This report describes a set of Fortran subroutines for solving linear systems with matrices of the form

\[ M = \begin{bmatrix} A & B \\ C^T & D \end{bmatrix} \]

where the \( n \times n \) matrix \( A \) may be singular but the \( n \) by \( m \) matrices \( B \) and \( C \) are such that \( M \) is nonsingular and well-conditioned. When \( A \) is large but has special structures (e.g., sparseness, band or profile structures), a block Gaussian elimination algorithm is commonly used because it allows solution of systems with \( M \) by solving only with \( A \). Unfortunately this algorithm is unstable \emph{numerically} when \( A \) is nearly singular and can produce inaccurate solutions. This report describes an implementation of a stable variant of the block elimination algorithm, written in Fortran-77 and built upon two popular linear-algebra packages (Linpack and YSMP).

**DBEPACK : A Program Package for Solving Bordered Singular Systems**

Tony F. Chan and Thomas A. Grossi
Research Report YALEU/DCS/RR-336
August 1985

This work was supported by the Department of Energy under contract DE-AC02-81ER10996, and by the Army Research Office under contract DAAG-83-0177.

**Keywords:** linear systems, bordered systems, singular matrices, deflation, LINPACK, YSMP
1 Introduction
This report describes a set of Fortran subroutines for solving linear systems of the form

\[
M \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} A & B \\ C^T & D \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} f \\ g \end{bmatrix}
\]

where the \( n \times n \) matrix \( A \) may be singular but the \( n \times m \) matrices \( B \) and \( C \) are such that \( M \) is nonsingular and well-conditioned. We assume that \( A \) has a nullity not greater than 1 and \( m \ll n \).

Systems of the form (1.1) arise, for example, in continuation methods, homotopy methods, and constrained optimization [1], and the solution of such systems often constitutes the most time-consuming part of the overall computation. Since \( M \) is assumed to be well-conditioned, the use of Gaussian Elimination on \( M \) with some form of pivoting is guaranteed to be stable. However, this approach is only suitable when \( n \) is small or when \( A \) is dense since the whole matrix \( M \) has to be stored to allow for fill-ins. When \( A \) is large but sparse, this approach is impractical: pivoting in (a) below could produce a matrix of the form (b), resulting in complete fill-in.

\[
(a) \begin{bmatrix} x & x & \cdots & x \\ x & 0 & \cdots & x \\ 0 & \cdots & \ddots & \vdots \\ x & x & \cdots & x \end{bmatrix} \quad \quad (b) \begin{bmatrix} x & x & \cdots & x \\ x & x & 0 & \vdots \\ \vdots & 0 & \cdots & \vdots \\ x & x & \cdots & x \end{bmatrix}
\]

In this case, or when \( A \) has already been factored, it becomes desirable to employ other algorithms for solving systems with \( M \) which involve solving systems with \( A \) only. The following block elimination algorithm has this desirable property:

**Algorithm BE:**

- solving a bordered system by block elimination
- Solve \( AV = B \), \( Aw = f \).
- Solve \( (D - C^TV)y = (g - C^Tw) \).
- Compute \( x = w - V y \).

Unfortunately, Algorithm BE is unstable numerically when \( A \) is nearly singular and can produce completely inaccurate solutions \((x,y)\). Consider, for example, the following matrix:

\[
\begin{bmatrix} 1 & 1 & 0 \\ 0 & \epsilon & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 + \epsilon \\ 1 \end{bmatrix},
\]

where \( |\epsilon| \) is small enough that \( 1 + \epsilon = 1 \) in floating-point arithmetic for the machine at hand. The correct values for \( v \) and \( w \) of Algorithm BE (with \( n = 2, m = 1 \)) are \( v = (-1/\epsilon, 1/\epsilon)^T \), and \( w = (1 - 1/\epsilon, 1 + 1/\epsilon)^T \), but when executed with the floating-point arithmetic of our machine, \( w \) will instead equal \( (-1/\epsilon, 1/\epsilon)^T \), producing the erroneous solution vector \( x = (0,0)^T \).

In [3, 4] a stable variant of BE, the Deflated Block Elimination Algorithm (or "Algorithm DBE"), is derived using deflation techniques developed in [2]. This algorithm retains the property that
only systems involving $A$ need be solved, allowing continued exploitation of $A$'s possibly special structure. For a survey of algorithms for solving (1.1), the reader is referred to [1].

In the remainder of this report, we will present in detail the algorithms used by the various subroutines. We will then discuss general implementation details, after which the calling sequences will be shown. These will be followed by a test driver which illustrates at once how our routines might be used and the advantages of our package when $A$ is nearly singular. A complete listing of the Fortran code is included at the end.

2 Algorithms

2.1 Deflated Decomposition of Solutions of Nearly Singular Systems

Consider the system

$$Az = p$$  

(2.1)

where $A$ may be singular with nullity of at most one. Let the singular values of $A$ be denoted by $\{\sigma_i\}_{i=1}^n$ and the corresponding left and right singular vectors by $\{u_i\}_{i=1}^n$ and $\{v_i\}_{i=1}^n$. Then the solution $z$ can be expressed as

$$z = \sum_{i=1}^{n-1} \frac{(u_i^TP)u_i}{\sigma_i} + \frac{(u_n^TP)v_n}{\sigma_n},$$

where we have isolated the term corresponding to the smallest singular value $\sigma_n$. Therefore, $z$ can be uniquely represented as

$$z = z_d + \frac{c_p}{\delta} \phi,$$  

(2.2)

where

$$z_d \equiv \sum_{i=1}^{n-1} \frac{(u_i^TP)u_i}{\sigma_i},$$

$\delta \equiv \sigma_n$, $\phi \equiv v_n$, $\psi \equiv u_n$, and $c_p \equiv \psi^TP$. We shall call $z_d$ the deflated solution of (2.1) and (2.2) the deflated decomposition of $z$ [2]. The basic idea behind the deflated decomposition is to break $z$ into two parts: a part in the approximate null space $\phi$ and a deflated part $z_d$ purged of $\phi$.

Computing the deflated decomposition according to its definition requires computing the singular value decomposition of $A$, which is expensive for large problems. It is therefore desirable to have alternative algorithms that require lower costs. For example, the algorithm implemented in this report requires only computing the $LU$ factorization of $A$.

It can be shown that $z_d$ is the unique solution to the following system:

$$P_\psi Az_d = P_\psi p$$  

$$P_\phi z_d = z_d,$$

where $P_u$ with $||u|| = 1$ denotes the orthogonal projector $I - uu^T$. Thus, $z_d$ is the solution to a singular but consistent system "close" to (2.1). This characterization can be used to derive the following simple algorithm for computing the deflated solution $z_d$. 

2
Algorithm DF:
Given $\phi, \psi$, and $\delta$, computes the deflated solution $z_d$ of $Ax = p$:

Compute $\hat{p} = P_{\phi}\psi \equiv p - (\psi^T p)\psi$
Solve $Az_d = \hat{p}$ for $z_d$.
$z_d \leftarrow P_{\phi}z_d$.

The singular vectors $\phi$ and $\psi$ and the singular value $\delta$ can be computed, for example, by the following inverse iteration algorithm:

Algorithm II

Make an initial guess for $\psi$
Repeat until convergence
Solve $A\phi' = \psi$ for $\phi'$
$\phi = \phi'/||\phi'||$
Solve $A^T\psi' = \phi$ for $\psi'$
$\psi = \psi'/||\psi'||$
End
Solve $A\phi' = \psi$ for $\phi'$ once more
$\delta = 1/||\phi'||$
$\phi = \phi'/||\phi'||$

In practice, this algorithm usually converges within two or three iterations if $A$ is nearly singular.

2.2 Deflated Block Elimination
Using Algorithm DF, we can compute the deflated decompositions of $V$ and $w$ of Algorithm BE:

$$V = V_d + \frac{1}{\delta} \phi(\psi^T B),$$
$$w = w_d + \frac{1}{\delta} \phi(\psi^T f).$$

Then it can be shown [4] that the solution to (1.1) is given by

$$x = w_d - V_d\beta + \alpha\phi$$
$$y = \beta$$

where $\alpha, \beta$ is the solution of the following $m + 1$ by $m + 1$ linear system:

$$E \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \psi^T f \\ g - C^T w_d \end{pmatrix}$$

where

$$E \equiv \begin{pmatrix} \delta & \psi^T B \\ C^T \phi & D - C^T V_d \end{pmatrix}.$$
Algorithm DBE:

Use Algorithm II to compute $\phi, \psi$ and $\delta$.
Compute deflated solution $V$ of $AV = B$.
Compute deflated solution $w$ of $Aw = f$.
Solve the $m + 1$ by $m + 1$ linear system:

$$E \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \psi^T f \\ g - CTw_d \end{pmatrix}.$$  

where

$$E \equiv \begin{pmatrix} \delta & \psi^T B \\ CT_\phi & D - CTV_d \end{pmatrix}.$$  

Assemble the solution by substituting into the following formula:

$$x = u_d - V_d \beta + \alpha \phi$$

$$y = \beta$$

It can be proven that Algorithm DBE is numerically stable and that $E$ is nonsingular if $M$ is nonsingular [4]. It also retains the desirable property of Algorithm BE of requiring only a solver for $A$ and hence can exploit special structures in $A$. It is almost as efficient as Algorithm BE, the only overhead being two extra backsolves for the deflated solutions and a few more for the null vectors. Our implementation is economical with respect to space as well, requiring only five vectors of size $n$ for working space. In light of these facts, it is entirely practical to use Algorithm DBE for any bordered system, regardless of whether $A$ is nearly singular or not, ensuring accurate results without the need for specific tests of singularity.

3 Implementation Notes

Our implementation of Algorithm DBE is based on two popular packages for linear systems: Linpack [5] and the Yale Sparse Matrix Package [7, 6]. We have included routines for use with Linpack's general, band and tridiagonal matrix routines, and routines for use with YSMP's general routines with and without compressed storage, as well as with its routines for symmetric matrices. The calling sequences have been designed to mirror closely those of the packages that they use, with the aim of making as painless as possible their incorporation into programs that may already use the base packages. The routines are written in standard Fortran-77 and are thus portable to any machine supporting that language. We have tested them on VAX 11/780, DEC-20 and Apollo DN-300. They assume the availability of the Basic Linear Algebra Subroutines (BLAS), which is included in LINPACK.

As described above, there are routines for use with Linpack's general, band, and tridiagonal matrix routines (SGE..., SBD..., and SGT..., respectively). The routines SYC..., SYN..., and SYS... are for use with YSMP's general routines with and without compressed storage, and for symmetric matrices, respectively. Within each set of routines, the driver S..DBE implements Algorithm DBE. This is the only required user interface. S..DBE calls the auxiliary routines S..DF for deflated decomposition and S..II for inverse iteration. These routines may also be
called independently for specialized applications. For example, the S..DF routines can be used to compute the deflated decomposition of linear systems with $A^T$. We include the S..BE routines, which do not use deflation, for comparison.

The routines S..DBE require two work arrays which upon return contain most of the intermediate computations. WORK1, whose leading dimension must be at least $n$, contains $V_d$ in its first $m$ columns, and $w_d$ in the $(m+1)^{th}$. The next two columns of WORK1 hold the left and right null vectors from the matrix $A$, $\psi$ and $\phi$. The small intermediate matrix $E$ is stored in the first $m+1$ columns of WORK2, whose leading dimension must be at least $m+1$. The vector $\psi^T B$ is placed in the $(m+1)^{th}$ column of the same array. Similarly, S..BE returns $v$ in the first $m$ columns of WORK1 and $w$ in the $m+1$-th column of WORK1. The Schur complement $D - C^T V$ is stored in WORK2.

The argument JOB (a character string of six characters or less) is used to indicate which inputs were the same as in the last call to S..DBE. When JOB contains 'A', 'B', 'C' 'D', 'F', and 'G' it indicates that the arrays A, B, CT, D, F, and G stay the same, respectively. If JOB contains 'S', it signifies that $A$ is new but already factored by the corresponding Linpack routines S..CO or S..FA.

The auxiliary routines also take a JOB parameter which allows precomputed results to be supplied to them. Since they are internal, and less complex, the parameter is an integer. S..DBE makes two calls to S..DF; the first call usually requires computation of singular vectors by S..II, but the second always makes use of the values obtained by the first call. SGEII (or SGBII) can profit from the singular vector computed by SGECO (or SGBCO) by accepting it as input and using it as its initial guess. S..II also allows specification of an upper limit on the number of iterations to perform. When the initial guess is the singular vector provided by SGECO or SGBCO, convergence generally occurs immediately; for other cases we permit a maximum of three iterations. S..BE can accept a matrix in factored form, and can also reuse the values which were returned to the user in the work arrays in a previous call to it.
4 Sample Driver

We now demonstrate use of our package with a test driver which calls each of our main routines. We have chosen three nearly singular matrices of suitably different forms:

1) matrix $T$, an upper-triangular matrix with +1 along the diagonal and -1 in all entries above it, used to test SGEDBE;

2) matrix $W$, a tridiagonal matrix used to test SGBDBE and SGTDBE: $W = \hat{W} - \lambda_{\text{max}}(\hat{W})I$, where

$$
\hat{W} = \begin{bmatrix}
10 & 1 \\
1 & 9 & 1 \\
1 & 8 & \ddots & 0 \\
\vdots & \ddots & \ddots & \ddots & 0 \\
0 & 1 & -1 & 1 \\
0 & 1 & -2 & \ddots & \ddots & 1 \\
0 & 1 & -10 & \ddots & \ddots & \ddots & \ddots & 1
\end{bmatrix}
$$

and $\lambda_{\text{max}}(\hat{W}) = 10.7461942$ ;

and 3) matrix $P$, of the form $P = \Delta_h - \lambda_{\text{min}}(\Delta_h)I$, where $\Delta_h$ is the 5-point discrete LaPlacian operator, used to test SYNDBE and SYSDBE.

A target solution is built using a random-number generator. This vector is multiplied by each matrix $M$ to produce a right-hand side from which to solve. We solve each system first with the regular (non-deflated) block elimination algorithm and then with our deflated block elimination algorithm, comparing each newly determined solution with the exact solution to calculate the error. The results of our tests are summarized in the following table.

<table>
<thead>
<tr>
<th>Matrix</th>
<th>DBE routine</th>
<th>Error without deflation</th>
<th>Error with deflation</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>SGE</td>
<td>5.70781400</td>
<td>0.00013622</td>
</tr>
<tr>
<td>W</td>
<td>SGB</td>
<td>1.94438300</td>
<td>0.00001813</td>
</tr>
<tr>
<td>W</td>
<td>SGT</td>
<td>1.49100000</td>
<td>0.00001882</td>
</tr>
<tr>
<td>P</td>
<td>SYN</td>
<td>***</td>
<td>0.000000368</td>
</tr>
<tr>
<td>P</td>
<td>SYS</td>
<td>5.36235400</td>
<td>0.00000174</td>
</tr>
</tbody>
</table>

*** produced division by 0

The Fortran code which produced this output follows. Included are several subroutines that demonstrate how to set up a matrix for the various representation schemes.
program tester

driver for testing the routines in DBEPACK

parameter (lda = 50)

real alpha, lambda, epsilon

integer n, ipvt(lda, 2)
real a(lda, lda), work1(lda, 10), work2(lda, 10)
real l(lda), di(lda), u(lda)
real ay(1000), ia(lda), ja(1000), rsp(1000)
integer r(lda), c(lda), ic(lda), isp(1000), nsp,
* case, path, flag
equivalence (isp, rsp)

real t
integer seed
real b(lda, lda), cT(lda, lda), d(lda, lda), f(lda), g(lda)
real x(lda), y(lda), xp(lda)
character*6 job1, job2
data seed / 111999 /

data nsp / 900/, lratio / 1/
do 5 i = 1, lda
   r(i) = i
c(i) = i
   ic(i) = i
5 continue

m = 5
job1 = ' ' 
job2 = 'a'

set up border and target solution

do 15 j = 1, lda
   do 10 k = 1, m
      b(j, k) = ranf(seed)
cT(k, j) = ranf(seed)
10 continue
\begin{verbatim}
x(j) = ranf(seed)
15 continue
do 25 j = 1,m
do 20 k = 1,m
d(j,k) = ranf(seed)
20 continue
25 continue

