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 *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-ACO2-81ER10996, and by the Army Research Office under contract DAAG-83-0177.
Keywords: *linear systems, bordered systems, singular matrices, deflation, LINPACK, YSMP*

Table of Contents

1 Introduction 1

2 Algorithms 2

2.1 Deflated Decomposition of Solutions of Nearly Singular Systems 2

2.2 Deflated Block Elimination 3

3 Implementation Notes 4

4 Sample Driver 6

Bibliography 15

5 Source Code Listings 16

5.1 SGEDBE 16

5.2 SGBDBE 27

5.3 SGTDBE 38

5.4 SYNDBE 43

5.5 SYCDBE 57

5.6 SYSDBE 71

1 Introduction

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

$$M \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} A & B \\ C^T & D \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} f \\ g \end{bmatrix} \tag{1.1}$$

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.

$$(a) \begin{bmatrix} x & & & x \\ & x & 0 & x \\ & 0 & \ddots & \vdots \\ x & x & \dots & x \end{bmatrix} \qquad (b) \begin{bmatrix} x & x & \dots & x \\ x & x & 0 & \\ \vdots & 0 & \ddots & \\ x & & & x \end{bmatrix}$$

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 $(D - C^T V)y = (g - C^T w)$.
> Compute $x = w - Vy$.

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

$$Az = p \tag{2.1}$$

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

$$z = \sum_{i=1}^{n-1} \left(\frac{u_i^T p}{\sigma_i}\right) v_i + \left(\frac{u_n^T p}{\sigma_n}\right) v_n,$$

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

$$z = z_d + \frac{c_p}{\delta}\phi, \tag{2.2}$$

where

$$z_d \equiv \sum_{i=1}^{n-1} \left(\frac{u_i^T p}{\sigma_i}\right) v_i,$$

$\delta \equiv \sigma_n$, $\phi \equiv v_n$, $\psi \equiv u_n$, and $c_p \equiv \psi^T p$. 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 ϕ and a deflated part z_d purged of ϕ.

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:

$$P_\psi A z_d = P_\psi p$$

$$P_\phi z_d = z_d,$$

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.

Algorithm DF:
Given ϕ, ψ, and δ, computes the deflated solution z_d of $Az = p$:

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

The singular vectors ϕ and ψ and the singular value δ can be computed, for example, by the following inverse iteration algorithm:

Algorithm II

Make an initial guess for ψ
Repeat until convergence
  Solve $A\phi' = \psi$ for ϕ'
  $\phi = \phi'/||\phi'||$
  Solve $A^T\psi' = \phi$ for ψ'
  $\psi = \psi'/||\psi'||$
End
Solve $A\phi' = \psi$ for ϕ' 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 α, β 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 ϕ, ψ and δ.
Compute deflated solution V of $AV = B$.
Compute deflated solution w of $Aw = f$.
Solve the $m+1$ by $m+1$ linear system:

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

where

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

Assemble the solution by substituting into the following formula:

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

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, ψ and ϕ. 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: $W = \hat{W} - \lambda_{\max}(\hat{W})I$, where

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

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

and 3) matrix P, of the form $P = \Delta_h - \lambda_{\min}(\Delta_h)I$, where Δ_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.

Matrix	DBE routine	Error without deflation	Error with deflation
T	SGE	5.70781400	0.00013622
W	SGB	1.94438300	0.00001813
W	SGT	1.49100000	0.00001882
P	SYN	***	0.00000368
P	SYS	5.36235400	0.00000174

*** 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
c
c     driver for testing the routines in DBEPACK
c
      parameter (lda = 50)

      real alpha,lambda,epsilo

      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'
c
c     set up border and target solution
c
      do 15 j = 1,lda
         do 10 k = 1,m
            b(j,k)  = ranf(seed)
            cT(k,j) = ranf(seed)
 10      continue
```

```
         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')
c.....................................................................

      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.....................................................................

      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, ml,mu)
      call sgbbe (n,m, a,lda, ml,mu,ipvt, b,lda,cT,lda,d,lda,
     *             f,g, xp,y, work1,lda,work2,lda, job1)
      write (6,2000)
```

```
      call compare (n,m,x,xp,y)
c
      call sgbdbe (n,m, a,lda, ml,mu,ipvt, b,lda,cT,lda,d,lda,
     *              f,g, xp,y, work1,lda,work2,lda, job2)
      write (6,2010)
      call compare (n,m,x,xp,y)
c
c     repeat procedure with Linpack routines for tridiagonal matrices
c
      call tbW (n, l,di,u)
      call sgtbe (N,M, L,DI,U, B,LDA, CT,LDA, D,LDA,
     *              F,G, Xp,Y, WORK1,LDA,WORK2,LDA, job1)
      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, job2)
      write (6,2010)
      call compare (n,m,x,xp,y)

c..................................................................

      write (6,1010)
 1010 format (//' building matrix P'/)
      n = 16
      call ebP (a,lda,n)
      call brhs (n,m, a,lda, b,lda,cT,lda,d,lda, x, f,g)
c
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, job1)
      write (6,2000)
      call compare (n,m,x,xp,y)
c
      call syndbe (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, job1)
      write (6,2010)
      call compare (n,m,x,xp,y)
c
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, job1)
      write (6,2000)
      call compare (n,m,x,xp,y)
c
```

```
      call sysdbe (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,x,xp,y)

      end

C-----------------------------------------------------------------------
C
      subroutine compare (n,m,x,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
      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)
      enddo

      call eaxb (a,lda,n+m,x,f)
      call scopy (m, f(n+1),1, g,1)
```

```
      end

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,epsilo

      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-1) = -1.
            a(i-1,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
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,ml + 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,l,d,u)
C
C     builds matrix W in SGT-format
C
      integer n
      real l(n),d(n),u(n)

      n = 21
      do 60 i = 1,n
         l(i)  = 1.
         d(i) = 11. - i - 10.7461942
         u(i)  = 1.
 60   continue

      end


C-----------------------------------------------------------------------
C
      subroutine nbP (n, ia,ja,a)
C
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
      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
```

```
      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
```

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

References

[1] T.F. Chan, Techniques for Large Sparse Systems Arising from Continuation Methods, T. Kupper, H. Mittelmann and H. Weber eds., *Numerical Methods for Bifurcation Problems,* International Series of Numerical Math., Vol. 70, Birkhauser Verlag, Basel, 1984, pp. 116–128.

[2] ———, *Deflated Decomposition of Solutions of Nearly Singular Systems,* Siam J. Numer. Anal., 1984, 21/4 August (1984), pp. 738–754.

[3] ———, *Deflation Techniques and Block-Elimination Algorithms for Solving Bordered Singular Systems,* Siam J. Sci. Stat. Comp., 5/1 March (1984).

[4] Tony F. Chan and Diana C. Resasco, *Generalized Deflated Block-Elimination,* Technical Report YALEU/DCS/RR-337, Dept. of Computer Science, Yale Univ., 1985.

[5] J.J. Dongarra, J.R. Bunch, C.B. Moler and G.W. Stewart, *LINPACK User's Guide,* SIAM, Philadelphia, 1979.

[6] S.C. Eisenstat, M.C. Gursky, M.H. Schultz and A.H. Sherman, *Yale sparse matrix package II: The nonsymmetric codes,* Technical Report 114, Dept. of Computer Science, Yale Univ., 1977.

[7] ———, *Yale sparse matrix package I: The symmetric codes,* International Journal for Numerical Methods in Engineering, 18 (1982), pp. 1145–1151.

5 Source Code Listings

5.1 SGEDBE

```
C
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     Computer Science, Yale Univ., 1985.
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/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
C><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><
C\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
C
      SUBROUTINE SGEDBE
     *           (N,M, A,LDA, IPVT, B,LDB, CT,LDC ,D,LDD, F,G, X,Y,
     *            WORK1,LDW1, WORK2,LDW2, JOB)
C
C     the deflated block elimination algorithm
C
C     arguments:
C
C     on entry:
C
C     N        INTEGER
C              the order of the matrix A
C
C     M        INTEGER
C              the order of the borders to A in M
C
C     A        REAL(LDA,N)
C              the matrix to be factored.
C
C     LDA      INTEGER
C              the leading dimension of the array A.  LDA >= N.
```

