Computational Statistics & Data Analysis 7 (1989) 217-235
North-Holland

All possible subset regressions using the QR decomposition

D.M. SMITH
Applied Statistics Research Unit, University of Kent, UK

J.M. BREMNER
Institute of Mathematics, University of Kent, UK

Received 23 March 1988
Revised 27 June 1988

Abstract: Existing algorithms for producing all possible subset regressions with the QR decomposition are compared, and a new algorithm is proposed with lower operation count and more accuracy.

Keywords: Linear regression, Givens' rotations.

1. Introduction

This paper discusses algorithms for evaluating all possible subset regression models using the QR decomposition for solution of the normal equations. After describing how the QR decomposition is used in regression and the role of Givens' rotations, the method of producing the residual sums of squares for subset models by a sequence of column transpositions of the R matrix is discussed. Following this details are given of a new algorithm for producing these residual sums of squares by dropping columns from the R matrix, after which the effects of using square root free forms of Givens' rotations are discussed and comparisons made between algorithms with respect to operation count, time and accuracy.

The major difficulty with evaluating all possible subset regressions is the number of regressions to be calculated. For p independent variables $2^p - 1$ regression models need fitting and as p increases the number of models increases exponentially e.g. $p = 15$ needs 32767 models fitted, $p = 20$ needs 1048575 models fitted. This means that efficient algorithms for generating the models to be fitted are a necessity. Often all that is required from the fitting are the residual sums of squares. In such circumstances it is possible to reduce the number of arithmetic operations performed to well below that for fully evaluating models.

Here attention is concentrated on evaluating just the residual sums of squares using the QR decomposition.

The common method of solving the normal equations produced by least squares fitting of linear regression models is through a variant of Gauss-Jordan elimination usually known as sweep, a term first used in [1]. Much development of the sweep method has taken place. A summary is given in [16] (pp. 349–355). Also, [12] gives a comprehensive review of the variety of sweep operators. The algorithm in [10] is used for the purposes of comparison as it is an efficient implementation of the sweep approach.

2. The QR decomposition

Details of fitting linear regression models using a QR decomposition are given in textbooks such as [16], [3] or [14]. These textbooks describe and contrast the alternative ways of forming the QR decomposition and compare it to using sweep. In the QR decomposition method the design matrix X of the regression equation

$$Y = X \cdot B + E$$

where Y is the dependent variable vector, B the regression coefficient vector, and E the vector of residual errors, is decomposed into two other matrices Q and R. Q^{T} is an orthogonal transformation matrix which when premultiplying X transforms it into the upper triangular matrix R. That is

$$X = Q \cdot R$$

where Q is an $(n \times p)$ matrix with orthogonal columns and R is a $(p \times p)$ upper triangular matrix, n being the number of observations, p the number of independent variables. The normal equations

$$(X^{\mathrm{T}} \cdot X) \cdot B = X^{\mathrm{T}} \cdot Y$$

become

$$R \cdot B = C \quad \text{where } C = Q^{\mathrm{T}} \cdot Y \text{ is a } (p \times 1) \text{ vector.}$$

If the matrix $(X \mid Y)$ is decomposed the upper triangular matrix R resulting can be written as

$$R^* = \begin{bmatrix} R & C \\ 0 & s \end{bmatrix}$$

where R and C are as above and s^2 is a scalar such that s^2 = the residual sum of squares for the fitted model. This matrix R^* contains all the information necessary to obtain the residual sums of squares for all possible subset regressions. The last column of R^* gives the residual sums of squares for the fitted model and also for other models fitted in descending order. For example if a four independent variable ($p = 4$) model has been fitted (labelled ABCD with the variables stored in that order) the following identities hold:

Model	residual sum of squares
ABCD	s^2
ABC	$s^2 + c_4^2$
AB	$s^2 + c_4^2 + c_3^2$
A	$s^2 + c_4^2 + c_3^2 + c_2^2$

where c_i, $i = 1,\dots,4$, are the elements of C. The total sum of squares is $s^2 + c_4^2 + c_3^2 + c_2^2 + c_1^2$.

The storage order of the variables is important. Although s for the order DCBA will be the same as above the elements of C will be different, as will the subsidiary models for which residual sums of squares are obtainable. Therefore transposing columns of the R^* matrix and re-triangularising (equivalent to changing the order of the variables) can be used to obtain residual sums of squares for other models. For example, to obtain the residual sum of squares for model ABD given the R^* matrix for ABCD, columns three and four would be exchanged and the resulting matrix re-triangularised. The residual sum of squares for ABD is then $s^2 + c_4^2$ of the new R^* matrix. As it is more efficient than recalculating the full QR decomposition for the new model transposition of columns and re-triangularisation of the R^* matrix is an obvious procedure to use for obtaining residual sums of squares for all subset regressions, although continually recalculating the elements of R^* will lead to some loss of accuracy due to compounding of rounding error.

