RC 20645 (91375) Dec 3, 1996
Computer Science/Mathematics

# IBM Research Report

## WSSMP: Watson Symmetric Sparse Matrix Package

Users Manual: Version S.0.2$\alpha$

**Anshul Gupta**

IBM Research Division
T.J. Watson Research Center
P. O. Box 218
Yorktown Heights, New York 10598
*anshul@watson.ibm.com*

**Fred Gustavson**

IBM Research Division
T.J. Watson Research Center
P. O. Box 218
Yorktown Heights, New York 10598
*gustav@watson.ibm.com*

LIMITED DISTRIBUTION NOTICE

This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g., payment of royalties).

Research Division
Almaden · Austin · China · Haifa · Tokyo · Watson · Zurich

# WSSMP: Watson Symmetric Sparse Matrix Package

Users Manual Version S.0..$\alpha$ *

Anshul Gupta and Fred Gustavson

IBM T. J. Watson Research Center
P. O. Box 218
Yorktown Heights, NY 10598

*anshul@watson.ibm.com*, *gustav@watson.ibm.com*

IBM Research Report RC 20645 (91375)

December 1996

## 1 Introduction

The Watson Symmetric Sparse Matrix Package *WSSMP* is a robust high performance software for solving large sparse symmetric positive definite systems of linear equations on modern work-station class computers. It uses a modified version of the multifrontal algorithm [7] for sparse Cholesky factorization. The current release of *WSSMP* can perform any or all of the following tasks: *ordering, symbolic factorization, Cholesky factorization, triangular solves,* and *iterative refinement.* We hope to enhance the functionality in the future, such as, the inclusion of $LDL^T$ factorization for semidefinite and indefinite matrices. A parallel version incorporating a highly scalable parallel sparse factorization algorithm [5, 3] and parallel triangular solver [6, 3] will also be released sometime in the future.

*Since this document describes software that might be upgraded from time to time, *WSSMP* users and the readers of this report are encouraged to request the latest version of the manual via e-mail from the author.

## 2 Availability

The *WSSMP* library is currently available for the following platforms (it will soon be available for other systems):

- Entire IBM RS6000 family (XLF 2.3.xx and higher), code WSSMPPX2.
- IBM RS6000 POWER (XLF 3.xx and higher), code WSSMPP1.
- IBM RS6000 POWER-2 (XLF 3.xx and higher), code WSSMPP2.
- IBM RS6000 POWER PC (XLF 3.xx and higher), code WSSMPPC.

While better ways of distributing the software are being worked upon, for the time being, the library can be obtained by sending e-mail to anshul@watson.ibm.com with the subject line containing the request code followed by one or more blanks followed by the e-mail address at which the appropriate library and the manual will be e-mailed. The body of the message must contain the requester's name and affiliation. A user must read Section 6 of this manual before using *WSSMP*. The manual is sent in the postscript format and the library is e-mailed as a uuencoded file that should yield a binary file with a *.a* extension upon successful decoding. It may be required to run *ranlib* on the decoded file.

**Note 2.1** *If the library is linked with C code and a C compiler is being used for linking, then the Fortran library must be linked in addition to the* WSSMP *library by -lf77 (or -lF77, depending on the system) or -lxlf (on an IBM RS600) option. In addition, the -L option may be required to indicate the paths to the various libraries. Otherwise, the C code may be compiled with the -c option and a Fortran compiler should be used for linking and generating the final executable.*

**Note 2.2** WSSMP *uses BLAS routines* DSYRK, DGEMM, DGEMV, DTRSM, *and* DTRSV. *Therefore, a BLAS library (by -lblas option) or a scientific library containing efficient BLAS routines (e.g. ESSL via -lessl or -lesslp2 option on IBM RS6000 machines) must be linked along with* WSSMP. *The* BLAS *library can be obtained from* http://www.netlib.org/blas, *but the on-line version should be used only if a tuned version provided by the hardware vendor is not available.*

**Note 2.3** WSSMP *uses the Watson Graph Partitioning (and sparse matrix ordering) Package [4] for ordering, although the user can supply any other ordering and force WSSMP to skip the ordering step. However, the appropriate* WGPP *library must be linked for propfier functioning of* WSSMP *because it uses some routines other than ordering from* WGPP. *Please refer to [4] for information on how to obtain a copy of* WGPP.

### 2.1 Using WSSMP with OSL

The users of the IBM Optimization Subroutine Library (*OSL*) can benefit from *WSSMP* by simply linking this libraryfi before *OSL* while compiling their programs. For example, on a POWER-2 based IBM RS6000 with XLF Version 3,

```
xlf -O3 -qstrict -o foo.out foo.f -lwssmpp2 -lwgppp2 -lesslp2 -losl
```

will automatically replace the sparse matrix ordering and factorization routines of *OSL* by the corresponding *WGPP* and *WSSMP* routines (called automatically with appropriate arguments). C users will need to use one of the following ways of using *WSSMP*:

```
xlc -O3 -o foo.out foo.c -lwssmpp2 -lwgppp2 -lxlf -lesslp2 -losl
```
or
```
xlc -O3 -c foo.c
xlf -o foo.out foo.o -lwssmpp2 -lwgppp2 -lesslp2 -losl
```
In addition, the -L option may be required to indicate the paths to the various libraries.

# 3 The *WSMMP* Subroutine

This section addresses some issues in the use of the *WSSMP* subroutine and describes its calling sequences in detail. There are five basic tasks that *WSSMP* is capable of performing, namely, *ordering*, *symbolic factorization*, *Cholesky factorization*, *forward and backward solve*, and *iterative refinement* (see Note 3.5). The same routine can perform all or any number of these functions in sequence depending on the options given by the user via parameter *IPARM* (see Section 3.2). In addition, a call to *WSSMP* can be used to get the default values of the options without any of the five basic tasks being performed. See the description of *IPARM(1)*, *IPARM(2)*, and *IPARM(3)* in Section 3.2.14 for more details. The ordering is performed using the Watson Graph Partitioning (and sparse matrix ordering) Package [4]. Please see Note 2.3. The user can skip *WSSMP*'s ordering step and use the output (vectors *PERM* and *INVP*) generated from any ordering code, including the *WMMRB* routine of *WGPP* called directly.

In addition to the advanced interface that the *WSSMP* library provides via the single subroutine *WSSMP*, there are a number of other routines that provide a simpler interface. These routines are described in detail in Section 4.

