Tensor product approximations to data defined on rectangular meshes in N-space

P. J. Hartley

Department of Mathematics, Lanchester Polytechnic, Priory Street, Coventry CV1 5FB

The problem of interpolating amongst data defined on a rectangular mesh in real N-space can be conveniently solved by constructing suitable tensor product hypersurfaces. In particular the problem can be neatly formulated with the use of Kronecker products of matrices. Greville (1961) has observed that the concept of the pseudo-inverse of a matrix allows this formulation to be extended to the corresponding least-squares problem. The purpose of this note is to review this approach and to relate it to the method of surface fitting described by Clenshaw and Hayes (1965), in which the surface is obtained by two stages of curve fitting.

There is a dearth of algorithms for solving the N-variable problem and two new ones are given here in outline. They are similar to those developed independently by Pereyra and Scherer (1973).

(Received November 1974)

1. Introduction

The problem we are considering is the representation of data which are values of some unknown function of N independent variables and which are given at all the vertices of a finite rectangular mesh in real N-space. The representation is to be by an element of a tensor product space of functions of N variables constructed from N finite spaces of functions of one variable.

For $N = 2$ we can represent the data by the triples (x_i, y_j, z_{ij}), $i = 1(1)m_1, j = 1(1)m_2$, and the function which approximates the z_{ij} is of the form

$$f(x, y) = \sum_{s=1}^{n_1} \sum_{t=1}^{n_2} a_{st}\phi_s(x)\,\psi_t(y) \tag{1}$$

thus belonging to the tensor product space (Halmos, 1958)

$$\langle\phi_1, \ldots, \phi_{n_1}\rangle \otimes \langle\psi_1, \ldots, \psi_{n_2}\rangle$$

where $\langle S\rangle$ means the span of the set S. In this paper $\{\phi_s\}$ and $\{\psi_t\}$ are bases for the corresponding linear function spaces so that $\langle\phi_1, \ldots, \phi_{n_k}\rangle$ is of dimension n_k.

This is of course a much restricted problem (in the sense that the data and the approximating function have special forms) but one which occurs with sufficient frequency to be of practical interest. Furthermore it is not essentially difficult to extend the methods considered in this paper to more generally defined data patterns. Particular variations of data structure are considered in the papers of Clenshaw and Hayes (1965) and Buchanan and Thomas (1968).

Rice (1969) discusses tensor product approximation in the L_2-norm and produces theoretical results related to the ones which follow.

2. Interpolation

In this case the number of data vertices and the number of basis functions is the same, for each variable, i.e. $n_k = m_k$, $k = 1(1)N$. For $N = 2$ we require .

$$\sum_{s=1}^{m_1} \sum_{t=1}^{m_2} a_{st}\phi_s(x_i)\,\psi_t(y_j) = z_{ij},\ i = 1(1)m_1, j = 1(1)m_2.$$

Defining the matrices

$$A = [a_{st}]$$
$$\Phi^T = [\phi_s(x_i)]$$
$$\Psi^T = [\psi_t(y_j)]$$
$$Z = [z_{ij}]$$

allows us to write the above as

$$\Phi A \Psi^T = Z\ . \tag{2}$$

If the interpolation problem is well-defined we can immediately write

$$A = \Phi^{-1} Z (\Psi^T)^{-1} \tag{3}$$

as the solution to that problem.

Although it is clear from the presence of the inverses of Φ and Ψ^T separately that the interpolation in the two variables is in some sense separated into two one-variable problems, (3) does not fully illuminate that separation. If we partition A and Z into rows $a^{(s)T}$ and $z^{(s)T}$, $s = 1(1)m_1$, then, after matrix transposition, (2) can be written in the form

$$\sum_{s=1}^{m_1} \phi_s(x_i)\,\Psi a^{(s)} = z^{(i)},\ i = 1(1)m_1\ .$$

By definition (Halmos, 1958) this is the same as the Kronecker product form

$$(\Phi \otimes \Psi)\,a = z \tag{4}$$

where a and z are $(m_1 m_2) \times 1$ column vectors constructed 'lexicographically' (in row order) from A and Z. Specifically

$$a = (a_{11}\, a_{12} \ldots a_{1m_2}\, a_{21} \ldots a_{2m_2} \ldots a_{m_1 1} \ldots a_{m_1 m_2})^T\ .$$

Using the properties of the Kronecker product of nonsingular matrices (assuming again that the interpolation problem is well-defined) we achieve

$$a = (\Phi^{-1} \otimes \Psi^{-1})z \tag{5}$$

corresponding to (3).

