FSPAK

An interface for public domain sparse matrix subroutines

Authors: Miguel Perez-Enciso, University of Wisconsin, USA
UdL-IRTA, Spain
Ignacy Misztal, University of Illinois, USA
Mauricio E. Elzo, University of Florida, USA

December, 1992. Last updated December 1994.

INTRODUCTION

FSPAK is a set to subroutines, consisting of about 1000 lines of fortran code, that utilize some public domain sparse matrix code from George and Liu (1981) to solve a symmetric positive definite linear system of equations. In addition,FSPAK incorporates other non-standard procedures in commercial packages such as sparse vector methods (Tinney et al., 1985), sparse matrix inversion using Takahashi et al. (1971) algorithm or computing the determinant. FSPAK is free to use for noncommercial applications.

PACKAGE OVERVIEW

FSPAK requires that the user provides the upper triangular matrix in the IA,JA,A form (i.e., Kincaid et al., 1982; Duff et al., 1989; George and Liu, 1981) where IA denotes the starting locations of each of the rows in JA and A, JA contains the column indices of the nonzero elements (NZE), and A the value of these elements.

As other direct solver packages, FSPAK breaks the solution process into several steps:

1. Ordering: a permutation vector and its inverse are found
2. Symbolic factorization: the compressed pointer data structures of U
   is obtained
3. Numerical factorization: computes the Cholesky decomposition U'D U
   FSPAK allows the factorization of non-full rank matrices

Once factorization is complete, several options are available:

4. Dense vector solving (full forward followed by full backward substitution)
5. Sparse vector solving (fast forward followed by fast backward substitution)

6. Compute the determinant or the log-determinant
7. Sparse inverse (memory- and speed-optimized)

In addition, FSPAK provides with a restart facility where the ordering
is read from a file previously written, and an option that provides with
timing, storage required and other statistics.There is also an option
that checks whether Ax = b, x being the solution provided by the dense vector solving.

SUBROUTINES REQUIRED

fspak routines
(they can be obtained from misz.animal.uiuc.edu)
   fspak.f (main driver)
   fspak_sub.f

sparspak routines:
(they can be obtained from netlib @ research.att.com)
   genqmd.f
   qmdrch.f
   qmdupd.f
   qmdqt.f
   qmdmrg.f
   qmdrch.f
   smbfct.f

test program
   ftest.f

GENERAL CALLING ARGUMENT

```
CALL FSPAK (OPTION,     N,      IA,     JA,     A,      B,
FLAG,
            IOUT,       IOOR,   AVAILABLE,      NEEDED, IS,
FNODE,
            BNODE,      FNSEG,  BNSEG,  FX,     FEQB,   IRANK)
```

where

```
OPTION    = Option to be run (integer scalar)
            The following options are available so far:
       10 = find order
       20 = symbolic factorization
       40 = numerical factorization
       50 = solve (full forward/backward)
       51 = initialize for sparse solving
       52 = sparse solving (fast forward/backward)
       54 = computes determinant
       55 = computes log-determinant
       56 = checks whether Ax = b
```

```
       60 = computes sparse inverse (optimized for speed)
       61 = computes sparse inverse (optimized for memory)
       70 = restart (read ordering)
       80 = print statistics

N         = dimension of the system (integer scalar)
IA        = vector specifying the pointers for the beginning of each
row
            of the upper triangular sparse stored (integer vector of
            dimension N+1)
JA        = vector specifying the column indices (integer of
dimension nze)
A         = vector with the numeric values of the NZE of the matrix
            (real*8 of dimension nze)
B         = vector with the right-hand-side (real*8 of dimension N)
            after option=50, B contains the solution vector of
dimension N
            after option=52, B contains the solution for the
elements
                  required (vector of dimension BNSEG)
            after option=54 or 55,  B is a scalar with the output
value
FLAG      = error flag (integer scalar)
            Possible values so far are:
      -99 = not implemented option
       -2 = error in OPTION=restart
       -1 = error in OPTION=restart
        0 = no error
        1 = error in checking solution ( Ax <> b)
  OPTION*
    100+1 = not enough memory in IS to perform option OPTION
  4000000
    + ROW = zero or negative pivot for row ROW during numerical
factoriz.

IOUT      = number of output unit (integer scalar)
IOOR      = number of unit where ordering is printed (integer
scalar)
AVAILABLE = declared dimension of working vector IS (integer scalar)
NEEDED    = needed dimension of IS to perform that particular option
            (integer scalar)
FNODE     = vector containing the nodes of the nonzero elements in
            the sparse vector for the fast forward (integer of
dimension
            FNSEG)
BNODE     = vector containing the nodes of the nonzero elements in B
            for which the solutions are wanted in the fast backward
            (integer of dimension BNSEG)
FNSEG     = NZE in B (integer scalar)
BNSEG     = number of elements in B for which solution is wanted
            (integer scalar)
FX        = (OPTION=52) numeric values of NZE in B
          = (OPTION=56) numeric values of the rhs, B
            (real*8 of dimension FNSEG, if OPTION=52, or N if
```

