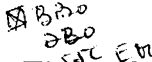


Use of sparse matrix absorption in animal breeding


 B. Tier and S.P. Smith

University of New England, Animal Genetics and Breeding Unit, Armidale, NSW 2351, Australia

(received 1 March 1988; accepted 1 June 1989)

Summary – Although the capacity of modern computers is increasing dramatically so too are the complexity of models that animal breeders employ, with the result that we still find computers limiting. This paper demonstrates the employment of linked lists for sparse matrix manipulations and their use in a number of relevant applications.

animal breeding – prediction of genetic merits – numerical methods – sparse matrix

Résumé – Utilisation en génétique animale de l'absorption dans des matrices creuses. Malgré l'accroissement de capacité des ordinateurs, la complexité également croissante des modèles employés en génétique animale nécessite le raffinement des méthodes numériques. Cet article explique l'utilisation de listes liées pour manipuler de très grandes matrices creuses, et illustre leur usage dans différents types d'application dans le cadre de l'évaluation des reproducteurs: absorption d'effets fixes, inversion de matrices, estimations de composantes de la variance par le maximum de vraisemblance.

génétique animale – évaluation des reproducteurs – analyse numérique – matrices creuses

INTRODUCTION

As the capacity of modern computers increases so does the quantity of data and the complexity of models that animal geneticists wish to use in their analyses. In the early years of computing when main memory was a major limitation, a variety of techniques were developed to utilise efficiently that memory which was available (Bunch and Rose, 1976). This paper illustrates the use of one of these techniques – linked lists – to store sparse matrices and eliminate (absorb) equations. Examples are given of how this technique can be useful.

TYPICAL MODELS

Linear models that are commonly used by animal geneticists have qualities that lend themselves to efficient methods of storage. Consider the model:

$$y = Xb + Zu + e$$

where y is a vector of observations; X and Z are incidence matrices; b is a vector of fixed effects; u is a vector of random animal (or sire) effects; and e is a vector of random residuals. Animals (sires) are related.

The mixed model equations (Henderson, 1974) are

$$\begin{bmatrix} \mathbf{X}'\mathbf{R}^{-1}\mathbf{X} & \mathbf{X}'\mathbf{R}^{-1}\mathbf{Z} \\ \mathbf{Z}'\mathbf{R}^{-1}\mathbf{X} & \mathbf{Z}'\mathbf{R}^{-1}\mathbf{Z} + \mathbf{G}^{-1} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{b}} \\ \hat{\mathbf{u}} \end{bmatrix} = \begin{bmatrix} \mathbf{X}'\mathbf{R}^{-1}\mathbf{y} \\ \mathbf{Z}'\mathbf{R}^{-1}\mathbf{y} \end{bmatrix}$$

where \mathbf{G} is the covariance matrix of \mathbf{u} , and \mathbf{R} is a covariance matrix of residuals. For a univariate analysis $\mathbf{G} = \gamma\mathbf{A}$ for some γ where \mathbf{A} is the numerator relationship matrix.

Let $\mathbf{Q} = \begin{bmatrix} \mathbf{X}'\mathbf{R}^{-1}\mathbf{X} & \mathbf{X}'\mathbf{R}^{-1}\mathbf{Z} & \mathbf{X}'\mathbf{R}^{-1}\mathbf{y} \\ \mathbf{Z}'\mathbf{R}^{-1}\mathbf{X} & \mathbf{Z}'\mathbf{R}^{-1}\mathbf{Z} + \mathbf{G}^{-1} & \mathbf{Z}'\mathbf{R}^{-1}\mathbf{y} \\ \mathbf{y}'\mathbf{R}^{-1}\mathbf{X} & \mathbf{y}'\mathbf{R}^{-1}\mathbf{Z} & \mathbf{y}'\mathbf{R}^{-1}\mathbf{y} \end{bmatrix}$

be the mixed model array. Because \mathbf{Q} is symmetric it is only necessary to store the upper (or lower) triangle. This means that more equations can be stored in the memory. When an animal model (or reduced animal model) is employed, then \mathbf{Q} is very sparse.

LINKED LISTS

A linked list consists of a list of elements linked together by pointers to their physical locations. The physical location of the first element in the row is stored and every element has associated with it a pointer to the location in the memory of the next element in the sequence. The pointer associated with the last element in the list is zero. Knuth (1968) provided a detailed explanation of linked lists.