The extension to $N > 2$ is straightforward, but complicated by the problems of notation. The data can be represented by $(N + 1)$-tuples $(x_{i_1}^{(1)}, x_{i_2}^{(2)}, \ldots, x_{i_N}^{(N)}, z_{i_1 i_2 \ldots i_N})$ with $i_k = 1(1)m_k$ and $k = 1(1)N$, and the hypersurface has the tensor product form

$$f(x^{(1)}, x^{(2)}, \ldots, x^{(N)}) = \sum_{s_1=1}^{m_1} \cdots \sum_{s_N=1}^{m_N} a_{s_1 \ldots s_N} \left(\prod_{k=1}^{N} \phi_{s_k}^{(k)}(x^{(k)})\right). \tag{6}$$

The solution to the (assumed well-defined) interpolation problem

$$f(x_{i_1}^{(1)}, \ldots, x_{i_N}^{(N)}) = z_{i_1 \ldots i_N},\ i_k = 1(1)m_k,\ k = 1(1)N\ ,$$

can then be written immediately as

$$a = (\Phi_1^{-1} \otimes \Phi_2^{-1} \otimes \ldots \otimes \Phi_N^{-1})z \tag{7}$$

where a and z are column vectors each with $\prod_{k=1}^{N} m_k$ elements, respectively containing all the $a_{s_1 \ldots s_N}$ and $z_{i_1 \ldots i_N}$ in lexicographic order. Also

$$\Phi_k^T = [\phi_{s_k}^{(k)}(x_{i_k}^{(k)})]$$

is the matrix that would be involved in interpolation in the variable $x^{(k)}$ alone with all other variables fixed, i.e. interpolation along mesh lines parallel to the $x^{(k)}$ co-ordinate axis. (7) shows clearly the separation of the N-variable problem into N one-variable problems.

3. Least-squares approximation

In this case the number of data vertices in each variable exceeds the number of basis functions in that variable, i.e. $m_k > n_k$, $k = 1(1)N$.

For $N = 2$ we are required to find the a_{st} that minimise

$$\sum_{i=1}^{m_1} \sum_{j=1}^{m_2} \left(z_{ij} - \sum_{s=1}^{n_1} \sum_{t=1}^{n_2} a_{st}\phi_s(x_i)\psi_t(y_j) \right)^2 , \qquad (8)$$

or, equivalently, to solve the now overdetermined system (4) in the least-squares sense. Peters and Wilkinson (1970) have given an excellent account of the problem of finding the least-squares solution of a linear system, showing that it may be solved by extensions of the familiar methods of Gauss-elimination or Householder transformations; the solution is represented symbolically by

$$a = (\Phi \otimes \Psi)^\dagger z \ . \qquad (9)$$

where $B^\dagger$ means the pseudo-inverse of B. Pseudo-inverses are discussed in this and related contexts by Greville (1959, 1960 and 1961), but from the more classical point of view as generalisations of the concept of the inverse of a nonsingular matrix. In particular in Greville (1961) he notes the result (9) and observes that the properties of pseudo-inverses enable us to write it in the manner equivalent to (5):

$$a = (\Phi^\dagger \otimes \Psi^\dagger)z \ . \qquad (10)$$

The generalisation to $N > 2$ is immediate:

$$a = (\Phi_1^\dagger \otimes \Phi_2^\dagger \otimes \ldots \otimes \Phi_N^\dagger)\, z \ . \qquad (11)$$

To complete the comparison we note that when $N = 2$ we could also write

$$A = \Phi^\dagger Z (\Psi^T)^\dagger \qquad (12)$$

corresponding to (3).

4. Computational methods

The separation of the N-variable problem into N one-variable problems is not simply a theoretical nicety, but neither is it a panacea for multivariable problems. In the simplest terms we have exchanged the problem of size for one of complexity: in effect the (pseudo-) inversion of a single matrix of size $\left(\prod_{k=1}^{N} m_k\right) \times \left(\prod_{k=1}^{N} n_k\right)$ for the (pseudo-) inversion of N matrices each of size $m_k \times n_k$ plus the evaluation of the Kronecker product.