```
C
C     IPVT    INTEGER(N+M)
C             an integer vector of pivot indices.  The last m spaces are
C             required for working space
C
C     B       REAL(LDB,M)
C             right-side border to matrix A in matrix M
C
C     LDB     INTEGER
C             the leading dimension of the array B.  LDB >= N.
C
C     CT      REAL(LDC,N)
C             bottom border to matrix A in matrix M
C
C     LDC     INTEGER
C             the leading dimension of the array CT.  LDC >= M.
C
C     D       REAL(LDD,M)
C             lower right-hand entries of M
C
C     LDD     INTEGER
C             the leading dimension of the array D.  LDD >= M.
C
C     F       REAL(N)
C     G       REAL(M)
C             right-hand side to solve with
C
C     WORK1   REAL(LDW1,M+4)
C             used to hold Vd, Wd, psiT B, psi and phi
C
C     LDW1    INTEGER
C             the leading dimension of the array WORK1.  LDW1 >= N.
C
C     WORK2   REAL(LDW2,M+3)
C             used to hold E, g',psiT f and delta
C
C     LDW2    INTEGER
C             the leading dimension of the array WORK2.  LDW2 >= M+1.
C
C     JOB     CHARACTER*6
C             indicates which inputs are the same as in the last call
C             to SGEDBE.  If there was no such call, set JOB =
C             '      ' or 'a     ' (see below).  Otherwise, JOB contains
C             as many of the following apply:
C              'A' if A stays the same
C              'S' if A is new but already factored by SGECO or SGEFA
C              'B' if B stays the same
C              'C' if CT stays the same
C              'D' if D stays the same
```

```
C              'F' if F stays the same
C              'G' if G stays the same
C
C     on exit:
C
C     A         REAL(LDA,N)
C               contains an upper triangular matrix and
C               the multipliers which were used to obtain it.  The
C               factorization can be written A = L*U, where L is the
C               product of permutation and unit lower triangular ma-
C               trices and U is upper triangular.
C
C     IPVT      INTEGER(N+M)
C               an integer vector of pivot indices.  The last m spaces are
C               required for working space.
C
C     X         REAL(N)
C     Y         REAL(M+1)
C               solution vector (Y(M+1) is an extra storage location).
C
C     WORK1     REAL(LDW1,M+4)
C               used to hold Vd, Wd, psiT B, psi and phi
C
C     WORK2     REAL(LDW2,M+3)
C               used to hold E, g',psiT f and delta
C
C     Savings on storage:
C     the following pairs of inputs may be equivalent:
C         (X,F)  (Y,G)  (B,WORK1)  (D,WORK2)
C     In general, if equivalent storage is used, then a change in any
C     of the inputs in either of the groups (A,B,C,D) or (F,G)
C     requires that the entire group be re-entered.  Specific
C     exceptions to this rule can be determined by examining
C     the algorithm.
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
      LOGICAL NEWA,NEWB,NEWC,NEWD,NEWF,NEWG
C
C     the following constants are used to partition WORK1 and WORK2
C     into their various vectors;  MP1 stands for the "extra" row and
C     column added to D in forming E.  WORK1 is primarily used for Vd,
C     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
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)
C
C     Algorithm:
C
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
C
C     if AJOB = 0 or 1, or B is new,  we start by solving A Vd = B;
C     this may imply factoring A, and/or computing psi, phi and delta
      IF (NEWA .OR. NEWB) THEN
C
C        for the first element of Vd, AJOB will tell sgeDF what to do
         CALL SGEDF (A,LDA,N,IPVT, B(1,1), WORK1(1,PSI),WORK1(1,PHI),
     *               DELTA, WORK1(1,1),WORK1(1,CB),AJOB)
C
C        compute remaining columns of Vd using results of first call
         IF (M .GT. 1) THEN
            DO 10 I = 2,M
               CALL SGEDF (A,LDA,N,IPVT, B(1,I), WORK1(1,PSI),
     *                WORK1(1,PHI), DELTA, WORK1(1,I),WORK1(I,CB),2)
 10         CONTINUE
         ENDIF
      ENDIF
C
C     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(1,PSI),WORK1(1,PHI),
     *              DELTA, WORK1(1,WD),WORK2(CF,GP),2)
      ENDIF
C
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
C
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(1,J),1)
 20         CONTINUE
C
C           compute cT PHI element by element
            WORK2(I,MP1) = SDOT(N, CT(I,1),LDC, WORK1(1,PHI),1)
 30      CONTINUE
C
C        factor E
         CALL SGEFA (WORK2,LDW2,MP1,IPVT(N+1),INFO)
      ENDIF
C
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(1,WD),1)
 40      CONTINUE
      ENDIF
C
C     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

C/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
C><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><
C\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
C
      SUBROUTINE SGEDF(A,LDA,N,IPVT,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             d          p
C
C     arguments are the same as for SGEDBE 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
C              smallest singular value for matrix A
C              (only on entry if JOB >= 2)
C
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 SYN[D]BE), and
C                        continue from there.
C              JOB > 1 : additionally, PSI, PHI and DELTA have already
C                        been computed.
C
C     on exit:
C
C     PSI      REAL(N)
C     PHI      REAL(N)
C              left and right null vectors to matrix A
C
C     DELTA    REAL
C              smallest singular value for matrix A
C
C     ZD       REAL(N)
C              deflated solution to system A z = p
C              Note that P and ZD may be the same vector
C
C     CP       REAL
C              phiT p.
C
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        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 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
C                                 T
C     Perform deflation with A
C         T
C        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 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/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
C><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><
C\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
C
      SUBROUTINE SGEII (A,LDA,N,IPVT,PSI,PHI,DELTA,JOB,ITER)
```

```
C
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     Numer. Anal., vol. 21 no. 4 (August, 1984)
C
C     arguments are the same as for SGEDF 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 SGEII.  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 A is nearly singular
C                         this usually occurs with 2 or 3 iterations.
C              ITER > 0 : do up to ITER many iterations.
C
C                                   resol = resolution of convergence
      REAL RESOL
      PARAMETER (RESOL = .0001)
      INTEGER LDA,N,IPVT(N),JOB,ITER
      REAL A(LDA,N),PSI(N),PHI(N),DELTA,PSILEN,PHILEN
C
C     note:  since SGESL destroys the rhs given to it, PSI and PHI
C            here are both computed in PSI, until the last step
C
C
      IF (JOB .EQ. 0) THEN
C
C        no initial guess; fill PSI with 1's
         DO 10 I = 1,N
            PSI(I) = 1.
 10      CONTINUE

      ELSEIF (JOB .EQ. 1) THEN
C
C        initial guess is in PHI;
C        move to PSI, then solve for initial PSI
C        phi' = phi' / ||phi'||
         CALL SCOPY (N,PHI,1,PSI,1)
         PHILEN = SNRM2(N,PSI,1)
         CALL SSCAL (N,1/PHILEN,PSI,1)
```

```
C
C          T
C         A  psi' = phi'
          CALL SGESL (A,LDA,N,IPVT,PSI,1)

      ENDIF
C
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 SGESL (A,LDA,N,IPVT,PSI,0)
C
C         phi' = phi' / ||phi'||
          PHILEN = SNRM2(N,PSI,1)
          CALL SSCAL (N,1/PHILEN,PSI,1)
C
C          T
C         A  psi' = phi'
          CALL SGESL (A,LDA,N,IPVT,PSI,1)
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 SCOPY (N,PSI,1,PHI,1)
      CALL SGESL (A,LDA,N,IPVT,PHI,0)
C
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)

      END

C/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
C><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><
C\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
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
C     solve A V  = B for V
      IF (AJOB .EQ. 0) CALL SGEFA (A,LDA,N,IPVT,INFO)
      IF (NEWA .OR. NEWB) THEN
         DO 10 I = 1,M
            CALL SCOPY (N, B(1,I),1, WORK1(1,I),1)
            CALL SGESL (A,LDA,N,IPVT,WORK1(1,I),0)
 10      CONTINUE
      ENDIF
C
C     solve A w = f for w
      IF (NEWA .OR. NEWF) THEN
         CALL SCOPY (N, F,1, WORK1(1,MP1),1)
         CALL SGESL (A,LDA,N,IPVT,WORK1(1,MP1),0)
```

```
      ENDIF
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(1,J),1)
 20         CONTINUE
 30      CONTINUE
         CALL SGEFA (WORK2,LDW2,M,IPVT(N+1),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(1,MP1),1)
 40      CONTINUE
      ENDIF
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
```

5.2 SGBDBE

```
C
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     Computer Science, Yale Univ., 1985.
C
C     This set of routines calls LINPACK's SGE- and SGB- routines, and
C     the SBLAs.  Implemented by Thomas A. Grossi, Yale University, 1985.
C
C/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
C><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><<
C\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
C
      SUBROUTINE SGBDBE
     *           (N,M, A,LDA,ML,MU, IPVT, B,LDB, CT,LDC ,D,LDD, F,G,
     *            X,Y,  WORK1,LDW1, WORK2,LDW2, JOB)
C
C     the deflated block elimination algorithm
C
C     arguments:
C
C     on entry:
C
C     N       INTEGER
C             the order of the matrix A
C
C     M       INTEGER
C             the order of the borders to A in M
C
C     A       REAL(LDA,N)
C             the matrix to be factored.
C
C     LDA     INTEGER
C             the leading dimension of the array A.  LDA >= N.
C
C     ML      INTEGER
C             number of diagonals below the main diagonal in A.
```

