Experience with NSPCG and SPARSKIT in relation to LIST

Russell K. Standish

December 8, 1995

Abstract

NSPCG is a package for performing iterative solutions to non-symmetric systems, not only preconditioned conjugate gradient algorithms as implied by the title. Provision is given for supplying additional algorithms, or for specifying different data structures. SPARSKIT is a utility package for performing such operations as transformations between different data representations, extraction of parts of matrices, obtaining properties such as the bandwidth, reading and writing Harwell/Boeing matrices, BLAS for sparse matrices, statistics and visualisation tools. In this work, SPARSKIT was used to generate test matrices, and to change them to different data formats for input to NSPCG.

1 NSPCG

NSPCG [5] is a package for performing iterative solutions to non-symmetric systems, not only preconditioned conjugate gradient algorithms as implied by the title. A variety of accelerator and preconditioning methods are available, and are selected by supplying the name of a subroutine which performs the given operation as parameters in the call to NSPCG. Other parameters in the call refer to the matrix describing the system, the right hand side, the solution vector, an exact solution (if known), parameters relating to the method, storage etc and workspace.

For anyone to use the package, a decision must be made which preconditioner / accelerator combination to use, from a combination of 312 possibilites. Many of these combinations require further parameters to be decided, such as allowed fill-in in the Incomplete LU methods. Then, a storage method must be chosen from 5 basic types, possibly with some kind of permutation or colouring scheme. Stopping criterion must be selected from amongst 10 different possibilities, and finally the workspace requirement must be calculated, and sufficient workspace allocated. Further possibilities are provided in that one can supply code to

perform matrix multiplication and preconditioning for arbitrary matrix formats, and also for new algorithms as they become available.

Clearly then, NSPCG is not intended to be a production code. Rather, it is intended as a research tool for experimenting with different methods. I have written a code stub which generates a test matrix according to some user specified parameters, then translates the matrix into one of three data formats (Ellpack, Diag and Coord), sets up the parameter arrays and then calls NSPCG. Optionally, one may also read in matrices from the Harwell-Boeing collection [2].

NSPCG vectorizes very well. The most time consuming parts of the package are matrix multiplications, and solutions of triangular systems. The general scenario is that one trades expensive preconditioning steps (i.e triangular solves in many cases) against the total number of iterations. For example, with Incomplete LU decomposition, the more fill-in allowed in the decomposition, the slower the preconditioning step, as the matrix solve is denser, however the fewer iterations are required to solve the system. In the limit of infinite fill-in, the whole method becomes a direct method, with the iteration count precisely 1.

On vector computers, this balance is worsened, because the only triangular solve method generally available (called the "English method") is inherently serial. Here, multifrontal methods may provide some kind of solution to vectorizing triangular solves, however, as these methods are related to cyclic reduction, they presumably are restricted to diagonally dominant matrices.

Much greater success is obtained if the preconditioner used only performs matrix multiplications, for example the polynomial methods. These, however, do have restrictions.

Of the three main storage modes provided in NSPCG, the matrix multiplication code for Ellpack and the Diag modes vectorize as is. The Coord code requires the addition of a NOVREC VOCL around the inner loop. There was already the equivalent Cray directive in the code. In terms of efficiency on a matrix with full diagonals, the diag code was the most efficient, followed by ellpack then the coord method. However, there was not such a great difference between them, that the number of zeros stored is probably a far greater factor in determining the efficiency of the method.

Another problem with NSPCG is that the code is not numerically stable, i.e. floating point exceptions can be easily generated if the methods selected are unsuited for the problem. Any production library code must be able to trap any such error before it arises so that a meaningful error message is returned to the user.

If the SPARSE BLAS proposal [3] is accepted, then NSPCG will need to be broken up into its component parts (if its code is to be used at all). At present, NSPCG is called with the preconditioner routine having a different name for each data format. The alternative that SPARSE BLAS opens up is to remove the need for the iterative algorithm to know anything about the internal algorithm. Some initial work setting up the splitting (e.g. computing Q in the

incomplete Cholesky decomposition) can be performed in any arbitrary sparse format (e.g. coord or csc) as efficiency is not so important - and then converted into the internal format for use by the iterator. Then the iterator merely needs to call matrix multiply and triangular solves without needing to know the matrix structure.