An interesting aspect of the Kronecker product form (11) of the solution is that it corresponds to a well-known method of surface fitting in which the Kronecker product does not appear explicitly. A typical description of this method for $N = 2$ can be found in Clenshaw and Hayes (1965). The authors observe that if the $\{\phi_s\}$ are orthogonal polynomials of degree s with respect to the usual summation inner-product over the data set $\{x_i\}$, and likewise the $\{\psi_t\}$ over the set $\{y_j\}$, then least-squares *curve* fitting with

(*a*) $f_j(x) = \sum_{s=1}^{n_1} b_{sj}\phi_s(x)$ to the data set $\{(x_i, z_{ij}): i = 1(1)m_1\}$ for each j, and then with

(*b*) $b_s(y) = \sum_{t=1}^{n_2} a_{st}\psi_t(y)$ to the data set $\{(y_j, b_{sj}): j = 1(1)m_2\}$

for each s, produces the true least-squares surface of the form (1) for the data $\{(x_i, y_j, z_{ij})\}$. This is clearly so because the orthogonality of the two sets $\{\phi_s\}$ and $\{\psi_t\}$ separately guarantees the orthogonality of the tensor product basis $\{\phi_s\psi_t\}$ with respect to the double sum inner-product over the set $\{(x_i, y_j)\}$ and the two diagonal systems of normal equations for the curve fitting processes:

$$(a)\ b_{sj} \sum_i \phi_s^2(x_i) = \sum_i z_{ij}\phi_s(x_i),\ \forall\, j, s, \qquad (13)$$

and

$$(b)\ a_{st} \sum_j \psi_t^2(y_j) = \sum_j b_{sj}\psi_t(y_j),\ \forall\, s, t, \qquad (14)$$

combine immediately into the diagonal system

$$a_{st} \sum_i \sum_j \phi_s^2(x_i)\, \psi_t^2(y_j) = \sum_i \sum_j z_{ij}\phi_s(x_i)\, \psi_t(y_j),\ \forall\, s, t\ ,$$

for the surface fitting process. That this separation into two curve fitting processes is possible for more general bases $\{\phi_s\}$ and $\{\psi_t\}$ is less clear. It is again caused by the transfer of the main properties of the ϕ_s and ψ_t to the product functions $\phi_s\psi_t$, in this case their linear independence.

To show this we note that if the surface is obtained by first fitting functions $\sum_s b_{sj}\phi_s(x)$ to $\{(x_i, z_{ij}): i = 1(1)m_1\}$, for each j, and then $\sum_t a_{st}\psi_t(y)$ to $\{(y_j, b_{sj}): j = 1(1)m_2\}$, for each s, the resulting form is as required (equation 1), and the two overdetermined sets of equations we have to solve (in the least-squares sense) have matrix forms

$$\Phi B = Z$$

and

$$A\Psi^T = B$$

i.e.

$$A = \Phi^\dagger Z(\Psi^T)^\dagger \ .$$

But this is the solution (12) of the surface fitting problem, and our proof is complete.

We conclude that if $\{\phi_s\}$ and $\{\psi_t\}$ are sets of independent functions, linear combinations of which are suitable for curve fitting, then least-squares surface fitting with linear combinations of $\phi_s\psi_t$ can be achieved by least-squares curve fitting in the two independent variables separately, provided also that the data are available at all the vertices of a rectangular grid. We also note that the order in which the variables are considered is immaterial. If the ψ_t are used first the separate equations are

$$B\Psi^T = Z$$

and

$$\Phi A = B$$

with the same overall result as before.

Weinstein (1971) has proved similar results for the more general problem of L_p-approximation of continuous functions on compact sets in E_k, $k \geqslant 2$.

It is also clear that this approach can be extended to higher dimensions. If the data are $\{(x_i, y_j, w_k, z_{ijk})\}$ we fit surfaces

$$\sum_s \sum_t b_{stk}\phi_s(x)\psi_t(y) \text{ to } \{(x_i, y_j, z_{ijk})\}$$

for each k and then curve fit the data $\{(w_k, b_{stk})\}$ with $\sum_u a_{stu}\chi_u(w)$ for each s and t, obtaining the true least-squares hypersurface $\sum_s \sum_t \sum_u a_{stu}\phi_s(x)\psi_t(y)\chi_u(w)$.