```
C             0 <= ML < N .
C
C     MU      INTEGER
C             number of diagonals above the main diagonal in A.
C             0 <= MU < N .
C             more efficient if  ML <= MU .
C
C     IPVT    INTEGER(N+M)
C             an integer vector of pivot indices.  The last M spaces are
C             required for working space
C
C     B       REAL(LDB,M)
C             right-side border to matrix A in matrix M
C
C     LDB     INTEGER
C             the leading dimension of the array B.  LDB >= N.
C
C     CT      REAL(LDC,N)
C             right and bottom borders to matrix A in matrix M
C
C     LDC     INTEGER
C             the leading dimension of the array CT.  LDC >= M.
C
C     D       REAL(LDD,M)
C             lower right-hand entries of M.
C
C     LDD     INTEGER
C             the leading dimension of the array D.  LDD >= M.
C
C     F       REAL(N)
C     G       REAL(M)
C             right-hand side to solve with
C
C     WORK1   REAL(LDW1,M+4)  LDW1 >= N
C             used to hold Vd, Wd, psiT B, psi and phi
C
C     LDW1    INTEGER
C             the leading dimension of the array WORK1.  LDW1 >= N.
C
C     WORK2   REAL(LDW2,M+3)
C             used to hold E, g',psiT f and delta
C
C     LDW2    INTEGER
C             the leading dimension of the array WORK2.  LDW2 >= M+1.
C
C     JOB     CHARACTER*6
C             indicates which inputs are the same as in the last call
C             to SGBDBE.  If there was no such call, set JOB =
C             '      ' or 'a     ' (see below).  Otherwise, JOB contains
```

```
C           as many of the following apply:
C            'A' if A stays the same
C            'S' if A is new but already factored by SGBCO or SGBFA
C            'B' if B stays the same
C            'C' if CT stays the same
C            'D' if D stays the same
C            'F' if F stays the same
C            'G' if G stays the same
C
C     on exit:
C
C     A       real(lda,n)
C             contains an upper triangular matrix and
C             the multipliers which were used to obtain it.  The
C             factorization can be written A = L*U, where L is the
C             product of permutation and unit lower triangular ma-
C             trices and U is upper triangular.
C
C     IPVT    INTEGER(N+M)
C             an integer vector of pivot indices.  The last m spaces are
C             required for working space.
C
C     X       REAL(N)
C     Y       REAL(M+1)
C             solution vector
C
C     WORK1   REAL(LDW1,M+4)
C             used to hold Vd, Wd, psiT B, psi and phi
C
C     WORK2   REAL(LDW2,M+3)
C             used to hold E, g',psiT f and delta
C
C     Savings on storage:
C     the following pairs of inputs may be equivalent:
C         (X,F)  (Y,G)  (B,WORK1)  (D,WORK2)
C     In general, if equivalent storage is used, then a change in any
C     of the inputs in either of the groups (A,B,C,D) or (F,G)
C     requires that the entire group be re-entered.  Specific
C     exceptions to this rule can be determined by examining
C     the algorithm.
C
      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),DELTA
      REAL F(N),G(M), X(N),Y(M), WORK1(LDW1,M),WORK2(LDW2,M)
      LOGICAL NEWA,NEWB,NEWC,NEWD,NEWF,NEWG
C
C     the following constants are used to partition WORK1 and WORK2
C     into their various vectors;  MP1 stands for the "extra" row and
```

```
C     column added to D in forming E.  WORK1 is primarily used for Vd,
C     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
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)
C
C     Algorithm:
C
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
C
C     if AJOB = 0 or 1, or B is new,  we start by solving A Vd = B;
C     this may imply factoring A, and/or computing psi, phi and delta
      IF (NEWA .OR. NEWB) THEN
C
C        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(1,1),WORK1(1,CB),AJOB)
C
C        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,I),WORK1(I,CB),2)
```

```
 10           CONTINUE
            ENDIF
         ENDIF
C
C        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
C
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
C
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(1,J),1)
 20            CONTINUE
C
C              compute cT PHI element by element
               WORK2(I,MP1) = SDOT(N, CT(I,1),LDC, WORK1(1,PHI),1)
 30         CONTINUE
C
C           factor E
            CALL SGEFA (WORK2,LDW2,MP1,IPVT(N+1),INFO)
         ENDIF
C
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(1,WD),1)
 40         CONTINUE
         ENDIF
C
C        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

C/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
C><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><
```

```
C\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
C
      SUBROUTINE SGBDF
     *      (A,LDA,N,ML,MU,IPVT, 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               d          p
C
C     arguments are the same as for SGBDBE 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
C              smallest singular value for matrix A
C              (only on entry if JOB >= 2)
C
C     JOB      INTEGER
C              JOB = 0 : start the deflation algorithm from scratch; i.e.,
C                        it factors the matrix, perform inverse iteration to
C                        determine PSI, PHI and DELTA, and the computes the
C                        deflated solution.
C              JOB = 1 : assume that A has already been factored by SGBCO
C                        or SGBFA, (or by a previous call to SGB[D]BE)
C                        and continue from there.
C              JOB > 1 : additionally, PSI, PHI and DELTA have already
C                        been computed.
C
C     on exit:
C
C     PSI      REAL(N)
C     PHI      REAL(N)
C              left and right null vectors to matrix A
C
C     DELTA    REAL
C              smallest singular value for matrix A
C
C     ZD       REAL(N)
C              deflated solution to system A z = p
```

```
C             Note that P and ZD may be the same vector
C
C     CP      REAL
C             phiT p.
C
C
      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
C
C        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)
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
C                                 T
C     Perform deflation with A
C         T
C        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)
C
C        orthogonalize Zd wrt psi
         CALL SAXPY(N,-SDOT(N,PSI,1,ZD,1),PSI,1,ZD,1)

 30   CONTINUE
      END

C/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
C><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><
```

```
C\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
C
      SUBROUTINE SGBII (A,LDA,N,ML,MU,IPVT,PSI,PHI,DELTA,JOB,ITER)
C
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     Numer. Anal., vol. 21 no. 4 (August, 1984)
C
C     arguments are the same as for SGBDF 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 SGII.  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                                 resol = resolution of convergence
      REAL RESOL
      PARAMETER (RESOL = .0001)
      INTEGER LDA,N,IPVT(LDA,2),JOB,ITER
      REAL A(LDA,LDA),PSI(N),PHI(N),DELTA,PSILEN,PHILEN
C
C     note:  since SGBSL destroys the rhs given to it, PSI and PHI
C            here are both computed in PSI, until the last step
C
      IF (JOB .EQ. 0) THEN
C
C        no initial guess; fill PSI with 1's
         DO 10 I = 1,N
            PSI(I) = 1.
 10      CONTINUE

      ELSEIF (JOB .EQ. 1) THEN
C
C           initial guess is in PHI;
C           move to PSI, then solve for initial PSI
C           phi' = phi' / ||phi'||
            CALL SCOPY (N,PHI,1,PSI,1)
```

```
            PHILEN = SNRM2(N,PSI,1)
            CALL SSCAL (N,1/PHILEN,PSI,1)
C
C            T
C           A  psi' = phi'
            CALL SGBSL (A,LDA,N,ML,MU,IPVT,PSI,1)

      ENDIF
C
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 SGBSL (A,LDA,N,ML,MU,IPVT,PSI,0)
C
C        phi' = phi' / ||phi'||
         PHILEN = SNRM2(N,PSI,1)
         CALL SSCAL (N,1/PHILEN,PSI,1)
C
C         T
C        A  psi' = phi'
         CALL SGBSL (A,LDA,N,ML,MU,IPVT,PSI,1)
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 -- this time for the record
      CALL SCOPY (N,PSI,1,PHI,1)
      CALL SGBSL (A,LDA,N,ML,MU,IPVT,PHI,0)
C
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)

      END

C/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
C><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><<
C\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
C
      SUBROUTINE SGBBE
     *           (N,M, A,LDA,ML,MU, 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 SGBDBE.
C
      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),DELTA
      REAL F(N),G(M), X(N),Y(M), WORK1(LDW1,M),WORK2(LDW2,M)
      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
C     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(1,I),1, WORK1(1,I),1)
            CALL SGBSL (A,LDA,N,ML,MU,IPVT,WORK1(1,I),0)
 10      CONTINUE
      ENDIF
C
C     solve A w = f for w
      IF (NEWA .OR. NEWF) THEN
```

```
         CALL SCOPY (N, F,1, WORK1(1,MP1),1)
         CALL SGBSL (A,LDA,N,ML,MU,IPVT,WORK1(1,MP1),0)
      ENDIF
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(1,J),1)
 20         CONTINUE
 30      CONTINUE
         CALL SGEFA (WORK2,LDW2,M,IPVT(N+1),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(1,MP1),1)
 40      CONTINUE
      ENDIF
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
```

5.3 SGTDBE