1.1 Changes made to NSPCG to install it on the VP

- Ran the code through the Toolpack tool apt to transform the default precision to double. This is not completely necessary on the VP, as one can use the DOUBLE option, but is necessary for using the code on the workstations.
- Used the BYNAME option. This is required, because in some places arrays are passed to subroutines using scalar variables as dummy parameters.
- VOCL LOOP,NOVREC added to inner loop of VADDS
- In routines SCAL1,SCAL2, SCAL3, loop 20 needed a VOCL LOOP, SCALAR, as otherwise a C7 abend occurs. Note that if diagonal scaling is turned off, then this error will occur in the MULTxx routines.

2 SPARSKIT

SPARSKIT [9] is a utility package for performing such operations as transformations between different data representations, extraction of parts of matrices, obtaining properties such as the bandwidth, reading and writing Harwell/Boeing matrices, BLAS for sparse matrices, statistics and visualisation tools. Unfortunately, it is hampered by inconsistancies in the user interface, poor documentation, and bugs in some places. This makes it essential to read the source code very carefully everytime one wishes to use a new routine from the package. The proposed Sparse BLAS routines covers a very important subset of of the routines in SPARSKIT, namely the data transformation, the matrix multiplication and triangular solves in a consistent way, and also allows these methods to work in the most efficient format appropriate for the given architecture. However, it is still useful for those tools such as the matrix properties module, and matrix plotting routines.

2.1 Changes made to SPARSKIT to install it on the VP

The following changes were made to correct for bugs in SPARSKIT:

1. In routine CSRELL, the statement in loop 41 should read

   ```
   JCOEF(I,J)=0
   ```

2. In routines APMBT and APLSBT, the ljob parameter should not exist in the call to COICSR.

3 Performance

The first test was to evaluate the performance of various data structure when used in a matrix multiplication. This is ultimately the most important operation on vector architectures, as it can be performed efficiently, whereas triangular solutions generally cannot be performed efficiently. The 3 test examples used in this work are ECN(5000,10) [6], astroph and jagmesh [2]. The matrix ECN(5000,10) is a 5000x5000 symmetric matrix with 4 down the diagonal and -1 on 1^{st} and 10^{th} offdiagonals. This would arise from a 5-point discretisation of a simple elliptic pde with constant coefficients. The non-zero structure for the latter two matrices are given in figures 1–2. Performance results in MFlops are shown in figure 3 for these three problems, where the storage schemes are given the following abreviations:

COO Coordinate format

CSR Compressed Sparse Row

DIA Diagonal format

ELL Ellpack format

JAD Jagged Diagonal format

Further descriptions of these formats may be found in the SPARSKIT manual [9]. Of the remaining data formats available in SPARSKIT, the dense format, the Linpack banded format and the Block Sparse Row did not seem general enough to consider. The problem with the Skyline format is that even though the nonsymmetric skyline format is described in the SPARSKIT manual, it is not actually implemented in SPARSKIT.

It can be seen from the graph, that for all problem types tested, the jagged diagonal storage scheme is the best method amongst those in SPARSKIT for vector architectures.

The second test is to use NSPCG to solve various problems by various methods, preconditioners and storage methods. Table 1 shows the iteration count, cpu time and vector speedup over scalar.

One must note that NSPCG always falls over with a floating exception with the astroph and jagmesh problems. It is unclear why - possibly the matrix is singular. In any case, NSPCG should have given diagnostics as to why it can't handle these cases. The net result is that it is impossible to give comparisons with problems other that the toy problem ECN(5000,5).