2000 format (' without deflation')
2010 format (' with deflation')

write (6,1000)
1000 format (' building upper-triangular matrix of 1's and -1's/)
n = 25
call ebl(a,lda,n)
c
c build rhs for system
call brhs(n,m,a,lda,b,lda,cT,lda,d,lda,x,f,g)
c
c solve system without deflation
call sgebe(n,m,a,lda,ipvt,b,lda,cT,lda,d,lda,f,g,xp,y,  
  *  work1,lda,work2,lda,job1)
c
c determine error
write (6,2000)
call compare(n,m,x,xp,y)
c
c solve system with deflation
call sgedbe(n,m,a,lda,ipvt,b,lda,cT,lda,d,lda,f,g,xp,y,  
  *  work1,lda,work2,lda,job2)
c
c determine error
write (6,2010)
call compare(n,m,x,xp,y)
c
c
write (6,1005)
1005 format (/' building matrix W'/)
call eBW(a,lda,n)
call brhs(n,m,a,lda,b,lda,cT,lda,d,lda,x,f,g)
c
c first test Linpack routines for band matrices
c
call bbW(a,lda,n,mi,nu)
call sgbbe(n,m,a,lda,mi,nu,ipvt,b,lda,cT,lda,d,lda,  
  *  f,g,xp,y,work1,lda,work2,lda,job1)
write (6,2000)
\end{verbatim}
call compare (n,m,x, xp,y)
call sgbdbe (n,m, a, lda, ml, mu, ipvt, b, lda, c, lda, d, lda,
  *       f, g, xp, y, work1, lda, work2, lda, job)
write (6, 2010)
call compare (n,m,x, xp,y)
c
repeat procedure with Linpack routines for tridiagonal matrices
c
  call tbW (n, l, di, u)
call sgtdbe (n,m, L, DI, U, B, LDA, CT, LDA, D, LDA,
  *       F, G, Xp, Y, WORK1, LDA, WORK2, LDA, job)
write (6, 2000)
call compare (n,m,x, xp,y)
c
call sgtdbe (n,m, L, DI, U, B, LDA, CT, LDA, D, LDA,
  *       F, G, Xp, Y, WORK1, LDA, WORK2, LDA, job)
write (6, 2010)
call compare (n,m,x, xp,y)
c
write (6, 11010)
1010 format (//" building matrix P")
n = 16
call ebP (a, lda, n)
call brhs (n, m, a, lda, b, lda, c, lda, d, lda, x, f, g)
c
test first with YSMP routines for nonsymmetric matrices
c
call nbP (n, ia, ja, a)
call synbe (n,m, R,C, IC, IA, JA, A, NSP, ISP, RSP, ESP,
  *       B, LDA, CT, LDA, D, LDA, F, G, Xp, Y, WORK1, LDA, WORK2, LDA, job)
write (6, 2000)
call compare (n,m,x, xp,y)
c
call synbe (n,m, R,C, IC, IA, JA, A, NSP, ISP, RSP, ESP,
  *       B, LDA, CT, LDA, D, LDA, F, G, Xp, Y, WORK1, LDA, WORK2, LDA, job)
write (6, 2010)
call compare (n,m,x, xp,y)
c
repeat procedure with YSMP routines for symmetric matrices
c
call sbP (n, ia, ja, a)
call sysbe (n,m, C, IC, IA, JA, A, NSP, ISP, RSP, ESP,
  *       B, LDA, CT, LDA, D, LDA, F, G, Xp, Y, WORK1, LDA, WORK2, LDA, job)
write (6, 2000)
call compare (n,m,x, xp,y)
c
call sysdb (N,M, C,IC, IA,JA,A, NSP,ISP,RSP,ESP, 
* B,LDA, CT,LDA, D,LDA, F,G, Xp,Y, WORK1,LDA,WORK2,LDA, job2)
write (6,2010)
call compare (n,m,xp,y)
end

C-----------------------------------------------------
C
C subroutine compare (n,m,xp,y)
C
C computes and prints the norm of the difference
C between x and (xp,y)
integer n
real x(n),xp(n),y(m)
real snrm

write (6,1040) (x(j),j=1,n+m)
1040 format (/t5,'original x,y:',/(t5,6f10.5))
write (6,1053) (xp(j),j=1,n),(y(j),j=1,m)
1053 format (/t5,'new x,y:',/(t5,6f10.5))
call scopy (m, y,1, xp(n+1),1)
call saxpy (n+m,-1.,x,1,xp,1)
snrm = snrm2(n+m,xp,1)
write (6,2000) snrm
2000 format (t10,'norm of error =',f15.8)
end

C-----------------------------------------------------
C
C subroutine brhs (n,m, a,lda, b, ldb, cT, ldc, d, ldd, x, f,g)
C
C builds a right hand side: copies b, cT and d into a, then
C multiplies the larger matrix (M) by x to produce f and g
C
integer n,m, lda, ldb, ldc, ldd
real a(lda,n)
real b(ldb,m), cT(ldc,n), d(ldd,m), x(lda), f(n), g(m)

do j = 1,m
   call scopy (n, cT(j,1), lda, a(n+j,1), lda)
call scopy (n, b(1,j),1, a(1, n+j),1)
call scopy (m, d(1,j),1, a(n+1, n+j),1)
endo

call eaxb (a, lda, n+m, x, f)
call scopy (m, f(n+1),1, g,1)
C
C subroutine ebl (a,lda,n)
C
C builds upper-triangular matrix of 1's and -1's in SGE-format
C
integer n,ipvt(n)
real a(lda,n)

do 20 i = 1,n-1
   a(i,i) = 1.
   do 10 j = i+1,n
      a(i,j) = -1.
      a(j,i) = 0.
10    continue
20    continue
   a(n,n) = 1.

end

C
C subroutine ebw (a,lda,n)
C
C builds matrix W in SGE-format
C
integer lda,n
real a(lda,lda)

n = 21
do 40 ii = 1,n
   do 50 jj = 1,n
      a(ii,jj) = 0.
50    continue
40    continue
   a(1,1) = -0.7461942
   do 60 ii = 2,n
      a(ii,ii) = 11. - ii - 10.7461942
      a(ii,ii-1) = 1.
      a(ii-1,ii) = 1.
60    continue
end

C
C subroutine ebp (a,lda,n)
C
C
builds matrix P in SGE-format
C
integer i,j,k
integer lda,n,nsqrt
real a(lda,lda), lambda, alpha, x, y, h, diaval, epsilon

nsqrt = sqrt(float(n))
h = float(nsqrt) + 1.
lambda = 8 * (sin (3.1415926 / (2 * h))) ** 2

do 50 i = 1,n
   do 60 j = 1,n
      a(i,j) = 0.
   60   continue
50  continue

i = 0
do 10 j = 1,nsqrt
   i = i + 1
   a(i,i) = 4. - lambda
   do 20 k = 2,nsqrt
      i = i + 1
      a(i,i) = 4. - lambda
      a(i,i-k) = -1.
      a(i-k,i) = -1.
   20   continue
10  continue

do 30 i = nsqrt+1,n
   a(i,i-nsqrt) = -1.
   a(i-nsqrt,i) = -1.
30  continue
end

C---------------------------------------------------------------------------
C
subroutine bbW (a,lda,n, ml,mu)
C
builds matrix W in SGB-format
C
integer lda,n, ml,mu,m
real a(lda,n)

k(i,j) = i - j + m

n = 21
ml = 1
mu = 1
m = ml + mu + 1

do 40 ii = 1,mi + ml + mu + 1
    do 50 jj = 1,n
        a(ii,jj) = 0.
    50 continue
40 continue
a(k(1,1),1) = -0.7461942
do 60 j = 2,n
    a(k(j,j),j) = 11. - j - 10.7461942
    a(k(j,j-1),j-1) = 1.
    a(k(j-1,j),j) = 1.
60 continue
end

C---------------------------------------------------------------------------
C subroutine tbW(n,1,d,u)
C builds matrix W in SGT-format
C
integer n
real l(n),d(n),u(n)

n = 21
do 50 i = 1,n
    l(i) = 1.
    d(i) = 11. - i - 10.7461942
    u(i) = 1.
50 continue
end

C---------------------------------------------------------------------------
C subroutine nbP (n, ia, ja, a)
C builds matrix P in NDRV-format
C
integer n,nsqrt, ia(1), ja(1), aptr
real a(1),h,lambda

nsqrt = sqrt(float(n))
h = float(nsqrt) + 1.
lambda = 8 * (sin(3.1415926 / (2*h))) ** 2
aptr = 1
do 100 i = 1, n
   ia(i) = aptr
   if ((i-nsqrt) .le. 0) goto 10
      a(aptr) = -1.
      ja(aptr) = i-nsqrt
      aptr = aptr + 1
   10 continue
   if (((mod(i-1, nsqrt) .eq. 0) .or. (i-1 .le. 0)) goto 20
      a(aptr) = -1.
      ja(aptr) = i-1
      aptr = aptr + 1
   20 continue
   a(aptr) = 4. - lambda
   ja(aptr) = i
   aptr = aptr + 1
   if ((mod(i, nsqrt) .eq. 0) .or. (i+1 .gt. n)) goto 30
      a(aptr) = -1.
      ja(aptr) = i+1
      aptr = aptr + 1
   30 continue
   if ((i+nsqrt) .gt. n) goto 40
      a(aptr) = -1.
      ja(aptr) = i+nsqrt
      aptr = aptr + 1
   40 continue
100 continue
   ia(n+1) = aptr
end

C------------------------------------------------------------------------
C
C subroutine sbP (n, ia, ja, a)
C
C builds matrix P in SDRV-format
C
integer n, nsqrt, ia(1), ja(1), aptr
real a(1), h, lambda

nsqrt = sqrt(float(n))
h = float(nsqrt) + 1.
lambda = 8 * (sin (3.1415926 / (2*h))) ** 2

aptr = 1
do 100 i = 1, n
   ia(i) = aptr
   a(aptr) = 4. - lambda
   ja(aptr) = i
   aptr = aptr + 1

14
if ((mod(i, nsqrt) .eq. 0) .or. (i+1 .gt. n)) goto 30
  s(aptr) = -1.
  ja(aptr) = i+1
  aptr = aptr + 1
30     continue
if ((i+nsqrt) .gt. n) goto 40
  s(aptr) = -1.
  ja(aptr) = i+nsqrt
  aptr = aptr + 1
40     continue
100    continue
      ia(n+1) = aptr
end

Acknowledgements: The authors wish to thank Mr. Leon Marr for his help in the preparation of the report.

References


5 Source Code Listings

5.1 SGEDBE

C The routines in this package implement the deflated block-
C elimination algorithm for solving systems of the form:
C
C     | x |   A   | B   | x | f |
C     | M | = | | | = | |
C     | y |   CT     | D   | y | g |
C
C discussed in T. F. Chan and D. Resasco, "Generalized Deflated
C Block-Elimination", Technical Report YALEU/DCS/RR-337, Dept. of
C
C This set of routines calls LINPACK's SGE- routines and the SBLAs.
C Implemented by Thomas A. Grossi, Yale University, 1985.
C
C SUBROUTINE SGEDBE
C * (N,M,A,LDA, IPVT, B,LDB, CT,LDCT, D,LDD, F,G, X,Y,
C * WORK1,LDW1, WORK2,LDW2, JOB)
C
C the deflated block elimination algorithm
C arguments:
C on entry:
C N INTEGER
C the order of the matrix A
C M INTEGER
C the order of the borders to A in M
C A REAL(LDA,N)
C the matrix to be factored.
C LDA INTEGER
C the leading dimension of the array A. LDA >= N.
IPVT INTEGER(N+M) 
an integer vector of pivot indices. The last m spaces are 
required for working space 

B REAL(LDB,M) 
right-side border to matrix A in matrix M 

LDB INTEGER 
the leading dimension of the array B. LDB >= N. 

CT REAL(LDC,N) 
bottom border to matrix A in matrix M 

LDC INTEGER 
the leading dimension of the array CT. LDC >= M. 

D REAL(LDD,M) 
lower right-hand entries of M 

LDD INTEGER 
the leading dimension of the array D. LDD >= M. 

F REAL(N) 
G REAL(M) 
right-hand side to solve with 

WORK1 REAL(LDW1,M+4) 
used to hold Vd, Wd, psi T B, psi and phi 

LDW1 INTEGER 
the leading dimension of the array WORK1. LDW1 >= N. 

WORK2 REAL(LDW2,M+3) 
used to hold E, g', psi T f and delta 

LDW2 INTEGER 
the leading dimension of the array WORK2. LDW2 >= M+1. 

JOB CHARACTER*6 
indicates which inputs are the same as in the last call 
to SGEBBE. If there was no such call, set JOB = 
' ' or 'a ' (see below). Otherwise, JOB contains 
as many of the following apply: 
'A' if A stays the same 
'S' if A is new but already factored by SGECO or SGEFA 
'B' if B stays the same 
'C' if CT stays the same 
'D' if D stays the same
'F' if F stays the same
'G' if G stays the same

on exit:

A    REAL(LDA,N)
contains an upper triangular matrix and
the multipliers which were used to obtain it. The
factorization can be written A = L*U, where L is the
product of permutation and unit lower triangular ma-
trices and U is upper triangular.

IPVT INTEGER(N+M)
an integer vector of pivot indices. The last m spaces are
required for working space.

X    REAL(N)
Y    REAL(M+1)
solution vector (Y(M+1) is an extra storage location).

WORK1 REAL(LDW1,M+4)
used to hold Vd, Wd, psi T B, psi and phi

WORK2 REAL(LDW2,M+3)
used to hold E, g',psi T f and delta

Savings on storage:
the following pairs of inputs may be equivalent:
(X,F)  (Y,G)  (B,WORK1)  (D,WORK2)
In general, if equivalent storage is used, then a change in any
of the inputs in either of the groups (A,B,C,D) or (F,G)
requires that the entire group be re-entered. Specific
exceptions to this rule can be determined by examining
the algorithm.

INTEGER N,M, LDA,LDB,LDC,LDD,LDW1,LDW2, IPVT(N), AJOB
REAL A(LDA,N), B(LDB,M), CT(LDC,N), D(LDD,M), F(N), G(M), X(N), Y(M)
REAL WORK1(LDW1,M), WORK2(LDW2,M), DELTA
CHARACTER*6 JOB
LOGICAL NEWA,NEWB,NEWC,NEWD,NEWF,NEWG

the following constants are used to partition WORK1 and WORK2
into their various vectors; MP1 stands for the 'extra' row and
column added to D in forming E. WORK1 is primarily used for Vd,
and WORK2 for E
INTEGER MP1, CB, WD, CF, PSI, PHI, GP, ALPHA
MP1 = M + 1
CB = MP1
WD = CB + 1
CF = MPI
PSI = WD + 1
PHI = PSI + 1
GP = MPI + 1
ALPHA = MPI

C

AJOB = 0
IF (INDEX(JOB,'A') .NE. 0) AJOB = 2
IF (INDEX(JOB,'S') .NE. 0) AJOB = 1
NEWA = (AJOB .NE. 2)
NEWB = (INDEX(JOB,'B') .EQ. 0)
NEWC = (INDEX(JOB,'C') .EQ. 0)
NEWD = (INDEX(JOB,'D') .EQ. 0)
NEWF = (INDEX(JOB,'F') .EQ. 0)
NEWG = (INDEX(JOB,'G') .EQ. 0)

Algorithm:

factor A, compute psi, phi, delta
compute deflated solution to A V = B
compute deflated solution to A w = f
build E: | (0 - cT Vd) (cT phi) |
          | CbT   delta |
build g': | g - cT Wd |
          | Cf     |
solve E | y | = g' for y
          | alpha |
x = Wd - Vd y + alpha phi

if AJOB = 0 or 1, or B is new, we start by solving A Vd = B;
this may imply factoring A, and/or computing psi, phi and delta
IF (NEWA .OR. NEWB) THEN

for the first element of Vd, AJOB will tell sgeDF what to do
CALL SGEFD (A,LDA,N,IPVT, B(1,1), WORK1(1,PSI), WORK1(1,PHI),
             * DELTA, WORK1(1,1), WORK1(1,CB), AJOB)

compute remaining columns of Vd using results of first call
IF (M .GT. 1) THEN
  DO 10 I = 2,M
    CALL SGEFD (A,LDA,N,IPVT, B(I,I), WORK1(I,PSI),
                * WORK1(I,PHI), DELTA, WORK1(I,1), WORK1(I,CB), 2)
  10    CONTINUE
ENDIF
ENDIF

We must recompute Wd and Cf if A or F have changed
IF (NEWA .OR. NEWF) THEN
CALL SGEDF (A,LDA,N,IPVT,F, WORK1(I,PSI),WORK1(I PHI),
         * DELTA, WORK1(I,WD),WORK2(CF,GP),2)
ENDIF

C build and factor E
IF (NEWA .OR. NEWB .OR. NEWC .OR. NEWD) THEN
   CALL SCOPY (M, WORK1(I,CB),1, WORK2(MP1,1),LDW2)
   WORK2(MP1,MP1) = DELTA
   DO 30 I = 1,M

      C compute D - cT Vd, column by column
      DO 20 J = 1,M
         WORK2(I,J) = D(I,J) - SDOT(N, CT(I,1),LDC, WORK1(I,J),1)
      CONTINUE

      C compute cT PHI element by element
      WORK2(I,MP1) = SDOT(N, CT(I,1),LDC, WORK1(I,PHI),1)
      CONTINUE

      C factor E
      CALL SGEFA (WORK2,LDW2,MP1,IPVT(N+1),INFO)
ENDIF

C g' depends on a lot of things
IF (NEWA .OR. NEWC .OR. NEWF .OR. NEWG) THEN
   DO 40 I = 1,M
      WORK2(I,GP) = G(I) - SDOT(N, CT(I,1),LDC, WORK1(I,WD),1)
   CONTINUE
ENDIF

C compute x and y
CALL SCOPY (MP1, WORK2(I,GP),1, Y,1)
CALL SGEQL (WORK2,LDW2,MP1,IPVT(N+1), Y, 0)
   DO 50 I = 1,N
      X(I) = WORK1(I,WD) - SDOT(M, WORK1(I,1),LDW1, Y,1)
   CONTINUE
CALL SAXPY (N,Y(ALPHA), WORK1(I,PHI),1, X,1)
WORK2(I,GP+1) = DELTA

END

C SUBROUTINE SGEDF(A,LDA,N,IPVT,P,PSI,PHI,DELTA,ZD,CP,JOB)
C computes the deflated decomposition of A z = p, returning
C solution in the form:


