LAIPE

Direct Solvers of [A]{X}={B}

Version 4.01

By Equation Solution

List of Contents

About This Manual

About

The letters LAIPE(TM) stands for "Link And In Parallel Execute". LAIPE is a symbol for high performance computing, and has a collection of subroutines for numerical analyses. All the functions in LAIPE are programmed in explicit parallelism, not optimized by auto-parallelizer. Some LAIPE solvers can yield almost perfect speedup, i.e., 1.99X on 2 processors. Link LAIPE to your programs, and then your applications not only can run on uniprocessor computer but also can speed up on multiprocessors. LAIPE provides powerful subroutines for users to efficiently take advantage of multiprocessors.

This manual covers parallel direct solvers, i.e., Cholesky decomposition, skyline solver, Crout decomposition, multiple entry solvers, and other popular and useful techniques. Solvers for dense and sparse systems are included. More than 90% of scientific and engineering problems are formulated into a system of equations. Solution of system equations is required in many scientific and engineering computing. LAIPE has the most useful and highly efficient solvers for scientific and engineering computing.

LAIPE is written in MTASK(TM) that is a parallel programming language. When building your application that links with LAIPE direct solvers, a copy of MTASK is necessary.

Assumptions About the Reader

This manual assumes that readers have knowledge on system equations. This manual focuses on how to apply LAIPE solvers, but does not discuss mathematical equations and parallel algorithms. This manual also assumes that users have experience writing, executing, and debugging Fortran, and assumes that user's computer is capable of parallel processing.

Overview of This Manual

This manual is organized as follows:

Chapter 1 **Introduction.** This chapter introduces terms and essential concepts that user will need to be familiar with before applying LAIPE solvers.

Chapter 2 **Constant-Bandwidth, Symmetric, and Positive Definite Systems.** This chapter describes calling syntax of LAIPE subroutines for a system in the category, with the definition of profile, data storage scheme, and example.

Chapter 3 **Variable-Bandwidth, Symmetric, and Positive Definite Systems.** This chapter describes calling syntax of LAIPE subroutines for a system in the category, with the definition of profile, data storage scheme, and example.

Chapter 4 **Dense, Symmetric, and Positive Definite Systems**. This chapter describes calling syntax of LAIPE subroutines for a system in the category, with the definition of profile, data storage scheme, and example.

Chapter 5 **Constant-Bandwidth and Symmetric Systems**. This chapter describes calling syntax of LAIPE subroutines for a system in the category, with the definition of profile, data storage scheme, and example.

Chapter 6 **Variable-Bandwidth and Symmetric Systems**. This chapter describes calling syntax of LAIPE subroutines for a system in the category, with the definition of profile, data storage scheme, and example.

Chapter 7 **Dense and Symmetric Systems.** This chapter describes calling syntax of LAIPE subroutines for a system in the category, with the definition of profile, data storage scheme, and example.

Chapter 8 **Constant-Bandwidth and Asymmetric Systems.** This chapter describes calling syntax of LAIPE subroutines for a system in the category, with the definition of profile, data storage scheme, and example.

Chapter 9 **Variable-Bandwidth and Asymmetric Systems.** This chapter describes calling syntax of LAIPE subroutines for a system in the category, with the definition of profile, data storage scheme, and example.

Chapter 10 **Dense and Asymmetric systems.** This chapter describes calling syntax of LAIPE subroutines for a system in the category, with the definition of profile, data storage scheme, and example.

Chapter 11 **Constant-Bandwidth and Asymmetric Solvers with Partial Pivoting.** This chapter describes calling syntax of LAIPE subroutines for a system in the category, with the definition of profile, data storage scheme, and example.

Chapter 12 **Constant-Bandwidth, Symmetric, and Positive Definite Solvers with Partial Pivoting.** This chapter describes calling syntax of LAIPE subroutines for a system in the category, with the definition of profile, data storage scheme, and example.

Chapter 13 **Constant-Bandwidth and Symmetric Solvers with Partial Pivoting.** This chapter describes calling syntax of LAIPE subroutines for a system in the category, with the definition of profile, data storage scheme, and example.

Chapter 14 **Dense Solvers with Partial Pivoting.** This chapter describes calling syntax of LAIPE subroutines for a system in the category, with the definition of profile, data storage scheme, and example.

Chapter 15 **Dense Solvers with full pivoting.** This chapter describes calling syntax of LAIPE subroutines for a system in the category, with the definition of profile, data storage scheme, and example.

Appendix A **Auxiliary Subroutine for Releasing System Resource**

Appendix B **Auxiliary Subroutines for Task Manipulations**

Chapter 1. Introduction

Parallel computing especially benefits to large-scaled problems, that distributes computing loads among employed processors and speeds up an individual application. It is an important technique for scientific and engineering computing. The executing speed of parallel computing is superior to sequential computing that executes instructions in order. Usually, more processors may produce better improvement.

LAIPE has high performance parallel solvers. On uniprocessor environments, LAIPE run as usual. When multiprocessors present, LAIPE may split itself to fit the multiprocessors. Users just link LAIPE to their applications. It is unnecessary for users to distribute computing instructions onto employ multiprocessors. LAIPE is a package for both small and large-scaled problems. The present release has solvers in the following categories:

1. sparse system (of constant bandwidth, and variable bandwidth)
2. dense system
3. symmetric system
4. asymmetric system
5. positive definite system
6. indefinite system
7. solution with partial pivoting
8. solution with full pivoting.

The following introduces essential terms and concept for applying LAIPE solvers.

1.1 Solution of System Equations

A system of linear equations may be written in the form

$$[A]\{X\}=\{B\} \tag{1.1}$$

where the left side matrix [A] is square and of order (NxN), and {B} is a given vector, and the vector {X} is the solution to be determined. Not every system in equation (1.1) is solvable. If the matrix [A] is singular, i.e., matrix [A] has zero eigenvalue or the determinant of [A] is zero, the solution {X} is not unique or even does not exist. This manual does not deal with singular systems, and provides solution to solvable systems.

In direct methods, solution procedure consists of two parts, decomposition and substitution. For example, the left side matrix [A] is decomposed into the product of [L][U] where matrix [L] is a lower triangular matrix and matrix [U] is an upper triangular matrix. Then, equation (1.1) is rewritten as

$$[L][U]\{X\}=\{B\}, \tag{1.2}$$

and is rewritten into the following

$$[L]\{Y\}=\{B\} \tag{1.3}$$
$$[U]\{X\}=\{Y\} \tag{1.4}$$

Equation (1.3) solves {Y}. Since [L] is the lower triangular matrix, equation (1.3) is called *forward substitution*. Equation (1.4) solves {X}, and is called *backward substitution*. The solution of equation (1.1) is obtained by decomposition, forward and backward substitutions. The solution costs depend on the nature of matrix [A], for example, sparsity or symmetry. Each type of matrix [A] will be briefly introduced in the following.

1.2 Symmetric and Asymmetric Systems

A symmetric matrix [A] means that $A_{ij} = A_{ji}$ for any i and j; otherwise matrix [A] is asymmetric. Solution of symmetric systems is cheaper than asymmetric systems. Most engineering and scientific applications can be approximated into a symmetric system. Symmetric systems only consider a triangular part of matrix [A]; While asymmetric systems must deal with the entire matrix.

1.3 Sparse and Dense Systems

In the situation that [A] has many zero coefficients, the row or column can be reordered such that the non-zero coefficients are clustered along the diagonal of [A]. The non-zero fill-ins generate a sparsity. This makes sparse matrix different from dense matrix. The sparse matrix can be classified into *constant or variable bandwidth*. The solution costs on sparse matrix may be far less than a corresponding dense system. If a system is sparse in nature, it is always better to apply sparse solvers.

1.4 Profile

Profile is a contiguous space to save a matrix. For a dense matrix that is the simplest example, the profile is the entire matrix size, i.e., an array of (NxN) coefficients. Sparse matrix has a profile less than (NxN) coefficients. A data storage scheme is associated with a profile. For an example of dense matrix, the profile is declared as

```
REAL (4) :: A(N,N)
```

The coefficient A_{ij} of matrix [A] is written as A(I,J) in a computer program. **Profile must be in a contiguous space**. Some Fortran compilers do not allocate 2-dimensional array in a contiguous space. That may create problems for LAIPE. It is always safe to initialize [A], in the main program, as a one-dimensional array, i.e., REAL (4) :: A(N*N), and then pass the reference of [A] to LAIPE solvers.

A sparse matrix has a profile smaller than the dense matrix, but the data storage scheme is more complex than dense matrix. The non-zero fill-ins are stored one by one in a contiguous space. For example,

$$
\begin{bmatrix}
* & * & & * & & & \\
& * & * & * & & & * \\
& & * & * & & & * \\
& & & * & * & * & * \\
& \text{sym.} & & & * & * & * \\
& & & & & * & * \\
& & & & & & *
\end{bmatrix}
$$

1.5 Definiteness

Definiteness is a mathematical condition. If all the eigenvalues are positive, the system is *positive definite*; If all the eigenvalues are negative, the system is *negative definite*; Others are *indefinite*. A solution procedure can be simplified if the system is definite. LAIPE has parallel solvers for positive definite systems. If a system is proved to be positive definite, it is better to apply a positive-definite solver.