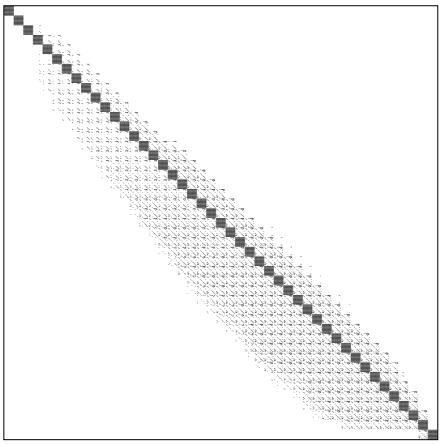

Figure 1: Non zero pattern of astroph

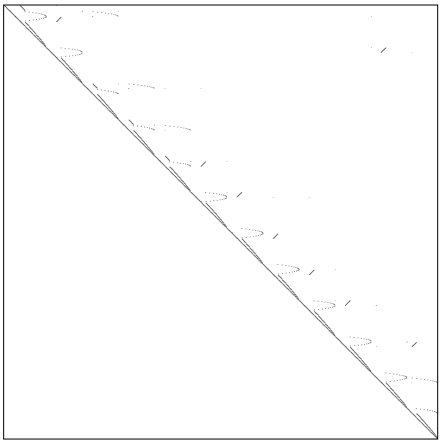

Figure 2: Non zero pattern of jagmesh

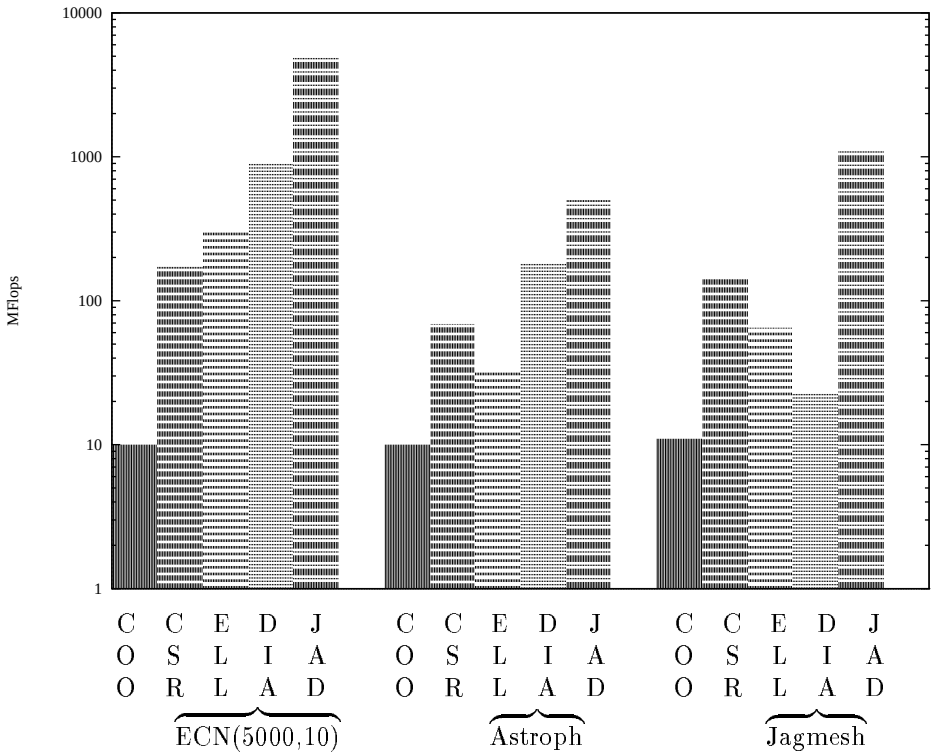

Figure 3: Speed (in MFlops) of a matrix multiply for different problems and data storage schemes

Problem	Preconditioner	Storage	Iterations	CPU (secs)	Speedup
ECN(5000,5)	Jacobi	Ellpack	726	0.76	8.6
(accel=CG)		Diagonal	726	0.73	6.8
diag scaling		Coord	726	1.01	6.6
	Incomplete	Ellpack	222	3.28	1.1
	Cholsesky	Diagonal	222	4.33	0.9
	(fill=2)				
	Least Squares	Ellpack	228	0.57	8.9
	Polynomial	Diagonal	228	0.41	10.4
	(ord=3)	Coord	228	0.71	13.9
	Neumann	Ellpack	504	0.89	12.6
	Polynomial	Diagonal	504	0.66	14.52
	(ord=3)	Coord	504	1.01	14

Table 1: Results of running NSPCG on selected problems

There are several comments to be made about these results. Firstly, the coordinate data format is internally translated into a data format which is equivalent to the jagged diagonal format. With the inclusion of an appropriate VOCL statement, this vectorizes well. Secondly, this chart illustrates how poorly the incomplete Cholesky decomposition performs on vector computers, due to the presence of triangular solves. It is expected that IC decomposition methods may have their place when the system is moderately dense. This is because the triangular solve can be vectorized along the row, and so vector lengths can be reasonable only if the matrix is moderately dense. Alternative strategies may be found with Markowitz and multi-frontal methods [4].