When using FORTRAN 3 vectors are required to store a matrix in this way – one for the element (a_{IJ}), one for the column (J) and one for the pointer to the next element. A scalar (NUSED) is used to point to the last occupied location in these vectors. As the list is being built, new elements are stored in the next available location in these vectors but the order of the row is maintained by adjusting the pointers (illustrated in Table I).

Table I. The state of the pointers in a linked list before and after the inclusion of a new element ($J=6$).

Memory Location	Before		After	
	Pointer P	Column J	Pointer P	Column J
1	3	1	3	1
2	0	7	0	7
3	2	3	4	3
4	0	0	2	6
	NUSED = 3		NUSED = 4	

Because matrices such as \mathbf{Q} and \mathbf{G}^{-1} are sparse, they lend themselves to this form of storage. To store a matrix of order N the first N elements in the storage

Table III. A linked list representation of the matrix Q derived from the three records in Table I. Coefficient (C), Column (J), and Pointer (P) to the next element are stored in Location (L).

<i>L</i>	<i>Record 1</i>			<i>Records 1 and 2</i>			<i>Records 1, 2 and 3</i>		
	<i>C</i>	<i>J</i>	<i>P</i>	<i>C</i>	<i>J</i>	<i>P</i>	<i>C</i>	<i>J</i>	<i>P</i>
1	1.	1	6	2.	1	6	2.	1	6
2	0.	0	0	0.	0	0	1.	2	11
3	1.	3	8	1.	3	8	1.	3	8
4	0.	0	0	1.	4	10	2.	4	10
5	9.	5	0	25.	5	0	50.	5	0
6	1.	3	7	1.	3	9	1.	3	9
7	3.	5	0	7.	5	0	7.	5	0
8	3.	5	0	3.	5	0	3.	5	0
9	0.	0	0	1.	4	7	1.	4	7
10	0.	0	0	4.	5	0	9.	5	0
11	0.	0	0	0.	0	0	1.	4	12
12	0.	0	0	0.	0	0	5.	5	0
13	0.	0	0	0.	0	0	0.	0	0
14	0.	0	0	0.	0	0	0.	0	0
15	0.	0	0	0.	0	0	0.	0	0
	NUSED = 8			NUSED = 10			NUSED = 12		

ABSORPTION OF EQUATIONS

Absorption or gaussian elimination is described in Smith and Graser (1986). If the sparsity of the matrix is to be preserved, as is desirable for a linked list to be useful, then it is important to choose pivots so that new elements do not proliferate. Gill and Murray (1974) suggest choosing rows with the least number of off-diagonal elements first.

As each row is absorbed, the space it occupied is released and can be made available to new elements that are created in other rows. Before absorbing any equations, it is useful to link the unoccupied space in the vectors into a separate linked list. As space is released, it can be added to the list of free space for reuse. Because the elements in the row are already connected by pointers, the complete row can be placed at the start of the list of free space by modifying the pointers at the end of the row and at the start of the free space. If backward substitution is to be implemented then the row should be written as an exterior file.

After absorbing the first row of Q

$$Q_1 = \begin{bmatrix} 0 & 0 & 0 & 0.0 \\ & 1 & 0 & 5.0 \\ & & 0.5 & -0.5 \\ & & & 1.5 \\ & & & & 5.5 \\ & & & & & 25.5 \end{bmatrix}$$

Table IV. A linked list representation of Q_1 , derived from eliminating the first row of Q (Table III).

<i>Location</i>	<i>Coefficient</i>	<i>Column</i>	<i>Pointer</i>
1	-0.5	4	8
2	1.0	2	11
3	0.5	3	1
4	1.5	4	10
5	25.5	5	0
6	0.0	0	9
7	0.0	0	13
8	-0.5	5	0
9	0.0	0	7
10	5.5	5	0
11	1.0	4	12
12	5.0	5	0
13	0.0	0	14
14	0.0	0	15
15	0.0	0	0

IHEAP = 6

and its linked list representation is shown in Table IV. There is no need to zero the column and coefficient vectors from the row being absorbed; when this space is reused they will be assigned new values.

During elimination of each row of Q , it is possible to design the algorithm so that subsequent rows and elements within rows are modified sequentially; redundant searching through Q can and should be avoided. If the selected pivot is zero, then the row can be regarded as having been preabsorbed. An algorithm to absorb equations in this manner is shown in the Appendix.

For large problems, it is possible and desirable to divide Q into 2 parts: an exterior file and a linked list. Absorption of a row entails: reading the row from the exterior file; merging the input with the linked list; and absorbing the row in the linked list. When the linked list is full, it can be merged with the exterior file so as to create a new exterior file. This clears the vectors for a new iteration.