C
C z = z + phi (c / delta)
C d = p
C
arguments are the same as for SGEDBE except:
C on entry:
C
P REAL(N)
C contains rhs to system of equations
C PSI REAL(N)
C PHI REAL(N)
C left and right null vectors to matrix A
C (only on entry if JOB >= 2)
C
DELTA REAL
C smallest singular value for matrix A
C (only on entry if JOB >= 2)
C
JOB INTEGER
C JOB = 0: start the deflation algorithm from scratch; i.e.,
C it factors the matrix, performs inverse iteration to
C determine PSI, PHI and DELTA, and then computes the
C deflated solution.
C JOB = 1: assume that A has already been factored by SGECO
C or SGEFA (or a previous call to SYNDBE), and
C continue from there.
C JOB > 1: additionally, PSI, PHI and DELTA have already
C been computed.
C
on exit:
C
PSI REAL(N)
C PHI REAL(N)
C left and right null vectors to matrix A
C
DELTA REAL
C smallest singular value for matrix A
C
ZD REAL(N)
C deflated solution to system A z = p
C Note that P and ZD may be the same vector
C
CP REAL
C phiT p.
C
INTEGER LDA,N,IPVT(N),JOB,IJOB
REAL A(LDA,N),P(N),PSI(N),PHI(N),DELTA,ZD(N),CP
INTEGER INFO
REAL PSITP,RCOND
LOGICAL TRANS

IJOB = JOB
TRANS = (IJOB .GE. 10)
IF (TRANS) IJOB = IJOB - 10

IF (IJOB .EQ. 0)
  * CALL SGEFA (A,LDA,N,IPVT,INFO)
IF (IJOB .LE. 1)
  * CALL SGEII (A,LDA,N,IPVT,PSI,PHI,DELTA,0,3)

IF (TRANS) GOTO 20

C
C       Perform deflation with A
C
C           Zd = p - (psiT p) psi ; solve for Zd ; Cp is approx (psiT p)
CP = SDOT (N,P,1,PSI,1)
CALL SCOPY (N,P,1,ZD,1)
CALL SAXPY (N,-CP,PSI,1,ZD,1)
CALL SGESL (A,LDA,N,IPVT,ZD,0)
C
C       orthogonalize Zd with respect to phi
CALL SAXPY(N,-SDOT(N,PHI,1,ZD,1),PHI,1,ZD,1)
GOTO 30

20 CONTINUE

T
C
C       Perform deflation with A
C
C           T
C
C           Zd = p - (phiT p) phi ; solve for Zd ; Cp = (phiT p)
CP = SDOT (N,P,1,PHI,1)
CALL SCOPY (N,P,1,ZD,1)
CALL SAXPY (N,-CP,PHI,1,ZD,1)
CALL SGESL (A,LDA,N,IPVT,ZD,1)
C
C       orthogonalize Zd wrt psi
CALL SAXPY(N,-SDOT(N,PSI,1,ZD,1),PSI,1,ZD,1)

30 CONTINUE
END

C\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\"
computes approximate left and right null vectors of A by applying
the inverse iteration algorithm described in T. F. Chan, "Deflated
Decomposition of Solutions of Nearly Singular Systems," SIAM J.
arguments are the same as for SGEDF except:
on entry:

JOB INTEGER
if an approximate null vector is already known, the user
may pass it to SGEII. JOB indicates where to find it.
JOB = 0 : no initial guess
JOB < 0 : approximate left null vector is passed in PSI
JOB > 0 : approximate right null vector is passed in PHI

ITER INTEGER
governs how many iterations are performed
ITER = 0 : continue iterating until PSI and PHI converge
on accurate values. If A is nearly singular
this usually occurs with 2 or 3 iterations.
ITER > 0 : do up to ITER many iterations.

resol = resolution of convergence

REAL RESOL
PARAMETER (RESOL = .0001)
INTEGER LDA,N,IPVT(N),JOB,ITER
REAL A(LDA,N),PSI(N),PHI(N),DELTAP,SILEN,PHILEN

note: since SGESL destroys the rhs given to it, PSI and PHI
here are both computed in PSI, until the last step

IF (JOB .EQ. 0) THEN
  no initial guess; fill PSI with 1's
  DO 10 I = 1,N
      PSI(I) = 1.
   10  CONTINUE
ELSEIF (JOB .EQ. 1) THEN
  initial guess is in PHI;
  move to PSI, then solve for initial PSI
  phi' = phi / ||phi'||
  CALL SCOPY (N,PHI,1,PSI,1)
  PHILEN = SNRM2(N,PSI,1)
  CALL SSCAL (N,1/PHILEN,PSI,1)
C
T
A psi' = phi'
CALL SGESL (A,LDA,N,IPVT,PSI,1)

ENDIF
C
PSI now contains initial guess; normalize it
psi' = psi' / ||psi'||
PSILEN = SNRM2(N,PSI,1)
CALL SSCAL (N,1/PSILEN,PSI,1)

C
................................................... main loop of routine
IINC = 0
IF (ITER .NE. 0) IINC = 1
I = IINC
C
C repeat until convergence
G0 CONTINUE
C
A phi' = psi
CALL SGESL (A,LDA,N,IPVT,PSI,0)
C
phi' = phi' / ||phi'||
PHILEN = SNRM2(N,PSI,1)
CALL SSCAL (N,1/PHILEN,PSI,1)
C
T
A psi' = phi'
CALL SGESL (A,LDA,N,IPVT,PSI,1)
C
psi' = psi' / ||psi'||
PSILEN = SNRM2(N,PSI,1)
CALL SSCAL (N,1/PSILEN,PSI,1)
C
increment counter
I = I + IINC
C
end
IF (I .LE. ITER .AND. ABS(1/PHILEN - 1/PSILEN) .GT. RESOL)
* GOTO 50
C
do phi' once more
CALL SCOPY (N,PSI,1,PHI,1)
CALL SGESL (A,LDA,N,IPVT,PHI,0)
C
DELTA = 1/||phi'||
C
DELTA gets a sign such that PSI(1) and PHI(1) have the same sign
C when A is symmetric, PSI = PHI, and DELTA is smallest eigenvalue
DELTA = SIGN(1/SNRM2(N,PHI,1),PSI(1)*PHI(1))
CALL SSCAL (N,DELTA,PHI,1)

C SUBROUTINE SGEBE
*     (N,M, A,LDA, IPVT, B,LDB, CT,LDC ,D,LDD, F,G, X,Y,
*      WORK1,LDW1, WORK2,LDW2, JOB)
C
C the ordinary (undeflated) block elimination algorithm
C
C all arguments are the same as in SGEDBE.
C
INTEGER N,M, LDA,LDB,LDC,LDD,LDW1,LDW2, IPVT(N), AJOB
REAL A(LDA,N), B(LDB,M), CT(LDC,N), D(LDD,M), F(N), G(M), X(N), Y(M)
REAL WORK1(LDW1,M), WORK2(LDW2,M), DELTA
CHARACTER*6 JOB
INTEGER MP1
LOGICAL NEWA,NEWB,NEWC,NEWD,NEWF,NEWG
MP1 = M + 1
C
AJOB = 0
IF (INDEX(JOB,'A') .NE. 0) AJOB = 2
IF (INDEX(JOB,'a') .NE. 0) AJOB = 1
NEWA = (AJOB .NE. 2)
NEWB = (INDEX(JOB,'B') .EQ. 0)
NEWC = (INDEX(JOB,'C') .EQ. 0)
NEWD = (INDEX(JOB,'D') .EQ. 0)
NEWF = (INDEX(JOB,'F') .EQ. 0)
NEWG = (INDEX(JOB,'G') .EQ. 0)
C
solve A V = B for V
IF (AJOB .EQ. 0) CALL SGFA (A,LDA,N,IPVT,INFO)
IF (NEWA .OR. NEWB) THEN
   DO 10 I = 1,M
      CALL SCOPY (N, B(I,I),1, WORK1(I,I),1)
      CALL SGESL (A,LDA,N,IPVT,WORK1(I,I),0)
   10 CONTINUE
ENDIF
C
solve A w = f for w
IF (NEWA .OR. NEWF) THEN
   CALL SCOPY (N, F,1, WORK1(MP1,1),1)
   CALL SGESL (A,LDA,N,IPVT,WORK1(MP1,0),0)
C
C compute E (= D - cT V)
IF (NEWA .OR. NEWB .OR. NEWC .OR. NEWD) THEN
   DO 30 I = 1,M
      DO 20 J = 1,M
         WORK2(I,J) = D(I,J) - SDOT(N, CT(I,1),LDC, WORK1(I,J),1)
      20 CONTINUE
   30 CONTINUE
   CALL SGEFA (WORK2,LDW2,M,IPVT(N+1),INFO)
C
C compute g' (= g - cT w)
IF (NEWA .OR. NEWC .OR. NEWF .OR. NEWG) THEN
   DO 40 I = 1,M
      WORK2(I,MP1) = G(I) - SDOT(N, CT(I,1),LDC, WORK1(1,MP1),1)
   40 CONTINUE
C
C solve for y
   CALL SCOPY (M, WORK2(1,MP1),1, Y,1)
   CALL SGESL (WORK2,LDW2,M,IPVT(N+1),Y,0)
C
C compute x
   DO 50 I = 1,N
      X(I) = WORK1(I,MP1) - SDOT(M, WORK1(I,1),LDW1, Y,1)
   50 CONTINUE
END
The routines in this package implement the deflated block-
 elimination algorithm for solving systems of the form:
\[
\begin{bmatrix}
  A & B \\
  M & 0
\end{bmatrix}
\begin{bmatrix}
  x \\
  y
\end{bmatrix}
= \begin{bmatrix}
  f \\
  g
\end{bmatrix}
\]

discussed in T. F. Chan and D. Resasco, "Generalized Deflated
Block-elimination", Technical Report YALEU/DCS/RR-337, Dept. of

This set of routines calls LINPACK's SGE- and SGB- routines, and
the SBLAs. Implemented by Thomas A. Grossi, Yale University, 1985.

SUBROUTINE SGBDBE
*(
  N,M,A,LDA,ML,IPVT,LDB,CT,LDC,DL,D,LDL,F,G,
  X,Y,WORK1,LDWORK1,WORK2,LDWORK2,JOB)
)*

the deflated block elimination algorithm
arguments:
on entry:

N   INTEGER
the order of the matrix A

M   INTEGER
the order of the borders to A in M

A   REAL(LDA,N)
the matrix to be factored.

LDA   INTEGER
the leading dimension of the array A. LDA >= N.

ML   INTEGER
number of diagonals below the main diagonal in A.
0 <= ML <= N.

MU INTEGER
number of diagonals above the main diagonal in A.
0 <= MU <= N.
more efficient if ML <= MU.

IPVT INTEGER(N+M)
an integer vector of pivot indices. The last M spaces are
required for working space

B REAL(LDB,M)
right-side border to matrix A in matrix M

LDB INTEGER
the leading dimension of the array B. LDB >= N.

CT REAL(LDC,N)
right and bottom borders to matrix A in matrix M

LDC INTEGER
the leading dimension of the array CT. LDC >= M.

D REAL(LDD,M)
lower right-hand entries of M.

LDD INTEGER
the leading dimension of the array D. LDD >= M.

F REAL(N)
G REAL(M)
right-hand side to solve with

WORK1 REAL(LDW1,M+4) LDW1 >= N
used to hold Vd, Wd, psi T B, psi and phi

LDW1 INTEGER
the leading dimension of the array WORK1. LDW1 >= N.

WORK2 REAL(LDW2,M+3)
used to hold E, g', psi T f and delta

LDW2 INTEGER
the leading dimension of the array WORK2. LDW2 >= M+1.

JOB CHARACTER*6
indicates which inputs are the same as in the last call
to SGDBDE. If there was no such call, set JOB =
' ' or 'a ' (see below). Otherwise, JOB contains
as many of the following apply:
'A' if A stays the same
'S' if A is new but already factored by SQBCO or SGBFA
'B' if B stays the same
'C' if CT stays the same
'D' if D stays the same
'F' if F stays the same
'G' if G stays the same

on exit:

A  real(lda,n)
contains an upper triangular matrix and
the multipliers which were used to obtain it. The
factorization can be written A = L*U, where L is the
product of permutation and unit lower triangular ma-
trices and U is upper triangular.

IPVT   INTEGER(n+m)
an integer vector of pivot indices. The last m spaces are
required for working space.

X   REAL(n)
Y   REAL(m+1)
solution vector

WORK1   REAL(ldw1,m+4)
used to hold Vd, Wd, psiT B, psi and phi

WORK2   REAL(ldw2,m+3)
used to hold E, g', psiT f and delta

Savings on storage:
the following pairs of inputs may be equivalent:
(X,F)   (Y,G)   (B,WORK1)   (D,WORK2)
In general, if equivalent storage is used, then a change in any
of the inputs in either of the groups (A,B,C,D) or (F,G)
requires that the entire group be re-entered. Specific
exceptions to this rule can be determined by examining
the algorithm.

INTEGER N,M, LDA,ML,MU,LDB,LDC,LDD,LDW1,LDW2, IPVT(N), AJOB
CHARACTER*6 JOB
REAL A(lda,n), B(ldb,m),CT(ldc,n),D(ldd,m),DELTAD
REAL F(n),G(m), X(n),Y(m), WORK1(ldw1,m),WORK2(ldw2,m)
LOGICAL NEWA,NEWB,NEWC,NEWD,NEWF,NEWG

the following constants are used to partition WORK1 and WORK2
into their various vectors; MPI stands for the "extra" row and
column added to D in forming E. WORK1 is primarily used for Vd, and WORK2 for E

INTEGER MP1,CB,WD,CF,PSI,PHI,GP,ALPHA
MP1 = M + 1
CB = MP1
WD = CB + 1
CF = MP1
PSI = WD + 1
PHI = PSI + 1
GP = MP1 + 1
ALPHA = MP1

AJOB = 0
IF (INDEX(JOB,'A') .NE. 0) AJOB = 2
IF (INDEX(JOB,'S') .NE. 0) AJOB = 1
NEWA = (AJOB .NE. 2) .
NEWB = (INDEX(JOB,'B') .EQ. 0)
NEWC = (INDEX(JOB,'C') .EQ. 0)
NEWD = (INDEX(JOB,'D') .EQ. 0)
NEWF = (INDEX(JOB,'F') .EQ. 0)
NEWG = (INDEX(JOB,'G') .EQ. 0)

Algorithm:

factor A, compute psi, phi, delta
compute deflated solution to A V = B
compute deflated solution to A w = f
build E: | (D - cT Vd) (cT phi) |
    | CbT       delta |
build g': | g - cT Wd |
    | Cf |
solve E | y | = g' for y
    | alpha |
x = Wd - Vd y + alpha phi

if AJOB = 0 or 1, or B is new, we start by solving A Vd = B;
this may imply factoring A, and/or computing psi, phi and delta
IF (NEWA .OR. NEWB) THEN

for the first element of Vd, AJOB will tell sgeDF what to do
CALL SGBDF (A, LDA, N, ML, MU, IPVT, B(1,1), WORK1(1,PSI),
            WORK1(1,PHI), DELTA, WORK1(1,CB),AJOB)

compute remaining columns of Vd using results of first call
IF (M .GT. 1) THEN
  DO 10 I = 2,M
    CALL SGBDF (A, LDA, N, ML, MU, IPVT, B(1,I), WORK1(1,PSI),
                WORK1(1,PHI), DELTA, WORK1(1,CB),2)
CONTINUE
ENDIF
ENDIF

We must recompute Wd and Cf if A or F have changed
IF (NEWA .OR. NEWF) THEN
    CALL SGBDF (A, LDA, N, ML, MU, IPVT, F, WORK1(1, PSI), WORK1(1, PHI),
               * DELTA, WORK1(1, WD), WORK2(CF, GP), 2)
ENDIF

build and factor E
IF (NEWA .OR. NEWB .OR. NEWC .OR. NEWD) THEN
    CALL SCOPY (M, WORK1(1, CB), 1, WORK2(MP1, 1), LDW2)
    WORK2(MP1, MP1) = DELTA
    DO 30 I = 1, M

compute D - cT Vd, column by column
DO 20 J = 1, M
    WORK2(I, J) = D(I, J) - SDOT(N, CT(I, 1), LDC, WORK1(1, J), 1)
20 CONTINUE

compute cT PHI element by element
    WORK2(I, MP1) = SDOT(N, CT(I, 1), LDC, WORK1(1, PHI), 1)
30 CONTINUE

factor E
    CALL SGEFA (WORK2, LDW2, MP1, IPVT(N+1), INFO)
ENDIF

g' depends on a lot of things
IF (NEWA .OR. NEWC .OR. NEWF .OR. NEWG) THEN
    DO 40 I = 1, M
        WORK2(I, GP) = G(I) - SDOT(N, CT(I, 1), LDC, WORK1(1, WD), 1)
40 CONTINUE
ENDIF

compute x and y
    CALL SCOPY (MP1, WORK2(1, GP), 1, Y, 1)
    CALL SGESL (WORK2, LDW2, MP1, IPVT(N+1), Y, 0)
    DO 50 I = 1, N
        X(I) = WORK1(I, WD) - SDOT(M, WORK1(I, 1), LDW1, Y, 1)
50 CONTINUE
    CALL SAXPY (N, Y(ALPHA), WORK1(1, PHI), 1, X, 1)
    WORK2(1, GP+1) = DELTA
END
SUBROUTINE SGDBF

* (A, LDA, N, ML, MU, IPVT, P, PSI, PHI, DELTA, ZD, CP, JOB)

computes the deflated decomposition of A z = p, returning
solution in the form:

\[ z = z + \phi (c / \delta) \]
\[ d \quad p \]

arguments are the same as for SGDBBE except:

on entry:

P REAL(N)
contains rhs to system of equations

PSI REAL(N)

PHI REAL(N)
left and right null vectors to matrix A
(only on entry if JOB >= 2)

DELTA REAL
smallest singular value for matrix A
(only on entry if JOB >= 2)

JOB INTEGER
JOB = 0 : start the deflation algorithm from scratch; i.e.,
it factors the matrix, perform inverse iteration to
determine PSI, PHI and DELTA, and the computes the
deflated solution.
JOB = 1 : assume that A has already been factored by SGBCO
or SGBFA. (or by a previous call to SGB[DBE])
and continue from there.
JOB > 1 : additionally, PSI, PHI and DELTA have already
been computed.

on exit:

PSI REAL(N)

PHI REAL(N)
left and right null vectors to matrix A

DELTA REAL
smallest singular value for matrix A

ZD REAL(N)
deflated solution to system A z = p
Note that P and ZD may be the same vector

CP REAL
phiT p.