4 Tools

A comment made at the recent Large Scale Scientific Computation workshop was that designers of numerical software should consider going beyond the concept of a monolithic library to that of a toolkit. It is clear that the monolithic library interface of NSPCG and SPARSKIT fail rather badly, as described above.

It is clear in the context of LIST that a tool is required to allow methods and preconditioners to be swapped in and out ("plug and play"). The fortran code given in the appendix does this for NSPCG, however it is cluttered by miscellaneous code related to such things as allocating space for matrices, vectors, workspace etc, conversion between different data formats, initialising parameters and such like. A superior testbed situation is given by Ellpack [8], which uses a customized preprocessor to implement a high level description of the problem. In terms of iterative solvers, one can imagine that the user can

- specify amongst a list of test examples by keywords, or alternatively give the filename of a matrix stored in Harwell Boeing format
- select a storage scheme by keyword, or specify an entirely new storage scheme by specifying three statement functions:

  $$\begin{aligned} \texttt{row} &= frow(\texttt{A}, \texttt{IA}, \texttt{JA}) \\ \texttt{col} &= fcol(\texttt{A}, \texttt{IA}, \texttt{JA}) \\ \texttt{val} &= fval(\texttt{A}, \texttt{IA}, \texttt{JA}) \end{aligned}$$

  This would be included into a suitable routine that performs the conversion. It may also be used to automatically generate matrix multiplication routines, and preconditioners, however it may be better to have a fixed number of "internal" formats, which can highly optimized.
- Specify the preconditioner and accelerator by keyword
- Give parameters values with meaningful parameter names.

Rather than implementing a fully customized preprocessor along the lines of the Ellpack model, the Fortran macro preprocessor [7] (included in the same package, or available from netlib in TOMS) could be used to simplify the preparation.

Further ancilliary tools include graphics facilities for plotting the sparsity structure, analysing matrix properties and so on that are included in SPARSKIT. Other tools which are quite important are some of the massaging routines that attempt to transform the matrix into a more dense banded or block structure by permutation of rows and columns.

5 Conclusion

The package NSPCG was examined with respect to its use as a production tool on the SNI S600 supercomputer. With the addition of a single VOCL statement, it runs with a vector to scalar speedup of between 6 to 15 times. However, the package is not only difficult to use, it is also not very robust. Of the three problems selected in this work, it could only solve the simplest (a SPD matrix). It otherwise would often fall over with floating exceptions, or, even worse, with a C7 abend code.

In respect to its possible use in the LIST project, NSPCG may be a source of code, but the package would need to be totally pulled apart and rewritten with attention paid to the robustness problem, and to make use of emerging standards such as the sparse level 3 BLAS.

Furthermore, it was reported at the LSSC conference that Jack Dongarra was working on an iterative solution package to be distributed by netlib. This will also need examination.

The package SPARSKIT is a useful collection of tools for manipulating sparse matrices. It is, however, marred by an inconsistent interface, and inadequate documentation, and occasional bugs. Further, the proposed sparse BLAS will make redundant the FORMATS and BLASSM modules of SPARSKIT.

From empirical tests conducted over the three test problems selected, it is clear that of all the data formats commonly in use, the jagged diagonal data format is the one most suited to vector architectures. This confirms the results of some other studies [1, 10].

The tests performed with NSPCG indicated the not too surprising result that incomplete Cholesky decomposition methods perform very poorly. This is because of the essentially serial nature of inverting a triangular matrix with a small number of elements per row. If the initial problem is moderately dense (a few hundred elements per row) then IC methods may hold some promise. On the other hand, polynomial approximations to the inverse matrix perform very efficiently, and may be used to great effect, at least for certain classes of problems.

