Genet Sel Evol (1990) 22, 419-430

Original article

Computing inbreeding coefficients quickly

B Tier

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

(Received 13 June 1988; accepted 12 September 1990)

Summary – An algorithm for computing inbreeding coefficients (F) for all animals of large populations is described. The technique relies on the subset of the numerator relationship matrix (A) required to compute its diagonal, which contains F. A simple example illustrates that this subset is a very small part of A.

inbreeding coefficients / fast algorithm

Résumé – Calcul rapide des coefficients de consanguinité. *On présente un algorithme pour calculer rapidement les coefficients de consanguinité de tous les animaux, dans une grande population. La méthode fait seulement intervenir le sous-ensemble de la matrice des coefficients de parenté qui est nécessaire au calcul de sa diagonale, qui contient les coefficients de consanguinité. Un exemple simple illustre que ce sous-ensemble n'est qu'une petite partie de la matrice des coefficients de parenté. L'algorithme est précisé sous la forme d'un codage symbolique.*

coefficients de consanguinité / calcul rapide

INTRODUCTION

Wright's (1922) coefficient of inbreeding (F) describes the probability that 2 alleles at any locus are identical by descent. Inbred offspring result from mating two animals which have one (or more) common ancestors. Breeders of livestock perceive inbreeding as being deleterious and consequently try to avoid mating close relatives. The relationship between prospective mates can be computed to determine the degree of inbreeding of the resulting offspring from such a mating.

For inbred populations, inbreeding coefficients are required to compute A^{-1} directly using Henderson's (1975) rules. Henderson (1976) showed the relationship between the inbreeding coefficients of an animal's parents and the "contributions"

* AGBU is a joint venture between the University of New England and NSW Department of Agriculture and Fisheries.

that each animal's pedigree makes to A^{-1}. For the ith animal with parents j and k, the following contributions are added to A^{-1}:

$$\begin{aligned} &d_{ii}^{-1} \text{ to } a^{ii} \\ &-0.5\ d_{ii}^{-1} \text{ to } a^{ij}, a^{ik}, a^{ji}, a^{ki}, \text{ and} \\ &0.25\ d_{ii}^{-1} \text{ to } a^{jj}, a^{kk}, a^{jk} \text{ and } a^{kj}, \end{aligned}$$

where $d_{ii} = 1.0 - 0.25(a_{jj} + a_{kk})$.

Two algorithms are commonly used to compute inbreeding coefficients. Originally they were calculated using the path coefficient method (Wright, 1922). While this method is easy to use for computing F for animals which have few ancestors and are only slightly inbred, it is a very complex method for animals with many common ancestors. A simpler approach is to generate the numerator relationship matrix (A) using the tabular method as attributed to Lush by Emik and Terrill (1949, cited by Hudson *et al*, 1982). Inbreeding coefficients can be computed from the diagonal elements of A : $F_i = a_{ii} - 1$. This paper describes an efficient implementation of the tabular method.

COMPUTING INBREEDING COEFFICIENTS USING THE TABULAR METHOD

To compute A by the tabular method animals in the population must be numbered from 1 to N so that parents precede their progeny. The pedigrees of all animals must be stored in memory. A zero in the pedigree denotes an unknown parent. The upper triangle of A can be computed on a row by row basis in consecutive order working from first to last. Diagonal elements are computed using the formula:

$$a_{jj} = 1 + 0.5a_{pq}. \tag{1}$$

The remainder of the row (to the right of the diagonal) can be computed by applying the formula:

$$a_{ij} = 0.5(a_{ip} + a_{iq}) \tag{2}$$

where p and q are the parents of the jth animal and $i < j$. If either parent is unknown (p or $q = 0$) then $a_{0i} = 0$. When a row is complete the corresponding column in the lower triangle can be completed by symmetry. Because A is symmetric it is only necessary to store the upper (or lower) triangle. Table II illustrates A for the sample population shown in table I.

The storage required to compute A is proportional to the square of the numbers of animals and so limits the size of the population for which A can be computed. Using the technique of Hudson *et al* (1982) memory is only required to store the non-zero elements of A and it can be computed for larger populations. When inbreeding coefficients only are required then it is not necessary to compute A completely, nor all its non-zero elements.

Table I. Pedigree of sample population.

Animal	*Parents*
1	unknown
2	unknown
3	unknown
4	1 and 2
5	1 and 3
6	4 and 3
7	5 and 6

Table II. Numerator relationship matrix (A) for the sample population shown in Table I.