APPLICATIONS

All the following examples have the form of manipulating a matrix

$$\begin{bmatrix} \tilde{U} & T \\ T' & W \end{bmatrix}$$

by absorbing \tilde{U} row by row to give

$$W^* = W - T' \tilde{U}^{-1} T$$

Row by row absorption is equivalent to repeated application of the formula for \mathbf{W}^* , where $\tilde{\mathbf{U}}$ is a scalar (the pivot) and \mathbf{T} is a column vector.

1) Sparse matrix inversion

For example, find \mathbf{E}^{-1} given the positive definite and symmetric matrix of \mathbf{E} .

$$\begin{aligned} \text{Set } \tilde{\mathbf{U}} &= \mathbf{E}_{n \times n} \\ \mathbf{T} &= \mathbf{I}_{n \times n} \\ \mathbf{W} &= \mathbf{O}_{n \times n} \\ \text{then } \mathbf{W}^* &= -\mathbf{E}^{-1} \end{aligned}$$

Sometimes only the diagonal of \mathbf{E}^{-1} may be required, in which case the calculation and storage of off-diagonal elements of \mathbf{E}^{-1} can be neglected.

2) Estimation of (co)variance components by maximum likelihood (ML) or restricted maximum likelihood (REML)

Many of the arrays in this section can be found in Searle (1979).

a) Evaluate $\mathbf{V}^{-1} = \mathbf{R}^{-1} - \mathbf{R}^{-1}\mathbf{Z}(\mathbf{Z}'\mathbf{R}^{-1}\mathbf{Z} + \mathbf{G}^{-1})^{-1}\mathbf{Z}'\mathbf{R}^{-1}$ in ML

$$\begin{aligned} \text{Set } \tilde{\mathbf{U}} &= \mathbf{Z}'\mathbf{R}^{-1}\mathbf{Z} + \mathbf{G}^{-1} \\ \mathbf{T} &= \mathbf{Z}'\mathbf{R}^{-1} \\ \mathbf{W} &= \mathbf{R}^{-1} \\ \text{then } \mathbf{W}^* &= \mathbf{V}^{-1} \end{aligned}$$

b) Evaluate $\mathbf{P} = \mathbf{V}^{-1} - \mathbf{V}^{-1}\mathbf{X}(\mathbf{X}'\mathbf{V}^{-1}\mathbf{X})^{-1}\mathbf{X}'\mathbf{V}^{-1}$ in REML

$$\begin{aligned} \text{Set } \tilde{\mathbf{U}} &= \begin{bmatrix} \mathbf{X}'\mathbf{R}^{-1}\mathbf{X} & \mathbf{X}'\mathbf{R}^{-1}\mathbf{Z} \\ \mathbf{Z}'\mathbf{R}^{-1}\mathbf{X} & \mathbf{Z}'\mathbf{R}^{-1}\mathbf{Z} + \mathbf{G}^{-1} \end{bmatrix} \\ \mathbf{T} &= \begin{bmatrix} \mathbf{X}'\mathbf{R}^{-1} \\ \mathbf{Z}'\mathbf{R}^{-1} \end{bmatrix} \\ \mathbf{W} &= \mathbf{R}^{-1} \end{aligned}$$

then $\mathbf{W}^* = \mathbf{P}$

c) Evaluate $\mathbf{Z}'\mathbf{P}\mathbf{Z}$ in REML : method 1

$$\begin{aligned} \tilde{\mathbf{U}} &= \begin{bmatrix} \mathbf{X}'\mathbf{R}^{-1}\mathbf{X} & \mathbf{X}'\mathbf{R}^{-1}\mathbf{Z} \\ \mathbf{Z}'\mathbf{R}^{-1}\mathbf{X} & \mathbf{Z}'\mathbf{R}^{-1}\mathbf{Z} + \mathbf{G}^{-1} \end{bmatrix} \\ \mathbf{T} &= \begin{bmatrix} \mathbf{X}'\mathbf{R}^{-1}\mathbf{Z} \\ \mathbf{Z}'\mathbf{R}^{-1}\mathbf{Z} \end{bmatrix} \\ \mathbf{W} &= \mathbf{Z}'\mathbf{R}^{-1}\mathbf{Z} \end{aligned}$$

then $\mathbf{W}^* = \mathbf{Z}'\mathbf{P}\mathbf{Z}$

d) Evaluate $Z'PZ$ in REML : method 2

$$\tilde{U} = \begin{bmatrix} X'R^{-1}X & Z'R^{-1}Z \\ Z'R^{-1}X & Z'R^{-1}Z + G^{-1} \end{bmatrix}$$

$$T = \begin{bmatrix} 0 \\ G^{-1} \end{bmatrix}$$

$$W = G^{-1}$$

then $W^* = Z'PZ$

e) Evaluate the log-likelihood (L) for REML using the derivative free search of Graser *et al.* (1987).