OPTION=56)
FEQB   = flag to indicate whether the forward and backward paths
            are equal
          0 = not equal
          1 = equal
IRANK   = rank of the matrix

PROCEDURES USAGE

FSPAK is written such that only one subroutine with the same arguments
has to be called in all steps. However, not all arguments are used in
all subroutines. In the following, dummy arguments are written in lower
case and needed arguments in UPPER case.

1. ORDERING

This should be the first step in solving the linear system, and it is
usually the most expensive one. Finding an ordering vector is achieved
by calling

```
CALL FSPAK (10,         N,      IA,     JA,     a,      b,
FLAG,
            IOUT,       IOOR,   AVAILABLE,      NEEDED, IS,
fnode,
            bnode,      fnseg,  bnseg,  fx,     feqb,  irank)
```

FSPAK calls the minimum degree ordering subroutines of George and Liu
(1981). It can also be easily modified to call the multiple minimum
degree subroutines of Liu (1985) (These subroutines are not provided).
Once the permutation vectors are found, these are written to unit IOOR
with format '(10i7)'.

2. SYMBOLIC FACTORIZATION

In this step the data structures for upper triangular factor, including
of course the "fill-ins", is created. The option is

```
CALL FSPAK (20,         N,      IA,     JA,     a,      b,
FLAG,
            IOUT,       ioor,   AVAILABLE,      NEEDED, IS,
fnode,
            bnode,      fnseg,  bnseg,  fx,     feqb,  irank)
```

FSPAK calls SMBFCT (George and Liu, 1981).

3. NUMERICAL FACTORIZATION

```
CALL FSPAK (40,         N,      IA,     JA,     A,      b,
FLAG,
            IOUT,       ioor,   AVAILABLE,      NEEDED, IS,
fnode,
            bnode,      fnseg,  bnseg,  fx,     feqb,  IRANK)
```

FSPAK calls a rearranged version of subroutine GSFCT (George and Liu, 1981), that resulted in an increase of efficiency of about 15%. Vector A should contain current numerical values. IRANK returns the rank of the matrix.

4. DENSE VECTOR SOLVING

```
CALL FSPAK (50,         N,      ia,     ja,     a,      B,
FLAG,
            IOUT,       ioor,   AVAILABLE,      NEEDED, IS,
fnode,
            bnode,      fnseg,  bnseg,  fx,     feqb,  irank)
```

FSPAK calls a full forward/backward solving routine. On output, B is overwritten by the vector of solutions. This is the cheapest step and, thus, it is fairly inexpensive to solve the same system of equations with different right hand sides.

5. SPARSE VECTOR SOLVING

This is one of the options in FSPAK not implemented so far in commercial packages.It uses the sparse vector methods well described in Tinney et al. (1985). It is of interest when either the right hand side is a sparse vector (only a few elements are not zero) or only a subset of the whole solution vector is required, or a combination of both. In all cases, dramatic increases in speed can be achieved. The basic idea is that if the rhs vector is sparse, only an ordered subset of the columns of U' and D are required. This subset is known as the factorization path if the vector is a singleton (only one nze) or as the composite or segmented factorization path if there are more than one nze. Forward substitution following a factorization path is called 'fast-forward'. Similarly, if

only certain elements of the solution vector have to be computed, backward
substitution has to be done only with a subset of the rows of U, given by
the segmented factorization path of the elements required. This is called
'fast-backward'. Computing factorization paths is inexpensive provided that
the number of the lowest-numbered nonzero off-diagonal element in each row
of U can be accessed without searching. This is the case in the data
structures considered here. In FSPAK, sparse vector solving is preceded
by an intilialization step that computes the path table:

```
CALL FSPAK (51,         N,      ia,     ja,     a,      b,
FLAG,
            IOUT,       ioor,   AVAILABLE,      NEEDED, IS,
fnode,
            bnode,      fnseg,  bnseg,  fx,     feqb,  irank)
```