INTEGER LDA,N,IPVT(N),JOB,IJOB, INFO
REAL A(LDA,N),P(N),PSI(N),PHI(N),DELTA,ZD(N),CP
REAL PSITP
LOGICAL TRANS

IJOB = JOB
TRANS = (IJOB .GE. 10)
IF (TRANS) IJOB = IJOB - 10

IF (IJOB .EQ. 0)
* CALL SGBFA (A,LDA,N,ML,MU,IPVT,INFO)
IF (IJOB .LE. 1)
* CALL SGBII (A,LDA,N,ML,MU,IPVT,PSI,PHI,DELTA,0,3)

IF (TRANS) GOTO 20

A Zd = p - (psiT p) psi ; solve for Zd ; Cp is approx (psiT p)
CP = SDOT (N,P,1,PSI,1)
CALL SCOPY (N,P,1,ZD,1)
CALL SAXPY (N,-CP,PSI,1,ZD,1)
CALL SGBSL (A,LDA,N,ML,MU,IPVT,ZD,0)

orthogonalize Zd with respect to phi
CALL SAXPY(N,-SDOT(N,PHI,1,ZD,1),PHI,1,ZD,1)
GOTO 30

CONTINUE 20

Perform deflation with A

T

A Zd = p - (phiT p) phi ; solve for Zd ; Cp == (phiT p)
CP = SDOT (N,P,1,PHI,1)
CALL SCOPY (N,P,1,ZD,1)
CALL SAXPY (N,-CP,PHI,1,ZD,1)"
CALL SGBSL (A,LDA,N,ML,MU,IPVT,ZD,0)

orthogonalize Zd wrt psi
CALL SAXPY(N,-SDOT(N,PSI,1,ZD,1),PSI,1,ZD,1)

CONTINUE 30
END
SUBROUTINE SGBII (A,LDA,N,ML,MU,IPVT,PSI,PHI,DELTA,JOBI,ITER)
computes approximate left and right null vectors of A by applying
the inverse iteration algorithm described in T. F. Chan. "Deflated
Decomposition of Solutions of Nearly Singular Systems," SIAM J.
arguments are the same as for SGBDF except:
on entry:
JOB INTEGER
if an approximate null vector is already known, the user
may pass it to SGBII. JOB indicates where to find it.
JOB = 0 : no initial guess
JOB < 0 : approximate left null vector is passed in PSI
JOB > 0 : approximate right null vector is passed in PHI
ITER INTEGER
governs how many iterations are performed
ITER = 0 : continue iterating until PSI and PHI converge
        on accurate values. If M is nearly singular
        this usually occurs with 2 or 3 iterations.
ITER > 0 : do up to ITER many iterations.
RESOL = resolution of convergence
REAL RESOL
PARAMETER (RESOL = .0001)
INTEGER LDA,N,IPVT(LDA,2),JOBI,ITER
REAL A(LDA,LDA),PSI(N),PHI(N),DELTA,PSILEN,PHILEN
note: since SGBSL destroys the rhs given to it, PSI and PHI
here are both computed in PSI, until the last step
IF (JOBI.EQ. 0) THEN
no initial guess; fill PSI with 1's
DO 10 I = 1,N
    PSI(I) = 1.
  10 CONTINUE
ELSEIF (JOBI.EQ. 1) THEN
initial guess is in PHI;
move to PSI, then solve for initial PSI
phi' = phi' / ||phi'||
CALL SCOPY (N,PHI,1,PSI,1)
PHILEN = SNRM2(N,PSI,1)
CALL SCAL (N,1/PHILEN,PSI,1)

T
A psi' = phi'
CALL SGBSL (A,LDA,N,ML,MU,IPVT,PSI,1)

ENDIF

PSI now contains initial guess; normalize it
psi' = psi' / ||psi'||
PSILEN = SNRM2(N,PSI,1)
CALL SCAL (N,1/PSILEN,PSI,1)

C............................... main loop of routine
IINC = 0
IF (ITER .NE. 0) IINC = 1
I = IINC

C repeat until convergence
50 CONTINUE
A phi' = psi
CALL SGBSL (A,LDA,N,ML,MU,IPVT,PSI,0)
phi' = phi' / ||phi'||
PHILEN = SNRM2(N,PSI,1)
CALL SCAL (N,1/PHILEN,PSI,1)

C
T
A psi' = phi'
CALL SGBSL (A,LDA,N,ML,MU,IPVT,PSI,1)
psi' = psi' / ||psi'||
PSILEN = SNRM2(N,PSI,1)
CALL SCAL (N,1/PSILEN,PSI,1)
C increment counter
I = I + IINC

C end
IF (I .LE. ITER .AND. ABS(1/PHILEN - 1/PSILEN) .GT. RESOL) *
* GOTO 50
C do phi' once more -- this time for the record
CALL SCOPY (N,PSI,1,PHI,1)
CALL SGBSL (A,LDA,N,ML,MU,IPVT,PHI,0)
C DELTA = 1/||phi'||

35
DELTA gets a sign such that PSI(1) and PHI(1) have the same sign
when A is symmetric, PSI = PHI, and DELTA is smallest eigenvalue
DELTA = SIGN(1/SNRM2(N,PHI,1),PSI(1)*PHI(1))
CALL SSCAL (N,DELTA,PHI,1)

END

SUBROUTINE SGBBE
* (N,M, A,LDA,ML, MU, IPVT, B,LDB, CT,LC, O,LDD, F,G,
* X,Y, WORK1,LDW1, WORK2,LDW2, JOB)

the ordinary (deflated) block elimination algorithm
all arguments are the same as in SGBDBE.

INTEGER N,M, LDA,ML,MU,LDB,LC, O,LDD,LDW1,LDW2, IPVT(N), AJOB
CHARACTER*6 JOB
REAL A(LDA,N), B(LDB,M), CT(LC,N), O(LDD,M), DELTA
REAL F(N), O(M), X(N), Y(M), WORK1(LDW1), WORK2(LDW2)
INTEGER MP1
LOGICAL NEWA,NEWB,NEWC,NEWD,NEWF,NEWG
MP1 = M + 1

AJOB = 0
IF (INDEX(JOB,'A') .NE. 0) AJOB = 2
IF (INDEX(JOB,'a') .NE. 0) AJOB = 1
NEWA = (AJOB .NE. 2)
NEWB = (INDEX(JOB,'B') .EQ. 0)
NEWC = (INDEX(JOB,'C') .EQ. 0)
NEWD = (INDEX(JOB,'D') .EQ. 0)
NEWF = (INDEX(JOB,'F') .EQ. 0)
NEWG = (INDEX(JOB,'G') .EQ. 0)

solve A V = B for V
IF (AJOB .EQ. 0) CALL SGBFA (A,LDA,N,ML,MU,IPVT,INFO)
IF (NEWA .OR. NEWB) THEN
DO 10 I = 1,M
   CALL SCOPY (N, B(I,1),1, WORK1(1,1),1)
   CALL SGBSL (A,LDA,N,ML,MU,IPVT,WORK1(1,1),0)
10 CONTINUE
ENDIF

solve A w = f for w
IF (NEWA .OR. NEWF) THEN
CALL SCOPY (N, F, I, WORK1(I, MP1), 1)
CALL SGBSL (A, LDA, N, ML, MU, IPVT, WORK1(I, MP1), 0)
ENDIF

C
C compute E (= D - cTV)
IF (NEWA .OR. NEWB .OR. NEWC .OR. NEWD) THEN
DO 30 I = 1, M
    DO 20 J = 1, M
        WORK2(I, J) = D(I, J) - SDOT(N, CT(I, 1), LDC, WORK1(I, J), 1)
    CONTINUE
  20 CONTINUE
  30 CONTINUE
CALL SGEFA (WORK2, LDW2, M, IPVT(N+1), INFO)
ENDIF

C
C compute g' (= g - cTw)
IF (NEWA .OR. NEWC .OR. NEWF .OR. NEWG) THEN
DO 40 I = 1, M
    WORK2(I, MP1) = G(I) - SDOT(N, CT(I, 1), LDC, WORK1(I, MP1), 1)
  40 CONTINUE
ENDIF

C
C solve for y
CALL SCOPY (M, WORK2(I, MP1), 1, Y, 1)
CALL SGESL (WORK2, LDW2, M, IPVT(N+1), Y, 0)

C
C compute x
DO 50 I = 1, N
    X(I) = WORK1(I, MP1) - SDOT(M, WORK1(I, 1), LDW1, Y, 1)
  50 CONTINUE
END
The routines in this package implement the deflated block-
-elimination algorithm for solving systems of the form:

\[
\begin{bmatrix}
x & A & B & x \\
M & I & = & I \\
y & CT & D & y \\
g & & & 1
\end{bmatrix}
\]

discussed in T. F. Chan and D. Resasco, "Generalized Deflated
Block-Elimination", Technical Report YALEU/DCS/RR-337, Dept. of

This set of routines calls LINPACK's SGE- and SGT- routines,
and the SBLAs. Implemented by Thomas A. Grossi, Yale University, 1985.

---

SUBROUTINE SGDBE

* (N,M, L,DI,U, B,LDB, CT,LDCL, D,LDD, F,G, X,Y, 
  WORK1,LDW1, WORK2,LDW2, JOB)

the deflated block elimination algorithm

arguments:

on entry:

N INTEGER
the order of the matrix A

M INTEGER
the order of the borders to A in M

L REAL(N)
  is the subdiagonal of the tridiagonal matrix.
  L(2) through L(n) should contain the subdiagonal.
  Unlike LINPACK, our routines do not change the
  contents of L.

DI REAL(N)
  is the diagonal of the tridiagonal matrix.
The contents of DI remain unchanged by our routines.

U

REAL(N)
is the superdiagonal of the tridiagonal matrix.
U(1) through U(n-1) should contain the superdiagonal.
The contents of U remain unchanged by our routines.

B

REAL(LDB,M)
right-hand border to the matrix A in matrix M

LDB

INTEGER
the leading dimensions of the array B. LDB >= N.

CT

REAL(LDC,N)
bottom border to matrix A in matrix M

LDC

INTEGER
the leading dimension of the array CT. LDC >= M.

D

REAL(LDD,M)
lower right-hand entries of M.

LDD

INTEGER
the leading dimension of the array D. LDD >= M.

F

REAL(N)

G

REAL(M)
right-hand side to solve with

WORK1

REAL(LDW1,M+7)
used to hold Vd, Wd, psiT B, psi and phi, plus scratch space
for factoring A

LDW1

INTEGER
the leading dimension of the array WORK1. LDW1 >= N.

WORK2

REAL(LDW2,M+3)
used to hold E, g', psiT f and delta, and a list of pivot
indices.

LDW2

INTEGER
the leading dimension of the array WORK2. LDW2 >= M+1.

JOB

CHARACTER*6
indicates which inputs are the same as in the last call
to SGTDRE. If there was no such call, set JOB = 
' ' ' or 'a ' (see below). Otherwise, JOB contains
as many of the following apply:
'A' if A stays the same
'S' if A is new but already factored by SGTCO or SGTFI
'B' if B stays the same
'C' if CT stays the same
'D' if D stays the same
'E' if F stays the same
'G' if G stays the same

on exit:

X    REAL(N)
Y    REAL(M+1)
solution vector

WORK1  REAL(LDW1,M+7)
used to hold Vd, Wd, psi, B, psi and phi, plus scratch space
for factoring A

WORK2  REAL(LDW2,M+3)
used to hold E, g', psi, f and delta

Savings on storage:
the following pairs of inputs may be equivalent:
(X,F)    (Y,G)    (B,WORK1)    (D,WORK2)
In general, if equivalent storage is used, then a change in any
of the inputs in either of the groups (A,B,C,D) or (F,G)
requires that the entire group be re-entered. Specific
exceptions to this rule can be determined by examining
the algorithm.

INTEGER N,M, LDA,LDB,LDC,LDD,LDW1,LDW2, AJOB
CHARACTER*6 JOB
REAL L(N),DI(N),U(N), B(LDB,M),CT(LDC,N),D(LDD,M),DELTA
REAL F(N),G(M), X(N),Y(M), WORK1(LDW1,M),WORK2(LDW2,M)
LOGICAL NEWA,NEWB,NEWC,NEWD,NEWF,NEWG

the following constants are used to partition WORK1 and WORK2
into their various vectors; MP1 stands for the "extra" row and
column added to D in forming E. WORK1 is primarily used for Vd,
and WORK2 for E
INTEGER MP1,CB,WD,CF,PSI,PHI,SAVE,IPVT,GP,ALPHA
MP1 = M + 1
CB = Mp1
WD = CB + 1
CF = MP1
PSI = WD + 1
PHI = PSI + 1
SAVE = PHI + 1
GP = MP1 + 1
IPVT = GP + 1
\[ \text{ALPHA} = \text{MP1} \]

\[ \text{AJOB} = 0 \]
\[ \text{IF} (\text{INDEX}(\text{JOB},'A') \neq 0) \text{ AJOB} = 2 \]
\[ \text{IF} (\text{INDEX}(\text{JOB},'S') \neq 0) \text{ AJOB} = 1 \]
\[ \text{NEWA} = (\text{AJOB} \neq 2) \]
\[ \text{NEWB} = (\text{INDEX}(\text{JOB},'B') \neq 0) \]
\[ \text{NEWC} = (\text{INDEX}(\text{JOB},'C') \neq 0) \]
\[ \text{NEWD} = (\text{INDEX}(\text{JOB},'D') \neq 0) \]
\[ \text{NEWF} = (\text{INDEX}(\text{JOB},'F') \neq 0) \]
\[ \text{NEWG} = (\text{INDEX}(\text{JOB},'G') \neq 0) \]

C

Algorithm:

factor A, compute psi, phi, delta
compute deflated solution to A V = B
compute deflated solution to A w = f
build E: | (D - cT Vd) (cT phi) |
         | CtB     delta |
build g': | g - cT Wd |
         | Cf      |
solve E | y     | = g' for y
         | alpha |
x = Wd - Vd y + alpha phi

if AJOB = 0 or 1, or B is new, we start by solving A Vd = B;
this may imply factoring A, and/or computing psi, phi and delta
IF (NEWA .OR. NEWB) THEN
for the first element of Vd, AJOB will tell sgeDF what to do
CALL SGTDF (N, L,DI,U,WORK1(1,SAVE), B(1,1), WORK1(1,PSI),
             WORK1(1,PHI),DELTA, WORK1(1,1),WORK1(1,CB),AJOB)
compute remaining columns of Vd using results of first call
IF (M .GT. 1) THEN
DO 10 I = 2,M
     CALL SGTDF(N, L,DI,U,WORK1(1,SAVE), B(1,I),WORK1(1,PSI),
                WORK1(1,PHI), DELTA, WORK1(1,I),WORK1(1,CB),2)
10    CONTINUE
ENDIF
ENDIF

We must recompute Wd and Cf if A or F have changed
IF (NEWA .OR. NEWF)
* CALL SGTDF (N, L,DI,U,WORK1(1,SAVE), F, WORK1(1,PSI),
*              WORK1(1,PHI),DELTA, WORK1(1,Wd),WORK2(CF,GP),2)
BUILD and factor E
IF (NEWA .OR. NEWB .OR. NEWC .OR. NEWD) THEN
CALL SCOPY (M, WORK1(1,CD),1, WORK2(MP1,1),LDW2)
The routines in this package implement the deflated block-
 elimination algorithm for solving systems of the form:

\[ \begin{bmatrix} x \end{bmatrix} = \begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} x \end{bmatrix} + \begin{bmatrix} f \end{bmatrix} \]

\[ M \begin{bmatrix} y \end{bmatrix} = \begin{bmatrix} CT & D \end{bmatrix} \begin{bmatrix} y \end{bmatrix} + \begin{bmatrix} g \end{bmatrix} \]

discussed in T. F. Chan and D. Resasco, "Generalized Deflated
Block-Elimination", Technical Report YALEU/DCS/RR-337, Dept. of

This set of routines calls Y SMP's general routines with non-
compressed storage, Linpack's SG E- routines, and the SBLAs.
Implemented by Thomas A. Grossi, Yale University, 1985.