$$\text{Set } \tilde{U} = \begin{bmatrix} X'R^{-1}X & X'R^{-1}Z \\ Z'R^{-1}X & Z'R^{-1}Z + G^{-1} \end{bmatrix}$$

$$T = \begin{bmatrix} X'R^{-1}y \\ Z'R^{-1}y \end{bmatrix}$$

$$W = y'R^{-1}y$$

then $W^* = y'Py$

To evaluate L we note that

$$L = -\frac{1}{2} \{ \log |R| + \log |G| + \log |\tilde{U}| + y'Py \}$$

where $|\tilde{U}|$ is the determinant of one of the largest non-singular submatrices of \tilde{U} ; and $|\tilde{U}|$ is evaluated by the product of the non-zero pivots.

To implement a derivative-free search sometimes we need $\text{rank}(X)$ which is the number of non-zero pivots minus the order of G^{-1} .

3) Calculating the exact A^{-1} for sub-populations

When a sire model is used it is possible to build A^{-1} for the full pedigree of the sires and then absorb all female relatives. Some sires may be absorbed as well if they are not part of the subpopulation. Partition A^{-1} into 2 parts: animals to be absorbed in \tilde{U} , and animals that are to remain in W . The W^* is the exact inverse relationship matrix for the remaining selected animals. Experience has shown that absorption seems to create many elements that are essentially zero. Linked-list absorption works well when these zero elements are released from storage, particularly from a row before it is absorbed.

4) Conducting secondary absorptions

Sometimes it is necessary to absorb 2 groups of factors out of the model. The model used by Smith (1987) included 2765 effects representing contemporary groups, 2611 fixed sire effects and 539 random sires. Rows representing contemporary groups were absorbed as the data were read. Then rows representing fixed sires were absorbed in a reasonable time using sparse matrix techniques. The absorption of the fixed-sire effects would have been impossible using matrix inversion.

The order used for the secondary absorption was determined by the size of the diagonals after the primary absorption. Rows with smaller diagonals were absorbed first. This order is opposite to the usual practice, however, it minimizes the creation of non-zero elements and hence preserves the efficient use of memory. Use of the traditional approach would have been as impossible as matrix inversion.

5) *Partial absorption prior to iteration*

Sometimes it may be advisable to absorb some equations prior to iteration, such as is implicitly done using the reduced animal model (Quaas and Pollak, 1980).

CONCLUSION

Some of the applications we have described may not be practical. For example, evaluating V^{-1} , P and $Z'PZ$ may be beyond current computing capabilities even with a linked list. However, some of the applications (e.g., evaluating L , constructing A^{-1} for sub-populations, and secondary absorptions) are realistic and have been tested on real data structures. Without linked lists these applications may not be feasible.

A common misconception is that evaluating Q^{-1} is about as difficult as absorbing all rows of Q . For non-sparse matrices, inversion requires 3 times the work of absorption. For sparse matrices the comparison is typically much more extreme. Inversion can be prohibitive even with a linked list, while absorption of the same matrix may be a relatively simple operation.

REFERENCES

- Bunch J.R. & Rose D.J. (ed.) (1976) *Sparse Matrix Computations*. Academic Press, New York
- Gill P.E. & Murray W. (1974) Methods for larger scale linearly constrained problems. In: *Numerical Methods for Constrained Optimisation*. (Gill P.E. & Murray W. (ed.), Academic Press, London, pp. 93-147
- Graser H.-U., Smith S.P. & Tier B. (1987) A derivative free approach for estimating variance components in animal models by REML. *J. Anim. Sci.* 64, 1362-1370
- Henderson C.R. (1974) Sire evaluation and genetic trends. In: *Proceedings of the Animal Breeding and Genetics Symposium, Blacksburg, July 29, 1972*, Am. Soc. Anim. Sci. and Am. Dairy Sci. Assoc., Champaign, IL., p. 10
- Knuth D.E. (1968) *The Art of Computer Programming*, vol. 1. *Fundamental Algorithms*, Addison Wesley, Reading, MA
- Quaas R.L. & Pollak E.J. (1980) Mixed model methodology for farm and ranch beef cattle testing programs. *J. Anim. Sci.* 51, 1277-1287
- Searle S.R. (1979) Notes on variance components estimation: a detailed account of maximum likelihood and kindred methodology. *Paper BU-673M*. Biometrics Unit, Cornell University, Ithaca, NY