```
C
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     Computer Science, Yale Univ., 1985.
C
C     This set of routines calls LINPACK's SGE- and SGT- routines,
C     and the SBLAs.  Implemented by Thomas A. Grossi, Yale University, 1985.
C
C/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
C><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><
C\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
C
      SUBROUTINE SGTDBE
     *           (N,M, L,DI,U, B,LDB, CT,LDC ,D,LDD, F,G, X,Y,
     *            WORK1,LDW1, WORK2,LDW2, JOB)
C
C     the deflated block elimination algorithm
C
C     arguments:
C
C     on entry:
C
C     N        INTEGER
C              the order of the matrix A
C
C     M        INTEGER
C              the order of the borders to A in M
C
C     L        REAL(N)
C            . is the subdiagonal of the tridiagonal matrix.
C              L(2) through L(n) should contain the subdiagonal.
C              Unlike LINPACK, our routines do not change the
C              the contents of L.
C
C     DI       REAL(N)
C              is the diagonal of the tridiagonal matrix.
```

```
C           The contents of DI remain unchanged by our routines.
C
C     U     REAL(N)
C           is the superdiagonal of the tridiagonal matrix.
C           U(1) through U(n-1) should contain the superdiagonal.
C           The contents of U remain unchanged by our routines.
C
C     B     REAL(LDB,M)
C           right-hand border to the matrix A in matrix M
C
C     LDB   INTEGER
C           the leading dimensions of the array B.  LDB >= N.
C
C     CT    REAL(LDC,N)
C           bottom border to matrix A in matrix M
C
C     LDC   INTEGER
C           the leading dimension of the array CT.  LDC >= M.
C
C     D     REAL(LDD,M)
C           lower right-hand entries of M.
C
C     LDD   INTEGER
C           the leading dimension of the array D.  LDD >= M.
C
C     F     REAL(N)
C     G     REAL(M)
C           right-hand side to solve with
C
C     WORK1 REAL(LDW1,M+7)
C           used to hold Vd, Wd, psiT B, psi and phi, plus scratch space
C           for factoring A
C
C     LDW1  INTEGER
C           the leading dimension of the array WORK1.  LDW1 >= N.
C
C     WORK2 REAL(LDW2,M+3)
C           used to hold E, g',psiT f and delta, and a list of pivot
C           indices.
C
C     LDW2  INTEGER
C           the leading dimension of the array WORK2.  LDW2 >= M+1.
C
C     JOB   CHARACTER*6
C           indicates which inputs are the same as in the last call
C           to SGTDBE.  If there was no such call, set JOB =
C           '      ' or 'a     ' (see below).  Otherwise, JOB contains
C           as many of the following apply:
C            'A' if A stays the same
```

```
C             'S' if A is new but already factored by SGTCO or SGTFA
C             'B' if B stays the same
C             'C' if CT stays the same
C             'D' if D stays the same
C             'F' if F stays the same
C             'G' if G stays the same
C
C     on exit:
C
C     X        REAL(N)
C     Y        REAL(M+1)
C              solution vector
C
C     WORK1    REAL(LDW1,M+7)
C              used to hold Vd, Wd, psiT B, psi and phi, plus scratch space
C              for factoring A
C
C     WORK2    REAL(LDW2,M+3)
C              used to hold E, g',psiT f and delta
C
C     Savings on storage:
C     the following pairs of inputs may be equivalent:
C         (X,F)   (Y,G)   (B,WORK1)   (D,WORK2)
C     In general, if equivalent storage is used, then a change in any
C     of the inputs in either of the groups (A,B,C,D) or (F,G)
C     requires that the entire group be re-entered.  Specific
C     exceptions to this rule can be determined by examining
C     the algorithm.
C
      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
C
C     the following constants are used to partition WORK1 and WORK2
C     into their various vectors;  MP1 stands for the "extra" row and
C     column added to D in forming E.  WORK1 is primarily used for Vd,
C     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
```

```
      ALPHA = MP1
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)
C
C     Algorithm:
C
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
C
C     if AJOB = 0 or 1, or B is new,  we start by solving A Vd = B;
C     this may imply factoring A, and/or computing psi, phi and delta
      IF (NEWA .OR. NEWB) THEN
C
C        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)
C
C        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(I,CB),2)
 10         CONTINUE
         ENDIF
      ENDIF
C
C     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)
C
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)
```

5.4 SYNDBE

```
C
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     Computer Science, Yale Univ., 1985.
C
C     This set of routines calls YSMP's general routines with non-
C     -compressed storage, Linpack's SGE- routines, and the SBLAs.
C     Implemented by Thomas A. Grossi, Yale University, 1985.
C-----------------------------------------------------------------------
C
C     STORAGE SCHEME FOR THESE ROUTINES
C
C          The nonzero entries of the coefficient matrix M are stored
C     row-by-row in the array A.  To identify the individual nonzero
C     entries in each row, we need to know in which column each entry
C     lies.  The column indices which correspond to the nonzero entries
C     of M are stored in the array JA;  i.e., if  A(K) = M(I,J),  then
C     JA(K) = J.  In addition, we need to know where each row starts and
C     how long it is.  The index positions in JA and A where the rows of
C     M begin are stored in the array IA;  i.e., if M(I,J) is the first
C     nonzero entry (stored) in the I-th row and A(K) = M(I,J),  then
C     IA(I) = K.  moreover, the index in JA and A of the first location
C     following the last element in the last row is stored in IA(N+1).
C     thus, the number of entries in the I-th row is given by
C     IA(I+1) - IA(I),  the nonzero entries of the I-th row are stored
C     consecutively in
C             A(IA(I)),  A(IA(I)+1),  ..., A(IA(I+1)-1),
C     and the corresponding column indices are stored consecutively in
C             JA(IA(I)), JA(IA(I)+1), ..., JA(IA(I+1)-1).
C     for example, the 5 by 5 matrix
C                 | 1. 0. 2. 0. 0.|
C                 | 0. 3. 0. 0. 0.|
C             M = | 0. 4. 5. 6. 0.|
C                 | 0. 0. 0. 7. 0.|
C                 | 0. 0. 0. 8. 9.|
C     would be stored as
```

```
C                |1  2  3  4  5  6  7  8  9
C             ---+---------------------------
C             IA | 1  3  4  7  8 10
C             JA | 1  3  2  2  3  4  4  4  5
C              A | 1. 2. 3. 4. 5. 6. 7. 8. 9.
C
C     N       INTEGER
C             number of variables/equations.
C
C     A       INTEGER(*)
C             nonzero entries of the coefficient matrix A, stored
C             by rows.
C             size = number of nonzero entries in A.
C
C     IA      INTEGER(N+1)
C             pointers to delimit the rows in A.
C
C     JA      INTEGER(*)
C             column numbers corresponding to the elements of A.
C             size = size of A.
C
C
C          The rows and columns of the original matrix A can be
C     reordered (e.g., to reduce fill-in or ensure numerical stability)
C     before calling the driver.  If no reordering is done, then set
C     R(I) = C(I) = IC(I) = I  FOR I=1,...,N.  The solution A is
C     returned in the original order.
C
C     R       INTEGER(N)
C             ordering of the rows of A.
C
C     C       INTEGER(N)
C             ordering of the columns of A.
C
C     IC      INTEGER(N)
C             inverse of the ordering of the columns of A;  i.e.,
C             IC(C(I)) = I  for I=1,...,n.
C
C          Working storage is needed for the factored form of the matrix
C     A plus various temporary vectors.  The arrays ISP and RSP should
C     be equivalenced;  integer storage is allocated from the beginning
C     of ISP and real storage from the end of RSP.
C
C     NSP     INTEGER
C             declared dimension of RSP;
C             the exact value of NSP will be specified below
C
C     ISP     INTEGER(*)
C             integer working storage divided up into various arrays
```

```
C               needed by the subroutines;  ISP and RSP should be
C               equivalenced.
C               size = LRATIO*NSP, where LRATIO = size of storage for a
C               real number divided by the size of storage for an integer.
C
C       RSP     REAL(NSP)
C               real working storage divided up into various arrays
C               needed by the subroutines;  ISP and RSP should be
C               equivalenced.
C
C       ESP     INTEGER
C               if sufficient storage was available to perform the
C               symbolic factorization (NSFC), then ESP is set to the
C               amount of excess storage provided (negative if
C               insufficient storage was available to perform the
C               numeric factorization (NNFC)).
C               if ESP > 2*N, then those last 2n position of RSP will
C               contain approximate left and right null vectors for A.
C
C/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
C><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><
C\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
C
      SUBROUTINE SYNDBE
     *   (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)
C
C       the deflated block elimination algorithm
C
C       arguments:
C
C       on entry:
C
C       N       INTEGER
C               the order of the matrix A
C
C       M       INTEGER
C               the order of the borders to A in M
C
C       R       INTEGER(N)
C               ordering of the rows of A.
C
C       C       INTEGER(N)
C               ordering of the columns of A.
C
C       IC      INTEGER(N)
C               inverse of the ordering of the columns of A;  i.e.,
C               IC(C(I)) = I  for I=1,...,n.
C
```