Animal	*1*	*2*	*3*	*4*	*5*	*6*	*7*
1	1	0	0	1/2	1/2	1/4	3/8
2	0	1	0	1/2	0	1/4	1/8
3	0	0	1	0	1/2	1/2	1/2
4	1/2	1/2	0	1	1/4	1/2	3/8
5	1/2	0	1/2	1/4	1	3/8	11/16
6	1/4	1/4	1/2	1/2	3/8	1	11/16
7	3/8	1/8	1/2	3/8	11/16	11/16	19/16

THE RECURSIVE PEDIGREE METHOD

This is an adaptation of the tabular method and depends upon computing the subset of A required to compute its diagonal.

As each animal's pedigree is read, Equation 1 is used to identify the element which describes the relationship between its parents. If the animal's inbreeding coefficient is already known and no new ancestral information is available it is stored. If not, then the element (a_{pq}) is placed in the subset (flagged) to be computed.

Equation 2 can now be applied to identify those elements required to compute elements already flagged. This requires searching the matrix (upper triangle) from bottom to top and from right to left. As each identified element is found the two elements required to apply equation 2 can also be flagged. Because these two elements lie to the left of the flagged element – animals are numbered so that parents precede their progeny – searching A in this manner results in all the elements required to compute the diagonal of A being identified.

The flagged subset can now be computed by applying equations 1 and 2 starting with the first row, and proceeding sequentially to the last. Computation of each row of A can be considered in 3 parts. Firstly, elements on the left of the diagonal have already been computed – they were on the right of the diagonal in earlier rows. Secondly, the off-diagonal element (a_{jk}) required to compute the diagonal element (a_{ii}) has already been computed ($j, k < i$) and a_{ii} can be computed by equation

1. Lastly, the flagged elements on the right of the diagonal can be computed using equation 2.

Table III illustrates the recursive pedigree method for the sample population. Firstly, off diagonal elements from (1) are identified and flagged. Because animals 1, 2 and 3 have at least 1 unknown parent they are not inbred and 1 is stored in the corresponding diagonal element. The diagonal elements of animals 4, 5, 6 and 7 are represented by the letters P, Q, R and S respectively. The offdiagonal elements required to compute them are represented by the same letter in lowercase. Other elements identified by t, u, v and w are required to compute p, q, r and s.

Table III. The subset of A required to compute F.

Animal	*1*	*2*	*3*	*4*	*5*	*6*	*7*
1	1	p	q	v	–	–	–
2	–	1	w	–	–	–	–
3	–	–	1	r	t	–	–
4	–	–	–	P	u	–	–
5	–	–	–	–	Q	s	–
6	–	–	–	–	–	R	–
7	–	–	–	–	–	–	S

As the rows are processed, elements that must be computed before the current element can be computed are identified and flagged. $a_{56}(s)$ is the first flagged element to be found. It requires that $a_{35}(t)$ and $a_{45}(u)$ are known. Table IV illustrates the identification and flagging of new elements as they were found using the search procedure described above.

Table IV. Elements of A required for the flagged element to be computed.

Flagged element	*Elements Required*	
$a_{56}(s)$	$a_{35}(t)$	$a_{45}(u)$
$a_{45}(u)$	$a_{14}(v)$	$a_{34}(r)^*$
$a_{35}(t)$	$a_{13}(q)^*$	$a_{33}+$
$a_{34}(r)$	$a_{13}(q)^*$	$a_{23}(w)$
$a_{23}(w)$	–	–
$a_{14}(v)$	$a_{11}+$	$a_{12}(p)^*$
$a_{13}(q)$	–	–
$a_{12}(p)$	–	–

* Previously flagged + Diagonal element

COMPUTATIONAL CONSIDERATIONS

To take advantage of the saving in space offered by this method it is necessary to use a sparse storage technique to store the subset of A. In this case, elements in the subset to the right of the diagonal in each row of A can be stored as a linked list.

Elements in a linked list are not stored in memory in any particular order but are linked together in some sequence by pointers. There is a pointer to the location in memory of the first element in each row. Each element in the list has an associated pointer to the location in memory of the next element in the sequence. The pointer associated with the last element in each row (list) is 0. To find any element in such a list it is necessary to "follow" the pointers until the required element is found. As new elements are stored in a list the pointers are adjusted to maintain the integrity of the sequence (for a detailed explanation of linked lists see Knuth, 1968).