3. Givens' rotations

The initial R^* matrix can be formed by any appropriate procedure (modified Gram-Schmidt, Householder, Givens rotations, ordinary Gram-Schmidt or Cholesky decomposition). Although columns can be shifted by more than one position and re-triangularised by means of modified Gram-Schmidt or Householder procedures, it is simplest to shift columns by only one position and use Givens' rotations to re-triangularise. Also, Givens' rotation is recommended for removing an independent variable from a model [16, pp. 336–342]. Clarke [5] has published an algorithm for doing the transposition and re-triangularisation using Givens' rotations. Givens' rotation is involved in all the algorithms for all possible subset regressions to be discussed, and details can be found in textbooks such as [16], [3], or [14]. Appendix A of the LINPACK Users' Guide [7] contains a concise summary of how to construct and apply a Givens' rotation. Construction of a Givens' rotation is calculation of the scalars c, s and r in the following matrix equation, where a and b are known $c^2 + s^2 = 1$:

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \cdot \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}$$

Basically it is calculation of the matrix necessary to transform the two element vector $[{}^a_b]$ into the two element vector $[{}^r_0]$. Applying a Givens' rotation consists of pre-multiplying a $2 \times j$ matrix by the calculated 2×2 matrix containing c and s, i.e.

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} x_{11} & \cdots & x_{1j} \\ x_{21} & \cdots & x_{2j} \end{bmatrix}.$$

For exchanging adjacent columns of the R^* matrix and re-triangularising the application of Givens' rotations is clear. For example if the four independent variable model ABCD has been fitted and the model ACB is wanted columns two and three of R^* will be exchanged resulting in

$$\begin{bmatrix} 0 & r_{23} & r_{22} & r_{24} & c_2 \\ 0 & r_{33} & 0 & r_{34} & c_3 \end{bmatrix}$$

in rows two and three. A Givens' rotation matrix is calculated on $[{}^{r_{23}}_{r_{33}}]$ to produce $[{}^{r_{22}^*}_{0}]$ where r_{22}^* is the new element in position row 2, column 2. This rotation matrix then pre-multiplies the (2×3) matrix consisting of the last three columns of the above (2×5) matrix to produce the new R^* matrix for order ACBD. The c, s and r of the Givens' rotation are given by

$$r = \text{SIGN} \cdot (a^2 + b^2)^{1/2}$$

where SIGN = the sign of a if $|a| \geqslant |b|$ or the sign of b if $|b| \geqslant |a|$,

$$\begin{aligned} c &= a/r \quad & s &= b/r \quad & &\text{if } r \neq 0, \\ &= 1 & &= 0 & &\text{if } r = 0. \end{aligned}$$

4. Sequences of column transpositions

Clarke [5] gives a sequence that produces all subset regression models in a total of $2^p - p - 1$ transpositions (p being the number of independent variables). This is the number of possible subsets $2^p - 1$ (excluding the null case) minus those given by the initial QR decomposition. Since each Givens' rotation gives only one new model this figure is a minimum for the number of transpositions required to produce all subset models given an initial R^* matrix. Clarke also gives a recursive procedure for generating the sequences of transpositions, stating that in Fortran this is most easily coded by replacing the recursive call by access to a vector of the sequence generated to date. It will be shown that the sequence generation procedure can be coded in a comparably simple way but accessing a much smaller array. Also, it will be shown that the number of arithmetic operations needed to produce the regression information for all possible subsets can be reduced by changing the order of the sequence, although the number of Givens' rotations is the same.

Clarke [5] gave as an example the sequence of transpositions generated for $p = 4$. If the variables are ABCD the sequence is:

Columns interchanged	Order of variables	Models given
	ABCD	A, AB, ABC, ABCD
1, 2	BACD	B
2, 3	BCAD	BC
1, 2	CBAD	C
2, 3	CABD	AC
3, 4	CADB	ACD
2, 3	CDAB	CD
1, 2	DCAB	D
2, 3	DACB	AD
3, 4	DABC	ABD
2, 3	DBAC	BD
3, 4	DBCA	BCD

The essential features of the recursive procedure generating this sequence can be seen by considering the series of integers corresponding to the left-hand members of the pairs of columns interchanged:

1 2 1 2	3 2 1	2 3 2 3
S1	S2	S3

The central section S2 is a sequence decreasing from 3 to 1 in steps of -1; S1 is the sequence of transpositions for 3 independent variables; and S3 is the same sequence incremented by 1. S1 can be similarly broken down into

1 | 2 1 | 2

The procedure for p variables is to take the sequence for $(p-1)$ variables, add a descending sequence from $(p-1)$ to 1 to it, and follow that with the sequence for $(p-1)$ incremented by 1.

The rationale underlying the sequence is based on an inductive argument. Suppose that, given an initial QR decomposition for $p-1$ variables, a sequence of $2^{p-1}-p$ adjacent column transpositions can be found which yields all $2^{p-1}-1$ subset models for these variables. The p variable case can then be split into two stages (a QR decomposition for the p variables having been obtained). The first stage is to obtain all $2^{p-1}-1$ models not involving the last variable, and the second stage is to obtain the remaining 2^{p-1} models involving the last variable.