Smith S.P. (1987) Genetic parameters for type in Australian *Holstein-Friesian* dairy cattle. In: *Proceedings of the Sixth Conference, Australian Assoc. of Anim. Breeding Genet., Perth, Australia, February 9-11, 1987*, Anim. Genet. Breeding Unit, University New England, Armidale, pp; 55-58

Smith S.P. & Graser H.-U. (1986) Estimating variance components in a class of mixed models by restricted maximum likelihood. *J. Dairy Sci.* 69, 1156-1165

APPENDIX

Subroutines *LINKAIJ*, *LINKFREE* and *ABSROW*

The following storage is required for these subroutines: *ELEMENT*(*LENGTH*) is a vector of elements. *POINTER*(*LENGTH*) is a vector holding pointers to the next element in the row. *JAY*(*LENGTH*) is a vector holding the column (*J*) of the element. *NUSED* is a pointer to the last occupied location in these vectors. *IThis* and *NEXT* are pointed to this and the next element respectively.

Subroutine *LINKAIJ* stores the contribution to the element a_{IJ} which are passed as parameters.

```

SUBROUTINE LINKAIJ(INI,INJ,AIJ)
If (INI=0) or (INJ=0) stop ' error message '
If (INI=INJ) then
  ELEMENT(INI)=ELEMENT(INI)+AIJ
else
  I=MIN(INI,INJ);J=MAX(INI,INJ);IThis=I;NEXT=POINTER(I)
  While (NEXT>0) and (J>JAY(NEXT)) do
    ITHIS=NEXT;NEXT=POINTER(NEXT);Endwhile
  If (J=JAY(NEXT)) then
    ELEMENT(NEXT)=ELEMENT(NEXT)+AIJ
  else
    NUSED=NUSED+1
    JAY(NUSED)=J
    ELEMENT(NUSED)=AIJ
    POINTER(NUSED)=POINTER(IThis)
    POINTER(IThis)=NUSED
  endif
endif
end

```

Subroutine *LINKFREE* links up the free space in *POINTER* and initialises *IHEAP* before *ABSROW* is called. *IHEAP* points to the next available location in the vectors.

```

SUBROUTINE LINKFREE
IHEAP=NUSED+1
LAST=LENGTH-1
DO I=IHEAP, LAST
  POINTER(I)=I+1
ENDDO

```

Subroutine ABSROW absorbs the i^{th} row. The i^{th} row is transferred to two work vectors (WORK which contains the elements and JWORK which contains the columns). Space released by this transfer is placed in the available heap and then the row is absorbed. OPZERO is the operational zero (if the absolute value of the element is less than OPZERO it is treated as being zero).

```

SUBROUTINE ABSROW(I)
K=0;NEXT=POINTER(I); ITHIS=I;DIAG=-1.0/ELEMENT(I)
While (NEXT>0) do
  If (ABS(ELEMENT(NEXT))>OPZERO) then
    K=K+1;WORK(K)=ELEMENT(NEXT);JWORK(K)=JAY(NEXT)
  Endif
  ITHIS=NEXT;NEXT=POINTER(NEXT);Endwhile
POINTER(ITHIS)=IHEAP;IHEAP=I

If (ABS(WORK(1))>OPZERO) then
  Do IK=1,K
    I=JWORK(IK);FACTOR=WORK(IK)*DIAG;IThis=I;NEXT=POINTER(
    ELEMENT(I)=ELEMENT(I)+WORK(IK)*FACTOR;IK1=IK+1
  Do JK=IK1,K
    J=JWORK(JK)
    While (J>JAY(NEXT)) and (POINTER(NEXT)>0) do
      ITHIS=NEXT;NEXT=POINTER(NEXT);Endwhile
    If (J=JAY(NEXT)) then
      ELEMENT(NEXT)=ELEMENT(NEXT)+WORK(JK)*FACTOR
    Else
      NEW=IHEAP
      IHEAP=POINTER(IHEAP)
      IF (IHEAP=0) stop 'list full'
      JAY(NEW)=J
      ELEMENT(NEW)=WORK(JK)*FACTOR
      POINTER(NEW)=POINTER(ITHIS)
      POINTER(ITHIS)=NEW
      ITHIS=NEW
    Endif
  Enddo
Enddo
Endif
End

```