However, the natural insight into the construction of the hypersurface that this gives us does not necessarily imply that we have the most sensible computing scheme. Indeed it is not obvious at first how we can control the above scheme since we do not wish to have arrays with an unknown number N of dimensions in order to cope with any number N of variables. The answer is to store the N-dimensional arrays as one-dimensional arrays with the elements stored in lexicographic order, and this leads us naturally back to the Kronecker product form (11). Rewriting (11) as

$$a = (\Phi_1^\dagger \otimes R_1)z \qquad (15)$$

where $R_1 = \Phi_2^\dagger \otimes \ldots \otimes \Phi_N^\dagger$, gives

$$a = (\Phi_1^\dagger \otimes I)(I \otimes R_1)z \qquad (16)$$

by a well-known identity of Kronecker products (Halmos, 1958). Representing $\Phi_1^\dagger$ by $[h_{ij}]$ gives the partitioned form

$$a = \begin{bmatrix} h_{11}I \dots h_{1m_1}I \\ \vdots \qquad \vdots \\ h_{n_11}I \dots h_{n_1m_1}I \end{bmatrix} \begin{bmatrix} R_1 & 0 & \dots 0 \\ 0 & R_1 & \dots 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & \dots R_1 \end{bmatrix} \begin{bmatrix} z_1 \\ \vdots \\ z_{m_1} \end{bmatrix}$$

where R_1 is of size $\left(\prod_{k=2}^N n_k\right) \times \left(\prod_{k=2}^N m_k\right)$ and each vector z_j has $\prod_{k=2}^N m_k$ elements. A little manipulation then shows that a has the partitioned form

$$a = \begin{bmatrix} R_1 v_1 \\ \vdots \\ R_1 v_{n_1} \end{bmatrix} ,$$

in which each vector v_i is given by

$$v_i = \sum_{j=1}^{m_1} h_{ij} z_j \ . \tag{17}$$

Now each component $R_1 v_i$ of the above partition is of the same general form as the right hand side of (11), and so can be manipulated in a similar manner. Thus, storing the n_1 vectors v_i (each with $\prod_{k=2}^N m_k$ elements) as one vector, we can operate on each of the n_1 parts with $\Phi_2^\dagger = [h_{ij}]$ exactly as in (17) we operated on z with $\Phi_1^\dagger = [h_{ij}]$. If we repeat the operation with $\Phi_3^\dagger = [h_{ij}]$ on each of the resulting $n_1 n_2$ parts (each with $\prod_{k=3}^N m_k$ elements), and so on, then we have the following scheme to produce finally the $\left(\prod_{k=1}^N n_k\right) \times 1$ vector a of surface parameters. The local procedures *COEFFICIENT MATRIX* and *PSEUDO INVERSE* calculate the elements of Φ_k and $\Phi_k^\dagger = [h_{ij}]$ respectively.

```
comment outline of tensor product hypersurface algorithm using
  explicit pseudo-inverses;
comment initialisation;
l2 := 1;
for k := 1 step 1 until N do l2 := l2 × m[k];
comment recursive calculation of hypersurface parameters;
for k := 1 step 1 until N do
  begin p := 1;
    l1 := if k = 1 then 1 else l1 × n[k − 1];
    l2 := l2 ÷ m[k];
    COEFFICIENT MATRIX;
    PSEUDO INVERSE;
    for s := 1 step 1 until l1 do
    for i := 1 step1 until n[k] do
    for t := 1 step 1 until l2 do
      begin a[p] := 0;
        for j := 1 step 1 until m[k] do
        a[p] := a[p] + h[i,j] × z[t + l2 × (j − 1 + m[k] ×
          (s − 1))];
        p := p + 1
      end
    for i := 1 step 1 until p − 1 do z[i] := a[i]
  end
```

ALGOL has been used here as a convenient medium for communication; actual coding may well be different. In particular the way in which *PSEUDO INVERSE* is defined and used may vary with the context, and the calculation of the inner-products $a[p]$ would be achieved with a double-precision procedure. Suitable array bounds for h, a and z are

$h[1:maxm, 1:maxn]$
$a[1:mp1]$
$z[1:mp1]$
where $maxm = \max\{m_k, k = 1(1)N\}$,
$maxn = \max\{n_k, k = 1(1)N\}$,

and $mp1 = \prod_{k=1}^n m_k$.

The other identifiers are self-explanatory, either being counting indices or corresponding in an obvious way to variables used in the text.