**Note 3.1** *Calling the* WMMRB *routine of* WGPP *directlfiy is faster if a call to* WSSMP *is being used to generate the ordering only. If* WSSMP *is being called to perform multiple functions in the same call and ordering is one of them, then using* WSSMP *for ordering is usually faster.*

**Note 3.2** *The original code for both* WGPP *and* WSSMP *is in Fortran and expects the parameters to be passed by reference. Therefore, when calling* WSSMP *from a C program, the addresses of the parameters described in Section 3.2 must be passed explicitly.*

## 3.1 Types of matrices accepted and input format

The *WSSMP* routine works only with symmetric positive definite matrices. If the matrix is negative definite, then it is the user's responsibility to scale the matrix. Please refer to the description of *IPARM(10)* for the kind of scaling that *WSSMP* can perform, optionally. All floating point values should be double precision. The actual size in bytes of double precision and integer values depends on the compiler and the hardware platform. On most systems, it is 8 bytes and 4 bytes, respectively.

Since the input matrix is symmetric, only a triangular part is accepted a input. Currently, three input formats are supported. In the first format, the input is the upper triangular part, including the diagonal, in compressed sparse row (CSR) or the lower triangular part in the compressed sparse column (CSC) format. The second format is modified compressed sparse row/column (MSR or MSC). The third format is the one used internally by *WSSMP*. CSR and the *WSSMP* internal formats are both slightly faster than MSR. The internal format is similar to CSR, but is somewhat more flexibles in that the column (row) indices of row (column) $i+1$ don't necessarily hafive to follow

those of row (column) $i$. Figure 1 illustrates all three input formats, which are also explained briefly in Sections 3.2.2, 3.2.3, 3.2.4, and 3.2.5.

*WSfiSMP* supports both C-style indexing starting from 0 and Fortran-style indexing starting from 1. C-style indexing is preferred and is used in the internal routines. Fortran-style numbering forces a conversion in the beginning and a reconversion at the end. Please refer to the description of *IPARM(5)* in Section 3.2.14.

## 3.2 Calling sequence

Please refer to Note 3.2 before reading this section. There are five types of arguments, namely input (type **I**), output (type **O**)fi, modifiable (type **M**), temporary (type **T**), and reserved (type **R**). The input arguments are read by the *WSSMP* and remain unchanged upon execution, the output arguments are not read but some useful information is returned via them, the modifiable arguments are read by *WSSMP* and modified to return some information, the temporary arguments are not read but their contents are overwritten by unpredictable values during execution, and the reserve arguments are just like temporary arguments which may change to one of the other types of arguments in the future serial and parallel releases of this software.

In the remainder of this document, the "system" refers to the sparse linear system of $N$ equations of the form $Ax = b$, where $A$ is the sparse symmetric positive definite coefficient matrix of dimension $N$, $b$ is the right-hand-side vector/matrix and $x$ is the solution vector/matrix, whose approximation $\overline{x}$ computed by *WSSMP* and overwrites $b$ when *WSSMP* is called to compute the solution of the system.

**Note 3.3** *Recall that* WSSMP *supports both C-style (starting from 0) and Fortran-style (starting from 1) numbortran-style numbering and C users*
*must interpreted it accordingly. For example,* IPARM(11) *will actually be* IPARM[10] *in a C program calling* WSSMP.

The calling sequence and description of the parameters of *WSSMP* is as follows:

*WSSMP ( N, IA, JA, AVALS, DIAG, PERM, INVP, B, LDB, NRHS, AUX, NAUX, MRP, IPARM, DPARM )*

### 3.2.1 N (type I): matrix dimension

```
INTEGER N
```

This is the number of rows and columns in the sparse matrix $A$ or the number of equations in the sparse linear system $Ax = b$.

### 3.2.2 IA, (type I or M): row pointers

- **Case 1:** If input format is CSR or MSR; i.e., *IPARM(4)* = 0 or 1 or 8 or 9 :

```
INTEGER IA (N + 1)
```

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 14. | | -1. | | | | -1. | -3. | |
| 2 | | 14. | -1. | | | | | -3. | -1. |
| 3 | -1. | -1. | 16. | | | | -2. | -4. | -2. |
| 4 | | | | 14. | | -1. | -1. | -3. | |
| 5 | | | | | 14. | -1. | | -3. | -1. |
| 6 | | | | -1. | -1. | 16. | -2. | -4. | -2. |
| 7 | -1. | | -2. | -1. | | -2. | 16. | -4. | |
| 8 | -3. | -3. | -4. | -3. | -3. | -4. | -4. | 71. | -4. |
| 9 | | -1. | -2. | | -1. | -2. | | -4. | 16. |

A 9 X 9 symmetric sparse matrix.

The storage of this matrix in the input formats accepted by WSSMP is shown in the table below:

| | CSR Format | | | MSR Format | | | | WSSMP Format | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| K | IA(K) | JA(K) | AVALS(K) | IA(K) | JA(K) | AVALS(K) | DIAG(K) | IA(1,K) | IA(2,k) | JA(K) | AVALS(K) |
| 1 | 1 | 1 | 14.0 | 1 | 3 | -1.0 | 14.0 | 1 | 4 | 1 | 14.0 |
| 2 | 5 | 3 | -1.0 | 4 | 7 | -1.0 | 14.0 | 5 | 4 | 3 | -1.0 |
| 3 | 9 | 7 | -1.0 | 7 | 8 | -3.0 | 16.0 | 9 | 4 | 7 | -1.0 |
| 4 | 13 | 8 | -3.0 | 10 | 3 | -1.0 | 14.0 | 13 | 4 | 8 | -3.0 |
| 5 | 17 | 2 | 14.0 | 13 | 8 | -3.0 | 14.0 | 17 | 4 | 2 | 14.0 |
| 6 | 21 | 3 | -1.0 | 16 | 9 | -1.0 | 16.0 | 21 | 4 | 3 | -1.0 |
| 7 | 25 | 8 | -3.0 | 18 | 7 | -2.0 | 16.0 | 25 | 2 | 8 | -3.0 |
| 8 | 27 | 9 | -1.0 | 19 | 8 | -4.0 | 71.0 | 27 | 2 | 9 | -1.0 |
| 9 | 29 | 3 | 16.0 | 20 | 9 | -2.0 | 16.0 | 29 | 1 | 3 | 16.0 |
| 10 | 30 | 7 | -2.0 | 20 | 6 | -1.0 | | | | 7 | -2.0 |
| 11 | | 8 | -4.0 | | 7 | -1.0 | | | | 8 | -4.0 |
| 12 | | 9 | -2.0 | | 8 | -3.0 | | | | 9 | -2.0 |
| 13 | | 4 | 14.0 | | 6 | -1.0 | | | | 4 | 14.0 |
| 14 | | 6 | -1.0 | | 8 | -3.0 | | | | 6 | -1.0 |
| 15 | | 7 | -1.0 | | 9 | -1.0 | | | | 7 | -1.0 |
| 16 | | 8 | -3.0 | | 7 | -2.0 | | | | 8 | -3.0 |
| 17 | | 5 | 14.0 | | 8 | -4.0 | | | | 5 | 14.0 |
| 18 | | 6 | -1.0 | | 9 | -2.0 | | | | 6 | -1.0 |
| 19 | | 8 | -3.0 | | 8 | -4.0 | | | | 8 | -3.0 |
| 20 | | 9 | -1.0 | | 9 | -4.0 | | | | 9 | -1.0 |
| 21 | | 6 | 16.0 | | | | | | | 6 | 16.0 |
| 22 | | 7 | -2.0 | | | | | | | 7 | -2.0 |
| 23 | | 8 | -4.0 | | | | | | | 8 | -4.0 |
| 24 | | 9 | -2.0 | | | | | | | 9 | -2.0 |
| 25 | | 7 | 16.0 | | | | | | | 7 | 16.0 |
| 26 | | 8 | -4.0 | | | | | | | 8 | -4.0 |
| 27 | | 8 | 71.0 | | | | | | | 8 | 71.0 |
| 28 | | 9 | -4.0 | | | | | | | 9 | -4.0 |
| 29 | | 9 | 16.0 | | | | | | | 9 | 16.0 |