```
C     IA       INTEGER(N+1)
C              pointers to delimit the rows in A.
C
C     JA       INTEGER(*)
C              column numbers corresponding to the elements of A.
C              size = size of A.
C
C     A        INTEGER(*)
C              nonzero entries of the coefficient matrix A, stored
C              by rows.
C              size = number of nonzero entries in A.
C
C     NSP      INTEGER
C              declared dimension of RSP;
C              the exact value of NSP will be specified below.
C
C     ISP      INTEGER(*)
C              integer working storage divided up into various arrays
C              needed by the subroutines;  ISP and RSP should be
C              equivalenced.
C              size = LRATIO*NSP, where LRATIO = size of storage for a
C              real number divided by the size of storage for an integer.
C
C     RSP      REAL(NSP)
C              real working storage divided up into various arrays
C              needed by the subroutines;  ISP and RSP should be
C              equivalenced.
C
C     B        REAL(LDB,M)
C              right-hand border to matrix A in matrix M.
C
C     LDB      INTEGER
C              the leading dimension of the array B.  LDB >= N.
C
C     CT       REAL(LDC,N)
C              bottom border to matrix A in matrix M
C
C     LDC      INTEGER
C              the leading dimension of the array CT.  LDC >= M.
C
C     D        REAL(LDD,M)
C              lower right-hand entries of M.
C
C     LDD      INTEGER
C              the leading dimension of the array D.  LDD >= M.
C
C     F        REAL(N)
C     G        REAL(M)
C              right-hand side to solve with
```

```
C
C     WORK1    REAL(LDW1,M+4)  LDW1 >= N
C              used to hold Vd, Wd, psiT B, psi and phi.
C
C     LDW1     INTEGER
C              the leading dimension of the array WORK1.  LDW1 >= N.
C
C     WORK2    REAL(LDW2,M+4)  LDW2 >= M+1
C              used to hold E, g',psiT f and delta, and pivot indices for E.
C
C     LDW2     INTEGER
C              the leading dimension of the array WORK2.  LDW2 >= M+1.
C
C     JOB      CHARACTER*6
C              indicates which inputs are the same as in the last call
C              to SGEDBE.  If there was no such call, set JOB =
C              '      ' or 'a     ' (see below).  Otherwise, JOB contains
C              as many of the following apply:
C               'A' if A stays the same
C               'S' if A is new but already factored by SGECO or SGEFA
C               'B' if B stays the same
C               'C' if CT stays the same
C               'D' if D stays the same
C               'F' if F stays the same
C               'G' if G stays the same
C
C     on exit:
C
C     RSP      REAL(NSP)
C              the last 2n positions of RSP contain approximate
C              left and right null vectors for A if ESP > 2*N.
C
C     ESP      INTEGER
C              if sufficient storage was available to perform the
C              symbolic factorization (NSFC), then ESP is set to the
C              amount of excess storage provided (negative if
C              insufficient storage was available to perform the
C              numeric factorization (NNFC)).
C              if ESP > 2*N, then those last 2n position of RSP will
C              contain approximate left and right null vectors for A.
C
C
C     X        REAL(N)
C     Y        REAL(M+1)
C              solution vector.
C
C     WORK1    REAL(LDW1,M+4)  LDW1 >= N
C              used to hold Vd, Wd, psiT B, psi and phi.
C
```

```
C      WORK2    REAL(LDW2,M+4)   LDW2 >= M+1
C               used to hold E, g',psiT f and delta, and pivot indices for E
C
C      Savings on storage:
C      the following pairs of inputs may be equivalent:
C          (X,F)   (Y,G)   (B,WORK1)   (D,WORK2)
C      in general if equivalent storage is used, then a change in one
C      of the inputs in either the left-hand-side group or the right-
C      -hand-side group requires that the entire group be re-entered.
C      Specific exceptions to this rule can be determined by examining
C      the algorithm.
C
       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
C
C      the following constants are used to partition WORK1 and WORK2
C      into their various vectors;  MP1 stands for the "extra" row and
C      column added to D in forming E.  WORK1 is primarily used for Vd,
C      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
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)
C
C      Algorithm:
C
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
C
C     if AJOB = 0 or 1, or B is new,  we start by solving A Vd = B;
C     this may imply factoring A, and/or computing psi, phi and delta
      IF (NEWA .OR. NEWB) THEN
C
C        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(1,1),
     *   WORK1(1,PSI),WORK1(1,PHI),DELTA, WORK1(1,1),WORK1(1,CB),AJOB)
C
C        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(1,I),
     *                     WORK1(1,PSI),WORK1(1,PHI),DELTA,
     *                     WORK1(1,I),WORK1(I,CB),2)
 10         CONTINUE
         ENDIF
      ENDIF
C
C     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)
C
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
C
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(1,J),1)
 20         CONTINUE
C
C           compute cT PHI element by element
            WORK2(I,MP1) = SDOT(N, CT(I,1),LDC, WORK1(1,PHI),1)
 30      CONTINUE
```

```
C
C         factor E
          CALL SGEFA (WORK2,LDW2,MP1,WORK2(1,IPVT),INFO)
       ENDIF
C
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(1,WD),1)
 40       CONTINUE
       ENDIF
C
C      compute 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

       END

C/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
C><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><
C\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
C
       SUBROUTINE SYNDF (N, R,C,IC, IA,JA,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                 d         p
C
C      arguments are the same as for SYNDBE 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
```

```
C           smallest singular value for matrix M (same as SV in SYNII)
C           (only on entry if JOB >= 2)
C
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 CDRV
C                     (or a previous call to SYC[D]BE) and continue
C                     from there.
C           JOB >= 2 : additionally, PSI, PHI and DELTA have already
C                     been computed.
C
C     on exit:
C
C     PSI   REAL(N)
C     PHI   REAL(N)
C           left and right null vectors to matrix A
C
C     DELTA REAL
C           smallest singular value for matrix M (same as SV in SYNII)
C
C     ZD    REAL(N)
C           deflated solution to system A z = p
C           Note that P and ZD may be the same vector
C
C     CP    REAL
C           coefficient of projection of Z onto right null vector (PHI)
C

      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,A, PHI,PHI,
     *              NSP,ISP,RSP,ESP, 1,FLAG)

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

```
      IF (TRANS) GOTO 20
C
C     Perform deflation with A
C        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 NDRV (N, R,C,IC, IA,JA,A, ZD,ZD, NSP,ISP,RSP,ESP, 3,FLAG)
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
C                             T
C     Perform deflation with A
C         T
C        A  Zd = p - (phiT p) phi ; solve for Zd ; Cp is approx (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 NDRV (N, R,C,IC, IA,JA,A, ZD,ZD, NSP,ISP,RSP,ESP, 3,FLAG)
C
C        orthogonalize Zd wrt psi
         CALL SAXPY(N,-SDOT(N,PSI,1,ZD,1),PSI,1,ZD,1)

 30   CONTINUE
      END

C/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
C><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><
C\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
C
      SUBROUTINE SYNII
     *(N, R,C,IC, IA,JA,A, NSP,ISP,RSP,ESP, PSI,PHI, DELTA, JOB,ITER)
C
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     Numer. Anal., vol. 21 no. 4 (August, 1984)
C
C     arguments are the same as for SYNDBE except:
C
C     on entry:
C
C     NSP      INTEGER
C     ISP      REAL(*)
C     RSP      REAL(NSP)
C     ESP      INTEGER
C              must contain a factorization of A, produced by NDRV with
```

```
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
      REAL RESOL
      PARAMETER (RESOL = .0001)
      INTEGER  N, R(1),C(1),IC(1),IA(1),JA(1),NSP,ISP(1),ESP, JOB,ITER
      INTEGER  FLAG
      REAL A(1), RSP(1), PSI(1), PHI(1), DELTA
      REAL PSILEN,PHILEN
C
      IF (JOB .EQ. 0) THEN
C
C        no initial guess; fill PSI with 1's
         DO 10 I = 1,N
            PSI(I) = 1.
 10      CONTINUE

      ELSEIF (JOB .EQ. 1) THEN
C
C        initial guess is in PHI; solve for initial PSI
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)
```

```
      ENDIF
C
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
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)
```

```
      END
C/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
C><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><
C\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
C
      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)
C
C    the ordinary (undeflated) block elimination algorithm
C
C    all arguments are the same as in SYNDBE.
C
      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
C
      INTEGER MP1,IPVT
      MP1 = M + 1
      IPVT = MP1 + 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
C     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
C
C     solve A w = f for w
      IF (NEWA .OR. NEWF)
     *   CALL NDRV (N, R,C,IC, IA,JA,A, F,WORK1(1,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(1,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(1,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
```