For the first stage all that is required is to permute the first $p-1$ variables so that each non-null subset of them occupies a set of consecutive leading positions at some time. The sequence of transpositions for $p-1$ variables suffices for this. To prepare for the second stage the last variable is moved into the first column by a sequence of $p-1$ transpositions. At the end of this sequence p models (including the full model and that with only the original last variable) are obtainable from the resulting R matrix. To complete the second stage the variables now occupying columns 2 to p are permuted so that each non-null subset of them occupies a set of consecutive positions from column 2 at some time. This is achieved by repeating the sequence of transpositions of the first stage shifted one position to the right. The total number of transpositions involved is then

$$2 \cdot (2^{p-1} - p) + p - 1 = 2^p - p - 1$$

which is the minimum number required. This is a recursive way of generating a sequence of transpositions for any number of variables taking as a starting point the null sequence appropriate for one independent variable.

Comparing the above explanation of Clarke's sequence with the preceding four variable example, it can be seen that in the former the full model does not enter until the second stage and the moving of the last variable to the first column does not produce models until the movement is complete. It is more natural to follow the example with the full model being obtained at the start from the p variable QR decomposition, and a new model being obtained from each transposition used to move the last variable to column one.

Clarke [5] suggested that the recursive procedure was most easily implemented in Fortran by repeatedly accessing a vector of the sequence generated to date. The following Fortran 77 statements do this. The vector accessed (ITR, length $2^p - p - 1$) contains the final sequence.

```
   DO 15 I = 2, P
      IF (I .EQ. 2) THEN
         L = 1
         ITR(L) = 1
      ELSE
         LK = L
         DO 5 J = I - 1, 1, -1
            L = L + 1
            ITR(L) = J
 5       CONTINUE
         DO 10 J = 1, LK
            L = L + 1
            ITR(L) = ITR(J) + 1
10       CONTINUE
      ENDIF
15 CONTINUE
```

The all possible subset regressions would be obtained by proceeding through the vector ITR at the ith step exchanging column ITR(i) with ITR(i) + 1. However ITR is nearly as large as the number of all possible subsets. It could be approximately halved by not storing the last sequence, even so for large p it would still be sizeable. Using recursion the same sequence can be obtained with only the column to be transposed stored, though a logical vector of length ($p - 1$) is required to track the state of the recursion. This vector is STATE(2 : P) in the following Fortran 77 statements. Initially STATE(P) is set to .FALSE., becoming .TRUE. after the central descending sequence (S2) in the p variable sequence has been completed. Similarly, STATE(P − 1) tracks the progress of the two $p - 1$ variable sequences which are part of the p variable sequence. This process continues on to STATE(2). The sequence ends when all elements of STATE are .TRUE.. The scalar LEVEL is the number of variables in the current sequence being generated. The scalar N acts both as the end of each descending sequence, and the number of .TRUE. elements in STATE after each sequence has been completed and STATE updated.

```
    DO 5 I = 2, P
       STATE(I) = .FALSE.
  5 CONTINUE
    N = 1
    LEVEL = 2
 10 DO 15 COL = N + LEVEL − 2, N, −1
       (Here insert routines to update R, transposing
       columns COL and COL + 1, and fit new model)
 15 CONTINUE
    STATE(LEVEL) = .TRUE.
    DO 20 I = 2, LEVEL − 1
       STATE(I) = .FALSE.
 20 CONTINUE
    IF (N .LT. P − 1) THEN
       DO 25 LEVEL = 2, P
          IF (.NOT. STATE(LEVEL)) GO TO 30
 25    CONTINUE
 30    N = N + 3 − LEVEL
       GO TO 10
    ENDIF
```

This code uses two features of Fortran 77 not available in earlier versions of Fortran. These are the IF...THEN statement and the fact that in Fortran 77 start and end indexes in DO loops are compared prior to execution of the loop, whereas in earlier Fortrans a loop is executed at least once.

For this sequence of transpositions each time a Givens' rotation matrix is calculated it is necessary to apply it to all columns in the relevant two rows of R to the right of those exchanged. For $p = 4$ this means that 110 multiplications are required, apart from those involved in calculating the 11 Givens' rotation matrices and forming the residual sums of squares from C and s. By using the mirror image of this sequence it is possible to reduce this number. The mirror image of the sequence for $p = 4$ is

Columns interchanged	Order of variables	
	ABCD	
3, 4	ABDC	
2, 3	ADBC	
3, 4	ADCB	
2, 3	ACDB	
1, 2	CADB	
2, 3	CDAB	
3, 4	CDBA	...............a)
2, 3	CBDA	
1, 2	BCDA	
2, 3	BDCA	...............b)
1, 2	DBCA	

With this mirror image sequence column 4 of the R^* matrix is not required after a), and columns 3 and 4 after b), so that there is no need to apply the Givens' rotation to the elements in these columns after those points. Not doing so saves 20 multiplications $(4 + 4 + 4 + 8)$.

