J. Austral. Math. Soc. **22** (Series B) (1980), 2–11

ON THE IMPLEMENTATION OF A SUBSET SELECTION ALGORITHM FOR THE RESTRICTED LEAST SQUARES PROBLEM

D. I. CLARK and M. R. OSBORNE

(Received 14 September 1979)

(Revised 26 November 1979)

Abstract

By noting that it is possible to interchange the roles of the solution vector **x** and the vector of Lagrange multipliers **λ** in the restricted least squares problem we are able to give a very efficient implementation of Clark's subset selection algorithm Numerical results are presented for several selection heuristics.

1. Introduction

Two main approaches have proved popular for the solution of the restricted least squares problem :

$$\min_{\mathbf{x} \geqslant 0} \tfrac{1}{2}(\mathbf{b} - A\mathbf{x})^T(\mathbf{b} - A\mathbf{x}), \tag{1}$$

where A is an $m \times n$ matrix with $m > n$, and $(\text{rank } A) = n$. The first approach identifies (1) as a quadratic programming problem of a rather simple kind and uses techniques familiar from linear programming to obtain a solution [2]. The second considers (1) as a subset selection problem and uses a branch and bound procedure to search for the optimum set of the x_i [1]. This latter approach has been advocated as it solves a standard regression problem at each step and so can make use of readily available computer software. Recently Clark has considered the subset selection approach carefully with a view to developing pruning rules to minimize the searching involved, and in [3] suggests an algorithm which appears to be efficient in terms of the number of different subproblems which have to be solved (the number of

https://doi.org/10.1017/S0334270000002496 Published online by Cambridge University Press

nodes of the search tree which have to be considered). In this paper we consider the implementation of Clark's algorithm. In particular we want to provide methods which avoid solving each of the unconstrained least squares problems *ab initio* when only one of the variables is changed at each step. The key to our approach is suggested by the Kuhn–Tucker conditions which characterize the unique minimum of (1), with uniqueness following from the strict convexity of the objective function and the linearity of the constraints. These conditions are

$$-A^{\mathrm{T}}(\mathbf{b}-A\mathbf{x}) = \boldsymbol{\lambda}, \tag{2a}$$

$$\boldsymbol{\lambda} \geqslant 0, \quad \mathbf{x} \geqslant 0, \tag{2b}$$

and

$$\lambda_i x_i = 0, \quad i = 1,2,\ldots,n, \tag{2c}$$

so that the subset selection problem can be restated as that of seeking among all solutions satisfying the system of equations (2a) and the complementarity condition (2c) the unique pair satisfying $\boldsymbol{\lambda} \geqslant 0$, $\mathbf{x} \geqslant 0$.

From our point of view the striking feature of this formulation is the symmetry between the roles of $\mathbf{x}$ and $\boldsymbol{\lambda}$. *In particular it is possible to interchange the roles of* $\mathbf{x}$ *and* $\boldsymbol{\lambda}$ *in Clark's algorithm.* This has the advantage that a certain amount of initial processing is avoided. For example, Clark started with $\mathbf{x}$ as the unconstrained minimizer of (1), and this is equivalent to rewriting (2a) in the form

$$(A^{\mathrm{T}}A)^{-1}\boldsymbol{\lambda} - \mathbf{x} = -(A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}}\mathbf{b} \tag{3}$$

and satisfying (2c) by setting $\boldsymbol{\lambda} = 0$.

By the complementarity condition (2c), fixing a particular component of $\mathbf{x}$ at zero is equivalent to freeing the corresponding component of $\boldsymbol{\lambda}$. Thus we consider at each stage a partition of $\mathbf{x}$,