1.6 Pivoting

Pivoting is a well known technique for improving accuracy. The idea of pivoting is well known. There are two kinds of pivoting; *partial pivoting* that finds the pivoting from the remaining elements in a column, and *full pivoting* that finds the pivoting element from the remaining columns and rows.

Floating variables always suffer from round-off error. Round-off error is a common problem in scientific and engineering computing. The problem can be enhanced if a number subtracts from another closed number. That may lose lots of significant digits. For example,

$$3.14160 - 3.14159 = 0.00001$$

The result does not have a significant digit, even both 3.14160 and 3.14159 have 5 significant digits. Any computations referring to the result become no significant digits, which is equivalent to no control of accuracy. Pivoting may keep significant digits as many as possible.

LAIPE has parallel solvers with pivoting. Solvers with pivoting, no doubt, take more execution time, and may lose the advantage of sparsity and symmetry. Pivoting is also a disadvantage to parallel processing.

1.7 Name Convention of LAIPE solvers

LAIPE has solvers in the following categories:

1. symmetric /asymmetric matrix
2. dense / sparse matrix
3. positive definite / indefinite system
4. single, double, and quad precision floating variables

LAIPE solvers can be identified by 5 elements. The name convention is as:

(Function)_#$%_^

Each element is introduced as follows.

§ Element 1

The symbol (Function) indicates the main purpose of the subroutine. That may be one of the following:

Decompose
Substitute
Solution
ppDecompose
ppSubstitute
ppSolution
fpDecompose
fpSubstitute
fpSolution
meSolution

where the prefix "pp" indicates a procedure with partial pivoting, and the prefix "fp" indicates a procedure with full pivoting, and the prefix "me" indicates a multiple entry direct solver. For example, “fpDecompose” is a procedure to decompose a matrix with full pivoting.

Multiple entry direct solvers have a higher degree of parallelism, but with a higher complexity. Multiple-entry direct solvers are most well suitable for systems with a small bandwidth, and are usually dealt with in a constant-bandwidth system, such as CSP, CSG, and CAG

§ Element 2

The symbol # is a single character. That indicates the type of sparsity, and may be one of the following:

C : sparse matrix with constant bandwidth
V : sparse matrix with variable bandwidth
D : dense matrix

§ Element 3

The symbol $ is a single character, and is a flag to indicate if matrix is symmetric or asymmetric. The flag is one of the following:

S : symmetric matrix
A : asymmetric matrix

§ Element 4

The symbol % is a single character, and is a flag to indicate if the matrix is positive definite or indefinite. The flag is one of the following:

P : positive definite system
G : general system without a consideration of definiteness

§ Element 5

The symbol ^ is for the kind of real or complex arguments. Argument is a variable or parameter, passed to LAIPE solvers. All the real or complex arguments must be in the type specified by the symbol. The symbol is one of the following:

4 : single precision real variables (4 bytes)
8 : double precision real variables (8 bytes)
10 : extended precision real variables (10 bytes)
16 : quad precision real variables (16 bytes)
Z4 : single precision complex variables (8 bytes)
Z8 : double precision complex variables (16 bytes)
Z10 : extended precision complex variables (10 bytes)
Z16 : quad precision complex variables (32 bytes)

Some Fortran compiler does not support quad precision variables. LAIPE subroutines are identified by those five elements. For the example of "Decompose_VSG_8", it is a subroutine for decomposing a variable-bandwidth, symmetric, and indefinite matrix. The REAL variables are in double precision.

The arguments passed to LAIPE functions are suffixed a "_i", "_o", "_io", or "_x". The suffix "_i" means the argument is an input. "_o" means an output. "_io" means that the argument inputs the data and returns the result. The suffix "_x" means that the argument provides a working space for temporary uses. For example,

Decompose_CSP_4(A_io, N_i, LowerBandwidth_i, NoGood_o)

The arguments "A_io", "N_i", and "LowerBandwidth_i" have to be defined before calling the function, and the result can be obtained from arguments "A_io" and "NoGood_o".

1.8 Data Storage Schemes

A data storage scheme is associated with profile, and has two specifications. The first one is to declare a dimension of profile, and the second one replaces the column index of coefficient of matrix with an address reference label. For example, a skyline matrix [A] is declared in a Fortran subroutine as

REAL (4) :: A(1,1)

And, the column index j of coefficient A_{ij} is programmed in a Fortran program as A(I,Label(J)) where Label(J) is the address reference label for column J.

Data storage scheme is applied to dummy arguments, for example in a subroutine, but not in the main program. The main program distributes a sufficient memory space for a profile, and then the main program passes the memory space to subroutine where data storage scheme is applied.

Chapter 2. Constant-Bandwidth, Symmetric, and Positive Definite Systems

2.1 Purpose

This chapter has subroutines for the solution of [A]{X}={B} where the left side matrix [A] is of constant bandwidth, symmetric, and positive definite. The non-zero fill-ins in the lower triangular part of matrix [A] have a shape, for example, as:

$$\begin{bmatrix} * & & & & & & \\ * & * & & & \text{sym.} & & \\ * & * & * & & & & \\ * & * & * & * & & & \\ & * & * & * & * & & \\ & & * & * & * & * & \\ & & & * & * & * & * \end{bmatrix}$$

Three types of subroutine are introduced in the chapter, which perform the following functions:

1. Decompose [A] into the product of $[L][L]^T$ where matrix [L] is the lower triangular matrix.
2. Perform forward and backward substitutions.
3. Solve [A]{X}={B} in a single call.

Decomposition and substitution must be called in order, and work together as a pair. No pivoting is applied to the functions introduced in this chapter. Subroutines are as:

Decompose_CSP_4
Decompose_CSP_8
Decompose_CSP_10
Decompose_CSP_16
Decompose_CSP_Z4
Decompose_CSP_Z8
Decompose_CSP_Z10
Decompose_CSP_Z16

Substitute_CSP_4
Substitute_CSP_8
Substitute_CSP_10
Substitute_CSP_16
Substitute_CSP_Z4
Substitute_CSP_Z8
Substitute_CSP_Z10
Substitute_CSP_Z16

Solution_CSP_4
Solution_CSP_8
Solution_CSP_10
Solution_CSP_16
Solution_CSP_Z4
Solution_CSP_Z8
Solution_CSP_Z10
Solution_CSP_Z16

meSolution_CSP_4
meSolution_CSP_8
meSolution_CSP_10
meSolution_CSP_16
meSolution_CSP_Z4
meSolution_CSP_Z8
meSolution_CSP_Z10
meSolution_CSP_Z16

The subroutines with a prefix "me", i.e., meSolution_CSP_4, are multiple-entry direct solvers that are most well suitable for systems with a small bandwidth.

2.2 Fortran Syntax for Subroutine Decompose

The following subroutines decompose a matrix $[A]$ into $[A]=[L][L]^T$:

Decompose_CSP_4 (A_io, N_i, LowerBandwidth_i, NoGood_o)
Decompose_CSP_8 (A_io, N_i, LowerBandwidth_i, NoGood_o)
Decompose_CSP_10 (A_io, N_i, LowerBandwidth_i, NoGood_o)
Decompose_CSP_16(A_io, N_i, LowerBandwidth_i, NoGood_o)
Decompose_CSP_Z4 (A_io, N_i, LowerBandwidth_i, NoGood_o)
Decompose_CSP_Z8 (A_io, N_i, LowerBandwidth_i, NoGood_o)
Decompose_CSP_Z10 (A_io, N_i, LowerBandwidth_i, NoGood_o)
Decompose_CSP_Z16 (A_io, N_i, LowerBandwidth_i, NoGood_o)

where
1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A], that inputs the original matrix and returns the result if the variable NoGood_o is false. For the definition of profile, please see section 2.6.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument LowerBandwidth_i, an INTEGER(4) variable, is the lower bandwidth of matrix [A]. The lower bandwidth is the maximal number of non-zero fill-ins below the diagonal in a column.
4. The argument NoGood_o, a LOGICAL(4) variable, is a flag that indicates if the input matrix [A] is suitable for the subroutine. If NoGood_o=.True., the input matrix [A] is not positive definite and there is no output from the subroutine; otherwise the profile A_io returns the decomposed matrix [L]. For the situation where NoGood_o=.True., please see section 2.8.