STORAGE SCHEME FOR THESE ROUTINES

The nonzero entries of the coefficient matrix \( M \) are stored
row-by-row in the array \( A \). To identify the individual nonzero
entries in each row, we need to know in which column each entry
lies. The column indices which correspond to the nonzero entries
of \( M \) are stored in the array \( JA \); i.e., if \( A(K) = M(I,J) \), then
\( JA(K) = J \). In addition, we need to know where each row starts and
how long it is. The index positions in \( JA \) and \( A \) where the rows of
\( M \) begin are stored in the array \( IA \); i.e., if \( M(I,J) \) is the first
nonzero entry (stored) in the \( I \)-th row and \( A(K) = M(I,J) \), then
\( IA(K) = K \). Moreover, the index in \( JA \) and \( A \) of the first location
following the last element in the last row is stored in \( IA(N+1) \).
Thus, the number of entries in the \( I \)-th row is given by
\[ IA(I+1) - IA(I) \], the nonzero entries of the \( I \)-th row are stored
consecutively in
\[ A(IA(I)), A(IA(I)+1), \ldots, A(IA(I)+1) \],
and the corresponding column indices are stored consecutively in
\[ JA(IA(I)), JA(IA(I)+1), \ldots, JA(IA(I)+1) \].
For example, the 5 by 5 matrix
\[
\begin{bmatrix}
0.1 & 0.2 & 0.0 & 0.0 \\
0.0 & 0.3 & 0.0 & 0.0 \\
0.0 & 0.4 & 0.5 & 0.6 \\
0.0 & 0.0 & 0.0 & 0.7 \\
0.0 & 0.0 & 0.8 & 0.9 \\
\end{bmatrix}
\]
would be stored as
<table>
<thead>
<tr>
<th>1 2 3 4 5 6 7 8 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA</td>
</tr>
<tr>
<td>JA</td>
</tr>
<tr>
<td>A</td>
</tr>
</tbody>
</table>

N INTEGER
number of variables/equations.

A INTEGER(*)
nonzero entries of the coefficient matrix A, stored
by rows.
size = number of nonzero entries in A.

IA INTEGER(N+1)
pointers to delimit the rows in A.

JA INTEGER(*)
column numbers corresponding to the elements of A.
size = size of A.

The rows and columns of the original matrix A can be
reordered (e.g., to reduce fill-in or ensure numerical stability)
before calling the driver. If no reordering is done, then set
R(I) = C(I) = IC(I) = I FOR I=1,...,N. The solution A is
returned in the original order.

R INTEGER(N)
ordering of the rows of A.

C INTEGER(N)
ordering of the columns of A.

IC INTEGER(N)
inverse of the ordering of the columns of A; i.e.,
IC(C(I)) = I for I=1,...,n.

Working storage is needed for the factored form of the matrix
A plus various temporary vectors. The arrays ISP and RSP should
be equivalenced; integer storage is allocated from the beginning
of ISP and real storage from the end of RSP.

NSP INTEGER
declared dimension of RSP;
the exact value of NSP will be specified below

ISP INTEGER(*)
integer working storage divided up into various arrays
needed by the subroutines; ISP and RSP should be
equivalenced.

size = LRATIO*NSP, where LRATIO = size of storage for a
real number divided by the size of storage for an integer.

RSP    REAL(NSP)
real working storage divided up into various arrays
needed by the subroutines; ISP and RSP should be
equivalenced.

ESP    INTEGER
if sufficient storage was available to perform the
symbolic factorization (NSFC), then ESP is set to the
amount of excess storage provided (negative if
insufficient storage was available to perform the
numeric factorization (NNFC)).
if ESP > 2*N, then those last 2*n position of RSP will
contain approximate left and right null vectors for A.

SUBROUTINE SYNODE
* (N,M, R,C,IC, IA,JA,A, NSP,ISP,RSP,ESP, B,LDB,
  * CT, LDC, D,LDD, F,G, X,Y, WORK1,LDW1, WORK2,LDW2, JOB)

the deflated block elimination algorithm

arguments:
on entry:
 N     INTEGER
  the order of the matrix A
 M     INTEGER
  the order of the borders to A in M
 R     INTEGER(N)
  ordering of the rows of A.
 C     INTEGER(N)
  ordering of the columns of A.
 IC    INTEGER(N)
  inverse of the ordering of the columns of A; i.e.,
  IC(C(I)) = I  for I=1,...,n.
IA  INTEGER(N+1)
pointers to delimit the rows in A.

JA  INTEGER(*)
column numbers corresponding to the elements of A.
    size = size of A.

A   INTEGER(*)
nonzero entries of the coefficient matrix A, stored
    by rows.
    size = number of nonzero entries in A.

NSP  INTEGER
declared dimension of RSP;
    the exact value of NSP will be specified below.

ISP  INTEGER(*)
integer working storage divided up into various arrays
    needed by the subroutines; ISP and RSP should be
    equivalenced.
    size = LRATIO*NSP, where LRATIO = size of storage for a
    real number divided by the size of storage for an integer.

RSP  REAL(NSP)
real working storage divided up into various arrays
    needed by the subroutines; ISP and RSP should be
    equivalenced.

B   REAL(LDB,M)
    right-hand border to matrix A in matrix M.

LDB  INTEGER
    the leading dimension of the array B.  LDB >= N.

CT  REAL(LDC,N)
    bottom border to matrix A in matrix M

LDC  INTEGER
    the leading dimension of the array CT.  LDC >= M.

D   REAL(LDD,M)
    lower right-hand entries of M.

LDD  INTEGER
    the leading dimension of the array D.  LDD >= M.

F   REAL(N)
    right-hand side to solve with
WORK1 REAL(LDW1,M+4) LDW1 >= N
used to hold Vd, Wd, psi T B, psi and phi.

LDW1 INTEGER
the leading dimension of the array WORK1. LDW1 >= N.

WORK2 REAL(LDW2,M+4) LDW2 >= M+1
used to hold E, g', psi T f and delta, and pivot indices for E.

LDW2 INTEGER
the leading dimension of the array WORK2. LDW2 >= M+1.

JOB CHARACTER*6
indicates which inputs are the same as in the last call
to SGEDBE. If there was no such call, set JOB =
' ' or 'a ' (see below). Otherwise, JOB contains
as many of the following apply:
'A' if A stays the same
'S' if A is new but already factored by SGECD or SGEFA
'B' if B stays the same
'C' if CT stays the same
'D' if D stays the same
'F' if F stays the same
'G' if G stays the same

on exit:

RSP REAL(NSP)
the last 2n positions of RSP contain approximate
left and right null vectors for A if ESP > 2*N.

ESP INTEGER
if sufficient storage was available to perform the
symbolic factorization (NSFC), then ESP is set to the
amount of excess storage provided (negative if
insufficient storage was available to perform the
numeric factorization (NNFC)).
if ESP > 2*N, then those last 2n position of RSP will
contain approximate left and right null vectors for A.

X REAL(N)
Y REAL(M+1)
solution vector.

WORK1 REAL(LDW1,M+4) LDW1 >= N
used to hold Vd, Wd, psi T B, psi and phi.
WORK2  REAL(LDW2,M+4)  LDW2 >= M+1
used to hold E, g', psiT f and delta, and pivot indices for E

Savings on storage:
the following pairs of inputs may be equivalent:
  (X,F)  (Y,G)  (B,WORK1)  (D,WORK2)
in general if equivalent storage is used, then a change in one
of the inputs in either the left-hand-side group or the right-
hand-side group requires that the entire group be re-entered.
Specific exceptions to this rule can be determined by examining
the algorithm.

INTEGER N,M, LDB,LDC,LDD,LDW1,LDW2, AJOB, LRATIO
INTEGER R(N),C(N),IC(N), IA(N),JA(*), NSP,ISP(NSP),ESP
REAL  A(*),RSP(NSP)
REAL  B(LDB,M),CT(LDC,N),D(LDD,M), F(N),G(M), X(N),Y(M)
REAL  WORK1(LDW1,M),WORK2(LDW2,M), DELTA
CHARACTER*8 JOB
LOGICAL NEWA,NEWB,NEWC,NEWD,NEWF,NEWG

the following constants are used to partition WORK1 and WORK2
into their various vectors; MPI stands for the 'extra' row and
column added to D in forming E. WORK1 is primarily used for Vd,
and WORK2 for E
INTEGER MPI,CB,WD,CF,PSI,PHI,GP,IPVT,ALPHA
DATA LRATIO /1/
MPI = M + 1
CB = MPI
WD = CB + 1
CF = MPI
PSI = WD + 1
PHI = PSI + 1
GP = MPI + 1
IPVT = GP + 1
ALPHA = MPI

AJOB = 0
IF (INDEX(JOB,'A') .NE. 0) AJOB = 2
IF (INDEX(JOB,'S') .NE. 0) AJOB = 1
NEWA = (AJOB .NE. 2)
NEWB = (INDEX(JOB,'B') .EQ. 0)
NEWC = (INDEX(JOB,'C') .EQ. 0)
NEWD = (INDEX(JOB,'D') .EQ. 0)
NEWF = (INDEX(JOB,'F') .EQ. 0)
NEWG = (INDEX(JOB,'G') .EQ. 0)

Algorithm:
factor A, compute psi, phi, delta

48
compute deflated solution to A V = B
compute deflated solution to A w = f
build E: | (D - cT Vd) (cT phi) |
| CbT | delta |
build g': | g - cT Wd |
cf |
solve E | y | = g' for y
| alpha |
x = Wd - Vd y + alpha phi

if AJOB = 0 or 1, or B is new, we start by solving A Vd = B;
this may imply factoring A, and/or computing psi, phi and delta
IF (NEWA .OR. NEWB) THEN
for the first element of Vd, AJOB will tell sgeDF what to do
CALL SYNDF (N, R,C,IC, IA,JA,A, NSP,ISP,RSP,ESP, B(I,I),
* WORK1(1,PSI),WORK1(1,PHI),DELTA, WORK1(1,1),WORK1(1,CB),AJOB)
compute remaining columns of Vd using results of first call
IF (M .GT. 1) THEN
DO 10 I = 2,M
   CALL SYNDF (N, R,C,IC, IA,JA,A, NSP,ISP,RSP,ESP, B(I,I),
* WORK1(1,PSI),WORK1(1,PHI),DELTA,
* WORK1(1,I),WORK1(I,CB),2)
10   CONTINUE
ENDIF
ENDIF

We must recompute Wd and Cf if A or F have changed
IF (NEWA .OR. NEWF) *
   CALL SYNDF (N, R,C,IC, IA,JA,A, NSP,ISP,RSP,ESP, F,
* WORK1(1,PSI),WORK1(1,PHI),DELTA,
* WORK1(1,WD),WORK2(CF,GP),2)
build and factor E
IF (NEWA .OR. NEWB .OR. NEWC .OR. NEWD) THEN
   CALL SCOPY (M, WORK1(1,CB),1, WORK2(MPI,1),LDW2)
   WORK2(MPI,MP1) = DELTA
   DO 30 I = 1,M
   compute D - cT Vd, column by column
   DO 20 J = 1,M
      WORK2(I,J) = D(I,J) - SDOT(N, CT(I,1),LDC, WORK1(1,J),1)
20   CONTINUE
compute cT PHI element by element
   WORK2(I,MP1) = SDOT(N, CT(I,1),LDC, WORK1(1,PHI),1)
30   CONTINUE
factor E
CALL SGEFA (WORK2, LDW2, MP1, WORK2(1, IPVT), INFO)
ENDIF

g' depends on a lot of things
IF (NEWA .OR. NEWC .OR. NEWF .OR. NEWG) THEN
DO 40 I = 1, N
   WORK2(I, GP) = G(I) - SDOT(N, CT(I, 1), LDC, WORK1(I, WD), 1)
40 CONTINUE
ENDIF

calculate x and y
CALL SCOPY (MP1, WORK2(1, GP), 1, Y, 1)
CALL SGESL (WORK2, LDW2, MP1, WORK2(1, IPVT), Y, 0)
DO 50 I = 1, N
   X(I) = WORK1(I, WD) - SDOT(M, WORK1(I, 1), LDW1, Y, 1)
50 CONTINUE
CALL SAXPY (N, Y(ALPHA), WORK1(1, PHI), 1, X, 1)
WORK2(1, IPVT+1) = DELTA

SUBROUTINE SYND (N, R, C, IC, IA, JA, A, NSP, ISP, RSP, ESP,
*                 P, PSI, PHI, DELTA, ZD, CP, JOB)

computes the deflated decomposition of \( A z = p \), returning
the solution in the form:

\[ z = z + \phi \left( \frac{c}{\delta} \right) \]

arguments are the same as for SYNDBE except:
on entry:
P REAL(N)
contains rhs to system of equations
PSI REAL(N)
PHI REAL(N)
left and right null vectors to matrix A
(only on entry if JOB >= 2)
DELTA REAL
smallest singular value for matrix M (same as SV in SYNII)
only on entry if JOB >= 2

JOB
INTEGER
JOB = 0 : start the deflation algorithm from scratch; i.e.,
it factors the matrix, performs inverse iteration to
determine PSI, PHI and DELTA, and then computes the
deflated solution.
JOB = 1 : assume that A has already been factored by CDRV
(or a previous call to SYC[DETBE] and continue
from there.
JOB >= 2 : additionally, PSI, PHI and DELTA have already
been computed.

on exit:

PSI
REAL(N)
left and right null vectors to matrix A

PHI
REAL(N)
smallest singular value for matrix M (same as SV in SYNII)

DELTA
REAL

ZD
REAL(N)
deflated solution to system A z = p
Note that P and ZD may be the same vector

CP
REAL
coefficient of projection of Z onto right null vector (PHI)

INTEGER N, R(N),C(N),IC(N),IA(N),JA(1),NSP,ISP(1),ESP
INTEGER JOB,IJOB, FLAG
REAL A(1), RSP(NSP), P(N), PSI(N), PHI(N), ZD(N), CP
REAL DELTA, PSITP, SV
LOGICAL TRANS

IJOB = JOB
TRANS = (IJOB .GE. 10)
IF (TRANS) IJOB = IJOB - 10

IF (IJOB .EQ. 0)
* CALL NDRV (N, R,C,IC, IA,JA, PHI, PHI,
* NSP,ISP,RSP,ESP, 1,FLAG)

IF (IJOB .LE. 1)
* CALL SYNII (N, R,C,IC, IA,JA, NSP,ISP,RSP,ESP,
* PSI,PHI,DELTA, 0,3)
IF (TRANS) GOTO 20

C
C Perform deflation with A
C A Zd = p - (psi T p) psi; solve for Zd; Cp is approx (psi T p)
C CP = SDOT(N,P,1,PSI,1)
C CALL SCOPY(N,P,1,ZD,1)
C CALL SAXPY(N,-CP,PSI,1,ZD,1)
C CALL NDRV(N,R,C,IC,IA,JA,A,ZD,ZD,NSP,ISP,RSP,ESP,3,FLAG)
C
C orthogonalize Zd with respect to phi
C CALL SAXPY(N,-SDOT(N,PHI,1,ZD,1),PHI,1,ZD,1)
GOTO 30

20 CONTINUE

C
C Perform deflation with A
C T
C A Zd = p - (phi T p) phi; solve for Zd; Cp is approx (phi T p)
C CP = SDOT(N,P,1,PHI,1)
C CALL SCOPY(N,P,1,ZD,1)
C CALL SAXPY(N,-CP,PHI,1,ZD,1)
C CALL NDRV(N,R,C,IC,IA,JA,A,ZD,ZD,NSP,ISP,RSP,ESP,3,FLAG)
C
C orthogonalize Zd wrt psi
C CALL SAXPY(N,-SDOT(N,PSI,1,ZD,1),PSI,1,ZD,1)

30 CONTINUE
END

C\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\\///\"
C PATH = 1. (which may have been done by SGEDBE)
C
C JOB INTEGER
C if an approximate null vector is already known, the user
C may pass it to SYNII. JOB indicates where to find it.
C JOB = 0 : no initial guess
C JOB < 0 : approximate left null vector is passed in PSI
C JOB > 0 : approximate right null vector is passed in PHI
C
C ITER INTEGER
C governs how many iterations are performed
C ITER = 0 : continue iterating until PSI and PHI converge
C on accurate values. If M is nearly singular
C this usually occurs with 2 or 3 iterations.
C ITER > 0 : do up to ITER many iterations.
C
C on exit:
C
C PSI, PHI REAL(N)
C on output, contain the left and right null vectors,
C respectively, of the matrix A.
C
C resol = resolution of convergence
C
C REAL RESOL
C PARAMETER (RESOL = .0001)
C INTEGER N, R(1), C(1), IC(1), IA(1), JA(1), NSP, ISP(1), ESP, JOB, ITER
C INTEGER FLAG
C REAL A(1), RSP(1), PSI(1), PHI(1), DELTA
C REAL PSILEN, PHILEN
C
C IF (JOB .EQ. 0) THEN
C no initial guess; fill PSI with 1's
D0 10 I = 1, N
   PSI(I) = 1.
10 CONTINUE
C
C ELSEIF (JOB .EQ. 1) THEN
C initial guess is in PHI; solve for initial PSI
C phi' = phi' / ||phi'||
C PHILEN = SNRM2(N, PHI, 1)
C CALL SSCAL (N, 1/PHILEN, PHI, 1)
C
C T
C A psi' = phi'
C CALL NDRV
C * (N, R, C, IC, IA, J, A, PHI, PSI, NSP, ISP, RSP, ESP, 4, FLAG)
ENDIF
C  PSI now contains initial guess; normalize it
C  psi' = psi' / ||psi'||
PSILEN = SNRM2(N,PSI,1)
CALL SSCAL (N,1/PSILEN,PSI,1)
C
C........................................................................... main loop of routine
IINC = 0
IF (ITER .NE. 0) IINC = 1
I = IINC
C
C repeat until convergence
50 CONTINUE
C
C  A phi' = psi
CALL NDRV
* (N, R,C,IC, IA,JA,A, PSI,PHI, NSP,ISP,RSP,ESP, 3,FLAG)
C
C  phi' = phi' / ||phi'||
PHILEN = SNRM2(N,PHI,1)
CALL SSCAL (N,1/PHILEN,PHI,1)
C
C  T
C  A psi' = phi'
CALL NDRV
* (N, R,C,IC, IA,JA,A, PHI,PSI, NSP,ISP,RSP,ESP, 4,FLAG)
C
C  psi' = psi' / ||psi'||
PSILEN = SNRM2(N,PSI,1)
CALL SSCAL (N,1/PSILEN,PSI,1)
C
C increment counter
I = I + IINC
C
C end
IF (I .LE. ITER .AND. ABS(1/PHILEN - 1/PSILEN) .GT. RESOL)
* GOTO 50
C
C do phi' once more
CALL NDRV (N, R,C,IC, IA,JA,A, PSI,PHI, NSP,ISP,RSP,ESP, 3,FLAG)
C
delta = 1/||phi'||
C
C DELTA gets a sign such that PSI(1) and PHI(1) have the same sign
C when A is symmetric, PSI = PHI, and DELTA is smallest eigenvalue
DELTA = SIGN(1/SNRM2(N,PHI,1),PSI(1)*PHI(1))
CALL SSCAL (N,DELTA,PHI,1)

54
SUBROUTINE SYNBE
  * (N,M, R,C,IC, IA,JA,A, NSP,ISP,RSP,ESP, B,LDB,
  *   CT, LDC, D,LDD, F,G, X,Y, WORK1,LDW1, WORK2,LDW2, JOB)
  
the ordinary (undflated) block elimination algorithm

all arguments are the same as in SYNDCE.