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} \quad \text{with } \mathbf{x}_2 = 0, \tag{4a}$$

and a corresponding partition of $\boldsymbol{\lambda}$,

$$\boldsymbol{\lambda} = \begin{bmatrix} \boldsymbol{\lambda}_1 \\ \boldsymbol{\lambda}_2 \end{bmatrix} \quad \text{with } \boldsymbol{\lambda}_1 = 0, \tag{4b}$$

which ensure automatically that (2c) is satisfied. If (2a) is now solved for the variables permitted to be nonzero at the current stage then it can be written

$$\boldsymbol{\sigma}_A - M\boldsymbol{\sigma}_I = -\mathbf{q}, \tag{5}$$

where

$$\boldsymbol{\sigma}_A = \begin{bmatrix} \mathbf{x}_1 \\ \boldsymbol{\lambda}_2 \end{bmatrix} \quad \text{and} \quad \boldsymbol{\sigma}_I = \begin{bmatrix} \boldsymbol{\lambda}_1 \\ \mathbf{x}_2 \end{bmatrix}. \tag{6}$$

Each step of the algorithm involves interchanging a component of $\mathbf{x}$ with the corresponding component of $\boldsymbol{\lambda}$. This results in a transformation of (5) which can be represented by multiplication by an elementary Jordan matrix followed by appropriate permutations to partition the new variables into the form (6). We define the Jordan matrix J_i by

$$J_i \kappa_i(M) = (I - \mathbf{j}_i \mathbf{e}_i^{\mathrm{T}}) \kappa_i(M) = -\mathbf{e}_i, \tag{7}$$

where i is the index of the element of $\boldsymbol{\sigma}_I$ to be exchanged, and $\kappa_i(\,.\,)$ indicates that the ith column is taken. Using a bar to denote transformed quantities we have

$$\bar{\mathbf{q}} = \mathbf{q} - q_i \mathbf{j}_i,$$

so

$$\left.\begin{aligned} \bar{q}_k &= q_k - \frac{q_i M_{ik}}{M_{ii}}, \quad k \neq i, \\ \bar{q}_i &= -\frac{q_i}{M_{ii}}, \end{aligned}\right\} \tag{8a}$$

and

$$\begin{aligned} \kappa_k(\overline{M}) &= (I - \mathbf{j}_i \mathbf{e}_i^{\mathrm{T}}) \kappa_k(M) \\ &= \kappa_k(M) - \frac{M_{ik}}{M_{ii}} \{\kappa_i(M) + \mathbf{e}_i\}, \quad k \neq i. \end{aligned} \tag{8b}$$

When $k = i$, the ith column of $\overline{M}$ comes as a result of the interchange $\lambda_i \leftrightarrow x_i$. This gives

$$\begin{aligned} \kappa_i(\overline{M}) &= -(I - \mathbf{j}_i \mathbf{e}_i^{\mathrm{T}}) \mathbf{e}_i \\ &= \frac{1}{M_{ii}} \{\kappa_i(M) + \mathbf{e}_i\} - \mathbf{e}_i. \end{aligned} \tag{8c}$$

This shows that the computations involved in Clark's algorithm can be carried out in a manner familiar from stepwise regression [4]. However, the problem set-up still involves the calculation of the normal matrix which is a significant initial computation.

An alternative to forming the normal matrix is to apply orthogonal transformations to the data matrix. This is known to have superior numerical properties [7], but it is interesting that in the stepwise regression case it is known to be more efficient for an important range of values of m and n [9]. This approach is considered in the next section. It turns out that set-up time can be considerably reduced by working with the multiplier vector $\boldsymbol{\lambda}$, and there is an unexpected bonus for $\boldsymbol{\lambda}_2$ turns out to be a numerically better determined quantity than $\mathbf{x}_1$. Numerical results, including a comparison with the quadratic programming approach, are presented in Section 3.

2. Use of orthogonal transformations

To derive the equations satisfied by $\mathbf{x}_1$ and λ_2 we assume that the orthogonal transformation of the data matrix is given by

$$A = Q\begin{bmatrix} U \\ 0 \end{bmatrix} \quad \text{and} \quad Q^{\mathrm{T}}\mathbf{b} = \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \end{bmatrix}, \tag{9}$$