5. An alternative to sequences of column transpositions

As each Givens' rotation only produces one new model the figure of $2^p - p - 1$ is a minimum for the number of rotations required. Therefore reduction in the number of arithmetic operation to obtain all models must be achieved by reducing the number of operations used applying the Givens' rotation. The fact central to the algorithm to be described is that dropping a column from the R^* matrix and re-triangularising involves Givens' rotation calculations and application, but this application is over a matrix reduced in size. The easiest way of describing the algorithm is by example. Again taking the case $p = 4$ the algorithm is

Step		Column dropped	No. Givens rotations	No. of multiplications	Models given
0	ABCD(Store)				A, AB, ABC, ABCD
1	ABCD(From store)				
	↓	3	1	2	
	AB′D				ABD
2	ABCD(From store)				
	↓	2	2	10	
	A′CD				AC, ACD
	↓	2	1	2	
	A′D				AD
3	ABCD(From store)				
	↓	1	3	22	
	′BCD(Store)				B, BC, BCD
	↓	1	2	10	
	′CD				C, CD
	↓	1	1	2	
	′D				D
4	′BCD(From store)				
	↓	2	1	2	
	B′D				BD

This algorithm uses a total of 11 Givens' rotations but only 50 multiplications, again excluding those to form the residual sums of squares from C and s. Two rather than four multiplications are required when applying the last Givens' rotation to the vector C because only one c needs calculating.

Regression trees like those of [9] can be drawn. The one for $p = 4$ is

ABCD

′BCD A′CD AB′D

′CD B′D A′D

′D

As in the algorithm in [9] temporary storage is used. This storage is of the original R^* matrix, together with all matrices along a branch with more than two columns to the right of the column dropped (in the tree diagram this is signified by a ′). As given here, within matrices that are not the root ABCD columns are dropped in a left to right order e.g. ′BCD drop B to give ′CD (evaluate all the ′CD branch and return to ′BCD), then drop C to give B′D. Within the root ABCD columns are dropped from right to left e.g. ABCD drop C to give AB′D (evaluate all the AB′D branch and return to ABCD), then drop B to give A′CD (evaluate all the A′CD branch and return to ABCD), etc. Other orders are possible and a range of such are given in [9]. Basically, regardless of order all columns except the last are sequentially dropped from the R^* matrix which is re-triangularised by Givens' rotations, those new R^* matrices with more than two columns to the right of that dropped being placed in temporary storage for future use.

So far the special nature of the last column of R^* (i.e. the vector C and scalar s) and the calculations on it necessary to construct the residual sums of squares have not been considered. The total sum of squares is obtainable from C and s and rather than storing the residual sums of squares in a work vector the fitted sums of squares can be stored instead. Dropping columns resulting in changing values of c can then produce the residual sum of squares with only one multiplication per model, although a subtraction is required. Thus two multiplications are required to calculate the residual sum of squares in addition to those involved in calculating the Givens' rotations and applying them. For example for $p = 4$ five multiplications form the total sum of squares, and work vector

$$\left(c_1^2,\ c_1^2 + c_2^2,\ c_1^2 + c_2^2 + c_3^2,\ c_1^2 + c_2^2 + c_3^2 + c_4^2\right)$$

whose elements are the fitted sums of squares for models A, AB, ABC, ABCD. Dropping column 3 to get ABD changes c_3 with the fitted sum of squares wanted being $c_1^2 + c_2^2 +$ new c_3^2. Therefore only one multiplication ($c_3 \times c_3$) is required since $c_1^2 + c_2^2$ is element 2 of the work vector. Similarly for the other models.

The total operation counts for the algorithms as discussed so far for $p = 4$ are those involved in calculating the 11 Givens' rotations, plus 125 multiplications for Clarke's [5] sequence of transpositions, 105 for the modified sequence of transpositions, and 66 for the dropping of columns. Therefore the dropping columns algorithm has a saving of 59 operations over Clarke's [5] sequence. Other savings of operations can be made but these depend on the form of the R^* matrix used and concern the calculation and application of Givens' rotations in such circumstances.

6. Alternative forms of Givens' rotation

The Givens' rotation as described in Section 3 uses square roots. As these can be expensive in computer time it is desirable to avoid doing them. For the QR decomposition square root free methods of calculating a modified version of the R^* matrix are known and algorithms have been published (e.g. [6,5,17,18]). Similarly the Givens' rotation algorithm of [5] for moving between models does not involve square roots, the process of transposition and re-triangularisation on this modified version of the R^* matrix only involving multiplication and division. If R^* is in standard form as described in Section 2 the modified version used by

Clarke for $p = 3$ is

$$\begin{bmatrix} d_1 & r_{12}/r_{11} & r_{13}/r_{11} & c_1/r_{11} \\ 0 & d_2 & r_{23}/r_{22} & c_2/r_{22} \\ 0 & 0 & d_3 & c_3/r_{33} \\ 0 & 0 & 0 & d_4 \end{bmatrix}$$

where $d_i = r_{ii}^2$, $i = 1, 2, 3$ and $d_4 = s^2$.