After this step, sparse solving can be called any number of times
without further explicit reinitializing. However, it should be borne
in mind that the sparse vector option destroys the data structures
for sparse solving and vice versa. Then, if sparse vector solving is
needed after sparse vector option, the initializing routine should be called
again. Calling the determinant routines dos not affect the sparse vector
data structures, except that B is overwritten. To call for sparse vector
solving:

```
CALL FSPAK (52,         N,      ia,     ja,     a,      B,
FLAG,
            IOUT,       ioor,   AVAILABLE,      NEEDED, IS,
FNODE,
            BNODE,      FNSEG,  BNSEG,  FX,     FEQB,  irank)
```

Sparse vector methods are of particular interest in obtaining certain
elements of the inverse. For instance, if only the diagonal elements
of the inverse of the matrix are needed the following piece of code
is likely to be the most efficient:

```
      do i=1,n
         call fspak (52, n, ia, ja, a, b, flag, iout, ioor,
available,
     +               needed, is,
     +               i, i, 1, 1, 1., 1 )
c       now b(1) contains the diagonal element of the i-th row of
the
c       inverse
      enddo
```

This variant stores the diagonal elements of the inverse in vector B

```
      do i=1,n
         call fspak (52, n, ia, ja, a, b(i), flag, iout, ioor,
available,
     +               needed, is,
     +               i, i, 1, 1, 1., 1 )
c       now b(i) contains the diagonal element of the i-th row of
the
c       inverse
      enddo
```

6. DETERMINANT

Once the factorization is complete, the determinant is obtained by multiplying the elements of the diagonal matrix D in U'DU.

```
CALL FSPAK (54,         N,      ia,     ja,     a,      B,
FLAG,
            IOUT,       ioor,   AVAILABLE,      NEEDED, IS,
fnode,
            bnode,      fnseg,  bnseg,  fx,     feqb,  irank)
```

gives in B(1) the determinant, while

```
CALL FSPAK (55,         N,      ia,     ja,     a,      B,
FLAG,
            IOUT,       ioor,   AVAILABLE,      NEEDED, IS,
fnode,
            bnode,      fnseg,  bnseg,  fx,     feqb,  irank)
```

gives in B(1) the logarithm of the determinant.

7. SPARSE INVERSE (speed-optimized)

The NZE of the inverse that are also non zero in the triangular factor U,
plus all the diagonal elements can be efficiently obtained using the Takahashi et al (1971) method. In FSPAK, the option is

```
CALL FSPAK (60,         N,      IA,     JA,     A,      b,
FLAG,
            IOUT,       ioor,   AVAILABLE,      NEEDED, IS,
fnode,
            bnode,      fnseg,  bnseg,  fx,     feqb,  irank)
```

This is the option likely to require more memory. Upper-triangular A,JA,A is
augmented by lower-triangular elements, and A is overwritten with the
corresponding elements of the inverse. Only those elements are overwritten
that are nonzero in the factorization. This includes all elements of

the
original matrix. Since, both upper and lower elements are stored in IA,JA,A
after return, it should be checked that there is enough storage available in
JA and A (see below in traps and tips). This options is fast, epecially on
vector computers, but it requires 250-400% more memory than that required by
the numerical factorization.

8. SPARSE INVERSE (memory-optimized)

This option is similar to the previous option, except that it is optimized for
memory.

```
CALL FSPAK (61,         N,      IA,     JA,     A,      b,
FLAG,
            IOUT,       ioor,   AVAILABLE,      NEEDED, IS,
fnode,
            bnode,      fnseg,  bnseg,  fx,     feqb,  irank)
```

In this option, IA is not changed, JA may be permuted, and A is overwritten
with the corresponding elements of the inverse. This options requires only
20-80% more memory than the factorization, but is up to 0-50% slower, the
difference being computer dependent. Inversion with option 61 (but not 60)
destroys the factor so that after option 61 any operations involving the
factor, e.g., solving or computing a determinant will yield incorrect results.

9. RESTART

Because ordering is the most expensive step in most cases, ordering vectors
are written to unit IOOR. During subsequent runs of the package, these vectors
can be recovered so that the ordering step can be
skipped. In order to do so replace Fspak (10, ...) with

```
CALL FSPAK (70,         N,      ia,     ja,     a,      b,
FLAG,
            IOUT,       IOOR,   AVAILABLE,      NEEDED, IS,
fnode,
            bnode,      fnseg,  bnseg,  fx,     feqb,  irank)
```

If the number of equations in unit IOOR is different from N, FLAG is

set to
-1. If there is an error in reading IOOR because of out of bounds reading
, FLAG is set to -2.