where Q is orthogonal and U upper triangular. Substituting in (2a) gives

$$\lambda - U^{\mathrm{T}}U\mathbf{x} = -U^{\mathrm{T}}\mathbf{c}_1$$

or

$$U^{-\mathrm{T}}\lambda - U\mathbf{x} = -\mathbf{c}_1. \tag{10}$$

We partition U and $\mathbf{c}_1$ to conform with (4) by setting

$$U = \begin{bmatrix} U_1 & U_{12} \\ 0 & U_2 \end{bmatrix} \quad \text{and} \quad \mathbf{c}_1 = \begin{bmatrix} \mathbf{c}_{11} \\ \mathbf{c}_{12} \end{bmatrix}, \tag{11}$$

so that (10) reduces to the pair of equations

$$U_1\mathbf{x}_1 = \mathbf{c}_{11} \quad \text{and} \quad \lambda_2 = -U_2^{\mathrm{T}}\mathbf{c}_{12}. \tag{12}$$

The interchange of a pair λ_i, x_i destroys the form of U unless the last element of $\mathbf{x}_1$ becomes the first element of λ_2 or vice versa. Thus the upper triangular form of U must be restored following an interchange, and this can be done using the now standard techniques treated in detail in [5]. For example, to drop the kth element of $\mathbf{x}_1$ which we assume to be of length $p > k$, we perform the interchanges $k+1 \to k$, $k+2 \to k+1, \ldots, k \to p$ on the columns of U_1 and then sweep out the elements introduced in the sub-diagonal positions using plane rotations $W\{j, j+1, (j+1, j)\}$, $j = k, k+1, \ldots, p-1$ where $W\{i, j, (p, q)\}$ is the plane rotation mixing rows i and j and making zero the element in the (p, q) position. Similarly, to add an element to $\mathbf{x}_1$ the corresponding column (say k) of U_2 is moved to column 1 by the sequence of interchanges $1 \to 2, 2 \to 3, \ldots, k \to 1$, and the upper triangular form is restored by the sequence of plane rotations $W\{j, j+1, (j+1, 1)\}, j = k-1, \ldots, 1$. These operations are shown schematically in Fig. 1. The interchanges are indicated by arrows, elements eliminated are circled, and elements introduced are labelled by the rotation number.

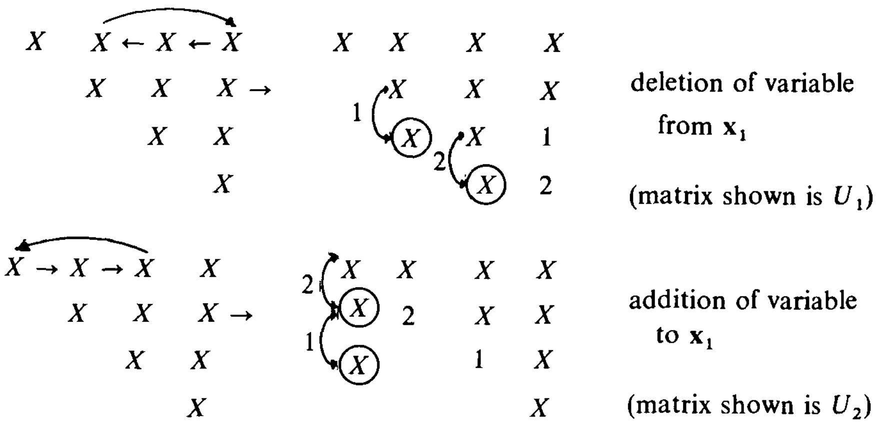
deletion of variable from $\mathbf{x}_1$
(matrix shown is U_1)
addition of variable to $\mathbf{x}_1$
(matrix shown is U_2)

Fig. 1. Transformations for addition and deletion of variables.

The algorithm can now proceed as before. However, although it appears from the above description that the initial set up time includes the factorization (9), the observation that it is possible to work with λ instead of $\mathbf{x}$ makes it possible to start the algorithm without any pre-processing of the data matrix A. The key point is that λ_2 can be determined once the transformation necessary for the calculation of the complementary set $\mathbf{x}_1$ has been carried out, although $\mathbf{x}_1$ need not be computed unless $\lambda_2 \geqslant 0$. The modification to the algorithm is explained by considering the first step which is typical. Note that initially $\mathbf{x} = \mathbf{x}_2^{(1)} = 0$ so that (2a) gives