2.3 Fortran Syntax for Subroutine Substitute

The following subroutines perform forward and backward substitutions:

Substitute_CSP_4 (A_i, N_i, LowerBandwidth_i, X_io)
Substitute_CSP_8 (A_i, N_i, LowerBandwidth_i, X_io)
Substitute_CSP_10 (A_i, N_i, LowerBandwidth_i, X_io)
Substitute_CSP_16 (A_i, N_i, LowerBandwidth_i, X_io)
Substitute_CSP_Z4 (A_i, N_i, LowerBandwidth_i, X_io)
Substitute_CSP_Z8 (A_i, N_i, LowerBandwidth_i, X_io)
Substitute_CSP_Z10 (A_i, N_i, LowerBandwidth_i, X_io)
Substitute_CSP_Z16 (A_i, N_i, LowerBandwidth_i, X_io)

where

1. The argument A_i, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A], that inputs the result from decomposition.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument LowerBandwidth_i, an INTEGER(4) variable, is the lower bandwidth of matrix [A]. The lower bandwidth is the maximal number of non-zero fill-ins below the diagonal in a column.
4. The argument X_io, array whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution.

2.4 Fortran Syntax for Subroutine Solution

The following subroutines first decompose matrix [A] into the product of $[L][L]^T$, and then perform forward and backward substitutions. Solve the system [A]{X}={B} in a single call. The syntax is as follows:

Solution_CSP_4 (A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)
Solution_CSP_8 (A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)
Solution_CSP_10 (A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)
Solution_CSP_16 (A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)
Solution_CSP_Z4 (A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)
Solution_CSP_Z8 (A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)
Solution_CSP_Z10 (A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)
Solution_CSP_Z16 (A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)

where

1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A], that inputs the original matrix and returns the decomposed result if the variable NoGood_o is false. For the definition of profile, please see section 2.6.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument LowerBandwidth_i, an INTEGER(4) variable, is the lower bandwidth of matrix [A]. The lower bandwidth is the maximal number of non-zero fill-ins below the diagonal in a column.
4. The argument X_io, array whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution if NoGood_o is false.
5. The argument NoGood_o, a LOGICAL(4) variable, is a flag that indicates if the input matrix [A] is suitable for the subroutine. If NoGood_o=.True., the input matrix [A] is not positive definite and there is no output from the subroutine; otherwise the profile A_io returns the

decomposed matrix [L] and vector X_io returns the solution. For the situation where NoGood_o=.True., please see section 2.8.

2.5 Fortran Syntax for meSolution

The following subroutines solve the system [A][X]=[B] by multiple-entry method, where [X] and [B] may be a matrix with multiple vectors, i.e., [X]=[{ X_1 } { X_2 } ...] and [B]=[{ B_1 } { B_2 } ...]. Syntax is as follows:

```
meSolution_CSP_4(A_io, N_i,LowerBandwidth_i, X_io, Nset_i,    &
                     WorkingSpace_x,  NoGood_o)
meSolution_CSP_8(A_io, N_i,LowerBandwidth_i, X_io, Nset_i,    &
                     WorkingSpace_x, NoGood_o)
meSolution_CSP_10(A_io, N_i,LowerBandwidth_i, X_io, Nset_i,    &
                      WorkingSpace_x, NoGood_o)
meSolution_CSP_16(A_io, N_i,LowerBandwidth_i, X_io, Nset_i,    &
                      WorkingSpace_x, NoGood_o)
meSolution_CSP_Z4(A_io, N_i,LowerBandwidth_i, X_io, Nset_i,    &
                      WorkingSpace_x, NoGood_o)
meSolution_CSP_Z8(A_io, N_i,LowerBandwidth_i, X_io, Nset_i,    &
                      WorkingSpace_x, NoGood_o)
meSolution_CSP_Z10(A_io, N_i,LowerBandwidth_i, X_io, Nset_i,    &
                       WorkingSpace_x, NoGood_o)
meSolution_CSP_Z16(A_io, N_i,LowerBandwidth_i, X_io, Nset_i,    &
                       WorkingSpace_x, NoGood_o)
```

where

1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A] that inputs the original matrix. After returning from this subroutine, the content in the profile is destroyed no matter if the calling request is successful or not. For the definition of profile, please see section 2.6.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument LowerBandwidth_i, an INTEGER(4) variable, is the lower bandwidth of matrix [A]. The lower bandwidth is the maximal number of non-zero fill-ins below the diagonal in a column. This subroutine is more efficient if the lower bandwidth is small.
4. The argument X_io, array whose kind must be consistent with subroutine name convention, inputs the right side vector(s), and returns the solution if NoGood_o is false.
5. The argument Nset_i, an INTEGER(4) variable, is the number of right side vectors.
6. The argument WorkingSpace_x, array whose kind must be consistent with subroutine name convention and providing a space of (2*N_i*LowerBandwidth_i) elements, is a working space.
7. The argument NoGood_o, a LOGICAL(4) variable, is a flag that indicates if the input matrix [A] is suitable for the subroutine. If NoGood_o=.True., the input matrix [A] is not positive definite and there is no output from the subroutine; otherwise the vector X_io returns the solution. For the situation where NoGood_o=.True., please see section 2.8.

2.6 Profile

The profile for a constant-bandwidth, symmetric, and positive definite matrix is as:

```
[ *                        ]
[ *   *        sym.        ]
[ *   *   *                ]
[ *   *   *   *            ]      (2.1)
[     *   *   *   *        ]
[         *   *   *   *    ]
[             *   *   *   *]
                  &   &
                      &
```

where the symbol * represents non-zero fill-ins and the symbol & indicates an extra memory space whose content is never used. Total length of profile is determined as

$$\text{profile size} = (N-1) * \text{LowerBandwidth} + N \qquad (2.2)$$

where N is the matrix order, and LowerBandwidth is the lower bandwidth.

2.7 Data Storage Scheme

Data storage scheme must be declared in a Fortran program, for example:

```
INTEGER (4) :: LowerBandwidth
REAL (4) :: A(LowerBandwidth,1)
```

where variable A here is a single precision profile. Other kinds of variable, profile must be properly declared. Then, the coefficient A_{ij} in the lower triangular part of matrix [A] is programmed in a Fortran program as A(I,J).

2.8 Failure of Calling Request

If a calling request fails, solving procedure meets a diagonal coefficient that is very small and is negligible compared to unity.

The subroutines introduced in this chapter deal with positive definite systems. Since a symmetric solver does not consider pivoting, failure of request does not mean the input matrix is absolutely not positive definite. A pivoting may continue execution. However, pivoting may destroy the symmetry. If you cannot get the solution by the subroutines introduced in this chapter, try the solvers with partial pivoting, i.e., ppDecompose_CSP_4 discussed in chapter 12. Pivoting procedure always takes more time, and is less efficient in parallel processing.

2.9 Fortran Example

For a given system [A]{X}={B}, the left side matrix [A] and the right side vector {B} are defined as follows:

$$\begin{bmatrix} 1 & & & & & & \\ 4 & 25 & & & \text{sym.} & & \\ 2 & 29 & 88 & & & & \\ & 9 & 34 & 89 & & & \\ & & 3 & 23 & 45 & & \\ & & & 11 & 7 & 22 & \\ & & & & 3 & 2 & 9 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 21 \\ 141 \\ 2 \\ 9 \\ 333 \\ 1 \\ 3 \end{bmatrix}$$

in which the order N=7 and the lower bandwidth, denoted by LowerBandwidth, is 2. A Fortran program for decomposition and substitution is as follows. Subroutines “Input” and “Output” have example of data storage scheme. Subroutine “Decompose_CSP_4” decomposes matrix [A], and subroutine “Substitute_CSP_4” performs forward and backward substitutions.

```
!  *** Example program ***
!  define variables where the length of A is determined by equation (2.2)
!
      Integer (4), PARAMETER :: N = 7
      Integer (4), PARAMETER :: LowerBandwidth=2
      REAL (4) :: A((N-1)*LowerBandwidth+N), X(N)
      LOGICAL (4) :: NoGood
      DATA X/21.0,141.0,2.0,9.0,333.0,1.0,3.0/
!
!  input the lower triangular part of [A]
!
      CALL Input(A,LowerBandwidth)
!
!  decompose in parallel
!
      CALL Decompose_CSP_4(A,N,LowerBandwidth, NoGood)
!
!  stop if NoGood=.True.
!
      IF(NoGood) STOP 'Cannot be decomposed'
!
!  perform substitutions in parallel
!
      CALL Substitute_CSP_4(A,N,LowerBandwidth,sX)
!
!  output decomposed matrix
!
      CALL Output(A,N,LowerBandwidth)
!
!  output the solution
!
      Write(*,'(" Solution is as:")')
      Write(*,*) X
```

```
!
! laipe done
!
      call laipeDone
!
      STOP
      END

      SUBROUTINE Input(A,LowerBandwidth)
!
!
! routine to demonstrate an application of data storage scheme
! (A)FORTRAN CALL: CALL Input(A,LowerBandwidth)
!   1.A: <R4> profile of matrix [A], dimension(*)
!   2.LowerBandwidth: <I4> lower bandwidth
!
! dummy arguments
!
      INTEGER (4) :: LowerBandwidth
      REAL (4) :: A(LowerBandwidth,1)

!
! input
!
      A(1,1)= 1.0
      A(2,1)= 4.0
      A(3,1)= 2.0
      A(2,2)=25.0
      A(3,2)=29.0
      A(4,2)= 9.0
      A(3,3)=88.0
      A(4,3)=34.0
      A(5,3)= 3.0
      A(4,4)=89.0
      A(5,4)=23.0
      A(6,4)=11.0
      A(5,5)=45.0
      A(6,5)= 7.0
      A(7,5)= 3.0
      A(6,6)=22.0
      A(7,6)= 2.0
      A(7,7)= 9.0
!
      RETURN
      END

      SUBROUTINE Output(A,N,LowerBandwidth)
!
!
! routine to output the decomposed matrix by data storage scheme
! (A)FORTRAN CALL: CALL Output(A,N,LowerBandwidth)
```

```
!     1.A: <R4> profile of matrix [A], dimension(*)
!     2.N: <I4> order of matrix [A]
!     3.LowerBandwidth: <I4> lower bandwidth
!
!  dummy arguments
!
      INTEGER (4) :: N,LowerBandwidth
      REAL (4) :: A(LowerBandwidth,1)

!
!  local variables
!
      INTEGER (4) :: Column,Row
!
!  output the coefficients on non-zero fill-ins
!
      WRITE(*,'(" Row  Column  Coefficient")')
      DO Column=1,N
         DO Row=Column, MIN0(Column+LowerBandwidth,N)
            WRITE(*,'(I4,I6,F9.3)') Row,Column, A(Row,Column)
         END DO
      END DO
!
      RETURN
      END
```