Figure 1: Illustration of the three input formats.

*IA* is an integer array of size one greater than $N$. *IA(I)* points to the first column index of row $I$ in the array *JA*. Refer to Figure 1 and description of *IPARM(4)* in Section 3.2.14 for more details.

- **Case 2:** If input format is WSSMP internal; i.e., *IPARM(4)* = 2 or 10 :

```
INTEGER IA (2, N)
```

*IA* is a two-dimensional integer array. *IA(1,I) points to the first column index of row I* (or first row index of column $I$) in the array *JA*. *IA(2,I)* stores the number of nonzero entries in the $I$-th row in the upper-triangular portion of the matrix $A$, which is the same as the number of entries in its $I$-th column of the lower triangular portion. Refer to Figure 1 and Section 3.2.14 for more details.

If *IPARM(8)* = 2, then the matrix is permuted on output. Please refer to the description of *IPARM(8)* and Note 3.6 for more details.

### 3.2.3 JA, (type I or M): column indices

```
INTEGER JA ( * )
```

The integer array *JA* confitain the column (row) indices of the upper (lower) triangular part of the symmetric sparse matrix $A$. The indices of a column (row) are stored in consecutive locations. In addition, if *IPARM(8)* = 2; i.e., the matrix is already permuted and the user does not want *WSSMP* to permute the matrix, then these consecutive column (row) indices of a row (column) must be sorted in increasing order upon input. If CSR or *WSSMP* internal input format is used, then the size of array *JA* is the total number of nonzeros in a triangular portion of the symmetric matrix $A$ (including the diagonal). If the MSR input format is used, then the size of *JA* is the number of nonzeros in a strictly triangular portion of $A$ excluding the diagonal.

If *IPARM(8)* = 2, then the matrix is permuted on output. Please refer to the description of *IPARM(8)* and Note 3.6 for more details.

### 3.2.4 AVALS, (type I or M): nonzero values of the coefficient matrix

```
DOUBLE PRECISION AVALS ( * )
```

The array *AVALS* contains the actual double precision values corresponding to the indices in *JA*. The size of *AVALS* is the same as that of *JA*. See Figure 1 for more details. The input *AVALS* may be modified if scaling is opted for; see the description of *IPARM(10)* for more details.

If *IPARM(8)* = 2, then the matrix is permuted on output. Please refer to the description of *IPARM(8)* and Note 3.6 for more details.

### 3.2.5 DIAG, (type I): diagonal values

```
DOUBLE PRECISION DIAG ( N )
```

If the MSR input format is used, then *DIAG(I,I)* contains the $I$-th diagonal element of the coefficient matrix. If CSR or WSSMP internal formats are used for input, then *DIAG* is not referenced and need not be a double precision array of size $N$; it can simply be a placeholder (e.g., a *NULL* can be passed in C). The input *DIAG* is not modified even if scaling is opted for because scaling does not modify the user input values if MSR format is used; instead it modifies an internal copy. See the description of *IPARM(10)* for more details.

If *IPARM(8)* = 2, then the matrix is permuted on output. Please refer to the description of *IPARM(8)* for more details.

### 3.2.6 PERM, (type I or O): permutation vector

```
INTEGER PERM ( N )
```

*PERM* is the permutation vector, as defined in Sparspak [1]. If $J$ = *PERM(I)*, then the $J$-th row and column of the original matrix become the $I$-th row and column in the permuted matrix to be factored. If ordering is one of the steps that the call to *WSSMP* is made to perform, the *PERM* is an output parameter. If *WSSMP* is called to perform a task or a set of tasks other than ordering, then *PERM* is an input parameter if either the matrix $A$ of the right-hand side $B$ is not permuted already. If both $A$ and $B$ are already permuted when passed in to *WSSMP*, then *PERM* is not referenced. See Section 3.2.14 for more details.

### 3.2.7 INVP, (type I or O): inverse permutation vector

```
INTEGER INVP ( N )
```

*INVP* is the inverse permutation vector, as defined in Sparspak [1]. If $J$ = *INVP(I)*, then the $I$-th row and column of the original matrix become the $J$-th row and column in the permuted matrix to be factored. If ordering is one of the steps that the call to *WSSMP* is made to perform, the *INVP* is an output parameter. If *WSSMP* is called to perform a task or a set of tasks other than ordering, then *INVP* is an input parameter if either the matrix $A$ of the right-hand side $B$ is not permuted already. If both $A$ and $B$ are already permuted when passed in to *WSSMP*, then *INVP* is not referenced. See Section 3.2.14 for more details.

### 3.2.8 B, (type M): right-hand side vector/matrix

```
DOUBLE PRECISION B ( LDB, NRHS )
```

The $N \times$ *NRHS* dense matrix $B$ contains the right-hand side of the system of equations $Ax = b$ to be solved. If the number of right-hand side vectors, *NRHS*, is one, then $B$ can simply be a vector of length $N$. $B$ is accessed only if a call to *WSSMP* includes solution or iterative refinement.

### 3.2.9 LDB, (type I): leading dimension of B

```
INTEGER LDB
```

*LDB* is the leading dimension of the right-hand side matrix if *NRHS* $> 1$. *LDB* must be greater than or equal to $N$.

### 3.2.10 NRHS, (type I): number of right-hand sides

```
INTEGER NRHS
```

*NRHS* is the second dimension of $B$; it is the number of right-hand sides that need to be solved for.