$$\lambda_2^{(1)} = -U^{\mathrm{T}}\mathbf{c}_1 = -A^{\mathrm{T}}\mathbf{b}, \tag{13}$$

where the superscript indicates step number. Using a Householder transformation (say) to sweep out the first column of A gives

$$H_1 A = \begin{bmatrix} U_{11} & U_{12} & \dots & U_{1n} \\ 0 & X & \dots & X \\ \vdots & \vdots & & \vdots \\ 0 & X & \dots & X \end{bmatrix} \quad \text{and} \quad H_1 \mathbf{b} = \begin{bmatrix} c_1 \\ X \\ \vdots \\ X \end{bmatrix}$$

Now, from (12), we have

$$\lambda_2^{(1)} = -\begin{bmatrix} U_{11} & \\ U_{12} & \\ \vdots & U_2^{(2)\mathrm{T}} \\ U_{1n} & \end{bmatrix} \begin{bmatrix} c_1 \\ \mathbf{c}_{12}^{(2)} \end{bmatrix}$$

$$= -c_1 \begin{bmatrix} U_{11} \\ U_{12} \\ \vdots \\ U_{1n} \end{bmatrix} + \begin{bmatrix} 0 \\ \lambda_2^{(2)} \end{bmatrix}, \tag{14}$$

showing that the Lagrange multipliers can be updated and decisions made on the order in which the remaining columns of A are swept out as the factorization of A proceeds. Essentially no set-up computations are required for this form of the algorithm.

This relation can be given a general form. We partition Q so that (9) is written

$$\begin{bmatrix} Q_1^T \\ Q_2^T \end{bmatrix} A = \begin{bmatrix} U \\ 0 \end{bmatrix} \quad \text{and} \quad Q_1^T \mathbf{b} = \mathbf{c}_1. \tag{15}$$

Partitioning A and Q_1^T in conformity with $\mathbf{x}$ we obtain

$$Q_{11}^T[A_1 \ \ A_2] = [U_1 \ \ U_{12}] \quad \text{and} \quad Q_{12}^T[A_1 \ \ A_2] = [0 \ \ U_2], \tag{16}$$

so that

$$-\lambda_2 = U_2^T \mathbf{c}_{12} = A_2^T Q_{12} Q_{12}^T \mathbf{b}$$

and, using (16),

$$-\begin{bmatrix} \mathbf{0} \\ \lambda_2 \end{bmatrix} = A^T Q_{12} Q_{12}^T \mathbf{b}$$

$$= A^T[I - Q_{11} Q_{11}^T] \mathbf{b}, \tag{17}$$

as

$$Q_{12} Q_{12}^T = I - Q_{11} Q_{11}^T - Q_2 Q_2^T \quad \text{and} \quad A^T Q_2 = 0.$$

In particular, the general form for (14) is

$$-\lambda_2 = A_2^T \mathbf{b} - U_{12}^T \mathbf{c}_{11}, \tag{18}$$

and this confirms that the multiplier vector is available when only the transformation of A necessary to compute $\mathbf{x}_1$ has been completed.

Equation (18) is useful also as it permits an error analysis for this method of computing λ_2 to be given. Indicating computed quantities by bars we have

$$\bar{\lambda}_2 - \lambda_2 = \delta\lambda = \bar{U}_{12}^T \bar{\mathbf{c}}_{11} - A_2^T Q_{11} Q_{11}^T \mathbf{b} + \varepsilon$$

$$= \{\bar{U}_{12}^T - A_2^T Q_{11}\} \bar{\mathbf{c}}_{11} + A_2^T Q_{11}\{\bar{\mathbf{c}}_{11} - \mathbf{c}_{11}\} + \varepsilon, \tag{19}$$

where ε is the evaluation error. This equation can be further expanded to give

$$\delta\lambda = \{\bar{U}_{12}^T - A_2^T Q'_{11}\} \bar{\mathbf{c}}_{11} + A_2^T\{Q'_{11} - Q_{11}\} \bar{\mathbf{c}}_{11}$$

