Raabo.

A/S REGNECENTRALEN
SCANDINAVIAN INFORMATION PROCESSING SYSTEMS

SYSTEM LIBRARY

RCSL NO: 53-M6
TYPE : Algol 5 procedure
AUTHOR : P. Mondrup
EDITION: November 1969 (E)

RC 4000 SOFTWARE

MATHEMATICAL PROCEDURE LIBRARY

solvesym

ABSTRACT:

The boolean procedure solvesym solves a set of n linear algebraic equations with symmetrical coefficient matrix. Only the lower half of the matrix has to be supplied. The procedure value will indicate whether the matrix is singular or not.

INFORMATION DEPARTMENT
DK-2500 VALBY · BJERREGAARDSVEJ 5 · PHONE: (01) 46 08 88 · TELEX: 6464 rcinf dk · CABLES: INFOCENTRALEN

boolean procedure solvesym(n, m, A, X);

1. Function and parameters.

The procedure solves the generalized linear algebraic equation

$$M \times X = B$$

where M is a symmetrical n × n matrix of coefficients, B is a given n × m matrix and X is the unknown n × m matrix.

In this procedure X and B are stored in the same array X, B on entry and X at return.

The lower half of M(1:n, 1:n) is stored in an array A such that M(s, r) = M(r, s) = A(r×(r-1)//2 + s). s <= r.

If M is singular then the procedure will come out with the value false. For each degree of degeneration one of the diagonal elements in M, say A(s×(s+1)//2), is zero, and the corresponding elements of X, X(s, k), k = 1, 2, ..., m, must be zero or very small if the given equation M × X = B has a solution.

Procedure heading:

```
boolean procedure solvesym(n, m, A, X);
value n, m;
integer n, m;
array A, X;
```

Call parameters:

integer n — The number of equations.

integer m — the number of right sides.

real array A(1:m×(n+1)//2) the lower half of the coefficient matrix M(r, s) = M(s, r) = A(r×(r-1)//2 + s), s <=r.

Call and Return parameter:

real array X(1:n, 1:m) is the right sides at call and the solutions at return.

Return parameter:

boolean solvesym. — false if A is singular, true if A is nonsingular.

2. Mathematical method.

The method is the usual Gauss reduction with diagonal pivoting. The pivoting criterion is the following:

In each step a new pivot index r is selected among the not used indices so that

abs M(r, r) / max abs M(r, s)

attains its maximum; and the reduction is carried out in the usual way by making the r'th column = 0 under the diagonal. However, if all possible diagonal elements are zero this can not be done. In that case an index r is found so that

max abs M(r, s)
s ≠ r

attains its minimum.

If this minimum is zero then the whole row is zero and the matrix is singular. In this case the procedure value is set to false and the corresponding r is set to 'has been pivot element', and the search for another r is continued. However, if the minimum is > 0 then row k is replaced by (row k) + (row r) × M(k, r) for all k which have not been pivot index. This will make at least one diagonal element ≠ 0 and the pivot index may be selected as above. The process can now go on until there are zeros under the whole diagonal of M and the solution obtained by simple backward elimination.

If M is singular some of the diagonal elements M(r, r) are zero. During the backward reduction the division by such a diagonal element is skipped. Moreover, the corresponding elements in the r'th row of X(r, k) = B(r, k) will have to be zero (or very small compared to the original values) in case the given equation has a solution.

3. Accuracy, time and storage Requirement.

Accuracy.

In practice the relative error measured as ||AxX - B||/||X|| has been found to be about 10^{-10}. This is not an errorbound, the errorbound has been discussed in detail in literature see e.g. Forsythe og Moler. (ref).

Time.

For m = 1 the time is .2×(n+1)××3 ms

Storage requirement.

The procedure is 4 segments long on backing-store. It uses 70 + 3.5 × n words in stack.

Typographical length: 103 lines, 4 segments.

4. Test and discussion

The procedure is intended for use in such cases where the total matrix M is too big for the available store. A program using decompose and solve will be faster than a program using solve_sym, even if the program must generate the matrix M from the half matrix A.

The procedure has been tested by some equations with coefficients chosen at random and by a representative set of singular equations.