5.5 SYCDBE

```
C
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     Computer Science, Yale Univ., 1985.
C
C     This set of routines calls YSMP's general routines with
C     compressed storage, Linpack's SGE- routines, and the SBLAs.
C     Implemented by Thomas A. Grossi, Yale University, 1985.
C-----------------------------------------------------------------------
C
C     STORAGE SCHEME FOR THESE ROUTINES
C
C          The nonzero entries of the coefficient matrix M are stored
C     row-by-row in the array A.  To identify the individual nonzero
C     entries in each row, we need to know in which column each entry
C     lies.  The column indices which correspond to the nonzero entries
C     of M are stored in the array JA;  i.e., if  A(K) = M(I,J),  then
C     JA(K) = J.  In addition, we need to know where each row starts and
C     how long it is.  The index positions in JA and A where the rows of
C     M begin are stored in the array IA;  i.e., if M(I,J) is the first
C     nonzero entry (stored) in the I-th row and A(K) = M(I,J),  then
C     IA(I) = K.  moreover, the index in JA and A of the first location
C     following the last element in the last row is stored in IA(N+1).
C     thus, the number of entries in the I-th row is given by
C     IA(I+1) - IA(I),  the nonzero entries of the I-th row are stored
C     consecutively in
C             A(IA(I)),  A(IA(I)+1),  ..., A(IA(I+1)-1),
C     and the corresponding column indices are stored consecutively in
C             JA(IA(I)), JA(IA(I)+1), ..., JA(IA(I+1)-1).
C     for example, the 5 by 5 matrix
C                 | 1. 0. 2. 0. 0.|
C                 | 0. 3. 0. 0. 0.|
C             M = | 0. 4. 5. 6. 0.|
C                 | 0. 0. 0. 7. 0.|
C                 | 0. 0. 0. 8. 9.|
C     would be stored as
```

```
C               | 1  2  3  4  5  6  7  8  9
C            ---+--------------------------
C            IA | 1  3  4  7  8 10
C            JA | 1  3  2  2  3  4  4  4  5
C             A | 1. 2. 3. 4. 5. 6. 7. 8. 9.
C
C     N      INTEGER
C            number of variables/equations.
C
C     A      INTEGER(*)
C            nonzero entries of the coefficient matrix A, stored
C            by rows.
C            size = number of nonzero entries in A.
C
C     IA     INTEGER(N+1)
C            pointers to delimit the rows in A.
C
C     JA     INTEGER(*)
C            column numbers corresponding to the elements of A.
C            size = size of A.
C
C
C         The rows and columns of the original matrix A can be
C     reordered (e.g., to reduce fill-in or ensure numerical stability)
C     before calling the driver.  If no reordering is done, then set
C     R(I) = C(I) = IC(I) = I  FOR I=1,...,N.  The solution A is
C     returned in the original order.
C
C     R      INTEGER(N)
C            ordering of the rows of A.
C
C     C      INTEGER(N)
C            ordering of the columns of A.
C
C     IC     INTEGER(N)
C            inverse of the ordering of the columns of m;  i.e.,
C            IC(C(I)) = I  for I=1,...,n.
C
C         Working storage is needed for the factored form of the matrix
C     m plus various temporary vectors.  The arrays ISP and RSP should
C     be equivalenced;  integer storage is allocated from the beginning
C     of ISP and real storage from the end of RSP.
C
C     NSP    INTEGER
C            declared dimension of RSP;
C            the exact value of NSP will be specified below
C
C     ISP    INTEGER(*)
C            integer working storage divided up into various arrays
```

```
C            needed by the subroutines;  ISP and RSP should be
C            equivalenced.
C            size = LRATIO*NSP, where LRATIO = size of storage for a
C            real number divided by the size of storage for an integer.
C
C     RSP    REAL(NSP)
C            real working storage divided up into various arrays
C            needed by the subroutines;  ISP and RSP should be
C            equivalenced.
C
C     ESP    INTEGER
C            if sufficient storage was available to perform the
C            symbolic factorization (NSFC), then ESP is set to the
C            amount of excess storage provided (negative if
C            insufficient storage was available to perform the
C            numeric factorization (NNFC)).
C            if ESP > 2*N, then those last 2n position of RSP will
C            contain approximate left and right null vectors for A.
C
C/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
C><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><<
C\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
C
      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)
C
C     the deflated block elimination algorithm
C
C     arguments:
C
C     on entry:
C
C     N      INTEGER
C            the order of the matrix A
C
C     M      INTEGER
C            the order of the borders to A in M
C
C     R      INTEGER(N)
C            ordering of the rows of A.
C
C     C      INTEGER(N)
C            ordering of the columns of A.
C
C     IC     INTEGER(N)
C            inverse of the ordering of the columns of m;  i.e.,
C            IC(C(I)) = I  for I=1,...,n.
C
```

```
C     IA      INTEGER(N+1)
C             pointers to delimit the rows in A.
C
C     JA      INTEGER(*)
C             column numbers corresponding to the elements of A.
C             size = size of A.
C
C     A       INTEGER(*)
C             nonzero entries of the coefficient matrix A, stored
C             by rows.
C             size = number of nonzero entries in A.
C
C     NSP     INTEGER
C             declared dimension of RSP;
C             the exact value of NSP will be specified below.
C
C     ISP     INTEGER(*)
C             integer working storage divided up into various arrays
C             needed by the subroutines;  ISP and RSP should be
C             equivalenced.
C             size = LRATIO*NSP, where LRATIO = size of storage for a
C             real number divided by the size of storage for an integer.
C
C     RSP     REAL(NSP)
C             real working storage divided up into various arrays
C             needed by the subroutines;  ISP and RSP should be
C             equivalenced.
C
C     B       REAL(LDB,M)
C             right-hand border to matrix A in matrix M.
C
C     LDB     INTEGER
C             the leading dimension of the array B.  LDB >= N.
C
C     CT      REAL(LDC,N)
C             bottom border to matrix A in matrix M.
C
C     LDC     INTEGER
C             the leading dimension of the array CT.  LDC >= M.
C
C     D       REAL(LDD,M)
C             lower right-hand entries of M.
C
C     LDD     INTEGER
C             the leading dimension of the array D.  LDD >= M.
C
C     F       REAL(N)
C     G       REAL(M)
C             right-hand side to solve with
```

```
C
C     WORK1    REAL(LDW1,M+4)  LDW1 >= N
C              used to hold Vd, Wd, psiT B, psi and phi
C
C     LDW1     INTEGER
C              the leading dimension of the array WORK1.  LDW1 >= N.
C
C     WORK2    REAL(LDW2,M+3  LDW2 >= M+1
C              used to hold E, g',psiT f and delta, and pivot indices for E.
C
C     LDW2     INTEGER
C              the leading dimension of the array WORK2.  LDW2 >= M+1.
C
C     JOB      CHARACTER*6
C              indicates which inputs are the same as in the last call
C              to SGEDBE.  If there was no such call, set JOB =
C              '      ' or 'a     ' (see below).  Otherwise, JOB contains
C              as many of the following apply:
C               'A' if A stays the same
C               'S' if A is new but already factored by SGECO or SGEFA
C               'B' if B stays the same
C               'C' if CT stays the same
C               'D' if D stays the same
C               'F' if F stays the same
C               'G' if G stays the same
C
C     on exit:
C
C     RSP      REAL(NSP)
C              the last 2n positions of RSP contain approximate
C              left and right null vectors for A if ESP > 2*N.
C
C     ESP      INTEGER
C              if sufficient storage was available to perform the
C              symbolic factorization (NSFC), then ESP is set to the
C              amount of excess storage provided (negative if
C              insufficient storage was available to perform the
C              numeric factorization (NNFC)).
C              if ESP > 2*N, then those last 2n position of RSP will
C              contain approximate left and right null vectors for A.
C
C     X        REAL(N)
C     Y        REAL(M+1)
C              solution vector
C
C     WORK1    REAL(LDW1,M+4)  LDW1 >= N
C              used to hold Vd, Wd, psiT B, psi and phi.
C
C     WORK2    REAL(LDW2,M+4)  LDW2 >= M+1
```

```
C           used to hold E, g',psiT f and delta, and pivot indices for E
C
C     Savings on storage:
C     the following pairs of inputs may be equivalent:
C        (X,F)  (Y,G)  (B,WORK1)  (D,WORK2)
C     in general if equivalent storage is used, then a change in one
C     of the inputs in either the left-hand-side group or the right-
C     -hand-side group requires that the entire group be re-entered.
C     Specific exceptions to this rule can be determined by examining
C     the algorithm.
C
      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
C
C     the following constants are used to partition WORK1 and WORK2
C     into their various vectors;  MP1 stands for the "extra" row and
C     column added to D in forming E.  WORK1 is primarily used for Vd,
C     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
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)
C
C     Algorithm:
C
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
C
C     if AJOB = 0 or 1, or B is new,  we start by solving A Vd = B;
C     this may imply factoring A, and/or computing psi, phi and delta
      IF (NEWA .OR. NEWB) THEN
C
C        for the first element of Vd, AJOB will tell sgeDF what to do
         CALL SYCDF (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)
C
C        compute remaining columns of Vd using results of first call
         IF (M .GT. 1) THEN
            DO 10 I = 2,M
               CALL SYCDF (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
C
C     We must recompute Wd and Cf if A or F have changed
      IF (NEWA .OR. NEWF) THEN
         CALL SYCDF (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
C
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
C
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(1,J),1)
 20         CONTINUE
C
C           compute cT PHI element by element
            WORK2(I,MP1) = SDOT(N, CT(I,1),LDC, WORK1(1,PHI),1)
 30      CONTINUE
```

```
C
C          factor E
           CALL SGEFA (WORK2,LDW2,MP1,WORK2(1,IPVT),INFO)
        ENDIF
C
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(1,WD),1)
 40        CONTINUE
        ENDIF
C
C       compute x and y
        CALL SCOPY (MP1, WORK2(1,GP),1, Y,1)
        CALL SGESL (WORK2,LDW2,MP1,WORK1(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/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
C><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><<
C\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
C
        SUBROUTINE SYCDF (N, R,C,IC, IA,JA,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                  d         p
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
```

```
C           smallest singular value for matrix M (same as SV in SYCII)
C           (only on entry if JOB >= 2)
C
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 CDRV
C                     (or a previous call to SYC[D]BE) and continue
C                     from there.
C           JOB >= 2 : additionally, PSI, PHI and DELTA have already
C                     been computed.
C
C     on exit:
C
C     PSI   REAL(N)
C     PHI   REAL(N)
C           left and right null vectors to matrix A
C
C     DELTA REAL
C           smallest singular value for matrix M (same as SV in SYNII)
C
C     ZD    REAL(N)
C           deflated solution to system A z = p
C           Note that P and ZD may be the same vector
C
C     CP    REAL
C           coefficient of projection of Z onto right null vector (PHI)
C

      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,JA,A, PHI,PHI,
     *              NSP,ISP,RSP,ESP, 1,FLAG)

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