Chapter 3. Variable-Bandwidth, Symmetric, and Positive Definite Systems

3.1 Purpose

This chapter has subroutines for the solution of [A]{X}={B} where the left side matrix [A] has a variable bandwidth, and is symmetric and positive definite. The non-zero fill-ins in the upper triangular part of matrix [A] have a shape, for example, as:

$$\begin{bmatrix} * & * & & & & & \\ & * & * & & * & & \\ & & * & * & * & & \\ & & & * & * & & * \\ & \multicolumn{2}{l}{\text{sym.}} & & * & * & * \\ & & & & & * & * \\ & & & & & & * \end{bmatrix}$$

which looks like a skyline in a city, and is sometimes called skyline solver. Three types of subroutine are introduced in the chapter, which have the following functions:

1. Decompose [A] into the product of $[U]^T[U]$ where matrix [U] is the upper triangular matrix.
2. Perform forward and backward substitutions.
3. Solve [A]{X}={B} in a single call.

Decomposition and substitution must be called in order, and work together as a pair. No pivoting is applied to the functions introduced in this chapter. This chapter has the following subroutines:

Decompose_VSP_4
Decompose_VSP_8
Decompose_VSP_10
Decompose_VSP_16
Decompose_VSP_Z4
Decompose_VSP_Z8
Decompose_VSP_Z10
Decompose_VSP_Z16

Substitute_VSP_4
Substitute_VSP_8
Substitute_VSP_10
Substitute_VSP_16
Substitute_VSP_Z4

Substitute_VSP_Z8
Substitute_VSP_Z10
Substitute_VSP_Z16

Solution_VSP_4
Solution_VSP_8
Solution_VSP_10
Solution_VSP_16
Solution_VSP_Z4
Solution_VSP_Z8
Solution_VSP_Z10
Solution_VSP_Z16

3.2 Fortran Syntax for Subroutine Decompose

The following subroutines decompose [A] into [A]= $[U]^T[U]$. Syntax is as follows:

```
Decompose_VSP_4(A_io, N_i, Label_i, NoGood_o)
Decompose_VSP_8(A_io, N_i, Label_i, NoGood_o)
Decompose_VSP_10(A_io, N_i, Label_i, NoGood_o)
Decompose_VSP_16(A_io, N_i, Label_i, NoGood_o)
Decompose_VSP_Z4(A_io, N_i, Label_i, NoGood_o)
Decompose_VSP_Z8(A_io, N_i, Label_i, NoGood_o)
Decompose_VSP_Z10(A_io, N_i, Label_i, NoGood_o)
Decompose_VSP_Z16(A_io, N_i, Label_i, NoGood_o)
```

where

1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A] that inputs the original matrix and returns the result if the variable NoGood_o is false. For the definition of profile, please see section 3.5.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument Label_i, an INTEGER(4) array, is the address reference label. For the definition of address reference label, please see section 3.6.
4. The argument NoGood_o, a LOGICAL(4) variable, is a flag that indicates if the input matrix [A] is suitable for the subroutine. If NoGood_o=.True., the input matrix [A] cannot be decomposed by the subroutine and there is no output from the subroutine; otherwise the profile A_io returns the decomposed matrix [U]. For the situation where NoGood_o=.True., please see section 3.7.

3.3 Fortran Syntax for Subroutine Substitute

The following subroutines perform forward and backward substitutions. Syntax is as follows:

```
Substitute_VSP_4( A_i, N_i, Label_i, X_io)
Substitute_VSP_8( A_i, N_i, Label_i, X_io)
Substitute_VSP_10( A_i, N_i, Label_i, X_io)
Substitute_VSP_16( A_i, N_i, Label_i, X_io)
```

Substitute_VSP_Z4(A_i, N_i, Label_i, X_io)
Substitute_VSP_Z8(A_i, N_i, Label_i, X_io)
Substitute_VSP_Z10(A_i, N_i, Label_i, X_io)
Substitute_VSP_Z16(A_i, N_i, Label_i, X_io)

where
1. The argument A_i,, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A], that inputs the result from decomposition.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument Label_i, an INTEGER(4) array, is the address reference label. For the definition of address reference label, please see section 3.6.
4. The argument X_io, array whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution.

3.4 Fortran Syntax for Subroutine Solution

The following subroutines first decompose [A] into the product of $[U]^T[U]$, and then perform forward and backward substitutions. Solve [A]{X}={B} in a single call. Syntax is as follows:

Solution_VSP_4 (A_io, N_i, Label_i, X_io, NoGood_o)
Solution_VSP_8 (A_io, N_i, Label_i, X_io, NoGood_o)
Solution_VSP_10 (A_io, N_i, Label_i, X_io, NoGood_o)
Solution_VSP_16 (A_io, N_i, Label_i, X_io, NoGood_o)
Solution_VSP_Z4 (A_io, N_i, Label_i, X_io, NoGood_o)
Solution_VSP_Z8 (A_io, N_i, Label_i, X_io, NoGood_o)
Solution_VSP_Z10 (A_io, N_i, Label_i, X_io, NoGood_o)
Solution_VSP_Z16 (A_io, N_i, Label_i, X_io, NoGood_o)

where
1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A], that inputs the original matrix and returns the decomposed result if the variable NoGood_o is false. For the definition of profile, please see section 3.5.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument Label_i, an INTEGER(4) array, is the address reference label. For the definition of address reference label, please see section 3.6.
4. The argument X_io, array whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution if NoGood_o is false.
5. The argument NoGood_o, a LOGICAL(4) variable, is a flag that indicates if the input matrix [A] is suitable for the subroutine. If NoGood_o=.True., the input matrix [A] is not positive definite and there is no output from the subroutine; otherwise the profile A_io returns the decomposed matrix [U] and vector X_io returns the solution. For the situation where NoGood_o=.True., please see section 3.7.

3.5 Profile

Profile for a variable-bandwidth, symmetric, and positive definite matrix is as:

$$
\begin{bmatrix}
* & * & & & & & \\
& * & * & & * & & \\
& & * & * & * & & \\
& & & * & * & & * \\
& \text{sym.} & & & * & * & * \\
& & & & & * & * \\
& & & & & & *
\end{bmatrix}
\qquad (3.1)
$$

$$\text{profile size} = \text{Label(N)-1+ N} \qquad (3.2)$$

where N is the matrix order, and Label(N) is the address reference label for the N-th column. The address reference label is discussed in the next section.

3.6 Data Storage Scheme

Data storage scheme must be declared in a Fortran program, for example:

```
REAL (4) :: A(1,1)
```

where variable A here is a single precision profile for matrix [A]. For other kinds of variable, profile must be properly declared. Then, replace the column index, for example j, with the address reference label, for example Label(J). The coefficient A_{ij} in the upper triangular part of matrix [A] is programmed in a Fortran program as A(I,Label(J)). The following algorithm defines the address reference labels:

(1) Set Label(1) = 1
(2) For i = 2 to N, do the following
    Label(i) = Label(i-1) + [number of non-zero fill-ins
        above the diagonal in the i-th column]

For the example in form (3.1), the address reference labels are 1, 2, 3, 4, 7, 8, and 11. Equation (3.2) computes 17 non-zero fill-ins that may be checked from the form (3.1). In the i-th column, the number of non-zero fill-ins above the diagonal is equal to the following:

$$\text{i-[the row index of the first non-zero fill-in]}$$

Therefore, the first non-zero fill-in in the i-th column is as:

$$\text{Label(i-1)-Label(i)+i} \qquad (3.3)$$

3.7 Failure of Calling Request

If a calling request fails, solving procedure meets a diagonal coefficient that is very small and is almost negligible compared to unity.