If the vector of diagonal elements is denoted by D, formulae for transposing columns and re-triangularising are given in [5].

Stirling [17] uses a different modified form of R^* with the same off diagonal elements as [5] but a diagonal of the reciprocal of r_{ii}^2. No subroutine for transposing adjacent columns is given in the collection published. However that of [5] (GTRANS) can be adapted. If D now denotes the vector of diagonal elements as stored by Stirling (i.e. the reciprocals of those stored by Clarke) the updating formulae are

$$\begin{aligned}
&d'_{i+1} = d_i + d_{i+1} \cdot r_{i,i+1}^2,\\
&e = d_{i+1}/d'_{i+1},\\
&d'_i = e \cdot d_i,\\
&r'_{i,i+1} = e \cdot r_{i,i+1},\\
&r'_{i,j} = r'_{i,i+1} \cdot r_{i,j} + e \cdot r_{i+1,j} \quad \text{for } j = i+2, \dots, p+1,\\
&r'_{i+1,j} = r_{i,j} - r_{i,i+1} \cdot r_{i+1,j} \quad \text{for } j = i+2, \dots, p+1,
\end{aligned}$$

where r'_{ij} is the element in the ith row, jth column of the modified R^*, and columns i and $i+1$ are transposed.

Similar updating formulae exist when column i is dropped although the situation is more complicated with that part of R^* with row value equal to or greater than i, and column value greater than i, requiring several Givens' rotations to be re-triangularised. For an R^* matrix stored in the form used in [5] the updating formulae for the first Givens' rotation when column i is dropped are

$$\begin{aligned}
&d'_k = d_{k+1} + d_k \cdot r_{k,k+1}^2,\\
&e_0 = -r_{k,k+2} + r_{k,k+1} \cdot r_{k+1,k+2},\\
&e_1 = d_k/d'_k,\\
&e_2 = d_{k+1}/d'_k,\\
&d'_{k+1} = e_1 \cdot d_{k+1},\\
&e_1 = e_1 \cdot r_{k,k+1},\\
&d'_{k+1} = e_0^2 \cdot d_{k+1},\\
&r'_{k,j} = r_{k,j+1} \cdot e_1 + r_{k+1,j+1} \cdot e_2 \quad \text{for } j = k+1, \dots, p-1,\\
&r'_{k+1,j} = (-r_{k,j+1} + r_{k,k+1} \cdot r_{k+1,j+1})/e_0 \quad \text{for } j = k+2, \dots, p-1,
\end{aligned}$$

where $k = i$ and $r'_{k,j}$ is the element in the kth row, jth column of the modified

R^*. Subsequent Givens' rotations with $k > i$ have the same updating formulae except that $r_{k,k+1}$ is replaced by 1.0 in all expressions where it occurs, e.g.

$$d'_k = d_{k+1} + d_k$$

Similarly for an R^* matrix stored in the form used by Stirling the updating formulae for the first Givens' rotation when column i is dropped are

$$e_0 = (-r_{k,k+2} + r_{k,k+1} \cdot r_{k+1,k+2}),$$
$$e_1 = d_k + d_{k+1} \cdot r^2_{k,k+1},$$
$$d'_k = d_k \cdot d_{k+1}/e_1,$$
$$e_2 = d_k/e_1,$$
$$e_3 = r_{k,k+1} \cdot d_{k+1}/e_1,$$
$$r'_{k,j} = r_{k,j+1} \cdot e_3 + r_{k+1,j+1} \cdot e_2 \quad \text{for } j = k+1, \ldots, p-1,$$
$$d'_{k+1} = e_1/e_0^2,$$
$$r'_{k+1,j} = (-r_{k,j+1} + r_{k,k+1} \cdot r_{k+1,j+1})/e_0 \quad \text{for } j = k+2, \ldots, p-1,$$

where $k = i$ and $r'_{k,j}$ is the element in the kth row, jth column of the modified R^*. Subsequent Givens' rotations with $k > i$ have the same updating formulae except that $r_{k,k+1}$ is replaced by 1.0 in all expressions where it occurs, e.g.

$$e_0 = -r_{k,k+2} + r_{k+1,k+2}$$

Also in both cases those elements of R^* with row value less than i but column value greater than i need shifting one position to the left in the row.

7. Comparisons of operation counts, times and accuracy

The operation count for the sweep method of [10] is given as

$$2^{p+3} - (p^3 + 9p^2 + 32p + 48)/6$$

where p is the number of independent variables. Because the correlation matrix was used this is actually the number of floating point multiplications as there were no divisions. If the sums of squares and cross products matrix is used

$$2^{p+2} - (p^2 + 5p + 8)/2$$

floating point divisions will also be required.

The different storage forms of R^* used in the QR decomposition methods affect the number and type of operations required. For the sequence of adjacent transpositions as in [5], with the R^* matrix stored in standard form, the number of operations is

$(p+3) \cdot 2^{p+1} - (2p^2 + 7p + 6)$ multiplications,

$2 \cdot (2^p - p - 1)$ divisions, and

$(2 - p - 1)$ square roots.