$$+ A_2^T Q_{11}\{Q'^T_{11} - Q_{11}^T\} \mathbf{b} + A_2^T Q_{11}\{\bar{Q}_{11}^T - Q'^T_{11}\} \mathbf{b} + \varepsilon, \tag{20}$$

where the prime indicates the exact orthogonal factorization defined by the actual numeric data at each stage. The quantities on the right-hand side of (20) can now be estimated using known inequalities. The most important terms are those involving $Q'_{11} - Q_{11}$, and in [8] it is shown that this can be bounded by an expression of the form $k_1 \operatorname{eps} \chi(A_1)$ where k_1 is a constant, eps is the machine precision, and $\chi(A_1)$ is the spectral condition number of A_1. This is a result which is *more favourable* than the corresponding result for $\mathbf{x}_1$ which shows a dependence also on

$$\operatorname{eps} \chi(A_1)^2 \left\| \begin{matrix} \mathbf{c}_{12} \\ \mathbf{c}_2 \end{matrix} \right\| \quad [7].$$

However, the bounds quoted in the error estimate are for the usual form of orthogonal factorization which takes no account of the possibility of the back-tracking which can and does occur in Clark's algorithm. If we assume the analysis is valid also in the case of back-tracking, then presumably we have to use the largest condition number encountered to the present stage rather than the condition number of the current partition A_1.

One further point in favour of this form of the algorithm is that it appears rarely to be necessary to compute the complete factorization (9) in the determination of the optimum subset $\mathbf{x}_1$. It is conceivable that the full system could be badly conditioned while the subproblems leading to the optimal subset could be well conditioned.

3. Numerical results

A subset selection algorithm for (1) proceeds essentially in two stages : an initial search for a feasible $\mathbf{x}_1$ (or λ_2) using a heuristic to determine at each stage the component to be set to zero, and subsequent back-tracking to explore other branches of the search tree if the first feasible solution is not optimal. It is in this second phase of the computation that the major improvements reported in [3] are achieved. In the first phase the heuristic commonly used is to fix at zero level the most negative of the current solution components. This procedure suffers from the disadvantage that it is not scale independent. For example, if the data matrix A is multiplied by a diagonal matrix D to rescale the column norms so that

$$A \leftarrow AD^{-1}, \tag{21a}$$

then it follows from (2a) that the solution vectors are transformed by

$$\mathbf{x} \leftarrow D\mathbf{x} \quad \text{and} \quad \lambda \leftarrow D^{-1}\lambda. \tag{21b}$$

Our numerical experiments have shown that the choice of the first phase heuristic is important because it can affect the amount of work that has to be done in the second phase of the computation. It seems reasonable that a good heuristic should not be affected by changes in scale, and for this reason we compare the choice of most negative component with a choice which corresponds to the test used in stepwise regression to determine the variable to enter the regression at each step and which has the property of invariance with respect to column scaling. If we consider the data matrix factorized so that at the ith step of the first phase of the computation we have

$$A^{(i)} = \begin{bmatrix} U_1 & U_{12} \\ 0 & B \end{bmatrix} \quad \text{and} \quad \mathbf{b}^{(i)} = \begin{bmatrix} \mathbf{c}_{11} \\ \mathbf{d} \end{bmatrix}, \tag{22}$$

then the stepwise regression test selects the variable to be introduced as that which maximizes

$$|\mathbf{d}^{\mathrm{T}} \kappa_j(B)| / \| \kappa_j(B) \|,$$

as this leads to the biggest reduction in the sum of squares of the residuals [6]. Here $\|\kappa_j(B)\|$ is the euclidean length of the jth column of B. Now, from (18),

$$-\lambda_2^{(i)} = A_2^{\mathrm{T}} \mathbf{b} - U_{12} \mathbf{c}_{11} = B^{\mathrm{T}} \mathbf{d}, \tag{23}$$

so that we can use the stepwise test in the form

$$\mathbf{x}_1 \leftarrow \begin{bmatrix} \mathbf{x}_1^{(i)} \\ x_s \end{bmatrix} \quad s \text{ maximizes } \gamma(j) = -\frac{(\lambda_2^{(i)})_j}{\|\kappa_j(B)\|} \tag{24}$$

for all j such that $\gamma(j) > 0$.