### 3.2.11 AUX, (type O, I, or T): auxiliary storage

```
DOUBLE PRECISION AUX ( NAUX )
```

The user can pass the storage required for factorization to *WSSMP* via the optional double precision array *AUX*. If *AUX* is passed in to *WSSMP* for factorization, then it must be passed in unmodified for triangular solutions and iterative refinement. Whether the user supplied storage *AUX* is used by *WSSMP* or not depends on the value of the input parameter *NAUX*. If $NAUX = 0$, then *WSSMP* allocates space dynamically, otherwise is $NAUX > 0$, the *AUX* array is used. If $NAUX > 0$, then *AUX* is also used as as integer working storage of size $r \times NAUX$ for the ordering routine *WMMRB*, where $r$ is the ratio of the size of a double precision number to the size of an integer on the hardware platform. If *AUX* is used by *WSSMP*, then it is of type T for ordering, unused for symbolic factorization, of type O for factorization, type I for triangular solves and iterative refinement.

**Note 3.4** *If* AUX *is used in call to* WSSMP *for Cholesky factorization, then it must be passed unaltered in subsequent calls to* WSSMP *for triangular solves and iterative refinement with the same matrix.*

### 3.2.12 NAUX, (type I): size of user supplied auxiliary storage

```
INTEGER NAUX
```

*NAUX* is the number of double words in *AUX*. If *NAUX* is zero, then *WSSMP* allocates space dynamically for factorization. If *NAUX* is a positive integer, then *WSSMP* attempts to use the first *NAUX* double words of the array *AUX* for factorization. If that is not sufficient, then *WSSMP* returns without completion. *IPARM(64)* returns the status of the call and tells the user whether the *WSSMP* call was successful or not. Whether *NAUX* is 0, or insufficient, or more than sufficient, *IPARM(23)* returns the actual size of *AUX* required. *IPARM(23)* is filled with the correct value of *NAUX* after symbolic factorization and the user has a chance to inspect this vfialue before calling *WSSMP* for Cholesky factorization. See Section 3.2.14 for more details.

### 3.2.13 MRP, (type M or O): pivot values or pivot info

- **Case 1:** If *IPARM(4)* $\geq 8$ and *IPARM(11)* $< 2$ :

  ```
  DOUBLE PRECISION MRP ( N )
  ```

  In this case, the double precision vector *MRP* contains the principal diagonal of the lower triangular Cholesky factor of $A$ upon return from *WSSMP* if factorization is performed.

- **Case 2:** If *IPARM(11)* $\geq 2$ :

```
INTEGER MRP ( N )
```

This case overrides case 1; i.e., if *IPARM(11)* $\geq 2$ and *IPARM(4)* $\geq 8$, then *IPARM(4)* is set to *IPARM(4)* - 8. In this case, on return from *WSSMP* factorization, *MRP(I)* is set to *IPARM(13)* if the $I$-th pivot was less than or equal to *DPARM(11)* during factorization. Otherwise, *MRP(I)* is not touched.

### 3.2.14 IPARM, (type I, O, M, and R): integer array of parameters

```
INTEGER IPARM ( 64 )
```

*IPARM* is an integer array of size 64 that is used to pass various optional parameters to *WSSMP* and to return some useful information about the execution of call to *WSSMP*. If *IPARM(1)* is 0, then *WSSMP* uses default values of *IPARM(4)* through *IPARM(64)* and *DPARM*. *IPARM(1)* through *IPARM(3)* are mandatory inputs, which should always be supplied by the user. If *IPARM(1)* is 1, then *WSSMP* uses the user supplied entries in the arrays *IPARM* and *DPARM*. The default initial values of *IPARM* and *DPARM* are shown in Table 1. Note that some of the entries in *IPARM* and *DPARM* are of type M or O. If *IPARM(1)* is 0, then the default values of Table 1 are used before *WSSMP* performs the user designated task(s) it has been called to perform. The contents of *IPARM* and *DPARM* at the end of the execution, depending on the task(s) performed by *WSSMP*, are the output or modified valfiues corresponding to the default input values. It is possible for a user to call *WSSMP* only to fill *IPARM* and *DPARM* with the default initial values. This is useful if the user needs to change only a few parameters in *IPARM* and *DPARM* and needs to use most of default values. Please refer to the description of *IPARM(2)* and *IPARM(3)* for more details. Note that there are no default values for *IPARM(2)* and *IPARM(3)* and these must always be supplied by the user, whether *IPARM(1)* is 0 or 1.

- **IPARM(1), type I or M:**

  If *IPARM(1)* is 0, then the remainder of the *IPARM* array and the *DPARM* array are filled with default values by *WSSMP* before further computation and *IPARM(1)* itself is set to 1. If *IPARM(1)* is 1 on input, then *WSSMP* uses the user supplied values in *IPARM* and *DPARM*.

- **IPARM(2), type M:**

  On input, *IPARM(2)* must contain the number of the starting task. On output, *IPARM(2)* contains 1 + number of the last task performefid by *WSSMP*, if any. This is to facilitate users to restart processing on a problem from where the last call to *WSSMP* left it. If *WSSMP* performs no task, then, on output, *IPARM(2)* is set to *max(IPARM(2),IPARM(3)*+ 1*)*. *WSSMP* can perform any set of consecutive tasks from the following list:

  | | |
  |---|---|
  | Task # 1: | Ordering |
  | Task # 2: | Symbolic Factorization |
  | Task # 3: | Cholesky Factorization |
  | Task # 4: | Forward and Backward Elimination |
  | Task # 5: | Iterative Refinement |

| | IPARM | | | DPARM | | |
|---|---|---|---|---|---|---|
| Index | Default input | Description | Type | Default input | Description | Type |
| 1 | mandatory I/P | default/user defined | M | - | run time | O |
| 2 | mandatory I/P | starting task | M | - | matrix norm | O |
| 3 | mandatory I/P | last task | I | - | mat. inv. norm | O |
| 4 | 0 | I/P format | I | - | max. $L_{i,i}$ | O |
| 5 | 1 | numbering style | I | - | min. $L_{i,i}$ | O |
| 6 | 1 | no. of iter. ref. | M | 1.0E-12 | rel. err. lim. | I |
| 7 | 0 | residual norm type | I | - | relative err. | O |
| 8 | 0 | mat. permut. option | M | - | unused | - |
| 9 | 0 | RHS permut. option | M | - | unused | - |
| 10 | 0 | scaling option | M | 1.0E-02 | scaling range | I |
| 11 | 0 | bad pivot handling | I | 1.0E-60 | bad piv. threshold | I |
| 12 | 0 | small piv. handling | I | 1.0E-30 | small piv. thresh. | M |
| 13 | 0 | bad pivot flag | I | - | unused | - |
| 14 | 0 | cache line size | I | - | unused | - |
| 15 | 0 | cache size | I | - | unused | - |
| 16 | 1 | WMMRB option 1 | I | - | unused | - |
| 17 | 0 | WMMRB option 2 | I | - | unused | - |
| 18 | 1 | WMMRB option 3 | I | - | unused | - |
| 19 | 0 | WMMRB option 4 | I | - | unused | - |
| 20 | 7 | WMMRB option 5 | I | - | unused | - |
| 21 | - | bad pivot count | O | 1.0E200 | bad pivot subst. | I |
| 22 | - | small pivot count | O | 1.0E-15 | small piv. subst. | I |
| 23 | - | minimum NAUX | O | - | factorization ops. | O |
| 24 | - | factor nonzeros | O | - | tri. solve ops. | O |
| 25 | 0 | cond. no. option | I | - | unused | - |
| 26 | 0 | N1 | I | - | unused | - |
| 27-63 | - | reserved | R | - | reserved | R |
| 64 | - | info O/P | O | - | first bad pivot | O |

Table 1: The default initial values of the various entries in *IPARM* and *DPARM* arrays. An '-' indicates that the value is not read by *WSSMP*.