The following program will read n, m, A, B, solve the linear algebraic equation A × X = B and write out the X:

```
Input, solution and output of a symmetrical set of linear algebraic equa-
tions
begin integer n, m, i, j, k, l; boolean s;
  read(in, n, m);
  begin array A(1:(n×(n+1)) shift (-1)), B(1:n, 1:m);
    read (in, A, B);
    s:= -, solvesym(n, m, A, B);
    if s then write(out, <:<10> A is singular:>);
    write(out, <:<10>:>);
    for i:= 1 step 5 until m do
    begin
      j:= if i+4 < m then i + 4 else m;
      for k:= i step 1 until j do write(out, <<_________ddd>, k);
      for k:= 1 step 1 until n do
      begin
        write(out, <:<10>:>, <<ddd>, k, if s then (if A((k×(k+1)) shift (-1))=0
                                    then <:×_:> else <:__:>) else <:__:>);
        for l := i step 1 until j do
          write(out, <<_-d.dddddd10-dd>, B(k, l))
      end k;
      write(out, <:<12><10>:>)
    end i
  end A
end program;
```

5. Reference

George Forsythe and Cleve B. Moler: Computer Solution of Linear Algebraic Systems. Prentice-Hall, Inc. (1967).

6. Procedure text.

```
solvesym = set 4
solvesym = algol
external

boolean procedure solve_sym(n,m,A,X);
message solve sym, version 18 11 69, RCSL 53-M6;
      value n,m; integer n,m; array A,X;
begin integer i,j,k,r,s,t;
      real ai,ak,ar,mi;
      array M(1:n); integer array R(1:n); boolean array B(1:n);

  j:=0; solve_sym:=true;
  for i:= 1 step 1 until n do
  begin
    mi:=0;
    for k:=i-1 step -1 until 1 do
    begin
      if abs A(k+j)> mi then mi:=abs A(k+j);
      if abs A(k+j)>M(k) then M(k):=abs A(k+j)
    end k;
    M(i):=mi;  B(i):=true; j:=j+i;
  end i;
  s:=1;
  for t:= 1 step 1 until n do
  begin
    mi:=ak:=-1;
    for i:= 1 step 1 until n do if B(i) then
```

```
      begin
        ai:=abs A((i×(i+1))shift(-1));
        if M(i)>0 then
        begin
          if mi×M(i)<ai then
          begin
            if ai<>0 then
            begin
              mi:=ai/M(i);  s:=i
            end else if M(i)×ak<1 then
            begin
              ak:=1/M(i);  s:=i
            end
          end
        end (i) > 0 else
        begin
          R(t):=i; B(i):=false; t:=t+1;
          if ai=0 then solve_sym := false
        end M(i)<0
      end i;
      if B(s) then
      begin
        r:=(s×(s-1))shift(-1); ar:= A(r+s);
        if ar=0 then begin ar:=-1; t:=t-1 end else R(t):=s;
        B(s):= false;
        for i:= 1 step 1 until n do if B(i) then
        begin
          j:=(i×(i-1))shift(-1);
          ai:=A(if i<s then r+i else j+s)/ar;  mi:=-1;
          for k:= i step -1 until 1 do if B(k) then
          begin
            ak:= A(j+k):=A(j+k)
              -ai×A(if k<s then r+k else (k×(k-1))shift(-1)+s);
            if abs ak>mi then
            begin
              if i=k then goto L1;
              mi:=abs ak
            end;
            if abs ak>M(k) then M(k):=abs ak;
L1:       end k;
          M(i):=mi;
          for k:=1 step 1 until m do X(i,k):=X(i,k)-ai×X(s,k)
        end i;
```

```
      if A(r+s)=0 then
      begin
        mi:=0;
        for k:=1 step 1 until n do if B(k) then
        begin
          ak:=abs A(if k<s then r+k else (k×(k-1))shift(-1)+s);
          if ak>mi then mi:=ak;
          if ak>M(k) then M(k):=ak;
        end k;
        B(s):= true
      end A(r+s)=0
    end B(s);
  end t;
  for t:= n step -1 until 1 do
  begin
    s:=R(t); r:=(s×(s-1))shift(-1);
    for i := t+1 step 1 until n do
    begin
      j:=R(i); ai:=A(if j<s then r+j else (j×(j-1))shift(-1)+s);
      for k:=1 step 1 until m do X(s,k):= X(s,k)-ai×X(j,k)
    end i;
    ai:= A(r+s);
    if ai<> 0 then for k:= 1 step 1 until m do X(s,k):=X(s,k)/ai
  end t
end solve sym;

comment
```

Call parameters:

integer n — the number of equations.

integer m — the number of right sides.

real array A(1:m×(n+1)//2) — the lower half of the coefficient matrix. M(r, s) = M(s, r) = A(r×(r-1)//2 + s)

Call and Return parameter:

real array X(1:n, 1:m) — is the right sides at call and the solutions at return.

Return parameter:

boolean solvesym. — false is A is singular, true if A is nonsingular;