INTEGER N,M, LDB,LDC,LDD,LDW1,LDW2, AJOB, LRATIO
INTEGER R(N),C(N),IC(N), IA(N),JA(*), NSP,ISP,NSP,ESP
REAL A(*),RSP,NSP
REAL B(LDB,M),CT(LDC,N),D(LDD,M), F(N),G(M), X(N),Y(M)
REAL WORK1(LDW1,1),WORK2(LDW2,1), DELTA
CHARACTER*6 JOB
LOGICAL NEWA,NEWB,NEWC,NEWD,NEWF,NEWG

INTEGER MP1,IPVT
MP1 = M + 1
IPVT = MP1 + 1

AJOB = 0
IF (INDEX(JOB,'A') .NE. 0) AJOB = 2
IF (INDEX(JOB,'A') .NE. 0) AJOB = 1
NEWA = (AJOB .NE. 2)
NEWB = (INDEX(JOB,'B') .EQ. 0)
NEWC = (INDEX(JOB,'C') .EQ. 0)
NEWD = (INDEX(JOB,'D') .EQ. 0)
NEWF = (INDEX(JOB,'F') .EQ. 0)
NEWG = (INDEX(JOB,'G') .EQ. 0)

solve A V = B for V
IF (AJOB .EQ. 0)
  * CALL NDRV (N, R,C,IC, IA,JA,A, X,X, NSP,ISP,RSP,ESP, 1, FLAG)

  IF (NEWA .OR. NEWB) THEN
    DO 10 I = 1,M
      CALL NDRV (N, R,C,IC, IA,JA,A, B(1,I),WORK1(1,I),
  *             NSP,ISP,RSP,ESP, 1, FLAG)
  10 CONTINUE
ENDIF

solve A w = f for w
IF (NEWA .OR. NEWF)
  * CALL NDRV (N, R,C,IC, IA,JA,A, F,WORK1(1,MP1),

55
C compute E (= D - cT V)
  IF (NEWA .OR. NEWB .OR. NEWC .OR. NEWD) THEN
    DO 30 I = 1,M
      DO 20 J = 1,M
        WORK2(I,J) = D(I,J) - SDOT(N, CT(I,1),LDC, WORK1(I,J),1)
      20 CONTINUE
    30 CONTINUE
    CALL SGEFA (WORK2,LDW2,M,WORK2(1,IPVT),INFO)
  ENDIF
C compute g' (= g - cT w)
  IF (NEWA .OR. NEWC .OR. NEWF .OR. NEWG) THEN
    DO 40 I = 1,M
      WORK2(I,MP1) = G(I) - SDOT(N, CT(I,1),LDC, WORK1(I,MP1),1)
    40 CONTINUE
  ENDIF
C solve for y
  CALL SCOPY (M, WORK2(1,MP1),1, Y,1)
  CALL SGESL (WORK2,LDW2,M,WORK2(1,IPVT),Y,0)
C compute x
  DO 50 I = 1,N
    X(I) = WORK1(I,MP1) - SDOT(M, WORK1(I,1),LDW1, Y,1)
  50 CONTINUE
END
The routines in this package implement the deflated block-
elimination algorithm for solving systems of the form:
\[
\begin{bmatrix}
\mathbf{x} \\
\mathbf{M} \\
\mathbf{y}
\end{bmatrix} =
\begin{bmatrix}
\mathbf{A} & \mathbf{b} & \mathbf{f} \\
\mathbf{c} & \mathbf{d} & \mathbf{g}
\end{bmatrix}
\begin{bmatrix}
\mathbf{x} \\
\mathbf{y}
\end{bmatrix}
\]

discussed in T. F. Chan and D. Reasorco, *Generalized Deflated
Block-Elimination*, Technical Report YALEU/DCS/RR-337, Dept. of

This set of routines calls YSMF's general routines with
compressed storage, Linpack's SGE- routines, and the SBLAs.
Implemented by Thomas A. Grossi, Yale University, 1985.

STORAGE SCHEME FOR THESE Routines

The nonzero entries of the coefficient matrix \( \mathbf{M} \) are stored
row-by-row in the array \( \mathbf{A} \). To identify the individual nonzero
entries in each row, we need to know in which column each entry
lies. The column indices which correspond to the nonzero entries
of \( \mathbf{M} \) are stored in the array \( \mathbf{J} \); i.e., if \( \mathbf{A}(k) = \mathbf{M}(i,j) \), then
\( \mathbf{J}(k) = j \). In addition, we need to know where each row starts and
how long it is. The index positions in \( \mathbf{J} \) and \( \mathbf{A} \) where the rows of
\( \mathbf{M} \) begin are stored in the array \( \mathbf{I} \); i.e., if \( \mathbf{M}(i,j) \) is the first
nonzero entry (stored) in the \( i \)-th row and \( \mathbf{A}(k) = \mathbf{M}(i,j) \), then
\( \mathbf{I}(i) = k \). Moreover, the index in \( \mathbf{J} \) and \( \mathbf{A} \) of the first location
following the last element in the last row is stored in \( \mathbf{I}(n+1) \).
Thus, the number of entries in the \( i \)-th row is given by
\( \mathbf{I}(i+1) - \mathbf{I}(i) \). The nonzero entries of the \( i \)-th row are stored
consecutively in
\( \mathbf{A}(\mathbf{I}(i)), \mathbf{A}((\mathbf{I}(i)+1), \ldots, \mathbf{A}(\mathbf{I}(i)+1)) \),
and the corresponding column indices are stored consecutively in
\( \mathbf{J}(\mathbf{I}(i)), \mathbf{J}((\mathbf{I}(i)+1), \ldots, \mathbf{J}(\mathbf{I}(i)+1)) \).
For example, the 5 by 5 matrix
\[
\begin{bmatrix}
1 & 0 & 2 & 0 & 0 \\
0 & 3 & 0 & 0 & 0 \\
M &=& 0 & 4 & 5 & 6 & 0 \\
0 & 0 & 0 & 7 & 0 \\
0 & 0 & 0 & 8 & 9
\end{bmatrix}
\]
would be stored as
The rows and columns of the original matrix $A$ can be reordered (e.g., to reduce fill-in or ensure numerical stability) before calling the driver. If no reordering is done, then set $R(I) = C(I) = IC(I) = I$ for $I=1,...,N$. The solution $A$ is returned in the original order.

Working storage is needed for the factored form of the matrix plus various temporary vectors. The arrays $ISP$ and $RSP$ should be equivalenced: integer storage is allocated from the beginning of $ISP$ and real storage from the end of $RSP$. 

Declared dimension of $RSP$; the exact value of $NSP$ will be specified below.

Integer working storage divided up into various arrays
needed by the subroutines; ISP and RSP should be
equivalenced.
size = LRATIO+NSP, where LRATIO = size of storage for a
real number divided by the size of storage for an integer.

RSP     REAL(NSP)
real working storage divided up into various arrays
needed by the subroutines; ISP and RSP should be
equivalenced.

ESP     INTEGER
if sufficient storage was available to perform the
symbolic factorization (NSFC), then ESP is set to the
amount of excess storage provided (negative if
insufficient storage was available to perform the
numeric factorization (NNFC)).
if ESP > 2*N, then those last 2n position of RSP will
contain approximate left and right null vectors for A.

SUBROUTINE SYCDBE
* (N,M, R,C,IC, IA,JA,A, NSP,ISP,RSP,ESP, B,LDB,
  CT, LDC, D,LDD, F,G, X,Y, WORK1,LDW1, WORK2,LDW2, JOB)

the deflated block elimination algorithm

arguments:
on entry:

N     INTEGER
the order of the matrix A

M     INTEGER
the order of the borders to A in M

R     INTEGER(N)
ordering of the rows of A.

C     INTEGER(N)
ordering of the columns of A.

IC     INTEGER(N)
inverse of the ordering of the columns of m; i.e.,
IC(C(I)) = I for I=1,...,n.

59
 INTEGER(N+1) pointers to delimit the rows in A.

 INTEGER(*) column numbers corresponding to the elements of A.
 size = size of A.

 INTEGER(*) nonzero entries of the coefficient matrix A, stored
 by rows.
 size = number of nonzero entries in A.

 INTEGER declared dimension of RSP;
 the exact value of NSP will be specified below.

 INTEGER(*) integer working storage divided up into various arrays
 needed by the subroutines; ISP and RSP should be
 equivalenced.
 size = LRATIO*NSP, where LRATIO = size of storage for a
 real number divided by the size of storage for an integer.

 REAL(NSP) real working storage divided up into various arrays
 needed by the subroutines; ISP and RSP should be
 equivalenced.

 REAL(LDB,M) right-hand border to matrix A in matrix M.

 INTEGER the leading dimension of the array B. LDB >= N.

 REAL(LDC,N) bottom border to matrix A in matrix M.

 INTEGER the leading dimension of the array CT. LDC >= M.

 REAL(LDD,M) lower right-hand entries of M.

 INTEGER the leading dimension of the array D. LDD >= M.

 REAL(N) right-hand side to solve with
WORK1 REAL(LDW1,M+4) LDW1 >= N
used to hold Vd, Wd, psiT B, psi and phi

LDW1 INTEGER
the leading dimension of the array WORK1. LDW1 >= N.

WORK2 REAL(LDW2,M+3) LDW2 >= M+1
used to hold E, g', psiT f and delta, and pivot indices for E.

LDW2 INTEGER
the leading dimension of the array WORK2. LDW2 >= M+1.

JOB CHARACTER*6
indicates which inputs are the same as in the last call
to SGEBDE. If there was no such call, set JOB =
' ' or 'a ' (see below). Otherwise, JOB contains
as many of the following apply:
'A' if A stays the same
'S' if A is new but already factored by SGECO or SGEEA
'B' if B stays the same
'C' if CT stays the same
'D' if D stays the same
'F' if F stays the same
'G' if G stays the same

on exit:

RSP REAL(NSP)
the last 2n positions of RSP contain approximate
left and right null vectors for A if ESP > 2*N.

ESP INTEGER
if sufficient storage was available to perform the
symbolic factorization (NSFC), then ESP is set to the
amount of excess storage provided (negative if
insufficient storage was available to perform the
numeric factorization (NNFC)).
if ESP > 2*N, then those last 2n position of RSP will
contain approximate left and right null vectors for A.

X REAL(N)
Y REAL(M+1)
solution vector

WORK1 REAL(LDW1,M+4) LDW1 >= N
used to hold Vd, Wd, psiT B, psi and phi.

WORK2 REAL(LDW2,M+4) LDW2 >= M+1
used to hold E, g', psi T f and delta, and pivot indices for E

Savings on storage:
the following pairs of inputs may be equivalent:
(X,F) (Y,G) (B,WORK1) (D,WORK2)
in general if equivalent storage is used, then a change in one
of the inputs in either the left-hand-side group or the right-
hand-side group requires that the entire group be re-entered.
Specific exceptions to this rule can be determined by examining
the algorithm.

INTEGER N,M, LDB, LDC, LDD, LDW1, LDW2, AJOB, LRATIO
INTEGER R(N), C(N), IC(N), IA(N), JA(*), NSP, ISP(NSP), ESP
REAL A(*), RSP(NSP)
REAL B(LDB,M), CT(LDC,N), D(LDD,M), F(N), G(M), X(N), Y(M)
REAL WORK1(LDW1,M), WORK2(LDW2,M), DELTA
CHARACTER*8 JOB
LOGICAL NEWA, NEWB, NEWC, NEWD, NEWF, NEWG

the following constants are used to partition WORK1 and WORK2
into their various vectors; MP1 stands for the "extra" row and
column added to D in forming E. WORK1 is primarily used for Vd,
and WORK2 for E

INTEGER MP1, CB, WD, CF, PSI, PHI, GP, IPVT, ALPHA
DATA LRATIO /1/
MP1 = M + 1
CB = MP1
WD = CB + 1
CF = MP1
PSI = WD + 1
PHI = PSI + 1
GP = MP1 + 1
IPVT = GP - 1
ALPHA = MP1

AJOB = 0
IF (INDEX(JOB,'A') .NE. 0) AJOB = 2
IF (INDEX(JOB,'S') .NE. 0) AJOB = 1
NEWA = (AJOB .NE. 2)
NEWB = (INDEX(JOB,'B') .EQ. 0)
NEWC = (INDEX(JOB,'C') .EQ. 0)
NEWD = (INDEX(JOB,'D') .EQ. 0)
NEWF = (INDEX(JOB,'F') .EQ. 0)
NEWG = (INDEX(JOB,'G') .EQ. 0)

Algorithm:

factor A, compute psi, phi, delta
compute deflated solution to A V = B
compute deflated solution to \( A \mathbf{w} = \mathbf{f} \)

build \( \mathbf{E} \):  
\[
| D - c^T \mathbf{Vd} | 
| c^T \mathbf{phi} | 
| c^T \mathbf{delta} |
\]

build \( g' \):  
\[
| g - c^T \mathbf{Vd} | 
| C | 
| \mathbf{f} |
\]

solve \( \mathbf{E} \mathbf{y} = g' \) for \( \mathbf{y} \)

\[
| \alpha |
| \mathbf{phi} |
\]

\[
x = \mathbf{Vd} \mathbf{y} + \alpha \mathbf{phi}
\]

if \( \text{AJOB} = 0 \) or 1, or \( B \) is new, we start by solving \( \mathbf{A} \mathbf{Vd} = \mathbf{B} \);
this may imply factoring \( \mathbf{A} \), and/or computing \( \mathbf{psi} \), \( \mathbf{phi} \) and \( \mathbf{delta} \)

IF (NEWA .OR. NEWB) THEN

for the first element of \( \mathbf{Vd} \), \( \text{AJOB} \) will tell \( \text{sgedf} \) what to do

CALL SYCDEF (N, R,C,IC, IA,JA,A, NSP,ISP,RSP,ESP, B(1,1),  
* WORK1(1,PSI),WORK1(1,PHI),DELTA, WORK1(1,1),WORK1(1,CB),AJOB)

compute remaining columns of \( \mathbf{Vd} \) using results of first call

IF (M .GT. 1) THEN

DO 10 I = 2,M

CALL SYCDEF (N, R,C,IC, IA,JA,A, NSP,ISP,RSP,ESP, B(1,I),  
* WORK1(1,PSI),WORK1(1,PHI),DELTA,  
* WORK1(1,I),WORK1(I,CB),2)

10 CONTINUE

ENDIF

ENDIF

we must recompute \( \mathbf{Wd} \) and \( \mathbf{Cf} \) if \( \mathbf{A} \) or \( \mathbf{F} \) have changed

IF (NEWA .OR. NEWF) THEN

CALL SYCDEF (N, R,C,IC, IA,JA,A, NSP,ISP,RSP,ESP, F,  
* WORK1(1,PSI),WORK1(1,PHI),DELTA,  
* WORK1(1,Wd),WORK2(CF,GP),2)

ENDIF

build and factor \( \mathbf{E} \)