A Matrix Multiplication Test

```
      PROGRAM MATMUL
C PERFORM TIMINGS OF MATRIX MULTIPLICATION FOR DIFFERENT STORAGE SCHEMES
      INTEGER MAXNZ,NRHS,NROW,NCOL,NNZ,IERR,MAXR
      PARAMETER(MAXNZ=200000,NTIMES=1000)
      DOUBLE PRECISION A(MAXNZ),B(MAXNZ),RESULT(MAXNZ),X(MAXNZ)
      DOUBLE PRECISION TRES(MAXNZ)
      INTEGER IA(MAXNZ),JA(MAXNZ),IB(MAXNZ),JB(MAXNZ),PERM(0:MAXNZ-1)
      CHARACTER GUESOL*2,TITLE*72,KEY*8,TYPE*3
      LOGICAL COMP
      NRHS=MAXNZ

C GENERATE MATRIX
      CALL READMT(MAXNZ,MAXNZ,1,10,A,JA,IA,RESULT,NRHS,GUESOL,NROW,
     $     NCOL,NNZ,TITLE,KEY,TYPE,IERR)
      IF (IERR.NE.0) CALL ERR('READMT',IERR)
C     NROW=5000
C     CALL ECN(NROW,10,NNZ,IA,JA,A,MAXNZ,IERR)
C      IF (IERR.NE.0) CALL ERR('ECN',IERR)
C      CALL COICSR(NROW,NNZ,A,JA,IA,RESULT)
      WRITE (*,*) 'NO OF NONZERO ELEMENTS =',NNZ

      DO 100 I=1,NROW
        X(I)=1
100     CONTINUE
C COMPRESSED SPARSE ROW FORMAT
      CALL ITIME
      DO 1 K=1,NTIMES
      MAXR=0
      DO 3 I=1,NROW
        MAXR=MAX(MAXR,IA(I+1)-IA(I))
        RESULT(I)=0
3       CONTINUE
      DO 1 J=0,MAXR-1
         DO 2 I=1,NROW
           IF (J+IA(I).LT.IA(I+1))
     &      TRES(I)=TRES(I)+A(J+IA(I))*X(JA(J+IA(I)))
 2       CONTINUE
 1    CONTINUE
      CALL TIMEIT('COMPRESSED SPARSE ROW')

C COORDINATE FORMAT
      CALL CSRCOO(NROW,3,MAXNZ,A,JA,IA,NNZ,B,IB,JB,IERR)
```

```
      IF (IERR.NE.0) CALL ERR('CSRCOO',IERR)
      CALL ITIME
      DO 12 K=1,NTIMES
      DO 11 I=1,NROW
         RESULT(I)=0
 11    CONTINUE
      DO 12 I=1,NNZ
         RESULT(IB(I))=RESULT(IB(I))+B(I)*X(JB(I))
 12    CONTINUE
      CALL TIMEIT('COORDINATE')
      WRITE (*,*) 'CORRECT MAT MUL IS ',COMP(TRES,RESULT,NROW)

C DIAG FORMAT
      NCOL=MAXNZ/NROW
C MAKE SURE NCOL IS ODD TO AVOID BANK CONFLICTS
      IF (.NOT.BTEST(NCOL,0)) NCOL=NCOL-1
      NCOL1=NCOL
C NOTE BECAUSE JOB=10, THE LAST FEW PARAMETERS ARE NOT USED
C HOWEVER, NCOL WILL CONTAIN THE NUMBER OF DIAGONALS, IB WILL CONTAIN TH
C OFFSETS, AND JB IS USED AS WORKSPACE
      CALL CSRDIA(NROW,NCOL,10,A,JA,IA,NROW,B,IB,B,JB,IB,JB)
      CALL ITIME
      DO 22 K=1,NTIMES
      DO 21 I=1,NROW
         RESULT(I)=0
 21    CONTINUE
      DO 22 I=1,NCOL