To expedite the searching phase, the elements in each row should be linked in reverse column order (from highest to lowest, fig 1a). After the subset has been identified the pointers can be adjusted so that the rows are linked in column order (fig 1b). For this application it is desirable to have a pointer (called RECENT in the Appendix) to the most recently added element as well as a pointer (ROW) to the first element in each list. As only the upper triangle is stored, elements on the left of the diagonal appear in the column above the diagonal. These elements must be added to the lists holding the rows above the diagonal. Because these new elements will generally be adjacent to the previous addition to that row using this pointer (RECENT) can avoid repeated searching through the lists. Similarly, repeated searching for the same new elements (arising from animals having the same parent) within a row and searching the row to the right of the current element should also be avoided.

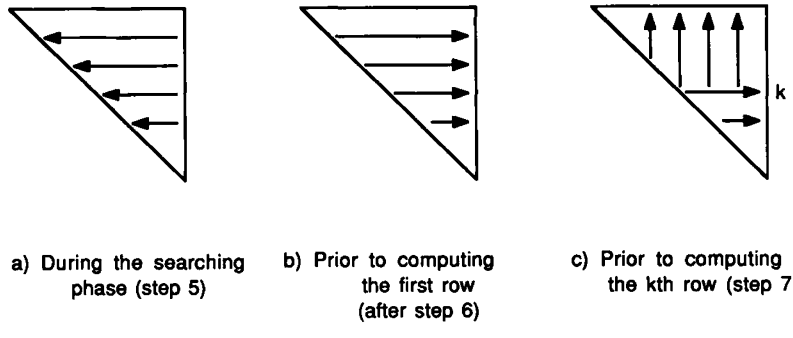

Fig 1. Arrangement of pointers in the linked list at various stages in the algorithm.

Diagonal elements can be stored in a separate vector (DIAG). This can be used to point to the physical location in the linked list of the element required to compute equation 1 until the diagonal element is determined. Because the theoretical maximum for a diagonal element is 2.0, the first 2 elements in the linked list must be reserved. Thus a value greater than 2 in the diagonal vector is a pointer, one less than 3 is a diagonal element. As each diagonal element is determined it can be stored in this vector. Subsequently, inbreeding coefficients can be derived from this vector.

An efficient method for finding elements on the left of the diagonal for any row is required. One way of doing this is to adjust the pointers so that after the elements in each row have been computed the elements on the right of the diagonal are relinked

on a column basis in reverse row order (fig 1c). This requires a vector (COLUMN) which points to the most recently added element of each column.

SIMULATION STUDY

For this example a population with the following characteristics was simulated. Starting with a base population of 20 sires and 500 dams, 40 years of progeny were generated. The adult population was held constant at 20 sires and 500 dams. Sires were mated randomly to the dams, all of which had 1 calf. After each year the oldest half of the sires were replaced and the oldest quarter of the dams were culled. Yearlings (offspring from the previous "year") were randomly selected to replace the culled animals. As a result, some animals had up to 20 generations of ancestors.

Three sets of inbreeding coefficients were computed for the animals, *viz*

– after each group of 10 years assuming that no inbreeding coefficients had been calculated on this population before;

– for each decade assuming that inbreeding coefficients had been calculated in the previous year *ie* only inbreeding coefficients for the latest group of calves were unknown;

Table V. Space and time (CPU seconds on a Gould NP1) required to compute inbreeding coefficients by the recursive pedigree method.

Years	*Animals*	*Elements required*	*% of A*	*Time (CPU seconds)*
a) When no inbreeding coefficients were known				
10	5 520	30 858	0.101	3
20	10 520	161 512	0.146	15
30	15 520	331 067	0.137	32
40	20 520	516 435	0.123	53
b) When inbreeding coefficients were known for all but the most recent calves				
10	5 520	16 488	0.059	3
20	10 520	93 609	0.085	10
30	15 520	209 767	0.085	24
40	20 520	347 506	0.080	34
c) When no inbreeding coefficients were known and they were only required for parents (when computing A^{-1})				
10	5 520	5 716	0.019	3
20	10 520	71 839	0.065	8
30	15 520	196 403	0.082	20
40	20 520	342 627	0.081	35

Table VI. State of the vectors at various stages of the algorithm when determining the diagonal of A for the sample population in table 1.