**Note 3.5** WSSMP *can process only one matrix at a time. A user cannot factor one matrix, then factor a second matrix, and then solve a system using the first factor. In other words,* WSSMP *can work on a system in increasing order of task numbers. If a call to* WSSMP *is mfiade with a starting task number in* IPARM(2) *that is less than or equal to the number of the last task performed by* WSSMP *in a previous call, then all or some of the results of the previous call are lost.*

- **IPARM(3), type I:**

  *IPARM(3)* must contain the number of the last task to be performed by *WSSMP*. In a call to *WSSMP*, all tasks from *IPARM(2)* to *IPARM(3)* are performed (both inclusive). If *IPARM(2)* $>$ *IPARM(3)* or both *IPARM(2)* and *IPARM(3)* is out of the range 1–5, then no task is performed. This can be used to fill *IPARM* and *DPARM* with default values; e.g., by calling *WSSMP* with *IPARM(1)* $= 0$, *IPARM(2)* $= 0$, and *IPARM(3)* $= 0$.

- **IPARM(4), type I:**

*IPARM(4)* denotes the format in which the coefficient matrix $A$ is stored and instructs *WSSMP* whether or not to output the diagonal of the Cholesky factor of the coefficient matrix. *IPARM(4)* = 0 denotes CSR, *IPARM(4)* = 1 denotes MSR, and *IPARM(4)* = 2 denotes *WSSMP* internal format. All three formats are illustrated in Figure 1.

*WSSMP* does not give user the access to the Cholesky factor of the coefficient matrix, which is stored in a non-standard format either in the user supplied storage area $AUX$, or in internally allocated data structures. However, occasionally the user might need to inspect the diagonal values of the factor to analyze the numerical accuracy of the factorization process. If 8 is added to the above mentioned values of *IPARM(4)*; i.e., if 8, 9, or 10 is used instead of 0, 1, and 2, then the parameter $MRP$ is tfireated as a double precision array of size $N$ and the diagonal of the Cholesky factor is copied into $MRP$ (provided that task number 3 is performed during that call to *WSSMP*. In terms of interpreting the input format of $A$, an 8, 9, or a 10 in *IPARM(4)* is treated exactly as a 0, 1, or a 2, respectively.

- **IPARM(5), type I:**

  If *IPARM(5)* = 0, then C-style numbering (starting from 0) is used; If *IPARM(5)* = 1, then Fortran-style numbering (starting from 1) is used. In C-style numbering, the matrix rows and columns are numbered from 0 to $N-1$ and the indices in $IA$ should point to entries in $JA$ starting from 0.

- **IPARM(6), type M:**

  On input, *IPARM(6)* should be set to the maximum number of steps of iterative refinement to be performed; on output, *IPARM(6)* contains the actual number of iterative refinement steps performed.

- **IPARM(7), type I:**

  If *IPARM(7)* = 0, 1, 2, or 3, then the residual in iterative refinement is computed in double precision (the same as the remainder of the computation). If *IPARM(7)* = 4, 5, 6, or 7, then the residual in iterative refinement is computed in quadruple precision (which twice the precision of the remainder of the computation). If *IPARM(7)* = 0 or 4, then exactly *IPARM(6)* number of iterative refinement steps are performed without checking for relative error. If *IPARM(7)* = 1, 2, 3, 5, 6, or 7, then iterative refinement is performed until the number of iterative refinement steps is equal to *IPARM(6)* or until the relative error given by $||b - A\overline{x}||/||b||$ falls below the input value in *DPARM(6)*. Here $A$ is the coefficient matrix, $\overline{x}$ is the computed solution, and $b$ is the right-hand side. If *IPARM(7)* = 1 or 5, then 1-norms are used in computing the relative error, if *IPARM(7)* = 2 or 6, then 2-norms are used, and if *IPARM(7)* = 3 or 7, then infinity-norms are used. Moreover, if *IPARM(7)* = 1, 2, 3, 5, 6, or 7, then the actual relative error given by $||b - A\overline{x}||/||b||$ at the end of the last iterative refinement step is placed in *DPARM(7)*. Even if iterative refinement is not performed (i.e. if *IPARM(6)* = 0 or if *IPARM(3)* < 5) the value of relative error is computed and placed in *DPARM(7)* if *IPARM(7)* = 1, 2, 3, 5, 6, or 7.

  If $NRHS > 1$, then the maximum of the relative errors amongst the $NRHS$ solution vectors is considered. Also note that, if scaling is opted for (i.e., if *IPARM(10)* = 1), then the relative errors are computed with respect to the scaled system and not the original system.

- **IPARM(8), type I or M:**

On input, *IPARM(8)* = 0 means that the matrix $A$ is not permuted and its permutation given by *PERM* and *INVP* must be used in symbolic factorization, Cholesky factorization, solution, and/or iterative refinement. *IPARM(8)* = 1 means that the matrix is permuted already and must be used as it is by *WSSMP*. *IPARM(8)* = 2 means that the matrix is not permuted and instructs *WSSMP* to permute the matrix and overwrite the data structures (*IA*, *JA*, *AVALS*, and *DIAG*) with the contents of the permuted matrix. In addition, if *IPARM(8)* = 2 on input, it is set to 1 on output. If *IPARM(8)* = 1 on input and ordering is one of the steps being performed, then *IPARM(8)* is treated as if it was equal to 2 on input. The input matrix is overwritten by the permuted matrix only if symbolic and/or Cholesky factorization is performed. Moreover, if only symbolic factorization is performed and Cholesky factorization is not performed, the array *AVALS* is not permuted and overwritten.

