyer for his help with the computer runs, and to the SSRC for provision of facilities to the spatial data project at the University of Bath.

References

Barnett, V. (1976). The ordering of multivariate data (with discussion), *J. Roy. Statist. Soc.* (*A*), Vol. 139, pp. 318-354.
Bentley, J. L., and Shamos, M. I. (1977). Divide and conquer for linear expected time, Carnegie-Mellon University Computer Science Technical Report.
Eddy, W. F. (1977). A new convex hull algorithm for planar sets, *ACM Transactions on Mathematical Software*, Vol. 3, pp. 398-403 and 411-412.
Graham, R. L. (1972). An efficient algorithm for determining the convex hull of a finite planar set, *Information Processing Letters*, Vol. 1, pp. 132-133.
Jarvis, R. A. (1973). On the identification of the convex hull of a finite planar set, *Information Processing Letters*, Vol. 2, pp. 18-21.
Preparata, F. P., and Hong, S. J. (1977). Convex hulls of finite sets of points in two and three dimensions, *CACM*, Vol. 20, pp. 87-93.

Book reviews

Natural Language Communication with Computers, edited by L. Bolc, 1978; 292 pages. (*Springer-Verlag*, $13·50)

Mainly, the book contains descriptions of systems which are capable of analysing, and transforming into internal code, classes of natural language sentences. The systems vary according to their purpose (application) and the extent of analysis performed (morphological/syntactical/semantical). Chapter 1 describes an ambitious system providing for the building and subsequent querying of a knowledge structure, including the possibility of temporal changes. Natural language sentences are parsed by a so-called feature grammar (a two-level extension of a Chomsky grammar), and coded in predicate logic with extra operators which reflect the 'state-of-the-world'. Chapter 2 is an account of a less general but perhaps more streamlined system designed for the use with, for example, a pharmacological data base. Queries are translated (again, by a feature grammar) into logical expressions (set language). The system described in Chapter 3 is designed for the monitoring of water pollution. Input statements or queries are analysed by an upgraded ATN (Augmented Transition Network) and translated into formulae. The book also includes a very detailed tutorial on ATNs (Chapter 5). Chapter 6 outlines a project whose aim is to analyse syntactically a subset of Polish, using the programming language PROLOG which, together with its formal semantics (or rather, a second approximation of that), is described in Chapter 4. Two sophisticated PROLOG programs are also given in Chapter 4, a complete compiler for a subset of ALGOL and an 'intelligent' interactive system.

In more than one respect, the book is less homogeneous than its title suggests; the subject inevitably attracts people with varying interests. On the whole I expect Chapters 1, 3, 5 and 6 to appeal to the linguist more than to the computer scientist, and vice versa for 2 and 4. In places, Chapter 5 reads like 'a linguist's first course in formal methods', whereas Chapter 4 is almost an anti-tutorial and 1 is also difficult to read. Although, in my opinion, the book does justice to its declared aim of 'providing information on the present state of the research in the area', there is only one contribution, namely Chapter 2, by Kraegeloh and Lockemann, which managed to convey to me quite clearly what has been achieved and, importantly, what has been left out. I regret that nowhere is the problem of what can or cannot be achieved discussed in detail. While the tenor of the contributions points to the recognition that 'syntax' and 'semantics' of natural languages tend to be inextricably interwoven, I also find traces of the opinion that the two can be considered separately. This touches the question of whether or not natural languages, or non-trivial subsets thereof, are suitable as machine languages. I think that a decision has to be made whether one is content with leaving natural languages to communication amongst humans (in which case the title of the book is a misnomer), or one strives for an improved COBOL, to put it in extreme terms (in which case the title is an over-estimation). The very notion 'natural' clashes with the term 'computer' which denotes, after all, an artificial device. It is a discussion of such issues which I most dearly miss, not being an expert in 'artificial intelligence'; it is also unfortunate that the responsibility of the editor did not extend to cover these problems and, thereby, to put the set of contributions into a more specific framework. For these reasons I can recommend the book in the first place only to those specialising in the area and wishing to find out what is happening 'elsewhere'.

E. Best (Newcastle)

Portability of Numerical Software, Workshop, Oak Brook, Illinois, 1976. Edited by W. Cowell, Lecture Notes in Computer Science Vol. 8, 1977. 539 pages. (*Springer-Verlag*, $18.30)

This volume should be compulsory reading for all members of ANSI X3J3, the Committee responsible for FORTRAN 77. Nonetheless, despite the considerable criticism of FORTRAN within this book, the overall feeling emanating from the papers is one of gratitude for its existence. Indeed the low point in the book comes with a predictable series of bleats in favour of ALGOL 68, but as L. M. Delves states in his paper 'ALGOL 68 compiler writers would give the biggest boost to the language if they could let the user call FORTRAN routines from his ALGOL 68 program'.

The book is a large volume and its very nature inevitably means that there is a fair amount of repetition, but the editors have found an extremely natural order for the papers and added some useful cohesive comment. The book gets off to a fairly slow start with a discussion of the restrictions created by the physical limitations of computers, in particular the word length. It starts to come to life with the papers from those responsible for successful general purpose libraries, in particular those concerning PFORT and PORT from members of Bell Telephone Laboratories. It contains an excellent paper from B. T. Smith on 'FORTRAN poisoning' and a valuable paper by J. M. Boyle surveying contemporary computerised source management systems. The NAG project emerges with considerable credit, and the final paper in the book contains, under a section headed 'Overflow control', a classic example of the kind of problem obstructing the path to program portability.

There is just one sour note to sound. The papers themselves are beautifully laid out. What a pity the examples of coding within them do not separate code and comment as well as the typist has separated examples and text.

D. L. Fisher (Leicester)