10. STATISTICS

At any time,

```
CALL FSPAK (80,         N,      ia,     ja,     a,      b,
flag,
            IOUT,       ioor,   AVAILABLE,      NEEDED, IS,
fnode,
            bnode,      fnseg,  bnseg,  fx,     feqb,  irank)
```

can be called to obtain a printout in unit IOUT of timing, storage and
other statistics.

11. CHECKING SOLUTION

The option

```
CALL FSPAK (56,         N,      IA,     JA,     A,      B,
FLAG,
            IOUT,       ioor,   AVAILABLE,      NEEDED, IS,
fnode,
            bnode,      fnseg,  bnseg,  FX,     feqb,  irank)
```

checks that the solution obtained from dense vector solving satisfies
the condition Ax=b. Here B contains the solution vector, x, and FX
contains the original right hand side.Thus, if this option is used,
the vector FX should a dimension of at least N. On output, FX(1) contains
the mean absolute difference between Ax and b. If this difference is
greater than TOL (set to 10d-6 in FSPAK), FLAG is set to 1.

CUSTOMIZING FSPAK

In the first call to FSPAK, the function FRATIO is called. This function
returns the value of 2 if the storage for integer is half of that for
real*8, as in most machines, or 1 if is the same (such as in the Cray
machines). The user is advised to check that this function returns
the proper value.

FSPAK uses a timing function called SECOND() that returns the cpu clock time.
This function should be specific to each system. See Appendix for examples of
this function on several computers.

TRAPS AND TIPS

1. IA,JA,A should contain only the NZE of the upper triangular of the
matrix. If the lower triangular elements are present, unpredictable results
can be expected.

2. All real variables in FSPAK are declared as REAL*8. For single precision
operation, all REAL*8 and DOUBLE PRECISION statements should be substituted by
REAL.

3. The error flag should remain 0 after calling FSPAK with any option. FSPAK
 with OPTION=80 can be called any time to check for maximum amount of
storage required in the previous calls to FSPAK.

4. Checks for storage bounds are performed by may be imperfect. If
unreasonable results are obtained, this could be due to too little, but
undetected, storage available.

5. It is responsability of the user to check that all diagonal elements must be
positive; otherwise unpredictable results may occur.

6. In the solving option, the rhs vector B is overwritten with solutions..

7. In the sparse vector solving option, if the forward path is equal to the
backward path, then

CALL FSPAK (. . . FNODE,FNODE,FNSEG,FNSEG,FX,1)

and

CALL FSPAK (. . . FNODE,BNODE,FNSEG,FNSEG,FX,0)

will work, though the last call will be slightly less efficient.

8. Different options use common storages in vector IS differently and
sometimes destroy vectors used by the other options.. For instance,

"sparse inverse" option (60) destroys vectors by "sparse matrix solving"
(51). Then, subsequent application of option 52 would necessitate calling
option 51 again.

9. None of the nodes in FNODE and BNODE should exceed the rank of the matrix, N.

10. In sparse inverse option (60) the original matrix is overwritten by
those of the inverse. In addition, lower triangulars are also stored, so the user should check that there is enough storage, that is, JA
and A have to be declared to have at least (IA(N+1)-1-N)*2 elements.

11. It can be found useful to insert the first lines of FSPAK to the application
program, so that mnemonic names can be used in calling FSPAK

```
      integer order,symfac,inval,numfac,solve,sinit,ssolve
     +       ,det,ldet,spin,restart,stat
      parameter (order=10,  symfac=20,  inval=30,  numfac=40,
solve=50,
     +           sinit=51,  ssolve=52,  det=54,    ldet =55,
spin=60,
     +           restart=70,stat=80  )
```

12. If no sparse vector methods are needed, and the rank is of no interest, FSPAK can be called using only the first 12 arguments, i.e.

```
CALL FSPAK (OPTION,     N,      IA,     JA,     A,      B,
FLAG,
            IOUT,       IOOR,   AVAILABLE,      NEEDED, IS)
```

MEMORY ALLOCATION AND REQUIREMENTS IN FSPAK

Guidelines for memory requirements in FSPAK. The actual maximum needed is written to unit IOUT under the heading "MAXIMUM NEEDED". Check the subroutine FIGUREOUT.