         JJ=I
         DO 23 J=1,NROW
           RESULT(J)=RESULT(J)+B(JJ) * X(J+IB(I))
           JJ=JJ+NCOL1
 23       CONTINUE
 22    CONTINUE
      CALL TIMEIT('DIAGONAL')
      WRITE (*,*) 'CORRECT MAT MUL IS ',COMP(TRES,RESULT,NROW)


C ELLPACK FORMAT
C NOTE BECAUSE JOB=11, THE LAST FEW PARAMETERS ARE NOT USED
C HOWEVER, NCOL WILL CONTAIN THE NUMBER OF DIAGONALS, IB WILL CONTAIN TH
C COLUMN NUMBERS, AND JB IS USED AS WORKSPACE
      CALL CSRELL(NROW,A,JA,IA,NCOL1,B,IB,NROW,NCOL,IERR)
      IF (IERR.NE.0) CALL ERR('CSRELL',IERR)
      CALL ITIME
```

```
      DO 32 K=1,NTIMES
      DO 31 I=1,NROW
         RESULT(I)=0
 31    CONTINUE
      DO 32 I=1,NCOL
         JJ=I
         DO 33 J=1,NROW
           RESULT(J)=RESULT(J)+B(JJ) * X(IB(JJ))
           JJ=JJ+NCOL1
 33       CONTINUE
 32    CONTINUE
      CALL TIMEIT('ELLPACK')
      WRITE (*,*) 'CORRECT MAT MUL IS ',COMP(TRES,RESULT,NROW)

C JAGGED DIAGONAL STORAGE
C NCOL WILL HAVE NUMBER OF DIAGS, PERM CONTAINS THE PERMUTATION VECTOR
      CALL CSRJAD(NROW,A,JA,IA,NCOL,PERM,B,JB,IB)
      CALL ITIME
      DO 41 K=1,NTIMES
      DO 43 I=1,NROW
         RESULT(I)=0
 43      CONTINUE
      DO 41 I=1,NCOL
*VOCL LOOP,NOVREC
         DO 42 J=IA(I),IA(I+1)-1
            RESULT(PERM(J-IA(I)))=RESULT(PERM(J-IA(I)))+B(J)*X(JA(J))
 42       CONTINUE
 41    CONTINUE
      CALL TIMEIT('JAGGED DIAGONAL')
      WRITE (*,*) 'CORRECT MAT MUL IS ',COMP(TRES,RESULT,NROW)


      STOP
      END


      SUBROUTINE ITIME
      DOUBLE PRECISION T0
      COMMON /TIME/T0
      CALL CLOCK(T0,0,2)
      RETURN
      END

      SUBROUTINE TIMEIT(C)
```

```
      CHARACTER*(*) C
      DOUBLE PRECISION T0,T1
      COMMON /TIME/T0
      CALL CLOCK(T1,0,2)
      WRITE (*,*) 'TIME FOR ',C,' IS ',T1-T0,' SECS'
      RETURN
      END



      SUBROUTINE ERR(SUB,ERRNO)
C     IMPLICIT NONE
      CHARACTER*(*) SUB
      INTEGER ERRNO
      WRITE (*,*) SUB,' IERR = ',ERRNO
      STOP
      END

      LOGICAL FUNCTION COMP(X,Y,N)
      INTEGER N,I
      LOGICAL RESULT
      DOUBLE PRECISION X(N),Y(N)
      RESULT=.TRUE.
      DO 1 I=1,N
        RESULT=RESULT.AND.X(I).EQ.Y(I)
1       CONTINUE
      COMP=RESULT
      RETURN
      END
```