A possible criticism of the scheme is the need for explicitly calculating all the $\Phi_k^\dagger$ when compared with the economies possible in those one-variable least-squares algorithms which calculate the solution $A^\dagger b$ of $Ax = b$ without actually exhibiting $A^\dagger$. Such an approach can be used as an alternative to the above scheme, but at greater cost in complexity, and the result is the 'natural' scheme of successive curve fitting described earlier, coupled with the lexicographic storage of the arrays.

The complexity arises from the need to do all the curve fitting in any one variable with only one implicit pseudo-inversion of the relevant Φ_k. For example at the first stage we have to solve the m_1 equations in n_1 unknowns

$$\sum_{i_1=1}^{n_1} z^{(1)}_{i_1 j_2 \dots j_n} \phi^{(1)}_{i_1}(x^{(1)}_{j_1}) = z_{j_1 \dots j_N}, j_1 = 1(1)m_1$$

for all $\prod_{k=2}^N m_k$ values of $j_2, \dots, j_N$. We can write this as

$$\Phi_1 Z^{(1)} = Z$$

where $Z^{(1)}$ is an array of size $n_1 \times \prod_{k=2}^N m_k$ and Z is an array of size $m_1 \times \prod_{k=2}^N m_k$ containing the data ordinates. At the second stage, however, $Z^{(1)}$ becomes the data on the right hand sides of the m_2 equations in n_2 unknowns

$$\sum_{i_2=1}^{n_2} z^{(2)}_{i_1 i_2 j_3 \dots j_N} \phi^{(2)}_{i_2}(x^{(2)}_{j_2}) = z^{(1)}_{i_1 j_2 \dots j_N}, j_2 = 1(1)m_2 \ ,$$

for all $n_1 \prod_{k=3}^N m_k$ values of $i_1, j_3, \dots, j_N$. To write this as

$$\Phi_2 Z^{(2)} = Z^{(1)}$$

it is necessary to permute the elements of $Z^{(1)}$ so that it becomes of size $m_2 \times n_1 \prod_{k=3}^N m_k$. Successive stages are similar, and the final array obtained, $Z^{(N)}$, is of size $n_N \times \prod_{k=1}^{N-1} n_k$, and taken in column order this is the vector of hypersurface parameters. An outline of this scheme is shown below, where the array b is initially set equal to the array Z above and the array c is finally the array $Z^{(N)}$ above.

```
comment outline of tensor product hypersurface algorithm using
          implicit pseudo-inverses;
comment initialisation;
l2 := 1;
for k := 2 step 1 until N do l2 := l2 × m[k];
comment recursive calculation of hypersurface parameters;
for k := 1 step 1 until N do
  begin q := 1;
    l1 := if k = 1 then 1 else l1 × n[k − 1];
    COEFFICIENT MATRIX;
    LEAST SQUARES SOLUTION (phi, c, b, m[k], n[k],
      l1 × l2, eta, coda);
    if k = N then goto coda;
    l2 := l2 ÷ m[k + 1];
    for $s := 1$ step 1 until $l1$ do
    for $i := 1$ step 1 until $n[k]$ do
    for $t := 1$ step 1 until $l2$ do
      begin for $j := 1$ step 1 until $m[k+1]$ do
                $b[j,q] := c[i, t + l2 \times (j - 1 + m[k+1] \times (s - 1))]$;
                $q := q + 1$
      end;
  coda: end
```

As with the previous algorithm the procedure *COEFFICIENT MATRIX* calculates the elements of the array $phi = \Phi_k$. The procedure *LEAST SQUARES SOLUTION* solves $(phi)(c) = b$ for c, in the least squares sense; the parameters used here are those required by the procedure of Businger and Golub (1965). For this second algorithm suitable array bounds for *phi*, *b* and *c* are

$phi[1{:}maxm, 1{:}maxn]$

$b[1{:}maxm, 1{:}mp2]$

$c[1{:}maxn, 1{:}mp2]$

where *maxm* and *maxn* are as previously defined and $mp2 = \prod_{k=2}^{N} m_k$; the other identifiers are self-explanatory. This coding is, of course, intended to describe the logic of the algorithm and, as before, is clearly not a finished program.

There is virtually no difference in the storage requirements of the two schemes because the arrays z and a of the first scheme and the arrays b and c of the second scheme store the same information in different forms. Note that at any one time only one of the Φ_k exists in storage. This is in contrast to the linear system for the complete N-dimensional problem obtained when no attempt is made to separate the variables. This latter method is used of necessity when the data are 'randomly' distributed in some finite region of N-space (Hayes and Halliday, 1972). The overdetermined system to be solved is

$$A\,a = z$$

where each row of A corresponds to a data point and each column to an element of the tensor product basis of functions; the solution is simply

$$a = A^{\dagger} z\ ,$$

but A has dimensions $\prod_{k=1}^{N} m_k \times \prod_{k=1}^{N} n_k$ and for $N > 2$ storage considerations predominate.