The subroutines introduced in this chapter deal with symmetric and positive definite systems without a consideration of pivoting. Failure of request does not mean that the input matrix is absolutely not positive definite. A pivoting may continue execution. However, pivoting not only destroys the symmetry but also disturbs sparsity. If a pivoting is necessary, try a constant-bandwidth solver with partial pivoting or a dense solver with pivoting.

3.8 Fortran Example

For a given system [A]{X}={B}, the left side matrix [A] and the right side vector {B} are defined as follows:

$$\begin{bmatrix} 1 & 4 & 2 & & & & \\ & 25 & 29 & 14 & & & \\ & & 88 & 34 & & & \\ & & & 89 & 23 & & 1 \\ & & & & 45 & 7 & 3 \\ & \text{sym.} & & & & 22 & 2 \\ & & & & & & 9 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 5 \\ 41 \\ 12 \\ 9 \\ 303 \\ 21 \\ 23 \end{bmatrix}$$

in which the order N=7. A Fortran program for decomposition and substitution is as follows. Subroutines “Input” and “Output” have data storage scheme. Subroutine “Decompose_VSP_4” decomposes matrix [A], and subroutine “Substitute_VSP_4” performs forward and backward substitutions.

```
!  *** Example program ***
!  define variables where the length of A is determined by equation (3.2)
!
     Integer (4), PARAMETER :: N = 7
     REAL (4) :: A(17),X(N)
     INTEGER (4) :: Label(N)
     LOGICAL (4) :: NoGood
     DATA X/5.0,41.0,12.0,9.0,303.0,21.0,23.0/
     DATA Label/1,2,4,6,7,8,11/
!
!  input the upper triangular part of [A]
!
     CALL Input(A,Label)
!
!  decompose in parallel
!
     CALL Decompose_VSP_4(A,N,Label,NoGood)
!
!  stop if NoGood=.True.
!
     IF(NoGood) STOP 'Cannot be decomposed'
```

```
!
!  perform substitutions in parallel
!
      CALL Substitute_VSP_4(A,N,Label,X)
!
!  output decomposed matrix
!
      CALL Output(A,N,Label)
!
!  output the solution
!
      Write(*,'(" Solution is as:")')
      Write(*,*) X
!
!  laipe done
!
      call laipeDone
!
      STOP
      END
      SUBROUTINE Input(A,Label)
!
!
!  routine to demonstrate an application of data storage scheme
!  (A)FORTRAN CALL: CALL Input(A,Label)
!     1.A: <R4> profile of matrix [A], dimension(*)
!     2.Label: <I4> address reference labels, dimension(*)
!
!  dummy arguments
!
      INTEGER (4) :: Label(1)
      REAL (4) :: A(1,1)
!
!  input
!
      A(1,Label(1))= 1.0
      A(1,Label(2))= 4.0
      A(2,Label(2))=25.0
      A(1,Label(3))= 2.0
      A(2,Label(3))=29.0
      A(3,Label(3))=88.0
      A(2,Label(4))=14.0
      A(3,Label(4))=34.0
      A(4,Label(4))=89.0
      A(4,Label(5))=23.0
      A(5,Label(5))=45.0
      A(5,Label(6))= 7.0
      A(6,Label(6))=22.0
      A(4,Label(7))= 1.0
      A(5,Label(7))= 3.0
      A(6,Label(7))= 2.0
```

```
      A(7,Label(7))= 9.0
!
      RETURN
      END
      SUBROUTINE Output(A,N,Label)
!
!
! routine to output the decomposed matrix by data storage scheme
! (A)FORTRAN CALL: CALL Output(A,N,Label)
!   1.A: <R4> profile of matrix [A], dimension(*)
!   2.N: <I4> order of matrix [A]
!   3.Label: <I4> address reference label, dimension(*)
!
! dummy arguments
!
      INTEGER (4) :: N,Label(1)
      REAL (4) :: A(1,1)
!
! local variables
!
      INTEGER (4) :: I4TEMP,Column,Row
!
! output the coefficients on non-zero fill-ins
! where the lower bound of "Row" is computed by equation (3.3)
!
      WRITE(*,'(" Row  Column  Coefficient")')
      WRITE(*,'(I4,I6,F9.3)') 1,1,A(1,1)
      DO I4TEMP=2,N
         Column=Label(I4TEMP)
         DO Row=Label(I4TEMP-1)-Column+I4TEMP, I4TEMP
            WRITE(*,'(I4,I6,F9.3)') Row,I4TEMP, A(Row,Column)
         END DO
      END DO
!
      RETURN
      END
```

Chapter 4. Dense, Symmetric, and Positive Definite Systems

4.1 Purpose

This chapter has subroutines for the solution of [A]{X}={B} where the left side matrix [A] is dense, symmetric, and positive definite. The non-zero fill-ins in the lower triangular part of matrix [A] have a shape, for example, as:

$$\begin{bmatrix} * & & & & & & \\ * & * & & & \text{sym.} & & \\ * & * & * & & & & \\ * & * & * & * & & & \\ * & * & * & * & * & & \\ * & * & * & * & * & * & \\ * & * & * & * & * & * & * \end{bmatrix}$$

where the symbol * indicates non-zero fill-ins. Three types of subroutine are introduced in the chapter, which perform the following functions:

1. Decompose matrix [A] into the product of $[L][L]^T$ where matrix [L] is the lower triangular matrix.
2. Perform forward and backward substitutions.
3. Solve [A]{X}={B} in a single call.

Decomposition and substitution must be called in order, and work together as a pair. No pivoting is applied to the subroutines introduced in this chapter. Subroutines are as follows:

Decompose_DSP_4
Decompose_DSP_8
Decompose_DSP_10
Decompose_DSP_16
Decompose_DSP_Z4
Decompose_DSP_Z8
Decompose_DSP_Z10
Decompose_DSP_Z16

Substitute_DSP_4
Substitute_DSP_8
Substitute_DSP_10
Substitute_DSP_16
Substitute_DSP_Z4
Substitute_DSP_Z8
Substitute_DSP_Z10

Substitute_DSP_Z16

Solution_DSP_4
Solution_DSP_8
Solution_DSP_10
Solution_DSP_16
Solution_DSP_Z4
Solution_DSP_Z8
Solution_DSP_Z10
Solution_DSP_Z16

4.2 Fortran Syntax for Subroutine Decompose

The following subroutines decompose [A] into [A]= $[L][L]^T$. Syntax is as follows:

```
Decompose_DSP_4(A_io, N_i, Label_i, NoGood_o)
Decompose_DSP_8(A_io, N_i, Label_i, NoGood_o)
Decompose_DSP_10(A_io, N_i, Label_i, NoGood_o)
Decompose_DSP_16(A_io, N_i, Label_i, NoGood_o)
Decompose_DSP_Z4(A_io, N_i, Label_i, NoGood_o)
Decompose_DSP_Z8(A_io, N_i, Label_i, NoGood_o)
Decompose_DSP_Z10(A_io, N_i, Label_i, NoGood_o)
Decompose_DSP_Z16(A_io, N_i, Label_i, NoGood_o)
```

where

1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A] that inputs the original matrix and returns the result if the variable NoGood_o is false. For the definition of profile, please see section 4.5.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument Label_i, an INTEGER(4) array, is the address reference label. For the definition of address reference label, please see section 4.6.
4. The argument NoGood_o, a LOGICAL(4) variable, is a flag that indicates if the input matrix [A] is suitable for the subroutine. If NoGood_o=.True., the input matrix [A] cannot be decomposed by the subroutine and there is no output from the subroutine; otherwise the profile A_io returns the decomposed matrix [L]. For the situation where NoGood_o=.True., please see section 4.7.

4.3 Fortran Syntax for Subroutine Substitute

The following subroutines perform forward and backward substitutions. Syntax is as follows:

```
Substitute_DSP_4(A_i, N_i, Label_i, sX_io)
Substitute_DSP_8(A_i, N_i, Label_i, sX_io)
Substitute_DSP_10(A_i, N_i, Label_i, sX_io)
Substitute_DSP_16(A_i, N_i, Label_i, sX_io)
Substitute_DSP_Z4(A_i, N_i, Label_i, sX_io)
Substitute_DSP_Z8(A_i, N_i, Label_i, sX_io)
```

Substitute_DSP_Z10(A_i, N_i, Label_i, sX_io)
Substitute_DSP_Z16(A_i, N_i, Label_i, sX_io)

where

1. The argument A_i, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A] that inputs the result from decomposition.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument Label_i, an INTEGER(4) array, is the address reference label. For the definition of address reference label, please see section 4.6.
4. The argument X_io, array whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution.