B NSPCG Test

```
      PROGRAM ITER
C THIS PROGRAM RUNS A TEST EXAMPLE THROUGH NSPCG IN ORDER TO GENERATE
C     SOMEPROFILES

      IMPLICIT CHARACTER (A-Z)
C IF HARBOE IS TRUE, THEN THE MATRIX WILL BE READ IN ON UNIT 10,
C OTHERWISE ECN(NMX,C) WILL BE USED
C DATAFT CONTAINS THE DATA FORMAT PASSED TO NSPCG:
C     1 = ELLPACK FORMAT
C     3 = DIAGONAL FORMAT
```

```
C     5 = COORDINATE FORMAT
C THE LAST DIGIT OF THE FIRST PARAMETER PASSED TO NSPCG SHOULD ALSO
C EQUAL DATAFT

      LOGICAL HARBOE
      INTEGER DATAFT
      PARAMETER( HARBOE=.TRUE.,DATAFT=1)

      INTEGER N,NN,C,NW,NCOL,NDIM,NMX,NC,NRHS
      PARAMETER(NMX=5000,NN=30000,C=5,NCOL=300,NW=500000)

      INTEGER IA(NN),JA(NN),IERR,NE,JCOEF(NMX,NCOL),NDIAG,NW1,INW1
      INTEGER IWORK(NW),IPARM(25),P(1),IP(1)
      DOUBLE PRECISION A(NN),COEF(NMX,NCOL),WORK(NW),RPARM(16),U(NMX),
     $     RHS(NMX),UBAR(1),TEMP
      CHARACTER GUESOL*2,TITLE*72,KEY*8,TYPE*3

C EXTERNAL DECLARATIONS FOR PRECONDITIONERS AND ACCELERATORS
      EXTERNAL RICH1,RICH2,RICH3,RICH4,RICH5,JAC1,JAC2,JAC3,JAC4,JAC5,
     $     LJAC2,LJAC3,LJACX2,LJACX3,SOR1,SOR2,SOR3,SOR6,SOR7,SSOR1,
     $     SSOR2,SSOR3,SSOR6,SSOR7,IC1,IC2,IC3,IC6,MIC1,MIC2,MIC3,MIC6,
     $     LSP1,LSP2,LSP3,LSP4,LSP5,NEU1,NEU2,NEU3,NEU4,NEU5,LSOR2,
     $     LSOR3,LSSOR2,LSSOR3,LLSP2,LLSP3,LNEU2,LNEU3,BIC2,BIC3,BIC7,
     $     BICX2,BICX3,BICX7,MBIC2,MBIC3,MBIC7,MBICX2,MBICX3,MBICX7,
     $     RS6,RS7
      EXTERNAL CG,SI,SOR,SRCG,SRSI,BASIC,ME,CGNR,LSQR,ODIR,OMIN,ORES,
     $     IOM,GMRES,USYMLQ,USYMQR,LANDIR,LANMIN,LANRES,CGCR,BCGS


C GENERATE TEST MATRIX
      IF (HARBOE)  THEN
        CALL READMT(NN,NN,1,10,A,JA,IA,RHS,NRHS,GUESOL,N,NC,
     $     NE,TITLE,KEY,TYPE,IERR)
        IF (IERR.NE.0) CALL ERR('READMT',IERR)
      ELSE
        N=NMX
        CALL ECN(N,C,NE,IA,JA,A,NN,IERR)
        IF (IERR.NE.0) CALL ERR('ECN',IERR)
        CALL COICSR(N,NE,A,JA,IA,IWORK)
        ENDIF

C      OPEN(UNIT=11,FILE='RZ69.MAT.PS')
C      CALL PLTMTPS(N,N,0,JA,IA,'DCN(100,5)','','',0,11)
```

```
C      CASE (DATAFMT)
       GOTO (100,300,500),DATAFT/2+1

100    CONTINUE
C CONVERT IT TO ELLPACK FORMAT
       WRITE (*,*) 'ELLPACK'
       CALL CSRELL(N,A,JA,IA,NCOL,COEF,JCOEF,N,NDIAG,IERR)
       IF (IERR.NE.0) CALL ERR('CSRELL',IERR)
       NDIM=N
       GOTO 200

300   CONTINUE
C CONVERT IT TO DIAGONAL FORMAT
      NDIAG=NCOL
       WRITE (*,*) 'DIAGONAL'
      CALL CSRDIA(N,NDIAG,10,A,JA,IA,N,COEF,JCOEF,A,JA,IA,WORK)
      IF (IERR.NE.0) CALL ERR('CSRDIA',IERR)
      NDIM=N
      GOTO 200

500   CONTINUE
C FOR COO FORMAT NSPCG
       WRITE (*,*) 'COORD'
      CALL CSRCOO(N,3,N*NCOL/2,A,JA,IA,NE,COEF,JCOEF,JCOEF(NE+1,1),IERR)
      IF (IERR.NE.0) CALL ERR('CSRCOO',IERR)
      NDIM=NE
      NDIAG=NE

C     END OF CASE STATEMENT
200   CONTINUE

      NW1=NW
      INW1=NW
      CALL DFAULT(IPARM,RPARM)

C ITMAX
      IPARM(2)=1000
C OUTPUT LEVEL
      IPARM(3)=3
C DIMENSION OF KRYLOV SPACE FOR TRUNCATED METHODS
      IPARM(9)=20
C RESTART FREQUENCY
      IPARM(10)=200
C SIZE OF LARGEST HESSENBERG MATRIX USED IN ORTHOMIN
```

```
      IPARM(11)=40
C SET THE STORAGE MODE
      IPARM(12)=DATAFT
C DIAGONAL SCALING
      IPARM(13)=0
C FILL IN
      IPARM(16)=3
C TRUNCATION FOR BANDED APPROX INVERSE
      IPARM(17)=2
C DEGREE FOR POLYNOMIAL PRECONDITIONER
      IPARM(25)=3
C STOPPING TEST ACCURACY (ZETA)
      RPARM(1)=1E-11

C SET RHS AND INITIAL U
      CALL VFILL(N,U,0.0)
      CALL VFILL(N,RHS,1.0)

      CALL NSPCG(LSP1,LSQR,NDIM,NCOL,N,NDIAG,COEF,JCOEF,P,IP,U,UBAR,
     $     RHS,
     $     WORK,IWORK,NW1,INW1,IPARM,RPARM,IERR)
      WRITE (*,*) 'INT WSP=',INW1,'REAL WSP=',NW1
      IF (IERR.NE.0) CALL ERR('NSPCG',IERR)


      STOP
      END

      SUBROUTINE ERR(SUB,ERRNO)
C     IMPLICIT NONE
      CHARACTER*(*) SUB
      INTEGER ERRNO
      WRITE (*,*) SUB,' IERR = ',ERRNO
      STOP
      END

      DOUBLE PRECISION FUNCTION SECOND()
      REAL*8 C
      CALL CLOCK(C,0,2)
      SECOND=C
      RETURN
      END
```