Moreover the number of long operations required to find $A^{\dagger}$ is of the order of $m^N n^{2N}$, whereas the number required to find all the $\Phi_k^{\dagger}$ and their Kronecker product with Z, as in our first scheme, is of the order of $N(n^2 m + m^{N+1})$, where m and n are typical values of m_k and n_k. Since m is unlikely to be of the order of n^{2N}, unless $N = 1$, and in view of the relatively small storage required, it seems to be worthwhile having a special algorithm for separating the variables when the data are defined on a rectangular mesh.

5. Examples of use

The simplest formulation occurs when the basis functions $\phi_i(x) = x^{i-1}$. However if polynomials are considered suitable then the orthogonal polynomial scheme described by Clenshaw and Hayes can be extended to higher dimensions, and would be more desirable for the usual reasons.

A formulation which has recently found considerable favour occurs when the ϕ_i are cubic B-splines (Cox, 1972; Hayes and Halliday, 1972). This results not only in a representation which deals effectively with local non-polynomial behaviour in the data but also causes the Φ_k to have a band structure, each row containing at most four non-zero entries. In this case the schemes given in Section 4 can be made even more efficient if the pseudo-inversion and least-squares solution procedures are specially tailored to the band structure (Reid, 1967).

The schemes given can be used to solve the classic spline interpolation problem where the knots are placed at the data vertices, or the more general interpolation problem where the knots are positioned to deal with the local behaviour of the data. In the latter case it is necessary for the knots and data vertices to satisfy the conditions derived by Schoenberg and Whitney (1953). When the schemes are being used in the least-squares sense the knots can, in theory, be positioned arbitrarily, again with the idea of dealing with the local behaviour of the data. In this case the Φ_k may not be of maximal rank n_k. The first scheme deals with this if *PSEUDO INVERSE* is not restricted to the maximal rank case. The second scheme will fail as it stands because the procedure of Businger and Golub assumes maximal rank. If maximal rank *is* required then the Schoenberg–Whitney conditions must be satisfied by some subset of the data (Cox and Hayes, 1973); if it is not, then the solution obtained by pseudo-inversion is that one of the linear manifold of possible solutions which has minimum norm, i.e. minimises

$$\left(\sum_{s_1,\ldots,s_n} a^2_{s_1\ldots s_N}\right)^{\frac{1}{2}}$$

(Peters and Wilkinson, 1970).

The schemes are designed solely for use with data which are values of a function which cannot reasonably be evaluated at other values of the independent variables. If the underlying data function can be evaluated anywhere in its domain then Gordon (1969) has shown that there are more efficient methods for finding hypersurface representations, in the sense of obtaining more accuracy per data point or obtaining a specific accuracy with fewer data points.

Finally, it is worth noting that, once obtained, the hypersurface representation (6) will be evaluated through a further computer routine. If B-spline bases are used then the evaluation routine, like the surface fitting routine, can be specially tailored to achieve economies by using the compact support property of the B-splines.

Acknowledgement

I would like to thank the referee for his helpful comments, and in particular for drawing my attention to the work of Call and Judd (1974) which covers similar ground to this paper.

References

Buchanan, J. E., and Thomas, D. H. (1968). On Least-Squares Fitting of 2-Dimensional Data with a Special Structure, *SIAM J. Numer. Anal.*, Vol. 5, pp. 252-257

Businger, P., and Golub, G. H. (1965). Linear Least-Squares Solutions by Householder Transformations, *Num. Math.*, Vol. 7, pp. 269-276

Call, E. S., and Judd, F. F. (1974). Surface Fitting by Separation, *J. Approx. Theory*, Vol. 12, pp. 283-290

Clenshaw, C. W., and Hayes, J. G. (1965). Curve and Surface Fitting, *J. Inst. Maths. Applics.*, Vol. 1, pp. 164-183

Cox, M. G. (1972). The Numerical Evaluation of B-Splines, *J. Inst. Maths. Applics.*, Vol. 10, pp. 134-149