For the same sequence but using the form of storage used by Clarke [5], the number of operations is

$$(3p+12)\cdot 2^{p-1}-(3p^2+11p+16)/2 \text{ multiplications}$$

and

$$2\cdot(2^p-p-1) \text{ divisions.}$$

For the same sequence but using the form of storage used by Stirling [17], the number of operations is

$$(3p+10)\cdot 2^{p-1}-(3p^2+11p+12)/2 \text{ multiplications}$$

and

$$3\cdot 2^p-2p-3 \text{ divisions.}$$

These operation counts for the sequence of adjacent column transpositions as in [5] implemented with different forms of storage of R^* were calculated using the fact that the number of transpositions of a particular column is the appropriate coefficient from the binomial series for p minus one e.g., for $p=4$ column 3, $(4!/(4-3)!3!)-1$ is 3, i.e. column 3 is transposed 3 times. The number of operations used to transpose each column is a function of column number so that a summation series results with additional counts sometimes added for forming residual sums of squares (depending on the operation being counted). Formulae for summing series of binomial coefficients (sum of binomial coefficients $=2^n$) and power series ($\Sigma_{i=1}^{i=n} i\cdot 2^i=2+(n-1)\cdot 2^{n+1}$) were used to produce simple expressions.

In a similar manner expressions for the other algorithms can be calculated. For the modified sequence of adjacent transpositions with R^* stored in standard form, the number of operations is

$$(p+1)\cdot 2^{p+1}-(5p+2) \text{ multiplications,}$$

$$2\cdot(2^p-p-1) \text{ divisions, and}$$

$$(2^p-p-1) \text{ square roots.}$$

For the dropping columns algorithm with R^* stored in standard form, the number of operations is

$$11\cdot 2^p-2p^2-9p-9 \text{ multiplications}$$

$$2\cdot(2^p-p-1) \text{ divisions, and}$$

$$(2^p-p-1) \text{ square roots.}$$

For the same algorithm but using the form of storage used by Clarke [5], the number of operations is

$$39\cdot 2^{p-2}-p^2-7p-10 \text{ multiplications}$$

and

$$6\cdot 2^{p-1}-(p^2+5p+6)/2 \text{ divisions.}$$

Table 1
Operation counts

Method & storage	Algorithm		No. of variables			
			5	10	15	20
SWEEP	GA	No. x	64	3797	130238	4192510
		No. /	42	1981	65399	2096920
QR Decomp. stored in standard form	S1	No. x	158	12057	556560	23067811
		No. /	22	1004	32738	1048536
		Sqrt	11	502	16369	524268
	S2	No. x	138	10193	491448	20971423
		No. /	22	1004	32738	1048536
		Sqrt	11	502	16369	524268
	DC	No. x	99	5380	179697	5766266
		No. /	22	1004	32738	1048536
		Sqrt	11	502	16369	524268
QR Decomp. stored in Clarke's form	S1	No. x	138	9805	441989	18087282
		No. /	22	1004	32738	1048536
	DC	No. x	102	4838	159440	5111304
		No. /	27	1470	49016	1572633
QR Decomp. stored in Stirling's form	S1	No. x	124	9295	425607	17562996
		No. /	37	1515	49121	1572823
	DC	No. x	90	4335	143070	4587035
		No. /	58	2731	89947	2883314

GA: sweep algorithm of [10]
S1: sequence of transpositions as [5]
S2: modified sequence
DC: the dropping columns algorithm

For the same algorithm but using the form of storage used by Stirling [17], the number of operations is

$$35 \cdot 2^{p-2} - p^2 - 6p - 10 \text{ multiplications}$$

and

$$11 \cdot 2p^{-1} - \left(p^2 + 9p + 8\right)/2 \text{ divisions.}$$

To illustrate the actual numbers of operations involved in the different methods Table 1 displays the operation counts for 5, 10, 15 and 20 variables (including the dependent).

To compare the various methods in terms of running times Fortran programs implementing them were written and timings of the appropriate parts of these

Table 2
Mean and s.d. (in seconds) of ten repeated runs of a 15 variable data set

Method & storage	Algo-rithm	Machine			
		IBM 3090		DEC VAX	
		Mean	s.d.	Mean	s.d.
SWEEP	GA	1.422	0.0072	10.176	0.145
QR Decomp.	S1	1.683	0.0060	15.498	0.587
standard	S2	1.980	0.0082	16.970	0.228
storage	DC	1.925	0.0073	18.482	0.267
QR Decomp.	S1	1.643	0.0073	13.088	0.567
Clarke's	DC	1.838	0.0076	11.782	0.485
storage					
QR Decomp.	S1	1.662	0.0054	12.850	0.167
Stirling's	DC	1.842	0.0065	11.428	0.099
storage					