**Note 3.6** *If* IAPRM(8) = *1; i.e., the user is supplying a pre-permuted matrix, then the indices of a particular row/column in the array* JA *must be sorted in* increasing *order. If the matrix is not pre-permuted; i.e.,* IPARM(8) *is 0 or 2, then the indices of a particular row/column in* JA *can be in any order.*

- **IPARM(9), type I or M:**

  On input, *IPARM(9)* = 0 means that the right-hand side $B$ is not permuted and its permutation given by *PERM* and *INVP* must be used by *WSSMP*. *IPARM(9)* = 1 means that $B$ is permuted already and must be used as it is by *WSSMP*. *IPARM(9)* = 2 means that the $B$ not permuted and instructs *WSSMP* to overwrite $B$ with its permuted version. In addition, if *IPARM(9)* = 2 on input, it is set to 1 on output. If *IPARM(9)* = 1 on input and ordering is one of the steps being performed, then *IPARM(9)* is treated fias if it was equal to 2 on input. If a solution is obtained or iterative refinement is performed, then the solution is in the original unpermuted order if *IPARM(9)* = 0 and it is permuted if *IPARM(9)* is 1 or 2 on input. The input *RHS* is overwritten by the permuted *RHS* only if the triangular solutions are performed.

- **IPARM(10), type M:**

  An input of *IPARM(10)* = 0 implies that *WSSMP* will not perform any scaling of the input matrix and vector(s). If *IPARM(10)* = 1, then a scaling of the form $(D^{-1}AD^{-1}Dx = D^{-1}b$ is performed, where $D = diag(\sqrt{a_{i,i}})$. The system $Hy = c$ is solved, where $H = D^{-1}AD^{-1}$ and $c = D^{-1}b$. The solution of the system, $\overline{y}$, is then scaled to compute the solution of the original system $\overline{x} = D^{-1}\overline{y}$. *IPARM(10)* = 2 instructs *WSSMP* to perform the scaling only if any diagonal element of the original matrix $A$ lies outside the range *[DPARM(10)*, 1.0/*DPARM(10)]*. Upon return, *IPARM(10)* is set to 0 or 1 to indicate whether or not the scaling was performed. If scaling is performed, then the values of the input matrix in *AVALS* are changed if the input format is *not* MSR *and* if the matrix either already permuted or is permuted on output; i.e., if *IAPRM(4)* is 0 or 2 and *IPARM(8)* is 1 or 2. If MSR input format is used or if *IPARM(8)* is 0, only an internal copy is scaled and the input matrix remains unchanged.

- **IPARM(11), type I:**

  *IPARM(11)* and *IPARM(12)* determine how very small, zero, or negative pivots are handled by *WSSMP*. If *IPARM(11)* = 0, then if *WSSMP* encounters a diagonal value less than or equal

to *DPARM(11)* during factorization (just before the square root), it sets *IPARM(64)* to the index of this pivot, puts the actual pivot value in *DPARM(64)* and returns without further processing. If *IPARM(11)* = 1, then if a pivot less than *DPARM(11)* is encountered, then it is replaced by the square of the value in *DPARM(21)* (in other words, it is replaced by *DPARM(21)* and no square root is performed) and the factorization continues. An input value of *IPARM(11)* = 2 is treated similar to *IPARM(11)* = 1, except that in addition to replacing a bad pivot with $DPARM(21)^2$, *MRP(I)* is set to the integer value in *IPARM(13)* to flag the occurrence of a bad $I$-th pivot. If *IPARM(11)* = 2 and *IPARM(4)* $\geq$ 8, then *IPARM(11)* takes precedence and *IPARM(4)* is set to *IPARM(4)* - 8. The number of occurrences of pivots less than or equal to *DPARM(11)* is reported in *IPARM(21)* on output.

- **IPARM(12), type I:**

  If *IPARM(12)* = 0, then *DPARM(12) DPARM(22)* are ignored. An input of *IPARM(12)* = 1 means that if the $I$-th pivot is greater than *DPARM(11)* but less than or equal to *DPARM(12)*, it will be replaced by $DPARM(22)^2$ during factorization. The number of pivots encountered in the range $>$ *DPARM(11)* and $\leq$ *DPARM(12)* is reported in *IPARM(22)* on output. If *DPARM(12)* $<$ *DPARM(11)* and *IPARM(12)* = 1, then .

- **IPARM(13), type I:**

  *IPARM(13)* is the integer flag used for marking bad pivots in the *MRP* array if *IPARM(11)* = 2.

- **IPARM(14), type I:**

  This should be the size of the cache line in bytes on input. If it is zero on input, *WSSMP* assigns a default value (the actual hardware value for IBM RS6000 work-stations). If it a positive integer, the user assigned value is used. An input value of 0 is recommended for IBM RS6000.

  **Note 3.7** *If the user supplies a positive integer in* IPARM(14) *and a zero in* IPARM(15)*, or vice-versa, then both* IPARM(14) *and* IPARM(15) *are treated a zero on input and are overwritten.*

- **IPARM(15), type M:**

  This should be the size of the cache in bytes on input. If it is zero on input, *WSSMP* assigns a default value (the actual hardware value for IBM RS6000 work-stations). If it a positive integer, the user assigned value is used. An input value of 0 is recommended for IBM RS6000. Please see Note 3.7.

- **IPARM(16:20), type M:**

  The five integer values in *IPARM(16)* through *IPARM(2)* are used as the five efintries in the *OPTIONS* array while calling the *WMMRB* routine for ordering. Please refer to *WGPP* Users Manual [4] for more details.

- **IPARM(21), type O:**

  Please refer to the description of *IPARM(11)*.

- **IPARM(22), type O:**

  Please refer to the description of *IPARM(12)*.

- **IPARM(23), type O:**

  On output, this contains the number of double words of storage requiredfi by Cholesky factorization to store the factor and as additional working space. This field is set only if symbolic factorization is performed.

- **IPARM(24), type O:**

  On output, *IPARM(24)* contains the number of nonzeros in the triangular factor. Thifis field is set only if symbolic factorization is performed.

- **IPARM(25), type I:**

  This parameter indicates whether or not a condition number estimate is to be performed. If *IPARM(25)* $= 0$, then a condition number estimate is not computed. If *IPARM(25)* $> 0$, then the value of $||A||$ is computed and placed as output in *DPARM(2)* and an estimate of $||A^{-1}||$ is computed and placed in *DPARM(3)*. The condition number of the matrix $A$ is the product *DPARM(2)* $\times$ *DPARM(3)*. The condition number is a measure of the reliability of the solution of a given linear system and can be used to compute error bounds. The algorithm described in [2] is used to estimate $||A^{-1}||$. If *IPARM(25)* $= 1$, then 1-norms are computed, if *IPARM(25)* $= 2$, then Frobenius-norms are computed, and if *IPARM(25)* $= 3$, then infinity-norms are computed. *WSSMP* can be made to do this computation either with factorization, or with solution, or with iterative refinement. *WSSMP* keeps track of the condition number estimation computation with respect to a particular norm for a given matrix and if *IPARM(25)* does not change between factorization, solution and iterative refinement steps, then the computation is not repeated.

  Note that if scaling is opted for (i.e., if *IPARM(10)* $= 1$) then the norms computed are those of the scaled matrix and not the original matrix.