Cox, M. G., and Hayes, J. G. (1973). Curve Fitting: A Guide and Suite of Algorithms for the Non-specialist User, *NPL Report NAC* 26, Teddington, Middlesex

Gordon, W. J. (1969). Distributive Lattices and the Approximation of Multivariate Functions, in Schoenberg (ed.). *Approximation with Special Emphasis on Spline Functions*, Academic Press

GREVILLE, T. N. E. (1959). The Pseudoinverse of a Rectangular or Singular Matrix and its Application to the Solution of Systems of Linear Equations, *SIAM Rev.*, Vol. 1, pp. 38-43

GREVILLE, T. N. E. (1960). Some Applications of the Pseudoinverse of a Matrix, *SIAM Rev.*, Vol. 2, pp. 15-22

GREVILLE, T. N. E. (1961). Note on Fitting of Functions of Several Independent Variables, *J. Soc. Indust. Appl. Math.*, Vol. 9, pp. 109-115

HALMOS, P. R. (1958). *Finite-Dimensional Vector Spaces* (Sections 24 and 52), Van Nostrand

HAYES, J. G., and HALLIDAY, J. (1974). The Least-Squares Fitting of Cubic Spline Surfaces to General Data Sets, *J. Inst. Maths. Applics.*, Vol. 14, pp. 89-103

PEREYRA, V., and SCHERER, G. (1973). Efficient Computer Manipulation of Tensor Products with Applications to Multidimensional Approximation, *Maths. Comp.*, Vol. 27, pp. 595-605

PETERS, G., and WILKINSON, J. H. (1970). The Least-Squares Problem and Pseudo-Inverses, *The Computer Journal*, Vol. 13, pp. 309-316

REID, J. K. (1967). A note on the Least-Squares Solution of a Band System of Linear Equations by Householder Reductions, *The Computer Journal*, Vol. 10, pp. 188-189

RICE, J. R. (1969). *The Approximation of Functions:* Vol. 2, Nonlinear and Multivariate Theory (Section 12-3), Addison-Wesley

SCHOENBERG, I. J., and WHITNEY, A. (1953). On Polya Frequency Functions III, *Trans. Amer. Math. Soc.*, Vol. 74, pp. 246-259

WEINSTEIN, S. E. (1971). Product Approximations of Functions of Several Variables, *SIAM J. Numer. Anal.*, Vol. 8, pp. 178-189

Book review

Fortran Programming—A spiral approach, by C. B. Kreitzburg and B. Schneiderman, 1976; 437 pages. (*Jarcourt Brace Jovanovitch*, £4·25)

Simplified ANSI FORTRAN IV Programming, second edition by G. A. and Joan B. Silver, 1976; 334 pages. (*Harcourt Brace Jovanovitch*, £5·50)

After preparing lecture notes on FORTRAN programming, many lecturers, including the above authors, publish their notes in book form. In a market already saturated by at least sixty very similar books, these additions seem superfluous. However, both books could form the basic material for an introductory course if supplemented with practical experience and tutorial advice.

By British standards, both books would be considered 'oversized'. Presumably the American market believes that the more pages, the better the book. British students, as well as hesitating at the price, would consider them tedious reading and tend to skip text in search of the next matter of substance. Both are slightly oriented towards the authors' local FORTRAN compilers.

Fortran Programming—A spiral approach has been thoughtfully prepared. During the gradual advance from the simple to the complex, the student is introduced to new concepts in a well ordered sequence. Good programming style is emphasised, and then illustrated in an ANSI standard FORTRAN context. It is one of the better text books on this topic. Lecturers would be encouraged to use it if the Instructor's Manual mentioned in the preface could be obtained.

The second edition of *Simplified ANSI FORTRAN IV programming* claims to adhere to ANSI Standard FORTRAN. It also contains a brief section on structured programming. Some inaccuracies (e.g. Page 59 TRACE and DISPLAY debugging facilities are erroneously attributed to ANSI) and misleading definitions (e.g. Page 32 'Top/down programming means that the logic in the main program can be followed by reading it from top to bottom') should be corrected. Overall I prefer the spiral approach.

P. A. CLARKE (Harpenden)

Erratum

Formula (3) of *Hit ratios* by S. J. Waters (*Computer Journal*, Volume 19, No. 1, February 1976) should read:

$$\mathrm{BHR} = 1 - \frac{\mathrm{N - B \subset H}}{\mathrm{N \subset H}}$$

Similarly in Appendix 3.