4.4 Fortran Syntax for Subroutine Solution

The following subroutines first decompose [A] into the product of $[L][L]^T$, and then perform forward and backward substitutions. Solve [A]{X}={B} in a single call. The syntax is as follows:

Solution_DSP_4(A_io, N_i, Label_i, X_io, NoGood_o)
Solution_DSP_8(A_io, N_i, Label_i, X_io, NoGood_o)
Solution_DSP_10(A_io, N_i, Label_i, X_io, NoGood_o)
Solution_DSP_16(A_io, N_i, Label_i, X_io, NoGood_o)
Solution_DSP_Z4(A_io, N_i, Label_i, X_io, NoGood_o)
Solution_DSP_Z8(A_io, N_i, Label_i, X_io, NoGood_o)
Solution_DSP_Z10(A_io, N_i, Label_i, X_io, NoGood_o)
Solution_DSP_Z16(A_io, N_i, Label_i, X_io, NoGood_o)

where

1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A], that inputs the original matrix and returns the decomposed result if the variable NoGood_o is false. For the definition of profile, please see section 4.5.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument Label_i, an INTEGER(4) array, is the address reference label. For the definition of address reference label, please see section 4.6.
4. The argument X_io, array whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution if NoGood_o is false.
5. The argument NoGood_o, a LOGICAL(4) variable, is a flag that indicates if the input matrix [A] is suitable for the subroutine. If NoGood_o=.True., the input system cannot be solved by the subroutine and there is no output from the subroutine; otherwise the profile A_io returns the decomposed matrix [L] and vector X_io returns the solution. For the situation where NoGood_o=.True., please see section 4.7.

4.5 Profile

Profile for a dense, symmetric, and positive definite matrix is as:

$$\begin{bmatrix} * & & & & & & \\ * & * & & & \text{sym.} & & \\ * & * & * & & & & \\ * & * & * & * & & & \\ * & * & * & * & * & & \\ * & * & * & * & * & * & \\ * & * & * & * & * & * & * \end{bmatrix} \qquad (4.1)$$

where the symbol * represents non-zero fill-ins. Total length of profile is determined as

$$\text{profile size} = ((N+1) * N) / 2 \qquad (4.2)$$

where N is the matrix order.

4.6 Data Storage Scheme

Data storage scheme for a dense and symmetric matrix must be declared in a Fortran program, for example:

```
REAL (4) :: A(1,1)
```

where variable A here is a single precision profile for a matrix [A]. For other kinds of variable, profile must be properly declared. Then, replace column index, for example j, with the address reference label, for example Label(J). The coefficient A_{ij} in the lower triangular part of matrix [A] is programmed in a Fortran program as A(I,Label(J)). The following algorithm defines the address reference labels:

(1) Set Label(1) = 1
(2) For i = 2 to N, do the following
    Label(i) = Label(i-1) + [number of non-zero fill-ins in the i-th column] (4.3)

For the example in form (4.1), the address reference labels are 1, 7, 12, 16, 19, 21, and 22. Equation (4.2) computes 28 non-zero fill-ins that may be checked from the form (4.1).

4.7 Failure of Calling Request

If a calling request fails, solving procedure meets a diagonal coefficient that is very small and is negligible compared to unity.

The subroutines introduced in this chapter deal with symmetric and positive definite systems without a consideration of pivoting. Failure of request does not mean that the input matrix is indefinite. A pivoting may continue execution. However, pivoting may destroy symmetry. If a pivoting is necessary, try a dense solver with pivoting. Pivoting procedure always takes more time, and is less efficient in parallel processing.

4.8 Fortran Example

For a given system [A]{X}={B}, the left side matrix [A] and the right side vector {B} are defined as follows:

$$\begin{bmatrix} 1 & & & & & & \\ 4 & 25 & & & \text{sym.} & & \\ 2 & 19 & 44 & & & & \\ 3 & 9 & 34 & 89 & & & \\ 1 & -2 & 3 & 0 & 45 & & \\ 4 & 2 & 2 & 11 & 7 & 68 & \\ 2 & 7 & 3 & 4 & 3 & 2 & 9 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 21 \\ 141 \\ 2 \\ 9 \\ 333 \\ 1 \\ 3 \end{bmatrix}$$

in which the order N=7. A Fortran program for decomposition and substitution is as follows. Subroutines "Input" and "Output" have data storage scheme. Subroutine "DenseLabel" based on equation (4.3) generates address reference labels. Two LAIPE subroutines are applied in this example: one is subroutine "Decompose_DSP_4" that decomposes matrix [A]; the other is subroutine "Substitute_DSP_4" that performs forward and backward substitutions.

```
! *** Example program ***
! define variables where the length of A is determined by equation (4.2)
!
      Integer (4),PARAMETER :: N=7
      REAL (4) :: A(((N+1)*N)/2),X(N)
      INTEGER (4) :: Label(N)
      LOGICAL (4) :: NoGood
      DATA X/21.0,141.0,2.0,9.0,333.0,1.0,3.0/
!
! generate address reference labels
!
      CALL DenseLabel(Label,N)
!
! input the lower triangular part of [A]
!
      CALL Input(A,Label)
!
! decompose in parallel
!
      CALL Decompose_DSP_4(A,N,Label,NoGood)
!
! stop if NoGood=.True.
!
      IF(NoGood) STOP 'Cannot be decomposed'
!
! perform substitutions in parallel
!
      CALL Substitute_DSP_4(A,N,Label,X)
!
```

```
!  output decomposed matrix
!
      CALL Output(A,N,Label)
!
!  output the solution
!
      Write(*,'(" Solution is as:")')
      Write(*,*) X
!
!  laipe done
!
      call laipeDone
!
      STOP
      END

      SUBROUTINE DenseLabel(Label,N)
!
!
!  routine to generate address reference labels for a dense lower triangular matrix
!  (A)FORTRAN CALL: CALL DenseLabel(Label,N)
!     1.Label: <I4> return address reference labels, dimension(N)
!     2.N: <I4> order of matrix
!
!  dummy arguments
!
     INTEGER*4 Label(1),N
!
!  local variables
!
     INTEGER*4 I4TEMP,J4TEMP
!
!  generate address label
!
     I4TEMP=N-1
     Label(1)=1
     DO J4TEMP=2,N
          Label(J4TEMP)=Label(J4TEMP-1)+I4TEMP
          I4TEMP=I4TEMP-1
     END DO
!
     RETURN
     END
     SUBROUTINE Input(A,Label)
!
!
!  routine to demonstrate an application of the data storage scheme
!  (A)FORTRAN CALL: CALL Input(A,Label)
!     1.A: <R4> profile of matrix [A], dimension(*)
!     2.Label: <I4> the address reference labels, dimension(N)
!
```

```
!  dummy arguments
!
       INTEGER*4 Label(1)
       REAL*4 A(1,1)
!
!  input
!

       A(1,Label(1))= 1.0
       A(2,Label(1))= 4.0
       A(3,Label(1))= 2.0
       A(4,Label(1))= 3.0
       A(5,Label(1))= 1.0
       A(6,Label(1))= 4.0
       A(7,Label(1))= 2.0
       A(2,Label(2))=25.0
       A(3,Label(2))=19.0
       A(4,Label(2))= 9.0
       A(5,Label(2))=-2.0
       A(6,Label(2))= 2.0
       A(7,Label(2))= 7.0
       A(3,Label(3))=44.0
       A(4,Label(3))=34.0
       A(5,Label(3))= 3.0
       A(6,Label(3))= 2.0
       A(7,Label(3))= 3.0
       A(4,Label(4))=89.0
       A(5,Label(4))= 0.0
       A(6,Label(4))=11.0
       A(7,Label(4))= 4.0
       A(5,Label(5))=45.0
       A(6,Label(5))= 7.0
       A(7,Label(5))= 3.0
       A(6,Label(6))=68.0
       A(7,Label(6))= 2.0
       A(7,Label(7))= 9.0
!
       RETURN
       END
       SUBROUTINE Output(A,N,Label)
!
!
!  routine to output the decomposed matrix by data storage scheme
!  (A)FORTRAN CALL: CALL Output(A,N,Label)
!     1.A: <R4> profile of matrix [A], dimension(*)
!     2.N: <I4> order of matrix [A]
!     3.Label: <I4> address reference labels, dimension(N)
!
!  dummy arguments
!
       INTEGER*4 N,Label(1)
```

```
      REAL*4 A(1,1)
!
!  local variables
!
      INTEGER*4 Column,Row,I4TEMP
!
!  output the coefficients on non-zero fill-ins
!
      WRITE(*,'(" Row  Column  Coefficient")')
      DO I4TEMP=1,N
          Column=Label(I4TEMP)
          DO Row=I4TEMP,N
               WRITE(*,'(I4,I6,F9.3)') Row, I4TEMP, A(Row,Column)
          END DO
      END DO
!
      RETURN
      END
```

Chapter 5. Constant-Bandwidth and Symmetric Systems

5.1 Purpose

This chapter has subroutines for the solution of [A]{X}={B} where the left side matrix [A] has a constant bandwidth and is symmetric. There is no consideration of definiteness of matrix [A]. The non-zero fill-ins in the lower triangular part of matrix [A] have a shape, for example, as:

$$\begin{bmatrix} * & & & & & & \\ * & * & & & \text{sym.} & & \\ * & * & * & & & & \\ * & * & * & * & & & \\ & * & * & * & * & & \\ & & * & * & * & * & \\ & & & * & * & * & * \end{bmatrix}$$

Three types of subroutine are introduced in this chapter, which perform the following functions:

1. Decompose matrix [A] into the product of $[L][D][L]^T$ where matrix [L] is the lower triangular matrix and matrix [D] is the diagonal matrix.
2. Perform forward and backward substitutions.
3. Solve [A]{X}={B} in a single call.