- **IPARM(26), type I:**

  *IPARM(26)* contains the input parameter *N1*, where $0 \leq N1 \leq N$. *N1* is used only if *WSSMP* is used for ordering and is ignored if it is less than or equal to 0 or greater than or equal to $N$. If *N1* is used, then the first *N1* columns of the matrix are factored before the remaining $N - N1$ columns.

- **IPARM(27:63), type R:**

  These are reserved for future use.

- **IPARM(64), type O:**

  In the event of a successful return from *WSSMP*, *IPARM(64)* is set to 0 on output. A nonzero value of *IPARM(64)* upon output indicates that *WSSMP* did not complete execution and detected an error condition. If the user supplies the memory for factorization in *AUX* and the size *NAUX* of this array is not sufficient, then *IPARM(64)* is -101 on output. If dynamic memory allocation by *WSSMP* fails (which can happen even if user supplies sufficiently large *AUX* because *WSSMP* allocates more memory for internal book-keeping), the *IPARM(64)* is

set to -102 on return. A positive integer value of *IPARM(64)* between 1 and $N$ on output indicates a computation error. In this case, *IPARM(64)* is the index of the first pivot that was less than or equal to *DPARM(11)* before computing the square root for Cholesky factorization. If C-style indexing is used and *IPARM(64)* $> 0$, then *IPARM(64)* is $1 +$ the index of the bad pivot. If an input argument error is detected, then *IPARM(64)* is set to a negative integer whose absolute value is the number of the erroneous input argument. Only minimal input argument checking is performed and a non-negative value of *IPARM(64)* does not guarantee that all input arguments have been verified to be correct.

### 3.2.15 DPARM, (type I, O, M, and R): double precision parameter array

```
DOUBLE PRECISION DPARM ( 64 )
```

The entries *DPARM(27)* through *DPARM(64)* are reserved. Unlike *IPARM*, only a few of the first 27 entries of *DPARM* are used. The description of only the relevant entries of *DPARM* is as follows:

- **DPARM(1), type O:**

  Returns the total wall clock time in seconds spent in an *WSSMP* call (only on IBM RS6000 systems). Since this is the elapsed time, it will vary depending on the load on the machine.

- **DPARM(2), type O:**

  Please refer to the description of *IPARM(25)*.

- **DPARM(3), type O:**

  Please refer to the description of *IPARM(25)*.

- **DPARM(4), type O:**

  On output, this contains the largest diagonal element of the factor; i.e., *DPARM(4)* $= \max_{i=1,N} L(i,i)$.

- **DPARM(5), type O:**

  On output, this contains the smallest diagonal element of the factor; i.e., *DPARM(4)* $= \min_{i=1,N} L(i,i)$.

- **DPARM(6), type I:**

  Iterative refinement is stopped if *IPARM(7)* $> 0$ and the relative error becomes less than *DPARM(6)*. *DPARM(6)* is not used if *IPARM(7)* $= 0$.

- **DPARM(7), type O:**

  If a solve or iterative refinement step is pfierformed, then *DPARM(7)* contains the relative error $||b - A\overline{x}||/||b||$ on output. The type of norm is determined by *IPARM(7)*. If *NRHS* $> 1$, then the this field contains the maximum of the relative errors amongst the *NRHS* right-hand side vectors.

- **DPARM(10), type I:**

  Please refer to the description of *IPARM(10)*.

- **DPARM(11), type I:**

  This is the lower threshold on value of a good pivot. If a pivot is less than or equal to *DPARM(11)*, then either an error is flagged or corrective action is taken as specified by *IPARM(11)*.

- **DPARM(12), type M:**

  *DPARM(12)* is also used to provide user some control over pivoting. See the description of *IPARM(12)* for more details.

- **DPARM(21), type I:**

  Please refer to the description of *IPARM(11)*. It is the user's responsibility to initialize *DPARM(21)* with a valid positive double precision value.

- **DPARM(22), type I:**

  Please refer to the description of *IPARM(12)*. If *DPARM(21)* is being used; i.e., if *IPARM(12)* = 1, then it is the user's responsibility to initialize *DPARM(21)* with a valid positive double precision value.

- **DPARM(23), type O:**

  This contains the number of floating point operations required in Cholesky factorization.

- **DPARM(24), type O:**

  This contains the total number of floating point operations required in forward and backward elimination for all the right-hand sides. The operation count if for a single right-hand side.

- **DPARM(64), type O:**

  If *IPARM(11)* = 0, then *DPARM(64)* contains the value of the first pivot (if any) encountered that was less than or equal to *DPARM(11)*.

# 4 Routines for a Simpler Interface

In this section, we describe a simpler interface to *WSSMP*. This interface accepts the input in only CSR and MSR formats and expects a Fortran style indexing starting from 1. The shape, size, attributes, and meaning of all data structures is the same as in the calling sequence of the *WSSMP* routine described in Section 3, *unless mentioned otherwise*. This interface is provided via the following subroutines:

## 4.1 WMMRB, WKKTORD, WN1ORD (ordering)

*WMMRB ( N, XADJ, ADJNCY, OPTIONS, NUMBERING, PERM, INVP, AUX, NAUX )*

*WKKTORD ( N, XADJ, ADJNCY, OPTIONS, NUMBERING, PERM, INVP, AUX, NAUX )*

*WN1ORD ( N, N1, XADJ, ADJNCY, OPTIONS, NUMBERING, PERM, INVP, AUX, NAUX )*

Please refer to the *WGPP* manual [4] for the details of the *WMMRB* routine.

The *WKKTORD* routine has exactly the same calling sequence as *WMMRB* and should be used for ordering what are known as KKT systems in which the coefficient matrix $K$ takes the following form:

$$K = \begin{pmatrix} D_1 & M^T \\ M & D_2 \end{pmatrix},$$

where $M$ is an $n \times m$ sparse matrix, $D_1$ is an $m \times m$ diagonal matrix, $D_2$ is an $n \times n$ diagonal matrix, and $n + m = N$.

The routine *WN1ORD* is also similar to *WMMRB*, except that *WN1ORD* has an extra parameter *N1*, where $0 \leq N1 \leq N$. Any permutation generated by *WN1ORD* is such that the first *N1* columns of the matrix are factored before the remaining $N - N1$ columns. *N1* is ignored if it is less than or equal to 0 or greater than or equal to $N$.

### 4.2 WSSMPCALZ (analyze, CSR input) and WSSMPMALZ (analyze, MSR input)

*WSSMPCALZ ( N, IA, JA, OPTIONS, PERM, INVP, NNZL, WSPACE, AUX, NAUX )*

*WSSMPMALZ ( N, IA, JA, OPTIONS, PERM, INVP, NNZL, WSPACE, AUX, NAUX )*

The description of *OPTIONS*, *AUX*, and *NAUX* inputs is the same as in the calling sequence of *WMMRB* [4].