References

[1] E. C. Anderson and Y. Saad. Solving sparse triangular systems on parallel computers. Technical Report 794, University of Illinois, CSRD, Urbana, IL, 1988.

[2] I. S. Duff, M. Marrone, and G. Radicati. Sparse matrix test problems. *ACM Transactions on Mathematical Software*, 15:1–14, 1989. available by anonymous ftp from:
orion.cerfacs.fr: /pub/HARWELL_BOEING/HAR_BOEING.

[3] I. S. Duff, M. Marrone, and G. Radicatti. A proposal for user level sparse blas. to appear, 1992.

[4] At the recent LSSC conference, Prof. Duff discussed his work on Markowitz methods and multifrontal methods. His latest Markowitz algorithm is called MA48, and is a successor to the successful MA28 program.

[5] T. C. Oppe, W. D. Joubert, and D. R. Kincaid. *NSPCG User's Guide.* available by anonymous ftp from emx.utexas.edu: /pub/mathlib/itpack/nspcg.

[6] O. Østerby and Z. Zlatev. *Direct Methods for Sparse Matrices*, volume 157 of *Lecture Notes in Computer Science.* Springer, NY, 1983.

[7] C. J. Ribbens, J. R. Rice, and W. A. Ward. A simple macro processor. *ACM Trans. Math. Software*, 10(4):410, December 1984.

[8] J. R. Rice and R. F. Boisvert. *Solving Elliptic Problems Using ELLPACK.* Springer, NY, 1985.

[9] Y. Saad. *SPARSKIT: a Basic Tool Kit for Sparse Matrix Computations.* available by anonymous ftp from icarus.riacs.edu: /pub/SPARSKIT.

[10] Y. Saad. Krylov subspace methods on supercomputers. *SIAM J. Scient. Stat. Comp.*, 10:1200–1232, 1989.