However, our implementation actually considers $\gamma(j)^2$ as $\|\kappa_j(B)\|^2$ is readily updated from step to step.

We report numerical results for two sets each of ten problems with $m = 50$, $n = 40$. The data are obtained by sampling from a normal distribution for the first set and from a uniform distribution for the second set, except that in all cases $\kappa_1(A)_j = 1$, $j = 1, 2, \ldots, m$. For each set we give results for each of the selection strategies already discussed and for the case in which the most negative strategy is used after the columns of A are scaled initially to have unit length. Also, for the data drawn from the uniform distribution, we consider scaling the columns of A to have unit L_1 norm as the most negative strategy proved particularly favourable in this case, and definitely superior to the corresponding scaling using the euclidean norm. Also, for comparison, we give results obtained using a quadratic programming subroutine QUADPR, based on the Cottle–Danzig principal pivoting algorithm, which was supplied by the Madison Academic Computing Center at the University of Wisconsin.

The results for the two data sets are given in Tables 1.1 and 1.2, respectively. We report the average time per problem as (cumulative time)/10 (recorded most unreliably on the computer used, a Univac 1100/42)†, and the total number of nodes visited. It will be seen that the variants of Clark's algorithm are superior to the

† Timings in a multiprogramming environment tend to be unreliable because compromises are made between keeping exhaustive records and efficiency. Part of the explanation in this case would appear to stem from the system executive's practice of continuing the internal timing of an interrupted program unless it is actually swapped out of core.

TABLE 1.1

Results for data from normal distribution

Method	Average time (ms)	Number of nodes
Quadratic programming	1187	404
Most negative λ_i	322	210
Stepwise	249	204*
$\|\kappa_j(A)\|_2 = 1$	437	204*
Most negative x_i	547	202

* First feasible solution is optimal for each problem.

TABLE 1.2

Results for data from uniform distribution

Method	Average time (ms)	Number of nodes
Quadratic programming	1340	416
Most negative λ_i	434	196-
Stepwise	390	262
$\|\kappa_j(A)\|_2$	385	262
$\|\kappa_j(A)\|_1$	307	168
Most negative x_i	518	240*

* First feasible solution is optimal for each problem.

quadratic programming algorithm. Also, the most negative heuristic is never too bad, while the stepwise heuristic is favoured for the data drawn from the normal distribution. There is some evidence that the statistical origin of the data is not irrelevant to the choice of a good heuristic. Starting with λ rather than $\mathbf{x}$ is clearly the superior strategy in terms of elapsed time despite the unreliability of the timings (for example, the stepwise and column scaling strategies should have returned approximately the same times in Table 1.1).

References

[1] R. D. Armstrong and E. L. Frome, "A branch-and-bound solution of a restricted least squares problem", *Technometrics* 18 (1976), 447–450.

[2] R. H. Bartels, G. H. Golub and M. A. Saunders, *Nonlinear programming* (Academic Press, 1970), pp. 123–176.

[3] David Clark, "An algorithm for solving the restricted least squares problem", *J. Austral. Math. Soc. B* 21 (1980), 345–356.

[4] M. A. Effroymson, *Numerical methods for digital computers* (Wiley, 1960), pp. 191–203

[5] P E. Gill, G. H. Golub, W. Murray and M. A. Saunders, "Methods for modifying matrix factorizations", *Math. Comp.* 28 (1974), 505–535.

[6] G. H. Golub, "Numerical methods for solving linear least squares problems", *Num. Math.* 7 (1965), 206–216.

[7] G. H. Golub and J. H. Wilkinson, "Note on the iterative refinement of least squares solutions", *Num. Math.* 9 (1966), 139–148.

[8] L. S. Jennings and M. R. Osborne, "A direct error analysis for least squares", *Num. Math.* 22 (1974), 325–332.

[9] M. R. Osborne, "On the computation of stepwise regressions", *Austral. Computer J.* 8 (1976), 61–68.

Department of Statistics
Research School of Social Sciences
Australian National University
Canberra
A.C.T. 2600