Notation:
- N=rank of the matrix
- NZE= no. of non zero elements in the upper triangular
- NZU=no. of nonzero elements in the factored matrix (upper triangular)
  = NZE + fillins
- RATIO=2 if real*8 requires double storage than integer, 1 otherwise

OPTION	APPROX. STORAGE REQUIRED FOR INTEGER VECTOR 'IS'
10 (order)	9*N+2*NZE
20 (sym. fact.)	8*N+2*NZE+NZU
40(num. fact.)	N*(5+2*RATIO)+NZE*(RATIO+1)+NZU*(RATIO+1)
50,51,52,54,55	less than num. fact.
60 (sparse inv.)	N*(9+4*RATIO)+NZU*(3+2*RATIO)
61 (sparse inv.)	N*(5+2*RATIO)+NZU*(RATIO+2)

With respect to what pointers can be nil, these are written in lower case
in the manual for each option, i.e. for OPTION=10 (order):

```
CALL FSPAK (10,         N,      IA,     JA,     a,      b,
FLAG,
            IOUT,       IOOR,   AVAILABLE,      NEEDED, IS,
fnode,
            bnode,      fnseg,  bnseg,  fx,     feqb,  irank)
```

will render exactly the same results as

```
CALL FSPAK (10,         N,      IA,     JA,     i,      i,
FLAG,
            IOUT,       IOOR,   AVAILABLE,      NEEDED, IS,
i    ,
            i    ,      i    ,  i    ,  i ,     i   )
```

OPTIMIZATION AND SUPERCOMPUTERS

During factorization, almost all the time is spent in subroutine FGSFCT in the
following loop:

```
                  DO II = ISTRT, ISTOP
                     ISUB = NZSUB(I)
                     TEMP(ISUB) = TEMP(ISUB) + LNZ(II)*LJK
                     I = I + 1
                  ENDDO
```

In the inversion routine (FSINVERSE), most of computing is in the loop below:

```
            do k=iu(l),iu(l+1)-1
               tmp(l)=tmp(l)+u(k)*tmp(ju(iju(l)-iu(l)+k))
            enddo
```

Optimizing for a particular architecture can be limited to the above loops.On
vector computers, these loops will not vectorize automatically, because
of possible dependencies. As in fact there are no dependencies, the compiler

should be told so. On the Cray computers, these loops can be preceded by a
line:
cdir$ ivdep
for up to 1.2-10 times better performance. The increase in performance is
greater if matrices are larger and denser.

COMPUTER-SPECIFIC PROBLEMS

FSPAK ends with a "bus error" on HP workstations when programs are compiled
with default options. That compiler expects real*8 aligned on a 8 byte
boundary. Option +A3 appears to solve the problem. Similar problems may occur
with other compilers. In this case please use an appropriate compiler option.

The test version of FSPAK (Dec 27, 1994) aligns real*8 variables to an even
address. This will alleviate the alignment problem If the work vector
passed to FSPAK is aligned. The work vector will always be aligned if it is
declared as real*8 (with halve the number of elements), or if it equivalenced
to any real*8 variable.

REFERENCES

Duff, I. S., Erisman A. M., Reid J. K. 1989. Direct
methods for sparse matrices. Clarendon Press, Oxford,
England.

George, A., Liu J. W. 1981. Computer solution of large
sparse positive definite systems. Prentice-Hall, Inc.,
Englewood Cliffs, New Jersey.

Kincaid, D. R., Respes J. R., Young D. M., Grimes R. G.
1982. Algorithm 586 ITPACK 2C: a FORTRAN package for solving
sparse linear systems by adaptive accelerated iterative
methods. ACM Trans. Math. Software 8:302.

Tinney, W.F., Brandwajn, V., Chan, S.M., 1985. Sparse vector
methods. IEEE Trans. Pow. App. Sys. Vol PAS-104 (2):295-301