Decomposition and substitution must be called in order, and work together as a pair. No pivoting is applied to the subroutines introduced in this chapter. Subroutines are as follows:

Decompose_CSG_4
Decompose_CSG_8
Decompose_CSG_10
Decompose_CSG_16
Decompose_CSG_Z4
Decompose_CSG_Z8
Decompose_CSG_Z10
Decompose_CSG_Z16

Substitute_CSG_4
Substitute_CSG_8
Substitute_CSG_10
Substitute_CSG_16
Substitute_CSG_Z4
Substitute_CSG_Z8
Substitute_CSG_Z10

Substitute_CSG_Z16

Solution_CSG_4
Solution_CSG_8
Solution_CSG_10
Solution_CSG_16
Solution_CSG_Z4
Solution_CSG_Z8
Solution_CSG_Z10
Solution_CSG_Z16

meSolution_CSG_4
meSolution_CSG_8
meSolution_CSG_10
meSolution_CSG_16
meSolution_CSG_Z4
meSolution_CSG_Z8
meSolution_CSG_Z10
meSolution_CSG_Z16

The subroutines with a prefix "me", i.e., meSolution_CSG_4, are multiple entry direct solvers that are most well suitable for systems with a small bandwidth. For more detailed discussions on multiple entry solvers, please see section 1.7.

5.2 Fortran Syntax for Subroutine Decompose

The following subroutines decompose matrix [A] into [A]= $[L][D][L]^T$. Syntax is as follows:

Decompose_CSG_4(A_io, N_i, LowerBandwidth_i, NoGood_o)
Decompose_CSG_8(A_io, N_i, LowerBandwidth_i, NoGood_o)
Decompose_CSG_10(A_io, N_i, LowerBandwidth_i, NoGood_o)
Decompose_CSG_16(A_io, N_i, LowerBandwidth_i, NoGood_o)
Decompose_CSG_Z4(A_io, N_i, LowerBandwidth_i, NoGood_o)
Decompose_CSG_Z8(A_io, N_i, LowerBandwidth_i, NoGood_o)
Decompose_CSG_Z10(A_io, N_i, LowerBandwidth_i, NoGood_o)
Decompose_CSG_Z16(A_io, N_i, LowerBandwidth_i, NoGood_o)

where

1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A] that inputs the original matrix and returns the result if the variable NoGood_o is false. For the definition of profile, please see section 5.6.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument LowerBandwidth_i, an INTEGER(4) variable, is the lower bandwidth of matrix [A]. The lower bandwidth is the maximal number of non-zero fill-ins below the diagonal in a column.
4. The argument NoGood_o, a LOGICAL(4) variable, is a flag that indicates if the input matrix [A] is suitable for the subroutine. If NoGood_o=.True., the input matrix [A] cannot be decomposed and there is no output returned; otherwise the profile A_io returns the

decomposed matrices [L] and [D]. For the situation where NoGood_o=.True., please see section 5.8.

5.3 Fortran Syntax for Subroutine Substitute

The following subroutines perform forward and backward substitutions. Syntax is as follows:

```
Substitute_CSG_4(A_i, N_i, LowerBandwidth_i, X_io)
Substitute_CSG_8(A_i, N_i, LowerBandwidth_i, X_io)
Substitute_CSG_10(A_i, N_i, LowerBandwidth_i, X_io)
Substitute_CSG_16(A_i, N_i, LowerBandwidth_i, X_io)
Substitute_CSG_Z4(A_i, N_i, LowerBandwidth_i, X_io)
Substitute_CSG_Z8(A_i, N_i, LowerBandwidth_i, X_io)
Substitute_CSG_Z10(A_i, N_i, LowerBandwidth_i, X_io)
Substitute_CSG_Z16(A_i, N_i, LowerBandwidth_i, X_io)
```

where

1. The argument A_i, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A] that inputs the result from decomposition.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument LowerBandwidth_i, an INTEGER(4) variable, is the lower bandwidth of matrix [A]. The lower bandwidth is the maximal number of non-zero fill-ins below the diagonal in a column.
4. The argument X_io, array whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution.

5.4 Fortran Syntax for Subroutine Solution

The following subroutines first decompose [A] into the product of $[L][D][L]^T$, and then perform forward and backward substitutions. Solve [A]{X}={B} in a single call. The syntax is as follows:

```
Solution_CSG_4(A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)
Solution_CSG_8(A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)
Solution_CSG_10(A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)
Solution_CSG_16(A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)
Solution_CSG_Z4(A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)
Solution_CSG_Z8(A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)
Solution_CSG_Z10(A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)
Solution_CSG_Z16(A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)
```

where

1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A], that inputs the original matrix and returns the decomposed result if the variable NoGood_o is false. For the definition of profile, please see section 5.6.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].

3. The argument LowerBandwidth_i, an INTEGER(4) variable, is the lower bandwidth of matrix [A]. The lower bandwidth is the maximal number of non-zero fill-ins below the diagonal in a column.
4. The argument X_io, array whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution if NoGood_o is false.
5. The argument NoGood_o, a LOGICAL(4) variable, is a flag that indicates if the input system is suitable for the subroutine. If NoGood_o=.True., the input system cannot be solved by the subroutine and there is no output returned; otherwise the profile A_io returns the decomposed matrices [L] and [D], and vector X_io returns the solution. For the situation where NoGood_o=.True., please see section 5.8.

5.5 Fortran Syntax for Subroutine meSolution

The following subroutines solve the system [A][X]=[B] by multiple entry procedure, where [X] and [B] may be a matrix with multiple vectors, i.e., [X]=[{ X_1 } { X_2 } ...] and [B]=[{ B_1 } { B_2 } ...]. Syntax is as follows:

```
meSolution_CSG_4(A_io,N_i,LowerBandwidth_i,X_io,Nset_i,WorkingSpace_x,NoGood_o)
meSolution_CSG_8(A_io,N_i,LowerBandwidth_i,X_io,Nset_i,WorkingSpace_x,NoGood_o)
meSolution_CSG_10(A_io,N_i,LowerBandwidth_i,X_io,Nset_i,WorkingSpace_x,NoGood_o)
meSolution_CSG_16(A_io,N_i,LowerBandwidth_i,X_io,Nset_i,WorkingSpace_x, NoGood_o)
meSolution_CSG_Z4(A_io,N_i,LowerBandwidth_i,X_io,Nset_i,WorkingSpace_x, NoGood_o)
meSolution_CSG_Z8(A_io,N_i,LowerBandwidth_i,X_io,Nset_i,WorkingSpace_x, NoGood_o)
meSolution_CSG_Z10(A_io,N_i,LowerBandwidth_i,X_io,Nset_i,WorkingSpace_x,NoGood_o)
meSolution_CSG_Z16(A_io,N_i,LowerBandwidth_i,X_io,Nset_i,WorkingSpace_x,NoGood_o)
```

where

1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A] that inputs the original matrix. After returning from this subroutine, the content in array A_io is destroyed no matter if the calling request is successful or not. For the definition of profile, please see section 5.6.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument LowerBandwidth_i, an INTEGER(4) variable, is the lower bandwidth of matrix [A]. The lower bandwidth is the maximal number of non-zero fill-ins below the diagonal in a column. This subroutine is more efficient if the lower bandwidth is small.
4. The argument X_io, array whose kind must be consistent with subroutine name convention, inputs the right side vector(s), and returns the solution if NoGood_o is false.
5. The argument Nset_i, an INTEGER(4) variable, is the number of right side vectors.
6. The argument WorkingSpace_x, array whose kind must be consistent with subroutine name convention and providing a space of (2*N_i*LowerBandwidth_I) elements, is a working space.
7. The argument NoGood_o, a LOGICAL(4) variable, is a flag that indicates if the input matrix [A] is suitable for the subroutine. If NoGood_o=.True., the input system cannot be solved by this function and there is no output; otherwise the vector "X_io" returns the solution. For the situation NoGood_o=.True., please see section 5.8.