On output, the integer *NNZL* contains the number of nonzeros in the Cholesky factor and the integer *WSPACE* contains the number of double words of memory required by Cholesky factorization, including the storage for the Cholesky factor.

### 4.3 WSSMPCSYM (symbolic, CSR input) and WSSMPMSYM (symbolic, MSR input)

This routine performs symbolic factorization only and uses the input permutation given by *PERM* and *INVP*.

*WSSMPCSYM ( N, IA, JA, PERM, INVP, NNZL, WSPACE )*

*WSSMPMSYM ( N, IA, JA, PERM, INVP, NNZL, WSPACE )*

On output, the integer *NNZL* contains the number of nonzeros in the Cholesky factor and the integer *WSPACE* contains the number of double words of memory required by Cholesky factorization, including the storage for the Cholesky factor.

### 4.4 WSSMPCCHF (Cholesky, CSR input) and WSSMPMCHF (Cholesky, MSR input)

This routine performs Cholesky factorization, assumes that the input matrix is *not* permuted, and uses its permutation for factorization. The input *NAUX* should either be zero, or at least as large

as the output *WSPACE* of the *analyze* or *symbolic routines*, either of which must be called before a *WSSMPxCHF* routine.

*WSSMPCCHF ( N, IA, JA, AVALS, PERM, INVP, AUX, NAUX, INFO )*

*WSSMPMCHF ( N, IA, JA, AVALS, DIAG, PERM, INVP, AUX, NAUX, INFO )*

The description of the output *INFO* is the same is that of *IAPRM(64)* in the calling sequence of the *WSSMP* routine described in Section 3.

### 4.5 WSSMPCSV(X and WSSMPMSVX (expert drivers with CSR and MSR inputs)

These routines perform Cholesky factorization, triangular solutions, and iterative refinement. A call to the appropriate *WSSMPxSYM* or *WSSMPxALZ* routine must precede the call to a *WSSMPxSVX* routine. *PERM* and *INVP* inputs determine the permutation of the input matrix that is used. The input matrix and $B$ are assumed to be unpermuted and the output $B$ is in the same order as the input.

*WSSMPCSVX ( N, IA, JA, AVALS, PERM, INVP, B, LDB, NRHS, AUX, NAUX, RCOND, INFO )*

*WSSMPMSVX ( N, IA, JA, AVALS, DIAG, PERM, INVP, B, LDB, NRHS, AUX, NAUX, RCOND, INFO )*

The double precision output *RCOND* is the inverse of the condition number estimate of the input matrix. The description of the output *INFO* is the same is that of *IAPRMfi(64)* in the calling sequence of the *WSSMP* routine described in Section 3.

### 4.6 WSSMPSLV (solve a system using a prior factorization)

The routine *WSSMPSLV* solves a system of equations with right hand size $B$, using a prior factorization by a call to either a *WSSMPxCHF* or a *WSSMPxSVX* routine.

*WSSMPSLV ( N, PERM, INVP, B, LDB, NRHS, AUX, NAUX )*

Note that if the user supplied *AUX* was used in factorization (i.e., $NAUX > 0$ in the call to the factorization routine) then the same *AUX* must be passed unaltered to *WSSMPSLV*.

## 5 Bugs and Problems

*WSSMP* performs minimal input argument error-checking and it is the user's responsibility to call *WSSMP* subroutines with correct arguments, especially, valid options and symmetric matrices. However, the numerical error checking capabilities of *WSSMP* are quite robust.

Version S.0 of *WSSMP* is an alpha release osfi the serial version and is not guaranteed to be bug free. Any bugs or other problems with the software or this manual may be reported via e-mail to *anshul@watson.ibm.com*.

Also refer to the section on "Bugs and Problems" in *WGPP* Users Manual [4].

# 6 Notice: Terms and Conditions for Use of *WSSMP*

IBM reserves the right to discontinue the free distribution of this software for any or all of the hardware platforms without any prior notice. IBM also reserves the right to reject a request for the software by any individual or organization. The information contained in this manual, including the calling sequences, are subject to change at any time without prior notice. The *WSSMP* library may be used solely for educational, research, and benchmarking purposes by non-profit organizations and the U.S. government. Commercial and other organizations may make use of the *WSSMP* library solely for benchmarking purposes only. Any other use of the libraries requires prior written permission. As unestablished research software, the *WSSMP* library is provided free of charge on an "as is" basis without warranty of any kind, either expressed or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose. IBM does not warrant that the functions contained in this library will meet the user's requirements or that the operation of its routines will be uninterrupted or error-free. Acceptance and use of this program constitutes the user's understanding that he/she will have no recourse to IBM for any actual or consequential damages, including, but not limited to, lost profits or savings, arising out of the use or inability to use this library. Even if the user informs IBM of the possibility of such damages, IBM expects the user to accept the risk of any such harm, or the user shall not attempt to use this library for any purpose. The library may not be sold, redistributed, or placed at a public site from where it can be copied by anyone other thafin the original requester. One may make copies of the library, provided that the copies are not distributed and are used under the same conditions. The use of the *WSSMP* library is an implicit agreement by the user to these conditions. These terms and conditions are subject to change at any time without prior notice.

# References


[1] E. Chu, Alan George, Joseph W.-H. Liu, and Esmond G.-Y. Ng. Users guide for SPARSPAK–A: Waterloo sparse linear equations package. Technical Report CS-84-36, University of Waterloo, Waterloo, IA, 1984.

[2] A. K. Cline, Cleve B. Moler, G. W. Stewart, and J. H. Wilkinson. An estimate for the condition number of a matrix. *SIAM Journal on Numerical Analysis*, 16:368–375, 1979.

[3] Anshul Gupta. *Analysis and Design of Scalable Parallel Algorithms for Scientific Computing*. PhD thesis, University of Minnesota, Minneapolis, MN, 1995.

[4] Anshul Gupta. WGPP: Watson graph partitioning (and sparse matrix ordering) package: Users manual. Technical Report RC 20453 (90427), IBM T. J. Watson Research Center, Yorktown Heights, NY, May 6, 1996.

[5] Anshul Gupta, George Karypis, and Vipin Kumar. Highly scalable parallel algorithms for sparse matrix factorization. Technical Report 94-63, Department of Computer Science, University of Minnesota, Minneapolis, MN, 1994. To appear in *IEEE Transactions on Paral-*

*lel and Distributed Systems*, 1997. Postscript file available via anonymous FTP from the site *ftp://ftp.cs.umn.edu/users/kumar*.

[6] Anshul Gupta and Vipin Kumar. Parallel algorithms for forward and back substitution in direct solution of sparse linear systems. In *Supercomputing '95 Proceedings*, December 1995.

[7] Joseph W.-H. Liu. The multifrontal method for sparse matrix solution: Theory and practice. Technical Report CS-90-04, York University, Ontario, Canada, 1990. Also appears in *SIAM Review*, 34:82–109, 1992.