IF (NEWA .OR. NEWB .OR. NEWC .OR. NEWD) THEN

CALL SCOPY (M, WORK1(1,CB),1, WORK2(MP1,1),LDW2)  
WORK2(MP1,MP1) = DELTA

DO 30 I = 1,M

compute \( D - c^T \mathbf{Vd} \), column by column

DO 20 J = 1,M

\[
\text{WORK2}(I,J) = D(I,J) - \text{SDOT}(N, C(T(I,1),LDC, \text{WORK1}(I,J),1)
\]

20 CONTINUE

compute \( c^T \mathbf{PHI} \) element by element

\[
\text{WORK2}(I,MP1) = \text{SDOT}(N, C(T(I,1),LDC, \text{WORK1}(I,PHI),1)
\]

30 CONTINUE

63
C
   factor E
   CALL SGFIA (WORK2, LDW2, MP1, WORK2(1, IPVT), INFO)
ENDIF

C
   g' depends on a lot of things
   IF (NEWA .OR. NEWC .OR. NEWF .OR. NEWQ) THEN
      DO 40 I = 1, M
         WORK2(I, GPR) = G(I) - SDOT(N, CT(I, I), LDC, WORK1(I, WD), 1)
      CONTINUE
   ENDIF

C
   compute x and y
   CALL SCOPI (MP1, WORK2(I, GPR), 1, Y, 1)
   CALL SGESL (WORK2, LDW2, MP1, WORK1(I, IPVT), Y, 0)
   DO 50 I = 1, N
      X(I) = WORK1(I, WD) - SDOT(M, WORK1(I, I), LDW1, Y, 1)
   CONTINUE
   CALL SAXPY (N, Y(ALPHA), WORK1(I, PHI), 1, X, 1)
   WORK2(I, IPVT+1) = DELTA

END

C----------------------------------------------------------------------------------------
C>-----------------------------------------------------------------------------------------------------------------------------------
C<-----------------------------------------------------------------------------------------------------------------------------------
C----------------------------------------------------------------------------------------

C SUBROUTINE SYCDF (N, R, IC, IA, J, A, NSP, ISP, RSP, ESP, *
*                   P, PSI, PHI, DELTA, ZD, CP, JOB)
C
C computes the deflated decomposition of A z = p, returning
C solution in the form:
C
C    z = z + phi (c / delta)
C
C arguments are the same as for SYCDBE except:
C
C on entry:
C
C P     REAL(N)
C    contains rhs to system of equations
C
C PSI    REAL(N)
C PHI    REAL(N)
C    left and right null vectors to matrix A
C    (only on entry if JOB >= 2)
C
C DELTA  REAL

64
smallest singular value for matrix M (same as SV in SYCII)
(only on entry if JOB >= 2)

JOB INTEGER
JOB = 0 : start the deflation algorithm from scratch; i.e.,
it factors the matrix, performs inverse iteration to
determine PSI, PHI and DELTA, and then computes the
deflated solution.
JOB = 1 : assume that A has already been factored by CDRV
(or a previous call to SYC[0]BE) and continue
from there.
JOB >= 2 : additionally, PSI, PHI and DELTA have already
been computed.

on exit:

PSI REAL(N)
PHI REAL(N)
left and right null vectors to matrix A

DELTA REAL
smallest singular value for matrix M (same as SV in SYNII)

ZD REAL(N)
deflated solution to system A z = p
Note that P and ZD may be the same vector

CP REAL
coefficient of projection of Z onto right null vector (PHI)

INTEGER N, R(N),C(N),IC(N),IA(N),JA(1),NSP,ISP(1),ESP
INTEGER JOB, IJOB, FLAG
REAL A(1), RSP(NSP), P(N), PSI(N), PHI(N), ZD(N), CP
REAL DELTA, PSITP, SV
LOGICAL TRANS

IJOB = JOB
TRANS = (IJOB .GE. 10)
IF (TRANS) IJOB = IJOB - 10
.

IF (IJOB .EQ. 0)
* CALL CDRV (N, R, C, IC, IA, A, PHI, PHI,
* NSP, ISP, RSP, ESP, 1, FLAG)

IF (IJOB .LE. 1)
* CALL SYCII (N, R, C, IC, IA, A, NSP, ISP, RSP, ESP,
* PSI, PHI, DELTA, 0, 3)
IF (TRANS) GOTO 20

C Perform deflation with A
C A Zd = p -(psi T p) psi ; solve for Zd ; Cp is approx (psi T p)
C CP = SDOT(N,P,1,PSI,1)
C CALL SCOPY (N,P,1,ZD,1)
C CALL SAXPY (N,-CP,PSI,1,ZD,1)
C CALL CDRV (N,R,C,IC,IA,JA,A,ZD,ZD,NSP,ISP,RSP,ESP,3,FLAG)

C orthogonalize Zd with respect to phi
C CALL SAXPY(N,-SDOT(N,PHI,1,ZD,1),PHI,1,ZD,1)
GOTO 30

20 CONTINUE

T

C Perform deflation with A
C T
C A Zd = p -(phi T p) phi ; solve for Zd ; Cp is approx (phi T p)
C CP = SDOT(N,P,1,PHI,1)
C CALL SCOPY (N,P,1,ZD,1)
C CALL SAXPY (N,-CP,PHI,1,ZD,1)
C CALL CDRV (N,R,C,IC,IA,JA,A,ZD,ZD,NSP,ISP,RSP,ESP,3,FLAG)

C orthogonalize Zd wrt psi
C CALL SAXPY(N,-SDOT(N,PSI,1,ZD,1),PSI,1,ZD,1)

30 CONTINUE
END

C SUBROUTINE SYCII
*(N, R,C,IC, IA,JA,A, NSP,ISP,RSP,ESP, PSI,PHI, DELTA, JOB,ITER)
C computes approximate left and right null vectors of A by applying
C the inverse iteration algorithm described in T. F. Chan, "Deflated
C Decomposition of Solutions of Nearly Singular Systems," SIAM J.
C arguments are the same as for SYCDBE except:
C on entry:
C NSP INTEGER
C ISP REAL(*)
C RSP REAL(NSP)
C ESP INTEGER
C must contain a factorization of A, produced by CDRV with
PATH = 1, (which may have been done by SGEDBE)

JOB INTEGER
if an approximate null vector is already known, the user
may pass it to SYCII. JOB indicates where to find it.
JOB = 0 : no initial guess
JOB < 0 : approximate left null vector is passed in PSI
JOB > 0 : approximate right null vector is passed in PHI

ITER INTEGER
governs how many iterations are performed
ITER = 0 : continue iterating until PSI and PHI converge
on accurate values. If M is nearly singular
this usually occurs with 2 or 3 iterations.
ITER > 0 : do up to ITER many iterations.

on exit:

PSI, PHI REAL(N)
on output, contain the left and right null vectors,
respectively, of the matrix A.

resol = resolution of convergence

REAL RESOL
PARAMETER (RESOL = .0001)
INTEGER N, R(I), C(I), IC(I), IA(I), JA(I), NSP, ISP(I), ESP, JOB, ITER
INTEGER FLAG
REAL A(I), RSP(I), PSI(I), PHI(I), DELTA
REAL PSILEN, PHILEN

IF (JOB .EQ. 0) THEN
  no initial guess; fill PSI with 1's
  DO 10 I = 1, N
    10    PSI(I) = 1.
ELSEIF (JOB .EQ. 1) THEN
  initial guess is in PHI; solve for initial PSI
  phi' = phi' / | |phi'||
  PHILEN = SNRM2(N, PHI, 1)
  CALL SSCAL (N, 1/PHILEN, PHI, 1)
  T
  A psi' = phi'
  CALL CDRV
  + (N, R, C, IC, IA, JA, A, PHI, PSI, NSP, ISP, RSP, ESP, 4, FLAG)
endif

C

psi' = psi / ||psi'||

PSILEN = SNRM2(N, PSI, 1)
CALL SSCAL (N, 1/PSILEN, PSI, 1)

C

main loop of routine

IINC = 0

IF (ITER .NE. 0) IINC = 1
I = IINC

C

repeat until convergence

50 CONTINUE

C

A phi' = psi
CALL CDRV

* (N, R, C, IC, IA, JA, A, PSI, PHI, NSP, ISP, RSP, ESP, 3, FLAG)

C

phi' = phi' / ||phi'||

PHILEN = SNRM2(N, PHI, 1)
CALL SSCAL (N, 1/PHILEN, PHI, 1)

C

T

A psi' = phi'
CALL CDRV

* (N, R, C, IC, IA, JA, A, PHI, PSI, NSP, ISP, RSP, ESP, 4, FLAG)

C

psi' = psi' / ||psi'||

PSILEN = SNRM2(N, PSI, 1)
CALL SSCAL (N, 1/PSILEN, PSI, 1)

C

increment counter
I = I + IINC

C

don end

IF (I .LE. ITER .AND. ABS(1/PHILEN - 1/PSILEN) .GT. RESOL)

* GOTO 50

C

do phi' once more

CALL CDRV (N, R, C, IC, IA, JA, A, PSI, PHI, NSP, ISP, RSP, ESP, 3, FLAG)

C

delta = 1/||phi'||

C

DELTA gets a sign such that PSI(1) and PHI(1) have the same sign
when A is symmetric, PSI = PHI, and DELTA is smallest eigenvalue
DELTA = SIGN(1/SNRM2(N, PHI, 1), PSI(1)*PHI(1))
CALL SSCAL (N, DELTA, PHI, 1)
SUBROUTINE SYCBE
* (N,M,  R,C,IC, IA,JA,A, NSP,ISP,RSP,ESP, B,LDB,
  *  CT, LDC, D,LDD, F,G, X,Y, WORK1,LDW1, WORK2,LDW2, JOB)

the ordinary (undeflated) block elimination algorithm

all arguments are the same as in SYCDBE.

INTEGER N,M, LDB,LDC,LDD,LDW1,LDW2, AJOB, LRATIO
INTEGER R(N),C(N),IC(N), IA(N),JA(*), NSP,ISP(NSP),ESP
REAL A(*),RSP(NSP)
REAL B(LDB,M),CT(LDC,N),D(LDD,M), F(N),G(M), X(N),Y(M)
REAL WORK1(LDW1,M),WORK2(LDW2,M), DELTA
CHARACTER*6 JOB
LOGICAL NEWA,NEWB,NEWC,NEWD,NEWF,NEWG

INTEGER MP1,IPVT
MP1 = M + 1
IPVT = MP1 + 1

AJOB = 0
IF (INDEX(JOB,'A') .NE. 0) AJOB = 2
IF (INDEX(JOB,'a') .NE. 0) AJOB = 1
NEWA = (AJOB ,NE. 2)
NEWB = (INDEX(JOB,'B') .EQ. 0)
NEWC = (INDEX(JOB,'C') .EQ. 0)
NEWD = (INDEX(JOB,'D') .EQ. 0)
NEWF = (INDEX(JOB,'F') .EQ. 0)
NEWG = (INDEX(JOB,'G') .EQ. 0)

solve A V = B for V
IF (AJOB .EQ. 0)
  * CALL CDRV (N, R,C,IC, IA,JA,A, X,X, NSP,ISP,RSP,ESP, 1, FLÅG)

IF (NEWA .OR. NEWB) THEN
  DO 10 I = 1,M
    CALL CDRV (N, R,C,IC, IA,JA,A, B(1,I),WORK1(1,I),
      *       NSP,ISP,RSP,ESP, 1, FLAG)
  10    CONTINUE
ENDIF

solve A w = f for w
IF (NEWA .OR. NEWF)
* CALL CDRV (N, R,C,IC, IA,J,A, A, F,WORK1(I,MP1),
* NSP,ISP,RSP,ESP, 1, FLAG)

C
C compute E (= D - cTv)
IF (NEWA .OR. NEWB .OR. NEWC .OR. NEWD) THEN
  DO 30 I = 1,M
      DO 20 J = 1,M
          WORK2(I,J) = D(I,J) - SDOT(N, CT(I,1),LDC, WORK1(I,J),1)
      20 CONTINUE
  30 CONTINUE
CALL SGEFA (WORK2,LDW2,M,WORK2(1,IPVT),INFO)
ENDIF
C
C compute g' (= g - cTw)
IF (NEWA .OR. NEWC .OR. NEWF .OR. NEWG) THEN
  DO 40 I = 1,M
      WORK2(I,MP1) = G(I) - SDOT(N, CT(I,1),LDC, WORK1(I,MP1),1)
  40 CONTINUE
ENDIF
C
C solve for y
CALL SCOPY (M, WORK2(1,MP1),1, Y,1)
CALL SGESL (WORK2,LDW2,M,WORK2(1,IPVT),Y,0)
C
C compute x
DO 50 I = 1,N
    X(I) = WORK1(I,MP1) - SDOT(M, WORK1(I,1),LDW1, Y,1)
  50 CONTINUE
END
The routines in this package implement the deflated block-
 elimination algorithm for solving systems of the form:

\[
\begin{bmatrix}
  x \\
  y \\
\end{bmatrix}
= 
\begin{bmatrix}
  A & B \\
  C & D \\
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
\end{bmatrix}
+ 
\begin{bmatrix}
  f \\
  g \\
\end{bmatrix}
\]

discussed in T. F. Chan and D. R. Massa, "Generalized Deflated
Block-Elimination," Technical Report YALEU/DCS/RR-3837, Dept. of

This set of routines calls YSPM's routines for symmetric matrices,
Linpack's SGE- routines, and the SBLAs.

Implemented by Thomas A. Gossi, Yale University, 1985.

STORAGE SCHEME FOR THESE ROUTINES

The nonzero entries of the coefficient matrix M are stored
row-by-row in the array A. To identify the individual nonzero
entries in each row, we need to know in which column each entry
lies. The column indices which correspond to the nonzero entries
of M are stored in the array JA; i.e., if A(k) = M(i, j), then
JA(k) = j. In addition, we need to know where each row starts and
how long it is. The index positions in JA and A where the rows of
M begin are stored in the array IA; i.e., if M(i, j) is the first
nonzero entry (stored) in the I-th row and A(k) = M(i, j), then
IA(I) = k. Moreover, the index in JA and A of the first location
following the last element in the last row is stored in IA(N+1).
Thus, the number of entries in the I-th row is given by
IA(I+1) - IA(I), the nonzero entries of the I-th row are stored
consecutively in

\[
A(IA(I)), A(IA(I)+1), \ldots, A(IA(I+1)-1)
\]

and the corresponding column indices are stored consecutively in

\[
JA(IA(I)), JA(IA(I)+1), \ldots, JA(IA(I+1)-1)
\]

Since the coefficient matrix is symmetric, only the nonzero entries
in the upper triangle need be stored, for example, the matrix
could be stored as

\[
\begin{bmatrix}
\end{bmatrix}
\]

\[\text{IA} \mid 1 \ 4 \ 5 \ 8 \ 12 \ 14\]
\[\text{JA} \mid 1 \ 3 \ 4 \ 2 \ 1 \ 3 \ 4 \ 1 \ 3 \ 4 \ 5 \ 4 \ 5\]
\[\text{A} \mid 1 \ 2 \ 3 \ 4 \ 2 \ 5 \ 6 \ 3 \ 6 \ 7 \ 8 \ 8 \ 9\]

or (symmetrically) as

\[
\begin{bmatrix}
\end{bmatrix}
\]

\[\text{IA} \mid 1 \ 4 \ 5 \ 7 \ 9 \ 10\]
\[\text{JA} \mid 1 \ 3 \ 4 \ 2 \ 3 \ 4 \ 4 \ 5 \ 5\]
\[\text{A} \mid 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9\]

N INTEGER
number of variables/equations.

A INTEGER(*)
nonzero entries of the coefficient matrix A, stored
by rows.
size = number of nonzero entries in A.

IA INTEGER(N+1)
pointers to delimit the rows in A.

JA INTEGER(*)
column numbers corresponding to the elements of A.
size = size of A.

The rows and columns of the original matrix A can be
reordered (e.g., to reduce fill-in or ensure numerical stability)
before calling the driver. If no reordering is done, then set
P(I) = IP(I) = I for I=1,...,N. The solution A is returned in
the original order.

P INTEGER(N)
ordering of the rows/columns of A.

IP INTEGER(N)
inverse of the ordering of the rows/columns of A; i.e.,
IC(I) = I for I = 1, ..., n.

Working storage is needed for the factored form of the matrix A plus various temporary vectors. The arrays ISP and RSP should be equivalenced; integer storage is allocated from the beginning of ISP and real storage from the end of RSP.

NSP INTEGER

declared dimension of RSP;
the exact value of NSP will be specified below

ISP INTEGER(*)

integer working storage divided up into various arrays needed by the subroutines; ISP and RSP should be equivalenced.

size = LRATIO*NSP, where LRATIO = size of storage for a real number divided by the size of storage for an integer.

RSP REAL(NSP)

real working storage divided up into various arrays needed by the subroutines; ISP and RSP should be equivalenced.

ESP INTEGER

if sufficient storage was available to perform the symbolic factorization (CSFC), then ESP is set to the amount of excess storage provided (negative if insufficient storage was available to perform the numeric factorization (CNFC)).

if ESP > 2*N, then those last 2n position of RSP will contain approximate left and right null vectors for A.

SUBROUTINE SYSDBE

* (N, M, P, IP, IA, JA, A, NSP, ISP, RSP, ESP, B, LDB,
* CT, LDC, D, LDD, F, G, X, Y, WORK1, LDW1, WORK2, LDW2, JOB)

c
arguments:

on entry:

N INTEGER
the order of the matrix A
M INTEGER
the order of the borders to A in M

P INTEGER(N)
ordering of the rows/columns of A.