```
      IF (TRANS) GOTO 20
C
C     Perform deflation with A
C        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 CDRV (N, R,C,IC, IA,JA,A, ZD,ZD, NSP,ISP,RSP,ESP, 3,FLAG)
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
C                             T
C     Perform deflation with A
C         T
C        A  Zd = p - (phiT p) phi ; solve for Zd ; Cp is approx (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 CDRV (N, R,C,IC, IA,JA,A, ZD,ZD, NSP,ISP,RSP,ESP, 3,FLAG)
C
C        orthogonalize Zd wrt psi
         CALL SAXPY(N,-SDOT(N,PSI,1,ZD,1),PSI,1,ZD,1)

 30   CONTINUE
      END

C/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
C><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><
C\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
C
      SUBROUTINE SYCII
     *(N, R,C,IC, IA,JA,A, NSP,ISP,RSP,ESP, PSI,PHI, DELTA, JOB,ITER)
C
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     Numer. Anal., vol. 21 no. 4 (August, 1984)
C
C     arguments are the same as for SYCDBE except:
C
C     on entry:
C
C     NSP      INTEGER
C     ISP      REAL(*)
C     RSP      REAL(NSP)
C     ESP      INTEGER
C              must contain a factorization of A, produced by CDRV with
```

```
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 SYCII.  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
      REAL RESOL
      PARAMETER (RESOL = .0001)
      INTEGER  N, R(1),C(1),IC(1),IA(1),JA(1),NSP,ISP(1),ESP, JOB,ITER
      INTEGER  FLAG
      REAL A(1), RSP(1), PSI(1), PHI(1), DELTA
      REAL PSILEN,PHILEN
C
      IF (JOB .EQ. 0) THEN
C
C        no initial guess; fill PSI with 1's
         DO 10 I = 1,N
            PSI(I) = 1.
 10      CONTINUE

      ELSEIF (JOB .EQ. 1) THEN
C
C        initial guess is in PHI; solve for initial PSI
C        phi' = phi' / ||phi'||
         PHILEN = SNRM2(N,PHI,1)
         CALL SSCAL (N,1/PHILEN,PHI,1)
C
C         T
C        A  psi' = phi'
         CALL CDRV
     *        (N, R,C,IC, IA,JA,A, PHI,PSI, NSP,ISP,RSP,ESP, 4,FLAG)
```

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

```
      END

C/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
C><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><
C\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
C
      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)
C
C    the ordinary (undeflated) block elimination algorithm
C
C    all arguments are the same as in SYCDBE.
C
      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
C
      INTEGER MP1,IPVT
      MP1 = M + 1
      IPVT = MP1 + 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
C     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, FLAG)

      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
C
C     solve A w = f for w
      IF (NEWA .OR. NEWF)
```

```
     *  CALL CDRV (N, R,C,IC, IA,JA,A, F,WORK1(1,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(1,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(1,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
```

5.6 SYSDBE

```
C
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     Computer Science, Yale Univ., 1985.
C
C     This set of routines calls YSMP's routines for symmetric matrices,
C     Linpack's SGE- routines, and the SBLAs.
C     Implemented by Thomas A. Grossi, Yale University, 1985.
C-----------------------------------------------------------------------
C
C     STORAGE SCHEME FOR THESE ROUTINES
C
C         The nonzero entries of the coefficient matrix M are stored
C     row-by-row in the array A.  To identify the individual nonzero
C     entries in each row, we need to know in which column each entry
C     lies.  The column indices which correspond to the nonzero entries
C     of M are stored in the array JA;  i.e., if  A(K) = M(I,J),  then
C     JA(K) = J.  In addition, we need to know where each row starts and
C     how long it is.  The index positions in JA and A where the rows of
C     M begin are stored in the array IA;  i.e., if M(I,J) is the first
C     nonzero entry (stored) in the I-th row and A(K) = M(I,J),  then
C     IA(I) = K.  moreover, the index in JA and A of the first location
C     following the last element in the last row is stored in IA(N+1).
C     thus, the number of entries in the I-th row is given by
C     IA(I+1) - IA(I),  the nonzero entries of the I-th row are stored
C     consecutively in
C
C             A(IA(I)),  A(IA(I)+1),  ...,  A(IA(I+1)-1),
C
C     and the corresponding column indices are stored consecutively in
C
C             JA(IA(I)), JA(IA(I)+1), ..., JA(IA(I+1)-1).
C
C     Since the coefficient matrix is symmetric, only the nonzero entries
C     in the upper triangle need be stored, for example, the matrix
C
```

```
C                         | 1  0  2  3  0 |
C                         | 0  4  0  0  0 |
C                     M = | 2  0  5  6  0 |
C                         | 3  0  6  7  8 |
C                         | 0  0  0  8  9 |
C
C     could be stored as
C
C            | 1  2  3  4  5  6  7  8  9 10 11 12 13
C         ---+--------------------------------------
C         IA | 1  4  5  8 12 14
C         JA | 1  3  4  2  1  3  4  1  3  4  5  4  5
C          A | 1  2  3  4  2  5  6  3  6  7  8  8  9
C
C     or (symmetrically) as
C
C            | 1  2  3  4  5  6  7  8  9
C         ---+--------------------------
C         IA | 1  4  5  7  9 10
C         JA | 1  3  4  2  3  4  4  5  5
C          A | 1  2  3  4  5  6  7  8  9
C
C     N       INTEGER
C             number of variables/equations.
C
C     A       INTEGER(*)
C             nonzero entries of the coefficient matrix A, stored
C             by rows.
C             size = number of nonzero entries in A.
C
C     IA      INTEGER(N+1)
C             pointers to delimit the rows in A.
C
C     JA      INTEGER(*)
C             column numbers corresponding to the elements of A.
C             size = size of A.
C
C
C         The rows and columns of the original matrix A can be
C     reordered (e.g., to reduce fill-in or ensure numerical stability)
C     before calling the driver.  If no reordering is done, then set
C     P(I) = IP(I) = I  for I=1,...,N.  The solution A is returned in
C     the original order.
C
C     P       INTEGER(N)
C             ordering of the rows/columns of A.
C
C     IP      INTEGER(N)
C             inverse of the ordering of the rows/columns of A;  i.e.,
```

```
C            IC(C(I)) = I  for I=1,...,n.
C
C         Working storage is needed for the factored form of the matrix
C     A plus various temporary vectors.  The arrays ISP and RSP should
C     be equivalenced;  integer storage is allocated from the beginning
C     of ISP and real storage from the end of RSP.
C
C     NSP      INTEGER
C              declared dimension of RSP;
C              the exact value of NSP will be specified below
C
C     ISP      INTEGER(*)
C              integer working storage divided up into various arrays
C              needed by the subroutines;  ISP and RSP should be
C              equivalenced.
C              size = LRATIO*NSP, where LRATIO = size of storage for a
C              real number divided by the size of storage for an integer.
C
C     RSP      REAL(NSP)
C              real working storage divided up into various arrays
C              needed by the subroutines;  ISP and RSP should be
C              equivalenced.
C
C     ESP      INTEGER
C              if sufficient storage was available to perform the
C              symbolic factorization (CSFC), then ESP is set to the
C              amount of excess storage provided (negative if
C              insufficient storage was available to perform the
C              numeric factorization (CNFC)).
C              if ESP > 2*N, then those last 2n position of RSP will
C              contain approximate left and right null vectors for A.
C
C/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
C><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><<
C\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
C
      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
C     the deflated block elimination algorithm
C
C     arguments:
C
C     on entry:
C
C     N        INTEGER
C              the order of the matrix A
C
```