Location	*DIAG*	*ROW*	*RECENT*	*JAY*	*NEXT*	*AIJ*
a) After step 4						
1	1.000	4	0		reserved	
2	1.000	0	0		reserved	
3	1.000	5	0	2	0	
4	3.000	0	0	3	3	
5	4.000	6	0	4	0	
6	5.000	0	0	6	0	
7	6.000	0	0	0	0	
b) After step 5						
1	1.000	9	9		reserved	
2	1.000	10	0		reserved	
3	1.000	8	8	2	0	
4	3.000	7	7	3	3	
5	4.000	6	6	4	0	
6	5.000	0	0	6	0	
7	6.000	0	0	5	0	
8		unused		5	5	
9		unused		4	4	
10		unused		3	0	
c) After computing the 3rd Row (I=3 in step 7)						
1	1.000	3	0		reserved	
2	1.000	10	3		reserved	
3	1.000	5	10	1	0	0.000
4	3.000	7	5	1	0	0.000
5	4.000	6	8	3	9	0.000
6	5.000	0	0	6	0	0.000
7	6.000	0	0	5	0	0.000
8		unused		3	0	0.500
9		unused		1	0	0.500
10		unused		2	4	0.000
d) On completion						
1	1.000	3	0		reserved	
2	1.000	10	3		reserved	
3	1.000	5	10	1	0	0.000
4	1.000	7	5	1	0	0.000
5	1.000	6	7	3	9	0.000
6	1.000	0	6	5	0	0.375
7	1.187	0	0	4	8	0.250
8		unused		3	0	0.500
9		unused		1	0	0.500
10		unused		2	4	0.000

– for parents only when no inbreeding coefficients were known. A detailed algorithm used to compute inbreeding is shown in the Appendix. These computations were carried out on a GOULD NP1 computer and the results from this are shown in table V.

For the population of 20 520 animals, the subset of A included 516 435 elements out of a total of 421 070 400 (0.123%). The population size which required computation of the largest proportion of A (0.146%) was 10 520. When inbreeding coefficients were known for all but the most recent group of calves, or were only required for parents, a significantly smaller proportion of A was required.

DISCUSSION

The results in table V illustrate how very small a subset of A is required to compute its diagonal for the simulated population. Although the subset is small as a proportion of A, more than 6 Mbytes of memory were required to store the largest subset (516 435 elements). Table Vb illustrates the computing resources required when inbreeding coefficients are known for all but the most recent group of calves. As this technique can make use of previously computed inbreeding coefficients, problems that are too large for a computer can be divided into a series of smaller problems. To avoid repeated computation of the same elements of A, subsets should not be chosen on a chronological basis but rather on a related group, herd or family basis. The technique could be readily adapted to compute any subset of A that is of interest.

ACKNOWLEDGMENTS

The financial support of the Australian Meat and Livestock Corporation is gratefully acknowledged, as are helpful comments and encouragement from colleagues and referees.

REFERENCES

Henderson CR (1975) Rapid method for computing the inverse of a relationship matrix. *J Dairy Sci* 58, 1727-1730

Henderson CR (1976) A simple method for computing the inverse of a numerator relationship matrix used in prediction of breeding values. *Biometrics* 32, 69-83

Hudson GFS, Quaas RL, Van Vleck LD (1982) Computer algorithm for the recursive method of calculating large numerator relationship matrices. *J Dairy Sci* 65, 2018-2022

Knuth DE (1968) *The art of Computer Programming. Vol 1. Fundamental Algorithms.* Addison Wesley, Reading, Massachussets. 634 p

Wright S (1922) Coefficients of inbreeding and relationships. *Am Nat* 56, 330-338

APPENDIX

Algorithm for the recursive pedigree method. Table VI illustrates the state of the storage at various stages of the algorithm.

The following variables and subroutine are required for this algorithm:

Integer scalars

N is the number of animals. LLSIZE is a pointer to the last element in the linked list at any time ((LLSIZE+1) is empty). LARGE is the size of the vectors used to store the lists. LASTEL holds the address of the last element passed to the subroutine.

Integer vectors

SIRE(0:N) and DAM(0:N) store parents (SIRE(0)=DAM(0)=0). During the searching phase, COLUMN(0:N) is used to keep a record of the flagged elements in a row; during the computation phase it stores pointers to the first element in the columns of elements above the diagonal. ROW(1:N) stores pointers to the first element in each row on the right of the diagonal. RECENT(1:N) stores pointers to the location of the elements preceding the most recently added element in each row. NEXT(1:LARGE) stores pointers to the next element in each row. JAY(1:LARGE) stores the column subscripts of the elements.

Real vectors

WORK(1:N) is used as workspace for computing a row. DIAG(1:N) stores pointers to the a_{pq}'s of equation [1] initially, and subsequently the diagonal elements when computed. AIJ(1:LARGE) stores the required elements of A.