GA: sweep algorithm of [10]
S1: sequence of transpositions as [5]
S2: modified sequence
DC: the dropping columns algorithm

programs were made. An optimising compiler was always used. For the sweep method an Algol version of the algorithm in [10] has been published [11] and a Fortran translation of this was used. The QR decompositions were formed using the appropriate routines from LINPACK [7] for the standard form; from Clarke [5] for the form he used; and from Stirling [17] for the form he used. Adjacent columns were transposed using a version of the Givens' algorithm as in LINPACK [7] with the operation count per rotation reduced (to 3 multiplications, 2 divisions and a square root); the appropriate routine from [5]; and a version of this routine modified to the form of storage used by Stirling [17]. Clarke's [5] sequence of column transpositions was produced using the second set of Fortran statements in Section 4. It was easy to modify these statements to produce the mirror image sequence. However, not applying the Givens' rotation to certain elements of R^* is more difficult as it requires additional coding inserted into the transposition/re-triangularisation routines. This was only done for the R^* matrix in standard form. These programs were run with counters summing the various operation counts to confirm the counts in Table 1.

A number of runs of these programs were made on two different machines, an IBM 3090 and a DEC VAX 11-780. The means and standard deviations of ten runs of a data set with 15 variables are given in Table 2. Apart from saying that the sweep method is faster than the QR based methods it is difficult to make any substantive statements about comparative performance from Table 2. There is a suggestion that using the standard storage form of R^* (and hence square roots) increases run time on the DEC VAX. Both operating systems under which the

Table 3
Draper & Smith steam target data. Single-double precision difference as hundredths of a percent of the residual sum of squares

Method & storage	Algo-rithm	Statistics of distribution of differences					
		IBM 3090			DEC VAX		
		Maximum	Minimum	Range	Maximum	Minimum	Range
SWEEP	GA	−0.043	−11.604	11.561	0.238	0.002	0.236
QR Decomp.	S1	0.003	−0.507	0.510	0.015	−0.009	0.024
standard	S2	−0.009	−0.438	0.429	0.044	−0.006	0.050
storage	DC	0.183	−0.097	0.280	0.020	−0.017	0.037
QR Decomp.	S1	0.142	−0.903	1.045	0.023	−0.031	0.054
Clarke's	DC	0.324	−0.198	0.522	0.016	−0.014	0.028
storage							
QR Decomp.	S1	0.284	−0.667	0.951	0.015	−0.012	0.027
Stirling's	DC	0.128	−0.112	0.240	0.018	−0.015	0.033
storage							

GA: sweep algorithm of [10]
S1: sequence of transpositions as [5]
S2: modified sequence
DC: the dropping column algorithm

Table 4
Random data. Single–double precision difference as hundredths of a percent of residual sums of squares

Method & storage	Algo-rithm	Statistics of distribution of differences					
		IBM 3090			DEC VAX		
		Maximum	Minimum	Range	Maximum	Minimum	Range
SWEEP	GA	−0.039	−4.133	4.094	0.0226	−0.0007	0.0233
QR Decomp.	S1	0.036	−0.076	0.112	0.0048	−0.0004	0.0052
standard	S2	−0.017	−0.077	0.060	0.0054	−0.0003	0.0057
storage	DC	−0.053	−0.096	0.043	0.0032	0.0010	0.0022
QR Decomp.	S1	−0.024	−0.126	0.102	0.0064	0.0023	0.0041
Clarke's	DC	−0.024	−0.060	0.036	0.0051	0.0020	0.0031
storage							
QR Decomp.	S1	0.018	−0.086	0.104	0.0043	0.0005	0.0038
Stirling's	DC	0.018	−0.021	0.039	0.0038	0.0012	0.0026
storage							

GA: sweep algorithm of [10]
S1: sequence of transpositions as [5]
S2: modified sequence
DC: the dropping columns algorithm

programs were run are multi-tasking systems and although the runs were made at times of light machine load other jobs would intrude and affect the timings, causing them to vary. The variability in run time was markedly worse for the DEC VAX.

To gain an idea of the comparative accuracy of these algorithms values of the residual sum of squares from single and double precision versions (32 and 64 bit words respectively) of the programs were compared for various sets of mean centred data. (It is common practice to use mean centred data [16, pp. 330–336].) The residual sums of squares from the double precision programs all agreed exactly with each other within the ranges of the Fortran list directed WRITE command. The set of double precision residual sums of squares was taken as the standard and differences between it and the sets of single precision results were calculated. Tables 3 and 4 give summary statistics of the distribution of these differences. The two data sets are the steam target data of [8, pp. 351–364] and a set of random numbers of equivalent size produced using the algorithm in [19]. Both data sets are 25 recordings of 10 variables ($p = 9$, number of models 511). As expected the QR based methods are all more accurate than sweep as assessed by range of double–single precision difference. An interesting feature of the results in these tables is the negative bias of the differences for the IBM 3090, a bias epitomised by the asymmetry of the maxima and minima about zero. This could be caused by the policy of IBM arithmetic to round numbers down.