```
C     M        INTEGER
C              the order of the borders to A in M
C
C     P        INTEGER(N)
C              ordering of the rows/columns of A.
C
C     IP       INTEGER(N)
C              inverse of the ordering of the rows/columns of A;  i.e.,
C              IC(C(I)) = I  for I=1,...,n.
C
C     IA       INTEGER(N+1)
C              pointers to delimit the rows in A.
C
C     JA       INTEGER(*)
C              column numbers corresponding to the elements of A.
C              size = size of A.
C
C     A        INTEGER(*)
C              nonzero entries of the coefficient matrix A, stored
C              by rows.
C              size = number of nonzero entries in A.
C
C     NSP      INTEGER
C              declared dimension of RSP;
C              the exact value of NSP will be specified below
C
C     ISP      INTEGER(*)
C              integer working storage divided up into various arrays
C              needed by the subroutines;  ISP and RSP should be
C              equivalenced.
C              size = LRATIO*NSP, where LRATIO = size of storage for a
C              real number divided by the size of storage for an integer.
C
C     RSP      REAL(NSP)
C              real working storage divided up into various arrays
C              needed by the subroutines;  ISP and RSP should be
C              equivalenced.
C
C     B        REAL(LDB,M)
C              right-hand border to matrix A in matrix M.
C
C     LDB      INTEGER
C              the leading dimension of the array B.  LDB >= N.
C
C     CT       REAL(LDC,N)
C              bottom border to matrix A in matrix M
C
C     LDC      INTEGER
C              the leading dimension of the array CT.  LDC >= M.
```

```
C
C     D        REAL(LDD,M)
C              lower right-hand entries of M
C
C     LDD      INTEGER
C              the leading dimension of the array D.  LDD >= M.
C
C     F        REAL(N)
C     G        REAL(M)
C              right-hand side to solve with
C
C     WORK1    REAL(LDW1,M+4)  LDW1 >= N
C              used to hold Vd, Wd, psiT B, psi and phi
C
C     LDW1     INTEGER
C              the leading dimension of the array WORK1.  LDW1 >= N.
C
C     WORK2    REAL(LDW2,M+4) LDW2 >= M+1
C              used to hold E, g',psiT f and delta, and pivot indices for E.
C
C     LDW2     INTEGER
C              the leading dimension of the array WORK2.  LDW2 >= M+1.
C
C     JOB      CHARACTER*6
C              indicates which inputs are the same as in the last call
C              to SGEDBE.  If there was no such call, set JOB =
C              '      ' or 'a     ' (see below).  Otherwise, JOB contains
C              as many of the following apply:
C               'A' if A stays the same
C               'S' if A is new but already factored by SGECO or SGEFA
C               'B' if B stays the same
C               'C' if CT stays the same
C               'D' if D stays the same
C               'F' if F stays the same
C               'G' if G stays the same
C
C     on exit:
C
C     RSP      REAL(NSP)
C              the last 2n positions of RSP contain approximate
C              left and right null vectors for A if ESP > 2*N.
C
C     ESP      INTEGER
C              if sufficient storage was available to perform the
C              symbolic factorization (CSFC), then ESP is set to the
C              amount of excess storage provided (negative if
C              insufficient storage was available to perform the
C              numeric factorization (CNFC)).
C              if ESP > 2*N, then those last 2n position of RSP will
```

```
C               contain approximate left and right null vectors for A.
C
C     X         REAL(N)
C     Y         REAL(M+1)
C               solution vector
C
C     WORK1     REAL(LDW1,M+4)  LDW1 >= N
C               used to hold Vd, Wd, psiT B, psi and phi
C
C     WORK2     REAL(LDW2,M+4) LDW2 >= M+1
C               used to hold E, g',psiT f and delta, and pivot indices for E.
C
C     Savings on storage:
C     the following pairs of inputs may be equivalent:
C         (X,F)  (Y,G)  (B,WORK1)  (D,WORK2)
C     in general if equivalent storage is used, then a change in one
C     of the inputs in either the left-hand-side group or the right-
C     -hand-side group requires that the entire group be re-entered.
C     Specific exceptions to this rule can be determined by examining
C     the algorithm.
C
      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
C
C     the following constants are used to partition WORK1 and WORK2
C     into their various vectors;  MP1 stands for the "extra" row and
C     column added to D in forming E.  WORK1 is primarily used for Vd,
C     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
      GP = MP1 + 1
      IPVT = GP + 1
      ALPHA = MP1
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)
C
C     Algorithm:
C
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
C
C     if AJOB = 0 or 1, or B is new,  we start by solving A Vd = B;
C     this may imply factoring A, and/or computing psi, phi and delta
      IF (NEWA .OR. NEWB) THEN
C
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
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,I),
     *              WORK1(1,PSI),DELTA, WORK1(1,I),WORK1(I,CB),2)
 10         CONTINUE
         ENDIF
      ENDIF
C
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
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
C
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(1,J),1)
 20         CONTINUE
C
C           compute cT PHI element by element
            WORK2(I,MP1) = SDOT(N, CT(I,1),LDC, WORK1(1,PHI),1)
 30      CONTINUE
C
C        factor E
         CALL SGEFA (WORK2,LDW2,MP1,WORK2(1,IPVT),INFO)
      ENDIF
C
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(1,WD),1)
 40      CONTINUE
      ENDIF
C
C     compute 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

      END

C/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
C><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><<
C\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
C
      SUBROUTINE SYSDF (N, P,IP, IA,JA,A, NSP,ISP,RSP,ESP,
     *                  RHS, PSI,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                d        p
C
C     arguments are the same as for SYSDBE except:
C
C     on entry:
C
C     RHS    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
C              smallest singular value for matrix A
C              (only on entry if JOB >= 2)
C
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 CDRV
C                        (or a previous call to SYC[D]BE) and continue
C                        from there.
C              JOB >= 1 : additionally, PSI, PHI and DELTA have already
C                        been computed.
C
C     on exit:
C
C     PSI      REAL(N)
C     PHI      REAL(N)
C              left and right null vectors to matrix A
C
C     DELTA    REAL
C              smallest singular value for matrix A
C
C     ZD       REAL(N)
C              deflated solution to system A z = p
C              Note that P and ZD may be the same vector
C
C     CP       REAL
C              psiT p
C
      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 SYSII (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

C/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
C><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><
C\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
C
      SUBROUTINE SYSII
     *(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     Numer. Anal., vol. 21 no. 4 (August, 1984)
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 SYSII.  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)
```

```
C              on output, contain the null vector of the matrix A.
C
C                                    resol = resolution of convergence
      REAL RESOL
      PARAMETER (RESOL = .0001)
      INTEGER  N, P(1),IP(1),IA(1),JA(1),NSP,ISP(1),ESP, JOB,ITER
      INTEGER  FLAG
      REAL A(1), RSP(1), PSI(1), SV
      REAL OLDLEN, NEWLEN
C
      IF (JOB .EQ. 0) THEN
C
C        no initial guess; fill PSI with 1's
         DO 10 I = 1,N
            PSI(I) = 1.
 10      CONTINUE

      ENDIF
C
C     PSI now contains initial guess; normalize it
C     psi' = psi' / ||psi'||
      NEWLEN = SNRM2(N,PSI,1)
      CALL SSCAL (N,1/NEWLEN,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        Apsi' = psi
         CALL SDRV (N, P,IP, IA,JA,A, PSI,PSI, NSP,ISP,RSP,ESP, 3,FLAG)
C
C        psi' = psi' / ||psi'||
         OLDLEN = NEWLEN
         HEWLEN = SNRM2(N,PSI,1)
         CALL SSCAL (N,1/NEWLEN,PSI,1)
C
C        increment counter
         I = I + IINC
C
C     end
      IF (I .LE. ITER .AND. ABS(1/OLDLEN - 1/NEWLEN) .GT. RESOL)
     *   GOTO 50
C
C     do psi' once more
      CALL SDRV (N, P,IP, IA,JA,A, PSI,PSI, NSP,ISP,RSP,ESP, 3,FLAG)
```

```
C
C     delta = 1/||psi'||
C
      DELTA = 1/SNRM2(N,PSI,1)
      CALL SSCAL (N, DELTA,PSI,1)

      END

C/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
C><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><<
C\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
C
      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)
C
C       the ordinary (undeflated) block elimination algorithm.
C
C       (All arguments are the same as in SYSDBE.)
C
      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
C
      INTEGER MP1,IPVT
      MP1 = M + 1
      IPVT = MP1 + 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
C     solve A V  = B for V
      IF (AJOB .EQ. 0)
     *   CALL SDRV (N, P,IP, IA,JA,A, X,X, NSP,ISP,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(1,I),
```

```
     *              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(1,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(1,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(1,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
```