Subroutine

RESERVE reserves space in the linked lists for the elements in the subset while maintaining the integrity of the lists. A check to ensure that there is sufficient space to complete the analysis could be included. RESERVE requires access to the global variables LLSIZE, LASTEL, LARGE, ROW, RECENT, NEXT, JAY and AIJ.

```
(SUBROUTINE RESERVE(INI,INJ)
If (INI=0) or (INJ=0) RETURN
If (INI=INJ)RETURN
I=MIN(INI,INJ);J=MAX(INI,INJ);
L=RECENT (I);M=NEXT(L);
IF (JAY(M)<J) THEN
  L=0;M=ROW(I)
Endif
If M=0 then
  LLSIZE=LLSIZE+1;JAY(LLSIZE)=J;ROW(I)=LLSIZE;LASTEL=LLSIZE
Else
  While (M>0) and (JAY(M)>J) do
    RECENT(I)=L;L=M;M=NEXT(M);Endwhile
  If JAY(M)≠J then
     LLSIZE=LLSIZE+1;JAY(LLSIZE)=J;LASTEL=LLSIZE
     IF L=0 Then
       NEXT(LLSIZE)=ROW(I);ROW(I)=LLSIZE
     Elseif M=0 then
       NEXT(L)=LLSIZE
     Else
       NEXT(LLSIZE)=NEXT(L);NEXT(L)=LLSIZE
     Endif
  Else
     LASTEL=M
  Endif
Endif
ENDreserve)
```

Algorithm

Table VI illustrates the state of the vector during the algorithm for the sample population. At the conclusion of the following steps DIAG contains the diagonal of A:

1 Number animals 1 to N so that parents precede their progeny.
2 Reserve the first two locations in the linked list (LLSIZE=2).

3 Read and store the pedigrees in SIRE and DAM. (3a. Read any known inbreeding coefficients, add 1 and store in DIAG (DIAG(I)=1+F_I)).
4 Reserve space in the lists for parental relationships when both parents are known. When either the sire or dam is unknown store 1.0 in DIAG.

```
(For I=1 to N do
If DIAG(I)=0.0 then
   If (SIRE(I)=0) or (DAM(I)=0) then
      DIAG(I)=1.0
   Else
      CALL RESERVE(SIRE(I),DAM(I));DIAG(I)=LASTEL
   Endif
Endif
Endfor)
```

5 Pass through each row from right to left starting with the Nth row and moving upwards, examine each flagged element and reserve space in the linked lists for any element that is required before the current element can be computed. Keep a record of all flagged elements in the row (in COLUMN). Confine searching to the left of the current element.

```
(For I=N downto 1 do
      K=ROW(I); COLUMN(0)=I
      While K≠0 do
            COLUMN(JAY(K))=I
            K=NEXT(K);Endwhile
      K=ROW(I)
      While K≠0 do
            JKS=SIRE(JAY(K))
            IF (COLUMN(JKS)≠I) then
                  If (JKS>JAY(RECENT(K)))RECENT(I)=K
                  CALL RESERVE(JKS,I)
                  COLUMN(JKS)=I
            Endif
            JKD=DAM(JAY(K))
            IF (COLUMN(JKD)≠I) then
                  If (JKD>JAY(RECENT(K)))RECENT(I)=K
                  CALL RESERVE(JKD,I)
                  COLUMN(JKD)=I
            Endif
            K=NEXT(K);Endwhile
      Endfor)
```

6 Adjust pointers so that the lists are linked in column order and reset COLUMN.

```
COLUMN.
COLUMN(0)=0
(For I=1 to N do
        COLUMN(I)=0
        K=ROW(I);NEW=0
        WHILE K>0 do
                IOLD=NEXT(K); NEXT(K)=NEW; NEW=K; K=IOLD;
                        Endwhile
        ROW(I)=NEW
        Endfor)
```

7 Compute each row starting with the first.

```
   [For I=1 to N do
```

a. Transfer elements on the left of the diagonal into WORK

```
        (K=COLUMN(I)
        While K≠0 do
           WORK(JAY(K))=AIJ(K);K=NEXT(K);Endwhile)
```

b. Compute the diagonal and store it in WORK.

```
        (If(DIAG(I)>2.0 )    DIAG(I)=1+AIJ(DIAG(I))/2;
        WORK(I)=DIAG(I))
```

c. Compute the elements on the right of the diagonal and store them in WORK and AIJ. Adjust the pointers so that elements on the right of the diagonal in the row are linked into columns.

```
        (K=ROW(I)
        While K≠0 do
                JK=JAY(K);WORK(JK)=[WORK(SIRE(JK))
                        +WORK(DAM(JK))]/2
                AIJ(K)=WORK(JK);K2=K;K=NEXT(K);
                NEXT(K2)=COLUMN(JK)
                COLUMN(JK)=K2;JAY(K2)=I;Endwhile)
        Endfor]
```