8. Discussion and conclusions

It is generally accepted that use of QR decomposition methods to solve regression equations gives more accurate results than sweep based methods. Although the degree of accuracy needed by statisticians when using regression methods can be debated [13,2] it is a major factor to be considered when comparing algorithms. Another major factor is cost which encompasses both time and storage. Miller [15] remarks that in his experience generating the residual sums of squares for all subset models by Givens' rotations of the R^* matrix from the QR decomposition gives more accurate results than the sweep algorithm, is only slightly slower (25%), and can be used with single precision arithmetic. [4] also compares Givens' rotations with sweeps although not specifically in the all possible subset context, and the views expressed match Miller's. The results presented here agree with their remarks.

To the best of our knowledge the only model generation procedure published for use with the QR decomposition is that of [5]. The dropping columns algorithm described is an alternative with certain advantages. Because it uses intermediate matrix storage with fewer operations being performed on a matrix the dropping columns algorithm should be most accurate. The results of Tables 3 and 4 support this. The formulae beginning Section 7 and the operation counts of Table 1 show a significant reduction in count has been achieved, and by implication run time should be reduced. Unfortunately more coding in the computer programs was required so that the net effect was only a small decrease in run time at best, although the results in Table 2 imply that at worst the dropping columns algorithm is little slower than Clarke's algorithm. Operation count and run time are two measures of cost, however, as remarked on in [3, pp. 9-13, pp. 118-120), there are other considerations such as convenience and computing strategy.

An important feature of the dropping columns algorithm is its similarity to the branch and bound algorithm of [9]. This algorithm using sweeps is the standard one for obtaining best-fitting subsets of different sizes. It is hoped that this similarity can be exploited to develop branch and bound algorithms for use with the QR decomposition methods and this is to be the subject of future work.

The conclusion is that the dropping columns algorithm using Givens' rotations is a viable procedure for producing all possible subset models.

Acknowledgements

Much of this work was performed by the first author whilst employed by the Wellcome Foundation Limited. Use of Wellcome facilities is gratefully acknowledged. We are indebted to Dr S. Pack for the original translation into Fortran of the algorithm in [11], to Dr B. Vowden and Dr T. Hopkins for their advice, and to Mr D. Noon for assistance with file transfer.

References

[1] A.E. Beaton, The use of special matrix operators in statistical calculus, *Research Bulletin, Educational Testing Service, Princeton, New Jersey* (1964).

[2] R.L. Branham, Jr., Are orthogonal transformations worthwhile for least squares problems?, *A.C.M. SIGNUM Newsletter* **22** (1) (1987) 14–19.

[3] J.M. Chambers, *Computational Methods for Data Analysis* (Wiley, New York, 1977).

[4] M.R.B. Clarke, Choice of algorithm for a model-fitting system, *COMPSTAT 80, Proceedings in Computational Statistics* (Physica-Verlag, Vienna, 1980).

[5] M.R.B. Clarke, Algorithm AS163. A Givens algorithm for moving from one linear model to another without going back to the data, *Appl. Statist.* **30** (1981) 198–203.

[6] D.G. Clayton, Algorithm AS46. Gram-Schmidt orthogonalization, *Appl. Statist.* **20** (1971) 335–338.

[7] J.J. Dongarra, C.B. Moler, J.R. Bunch and C.W. Stewart, *LINPACK USERS GUIDE* (SIAM, Philadelphia, 1979).

[8] N.R. Draper and H. Smith, *Applied Regression Analysis* (Wiley, New York, 1966).

[9] G.M. Furnival and R.W. Wilson Jr., Regression by leaps and bounds, *Technometrics* **16** (1974) 499–511.

[10] M.J. Garside, Some computational procedures for the best subset problem, *Appl. Statist.* **20** (1971) 8–15.

[11] M.J. Garside, Algorithm AS38. Best subset search, *Appl. Statist.* **20** (1971) 112–115.

[12] J.H. Goodnight, A tutorial on the SWEEP operator, *The American Statistician* **33** (1979) 149–158.

[13] N.J. Higham and G.W. Stewart, Numerical linear algebra in statistical computing, *Proceedings of the IMA/SIAM conference on "The State of the Art in Numerical Analysis"* (Birmingham, April 1986).

[14] C.L. Lawson and R.J. Hanson, *Solving Least Squares Problems* (Prentice-Hall, Englewood Cliffs, 1974).

[15] A.J. Miller, Selection of subsets of regression variables, *J.R. Statist. Soc. A* **147** (1984) 389–425.

[16] G.A.F. Seber, *Linear Regression Analysis* (Wiley, New York, 1977).

[17] W.D. Stirling, AS164: Least squares subject to linear constraints, *Appl. Statist.* **30** (1981) 204–212.

[18] W.D. Stirling, Correction to AS164: Least squares subject to linear constraints, *Appl. Statist.* **30** (1981) 357.

[19] B.A. Wichmann and I.D. Hill, AS183: An efficient and portable pseudo-random number generator, *Appl. Statist.* **31** (1982) 188–190.