EXAMPLE

This is the output from running FTEST.F, the test program.
This program produces two output files FSPAK.OUT and FSPAK.ORD

```
cat fspak.out

 1. DENSE VECTOR METHOD
    COLUMN     DIAG VALUE
         1    0.38889D+00
         2    0.41358D+00
         3    0.33951D+00
         4    0.52469D+00
         5    0.72222D+00
 The mean absolute difference between Ax and b
 is   0.527355936696949301E-16


 2. SPARSE VECTOR METHOD
    COLUMN     DIAG VALUE
         1    0.38889D+00
         2    0.41358D+00
         3    0.33951D+00
         4    0.52469D+00
         5    0.72222D+00


 3. TAKAHASHI ET AL.s METHOD
    COLUMN     DIAG VALUE
         1    0.38889D+00
         2    0.41358D+00
         3    0.33951D+00
         4    0.52469D+00
         5    0.72222D+00


Finally, the determinant is: .16200D+03

 I forgot that I need element (2,4) of the inverse |
   -0.30864D-01
 or, of course, element (4,2)
   -0.30864D-01
               **************
               *** FSPAK ***
               **************
                  MPE / IM
                 August 1992

              SPARSE STATISTICS
      RANK OF COEFF MATRIX    =                       5
      STORAGE AVAILABLE       =                    2000
      MAXIMUM NEEDED          =                     267
      NZE IN UPPER TRIANGULAR =                       9
      NZE IN FACTOR           =                       4
      NO. OF CALLS NUM FACT   =                       1
      NO. OF CALLS SOLVE      =                       5
      NO. OF CALLS SPARS SOLV =                       7
      NO. OF CALLS DET / LDET =                       1
      NO. OF CALLS SPARS INV  =                       1
```

```
      MAX NO. NODES           =                      1
      MAX PATH LENGTH         =                      3
      AVG NO. NODES FPATH     =                  1.000
      AVG NO. NODES BPATH     =                  1.000
      AVG FPATH LENGTH        =                  2.286
      AVG BPATH LENGTH        =                  2.286
      TOTAL CPU TIME IN FSPAK =               0.006918
      TIME FOR FINDING ORDER  =               0.000127
      TIME FOR SYMBOLIC FAC   =               0.000071
      TIME FOR NUMERICAL FAC  =               0.000121
      TIME FOR SOLVE          =               0.000226
      TIME FOR SPARSE SOLVE   =               0.000470
      TIME FOR SPARSE INVERSE =               0.000146

cat fspak.ord

          5           4
      2      1      5      4      3      2      1      5      4
3
```

APPENDIX

Function second on several computers

1. VM CMS (IBM mainframe)

```
       REAL FUNCTION SECOND()
       REAL*8 A,B
       LOGICAL FIRST/.TRUE./
       INTEGER I
       SAVE FIRST,B
       IF (FIRST) THEN
          CALL CPUTIME(B,I)
          FIRST=.FALSE.
       ENDIF
       CALL CPUTIME(A,I)
       SECOND=(A-B)/1.E6
       END
```

2. CRAY supercomputers and DEC workstations

Function second is part of the system. No extra function is needed.

3. PC - Microsoft Compiler ver. 5.0, Watcom compiler ver. 9.0, Lahey LF90 3.0

```
      function second()
      real x,secnds
      integer first
      save x,first
      data first/0/
c
```

```
      if (first.eq.0) then
         x=secnds(0.0)
         first=1
         second=0
       else
         second=secnds(x)
      endif
      end
C
C     INTEFRACE ROUTINE FROM DEC TO PC FOR GETTING TIME
C
      FUNCTION SECNDS(X)
      REAL*4    SECNDS, X
      INTEGER*2 IHOUR, IMINUT, ISECON, IHUND
      REAL*4    RHOUR, RMINUT, RSECON, RHUND
      REAL*4    X1
      CALL GETTIM(IHOUR, IMINUT, ISECON, IHUND)
      RHOUR     =  FLOAT( IHOUR )
      RMINUT    =  FLOAT( IMINUT)
      RSECON    =  FLOAT( ISECON)
      RHUND     =  FLOAT( IHUND )
      X1        =  RHOUR*3600.0 + RMINUT*60.0 + RSECON +
     &            RHUND/100.0
      SECNDS    =  X1 - X
      RETURN
      END
```

4. Sun workstation

```
       function second
       real second
       real*8 x(2)
       second=etime(x)
       end
```

5. IBM RS/6000 workstation

```
      real function second()
      integer mclock
      second = mclock()/100.
      end
```

6. Linux with Absoft f90 (and probably f77) compiler

```
 function second()
       real second
       real x(2)
       second=etime(x)
       end
```

Link as

```
f90 ..... -lU77
```

7. If neither of the above functions work on your computer, either write one yourself, or substitute the following "dummy" subroutine:

```
      real function second()
      second=0
      end
```

Please note that this subroutine does not return any CPU time; it merely lets the other programs run.