IP INTEGER(N)
inverse of the ordering of the rows/columns of A; i.e.,
IC(C(I)) = I for I=1,...,n.

IA INTEGER(N+1)
pointers to delimit the rows in A.

JA INTEGER(*)
column numbers corresponding to the elements of A.
size = size of A.

A INTEGER(*)
nonzero entries of the coefficient matrix A, stored
by rows.
size = number of nonzero entries in A.

NSP INTEGER
declared dimension of RSP;
the exact value of NSP will be specified below

ISP INTEGER(*)
integer working storage divided up into various arrays
needed by the subroutines; ISP and RSP should be
equivalenced.
size = LRATIO*NSP, where LRATIO = size of storage for a
real number divided by the size of storage for an integer.

RSP REAL(NSP)
real working storage divided up into various arrays
needed by the subroutines; ISP and RSP should be
equivalenced.

B REAL(LDB,M)
right-hand border to matrix A in matrix M.

LDB INTEGER
the leading dimension of the array B. LDB >= N.

CT REAL(LDC,N)
bottom border to matrix A in matrix M

LDC INTEGER
the leading dimension of the array CT. LDC >= M.
REAL(LDD,M)  
lower right-hand entries of M

INTEGER  
the leading dimension of the array D.  LDD >= M.

REAL(N)  
right-hand side to solve with

REAL(M)  
used to hold Vd, Wd, psiT B, psi and phi

REAL(LDW1,M+4)  
LDW1 >= N
used to hold E, g',psiT f and delta, and pivot indices for E.

INTEGER  
the leading dimension of the array WORK1.  LDW1 >= N.

REAL(LDW2,M+4)  
LDW2 >= M+1
used to hold E, g',psiT f and delta, and pivot indices for E.

INTEGER  
the leading dimension of the array WORK2.  LDW2 >= M+1.

CHARACTER*6  
indicates which inputs are the same as in the last call
to SGEDBE.  If there was no such call, set JOB =
' ' or 'a ' (see below).  Otherwise, JOB contains
as many of the following apply:
'A' if A stays the same
'S' if A is new but already factored by SGECO or SGEEA
'B' if B stays the same
'C' if CT stays the same
'D' if D stays the same
'F' if F stays the same
'G' if G stays the same

on exit:

REAL(NSP)  
the last 2n positions of RSP contain approximate
left and right null vectors for A if ESP > 2*N.

INTEGER  
if sufficient storage was available to perform the
symbolic factorization (CSFC), then ESP is set to the
amount of excess storage provided (negative if
insufficient storage was available to perform the
numeric factorization (CNFC)).
if ESP > 2*N, then those last 2n position of RSP will
contain approximate left and right null vectors for A.

\[ X \text{ REAL(N)} \]
\[ Y \text{ REAL(M+1)} \]
solution vector

\[ \text{WORK1 REAL(LDWI,M+4) \ LDWI} \geq N \]
used to hold Vd, Wd, psiT B, psi and phi

\[ \text{WORK2 REAL(LDW2,M+4) \ LDW2} \geq M+1 \]
used to hold E, g', psiT f and delta, and pivot indices for E.

Savings on storage:
the following pairs of inputs may be equivalent:
\[ (X,F) \text{ (Y,Q) (B,WORK1) (D,WORK2)} \]
in general if equivalent storage is used, then a change in one
of the inputs in either the left-hand-side group or the right-
hand-side group requires that the entire group be re-entered.
Specific exceptions to this rule can be determined by examining
the algorithm.

\begin{verbatim}
INTEGER N,M, LDB,LDC,LDD,LDWI,LDW2, AJOB, LRATIO
INTEGER P(N),IP(N), IA(N),JA(*), NSP,ISP(NSP),ESP
REAL A(*),RSP(NSP)
REAL B(LDB,M),CT(LDC,N),D(LDD,M), F(N),G(M), X(N),Y(M)
REAL WORK1(LDW1,M),WORK2(LDW2,M), DELTA
CHARACTER*6 JOB
LOGICAL NEWA,NEWB,NEWC,NEWD,NEWF,NEWG

the following constants are used to partition WORK1 and WORK2
into their various vectors; MP1 stands for the "extra" row and
column added to D in forming E. WORK1 is primarily used for Vd,
and WORK2 for E
\end{verbatim}

\begin{verbatim}
INTEGER MP1,CB,WD,CF,PSI,PHI,GP,IPVT,ALPHA
DATA LRATIO /1/
MP1 = M + 1
CB = MP1
WD = CB + 1
CF = MP1
PSI = WD + 1
PHI = PSI
GP = MP1 + 1
IPVT = GP + 1
ALPHA = MP1

AJOB = 0
IF (INDEX(JOB,'A') .NE. 0) AJOB = 2
IF (INDEX(JOB,'S') .NE. 0) AJOB = 1
NEWA = (AJOB .NE. 2)
\end{verbatim}
NEWB = (INDEX(JOB,'B') .EQ. 0)  
NEWC = (INDEX(JOB,'C') .EQ. 0)  
NEWD = (INDEX(JOB,'D') .EQ. 0)  
NEWF = (INDEX(JOB,'F') .EQ. 0)  
NEWG = (INDEX(JOB,'G') .EQ. 0)  

C

Algorithm:

C factor A, compute psi, phi, delta
C compute deflated solution to A V = B
C compute deflated solution to A w = f
C build E: | (D - cT Vd) (cT phi) |
C | CbT delta |
C build g': | g - cT Wd |
C | Cf |
C solve E | y | = g' for y
C | alpha |
C x = Wd - Vd y + alpha phi

C if AJOB = 0 or 1, or B is new, we start by solving A Vd = B;
this may imply factoring A, and/or computing psi, phi and delta
IF (NEWA .OR. NEWB) THEN

C for the first element of Vd, AJOB will tell sgeDF what to do
CALL SYSDF (N, P, IP, IA, JA, A, NSP, ISP, RSP, ESP, B(1,1)).
* WORK1(1,PSI),DELTA, WORK1(1,1),WORK1(1,CB),AJOB

C compute remaining columns of Vd using results of first call
IF (M .GT. 1) THEN
  DO 10 I = 2, M
    CALL SYSDF (N, P, IP, IA, JA, A, NSP, ISP, RSP, ESP, B(1,1)).
    * WORK1(1,PSI),DELTA, WORK1(1,1),WORK1(1,CB),2
  CONTINUE
10  ENDIF
ENDIF

C We must recompute Wd and Cf if A or F have changed
IF (NEWA .OR. NEWF)
* CALL SYSDF (N, P, IP, IA, JA, A, NSP, ISP, RSP, ESP, F,
* WORK1(1,PSI),DELTA, WORK1(1,WD),WORK2(CF,GP),2

C build and factor E
IF (NEWA .OR. NEWB .OR. NEWC .OR. NEWD) THEN
  CALL SCOPY (M, WORK1(1,CB), 1, WORK2(MP1,1), LDW2)
  WORK2(MP1,MP1) = DELTA
  DO 30 I = 1, M
  CONTINUE
  CALL SYSDF (N, P, IP, IA, JA, A, NSP, ISP, RSP, ESP, E,
  * WORK1(1,PSI),DELTA, WORK1(1,WD),WORK2(CF,GP),2

C compute D - cT Vd, column by column

77
DO 20 J = 1,M
    WORK2(I,J) = D(I,J) - SDOT(N, CT(I,1),LDC, WORK1(I,J),1)
20  CONTINUE

C compute cT PHI element by element
    WORK2(I,MP1) = SDOT(N, CT(I,1),LDC, WORK1(I PHI),1)
30  CONTINUE

C factor E
    CALL SGEFA (WORK2,LDW2,MP1,WORK2(1,IPVT),INFO)
ENDIF

C g' depends on a lot of things
IF (NEWA .OR. NEWC .OR. NEWF .OR. NEWG) THEN
    DO 40 I = 1,M
        WORK2(I,GP) = G(I) - SDOT(N, CT(I,1),LDC, WORK1(I,WD),1)
40  CONTINUE
ENDIF

C compute x and y
    CALL SCOPY (MP1, WORK2(1,GP),1, Y,1)
    CALL SGEFL (WORK2,LDW2,MP1,WORK2(1,IPVT), Y, 0)
    DO 50 I = 1,N
        X(I) = WORK1(I,WD) - SDOT(M, WORK1(I,1),LDW1, Y,1)
50  CONTINUE
    CALL SAXPY (N,Y(ALPHA), WORK1(1,PHI),1, X,1)
    WORK2(1,IPVT+1) = DELTA

END

C SUBROUTINE SYSDF (N, P, IP, IA, JA, A, NSP, ISP, RSP, ESP, *
                    RHS, PSI, DELTA, Z0, CP, JOB)
C computes the deflated decomposition of A z = p, returning
C solution in the form:
C
C   z = z + phi (c / delta)
C       d   p
C
C arguments are the same as for SYSDBE except:
C on entry:
C RHS REAL(N)
C contains rhs to system of equations
C PSI REAL(N)
P phi REAL(N)
left and right null vectors to matrix A
(only on entry if JOB = 2)
C DELTA REAL
smallest singular value for matrix A
(only on entry if JOB = 2)
C JOB INTEGER
JOB = 0: start the deflation algorithm from scratch; i.e.,
it factors the matrix, performs inverse iteration to
determine PSI, PHI and DELTA, and then computes the
deflated solution.
JOB = 1: assume that A has already been factored by CDRV
(or a previous call to SYC[DG]BE) and continue
from there.
JOB >= 1: additionally, PSI, PHI and DELTA have already
been computed.

ON exit:
C PSI REAL(N)
P phi REAL(N)
left and right null vectors to matrix A
C DELTA REAL
smallest singular value for matrix A
C ZD REAL(N)
deflated solution to system AX = P
Note that P and ZD may be the same vector
C CP REAL
psit p

INTEGER N, P(N), IP(N), IA(N), JA(1), NSP, ISP(1), ESP, JOB, IJOB, FLAG
REAL A(1), RSP(NSP), RHS(N), PSI(N), ZD(N), CP
REAL DELTA, PSITP, SV
LOGICAL TRANS

IJOB = JOB
IF (IJOB .GE. 10) IJOB = IJOB - 10

IF (IJOB .EQ. 0)
* CALL SDRV(N, P, IP, IA, JA, A, PSI, PSI,
*            NSP, ISP, RSP, ESP, 1, FLAG)
IF (IJOB .LE. 1)
  CALL SYSI2 (N, P, IP, IA, JA, A, NSP, ISP, RSP, ESP,
              PSI, DELTA, 0, 3)

C
C    A Zd = p - (psiT p) psi ; solve for Zd ; Cp is approx (psiT p)
CP = SDOT (N, RHS, 1, PSI, 1)
CALL SCOPY (N, RHS, 1, ZD, 1)
CALL SAXPY (N, -CP, PSI, 1, ZD, 1)
CALL SDRV (N, P, IP, IA, JA, A, ZD, ZD, NSP, ISP, RSP, ESP, 3, FLAG)
C
C    orthogonalize Zd with respect to psi
CALL SAXPY (N, -SDOT (N, PSI, 1, ZD, 1), PSI, 1, ZD, 1)

30 CONTINUE
END

SUBROUTINE SYSI2
*(N, P, IP, IA, JA, A, NSP, ISP, RSP, ESP, PSI, DELTA, JOB, ITER)
C
C    computes an approximate null vector of A by applying
C    the inverse iteration algorithm described in T. F. Chan. *Deflated
C    Decomposition of Solutions of Nearly Singular Systems.* SIAM J.
C
C    arguments are the same as for SYSDBE except:
C
C    on entry:
C
C    JOB     INTEGER
C      - if an approximate null vector is already known, the user
C      may pass it to SYSI2. JOB indicates where to find it.
C      JOB = 0 : no initial guess
C      JOB <> 0 : approximate null vector is passed in PSI
C
C    ITER    INTEGER
C      - governs how many iterations are performed
C      ITER = 0 : continue iterating until PSI and PHI converge
C      on accurate values. If M is nearly singular,
C      this usually occurs with 2 or 3 iterations.
C      ITER > 0 : do up to ITER many iterations.
C
C    on exit:
C
C    PSI, PHI REAL (N)
on output, contain the null vector of the matrix A.

REAL RESOL
PARAMETER (RESOL = .0001)
INTEGER N, P(I), IP(I), IA(I), JA(I), NSP, ISP(I), ESP, JOB, ITER
INTEGER FLAG
REAL A(I), RSP(I), PSI(I), SV
REAL OLDLEN, NEWLEN

IF (JOB .EQ. 0) THEN
  no initial guess; fill PSI with 1's
  DO 10 I = 1, N
   PSI(I) = 1.
  10 CONTINUE
ENDIF

PSI now contains initial guess; normalize it
psi' = psi' / ||psi'||
NEWLEN = SNRM2(N, PSI, 1)
CALL SSCAL (N, 1/NEWLEN, PSI, 1)

main loop of routine
IINC = 0
IF (ITER .NE. 0) IINC = 1
  I = IINC

repeat until convergence
50 CONTINUE

Apsi' = psi
CALL SDRV (N, P, IP, IA, JA, A, PSI, PSI, NSP, ISP, RSP, ESP, 3, FLAG)

psi' = psi' / ||psi'||
OLDLEN = NEWLEN
NEWLEN = SNRM2(N, PSI, 1)
CALL SSCAL (N, 1/NEWLEN, PSI, 1)

increment counter
  I = I + IINC

end
IF (I .LE. ITER .AND. ABS(1/OLDLEN - 1/NEWLEN) .GT. RESOL) *
  GOTO 50

do psi' once more
CALL SDRV (N, P, IP, IA, JA, A, PSI, PSI, NSP, ISP, RSP, ESP, 3, FLAG)
delta = 1/||psi'||

DELTA = 1/SNRM2(N, PSI, 1)
CALL SSCAL (N, DELTA, PSI, 1)

END

SUBROUTINE SYSBE
* (N,M, P,IP, IA, JA, A, NSP, ISP, RSP, ESP, B, LDB,
  * CT, LDC, D, LDD, F, G, X, Y, WORK1, LDW1, WORK2, LDW2, JOB)

the ordinary (undeflated) block elimination algorithm.

(All arguments are the same as in SYSDE.)

INTEGER N,M, LDB, LDC, LDD, LDW1, LDW2, AJOB, LRATIO
INTEGER P(N), IP(N), IA(N), JA(*), NSP, ISP(NSP), ESP
REAL A(*), RSP(NSP)
REAL B(LDB,M), CT(LDC,N), D(LDD,M), F(N), G(M), X(N), Y(M)
REAL WORK1(LDW1,M), WORK2(LDW2,M), DELTA
CHARACTER*6 JOB
LOGICAL NEWA, NEWB, NEWC, NEWD, NEWF, NEWG

INTEGER MP1, IPVT
MP1 = M + 1
IPVT = MP1 + 1

AJOB = 0
IF (INDEX(JOB,'A') .NE. 0) AJOB = 2
IF (INDEX(JOB,'a') .NE. 0) AJOB = 1
NEWA = (AJOB .NE. 2)
NEWB = (INDEX(JOB,'B') .EQ. 0)
NEWC = (INDEX(JOB,'C') .EQ. 0)
NEWD = (INDEX(JOB,'D') .EQ. 0)
NEWF = (INDEX(JOB,'F') .EQ. 0)
NEWG = (INDEX(JOB,'G') .EQ. 0)

solve A V = B for V
IF (AJOB .EQ. 0)
  * CALL SDRV (N, P, IP, IA, JA, A, X, X, NSP, RSP, ESP, 1, FLAG)

IF (NEWA .OR. NEWB) THEN
  DO 10 I = 1, M
    CALL SDRV (N, P, IP, IA, JA, A, B(1,I), WORK1(I,I),
2
* NSP,ISP,RSP,ESP, 1, FLAG)
10 CONTINUE
ENDIF

C
C solve A w = f for w
IF (NEWA .OR. NEWF)
* CALL SDRV (N, P, IP, IA, JA, A, F, WORK1(I,MP1),
* NSP,ISP,RSP,ESP, 1, FLAG)

C
C compute E (= D - cT V)
IF (NEWA .OR. NEWB .OR. NEWC .OR. NEWD) THEN
   DO 30 I = 1,M
      DO 20 J = 1,M
          WORK2(I,J) = D(I,J) - SDOT(N, CT(I,1),LDC, WORK1(I,J),1)
20 CONTINUE
30 CONTINUE
CALL SGEFA (WORK2,LDW2,M,WORK2(1,IPVT),INFO)
ENDIF

C
C compute g' (= g - cT w)
IF (NEWA .OR. NEWC .OR. NEWF .OR. NEWG) THEN
   DO 40 I = 1,M
      WORK2(I,MP1) = G(I) - SDOT(N, CT(I,1),LDC, WORK1(I,MP1),1)
40 CONTINUE
ENDIF

C
C solve for y
CALL SCOPY (M, WORK2(1,MP1),1, Y,1)
CALL SGESL (WORK2,LDW2,M,WORK2(1,IPVT),Y,0)

C
C compute x
DO 50 I = 1,N
    X(I) = WORK1(I,MP1) - SDOT(M, WORK1(I,1),LDW1, Y,1)
50 CONTINUE
CONTINUE
END