5.6 Profile

Profile for a constant-bandwidth and symmetric matrix is as:

$$
\begin{array}{l}
\left[\begin{array}{ccccccc}
* & & & & & & \\
* & * & & & \text{sym.} & & \\
* & * & * & & & & \\
* & * & * & * & & & \\
 & * & * & * & * & & \\
 & & * & * & * & * & \\
 & & & * & * & * & *
\end{array}\right] \\
\begin{array}{ccccccc}
 & & & & \& & \& & \\
 & & & & & \& &
\end{array}
\end{array}
\qquad (5.1)
$$

where the symbol * represents non-zero fill-ins and the symbol & indicates an extra memory space whose content is never used. Total length of profile is determined as

$$\text{profile size} = (N-1) * \text{LowerBandwidth} + N \qquad (5.2)$$

where N is the matrix order, and LowerBandwidth is the lower bandwidth.

5.7 Data Storage Scheme

Data storage scheme for a constant-bandwidth and symmetric matrix must be declared in a Fortran program, for example:

```
INTEGER (4) :: LowerBandwidth
REAL (4) :: A(LowerBandwidth,1)
```

where variable A here is a single precision profile for matrix [A]. For other kinds of variable, profile must be properly declared. Then, the coefficient A_{ij} in the lower triangular part of matrix [A] is programmed in a Fortran program as A(I,J).

5.8 Failure of Calling Request

If a calling request fails, solving procedure meets a diagonal coefficient whose absolute value is very small and is almost negligible compared to unity.

The subroutines introduced in this chapter deal with symmetric systems without a consideration of pivoting. Since a symmetric solver does not consider pivoting. Failure of request does not mean that the input matrix is absolutely singular. A pivoting may continue execution. However, pivoting may destroy symmetry. If a pivoting is necessary, try a solver with partial pivoting that will be discussed in chapter 13. A pivoting procedure always takes more time, and is less efficient in parallel processing.

5.9 Fortran Example

For a given system [A]{X}={B}, the left side matrix [A] and the right side vector {B} are defined as follows:

$$
\begin{bmatrix}
1 & & & & & & \\
4 & 25 & & \text{sym.} & & & \\
2 & 29 & 14 & & & & \\
 & 99 & 34 & 19 & & & \\
 & & 3 & 23 & 5 & & \\
 & & & 11 & 7 & 22 & \\
 & & & & 3 & 2 & 9
\end{bmatrix}
\quad \text{and} \quad
\begin{bmatrix}
21 \\ 11 \\ 122 \\ 19 \\ 333 \\ 1 \\ 3
\end{bmatrix}
$$

in which the order N=7 and the lower bandwidth, denoted by LowerBandwidth, is 2. A Fortran program for decomposition and substitution is as follows. Subroutines “Input” and “Output” have data storage scheme. Subroutine “Decompose_CSG_4” decomposes matrix [A], subroutine “Substitute_CSG_4” performs forward and backward substitutions.

```
!  *** Example program ***
!  define variables where the length of A is determined by equation (5.2)
!
      Integer (4) , PARAMETER :: N=7
      Integer (4), PARAMETER :: LowerBandwidth=2
      REAL (4) :: A((N-1)*LowerBandwidth+N),sX(N)
      LOGICAL*4 NoGood
      DATA sX/21.0,11.0,122.0,19.0,333.0,1.0,3.0/
!
!  input the lower triangular part of [A]
!
      CALL Input(A,LowerBandwidth)
!
!  decompose in parallel
!
       CALL Decompose_CSG_4(A,N,LowerBandwidth,NoGood)
!
!  stop if NoGood=.True.
!
      IF(NoGood) STOP 'Cannot be decomposed'
!
!  perform substitutions in parallel
!
      CALL Substitute_CSG_4(A,N,LowerBandwidth,sX)
!
!  output decomposed matrix
!
      CALL Output(A,N,LowerBandwidth)
!
!  output the solution
!
      Write(*,'(" Solution is as:")')
      Write(*,*) X
```

```
!
! laipe done
!
      call laipeDone
!
      STOP
      END
      SUBROUTINE Input(A,LowerBandwidth)
!
!
! routine to demonstrate an application of data storage scheme
! (A)FORTRAN CALL: CALL Input(A,LowerBandwidth)
!    1.A: <R4> profile of matrix [A], dimension(*)
!    2.LowerBandwidth: <I4> lower bandwidth
!
! dummy arguments
!
      INTEGER (4) :: LowerBandwidth
      REAL (4) :: A(LowerBandwidth,1)
!
! input
!
      A(1,1)= 1.0
      A(2,1)= 4.0
      A(3,1)= 2.0
      A(2,2)=25.0
      A(3,2)=29.0
      A(4,2)=99.0
      A(3,3)=14.0
      A(4,3)=34.0
      A(5,3)= 3.0
      A(4,4)=19.0
      A(5,4)=23.0
      A(6,4)=11.0
      A(5,5)= 5.0
      A(6,5)= 7.0
      A(7,5)= 3.0
      A(6,6)=22.0
      A(6,6)=22.0
      A(7,6)= 2.0
      A(7,7)= 9.0
!
      RETURN
      END
      SUBROUTINE Output(A,N,LowerBandwidth)
!
!
! routine to output the decomposed matrix by data storage scheme
! (A)FORTRAN CALL: CALL Output(A,N,LowerBandwidth)
!    1.A: <R4> profile of matrix [A], dimension(*)
!    2.N: <I4> order of matrix [A]
```

```
!    3.LowerBandwidth: <I4> lower bandwidth
!
!  dummy arguments
!
      INTEGER (4) :: N,LowerBandwidth
      REAL (4) :: A(LowerBandwidth,1)
!
!  local variables
!
      INTEGER*4 Column,Row
!
!  output the coefficients on non-zero fill-ins
!
      WRITE(*,'(" Row  Column  Coefficient")')
      DO Column=1,N
          DO Row=Column, MIN0(Column+LowerBandwidth,N)
              WRITE(*,'(I4,I6,F9.3)') Row,Column, A(Row,Column)
          END DO
      END DO
!
      RETURN
      END
```

Chapter 6. Variable-Bandwidth and Symmetric Systems

6.1 Purpose

This chapter has subroutines for the solution of [A]{X}={B} where the left side matrix [A] has a variable bandwidth and is symmetric. There is no consideration of definiteness of matrix [A]. The non-zero fill-ins in the upper triangular part of matrix [A] have a shape, for example, as:

$$\begin{bmatrix} * & * & & & & & \\ & * & * & & * & & \\ & & * & * & * & & \\ & & & * & * & & * \\ & \text{sym.} & & & * & * & * \\ & & & & & * & * \\ & & & & & & * \end{bmatrix}$$

which looks like a skyline in a city, and is sometimes called *skyline solver*. Three types of subroutine are introduced in the chapter, which perform the following functions:

1. Decompose matrix [A] into the product of $[U]^T[D][U]$ where matrix [U] is the upper triangular matrix and matrix [D] is the diagonal matrix.
2. Perform forward and backward substitutions.
3. Solve [A]{X}={B} in a single call.

Decomposition and substitution must be called in order, and work together as a pair. No pivoting is applied to the subroutines, which are as:

Decompose_VSG_4
Decompose_VSG_8
Decompose_VSG_10
Decompose_VSG_16
Decompose_VSG_Z4
Decompose_VSG_Z8
Decompose_VSG_Z10
Decompose_VSG_Z16

Substitute_VSG_4
Substitute_VSG_8
Substitute_VSG_10
Substitute_VSG_16
Substitute_VSG_Z4
Substitute_VSG_Z8
Substitute_VSG_Z10
Substitute_VSG_Z16

Solution_VSG_4
Solution_VSG_8
Solution_VSG_10
Solution_VSG_16
Solution_VSG_Z4
Solution_VSG_Z8
Solution_VSG_Z10
Solution_VSG_Z16

6.2 Fortran Syntax for Subroutine Decompose

The following subroutines decompose matrix [A] into [A]= $[U]^T[D][U]$. Syntax is as follows:

```
Decompose_VSG_4(A_io, N_i, Label_i, NoGood_o)
Decompose_VSG_8(A_io, N_i, Label_i, NoGood_o)
Decompose_VSG_10(A_io, N_i, Label_i, NoGood_o)
Decompose_VSG_16(A_io, N_i, Label_i, NoGood_o)
Decompose_VSG_Z4(A_io, N_i, Label_i, NoGood_o)
Decompose_VSG_Z8(A_io, N_i, Label_i, NoGood_o)
Decompose_VSG_Z10(A_io, N_i, Label_i, NoGood_o)
Decompose_VSG_Z16(A_io, N_i, Label_i, NoGood_o)
```

where

1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A] that inputs the original matrix and returns the result if the variable NoGood_o is false. For the definition of profile, please see section 6.5.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument Label_i, an INTEGER(4) array, is the address reference label. For the definition of address reference label, please see section 6.6.
4. The argument NoGood_o, a LOGICAL(4) variable, is a flag that indicates if the input matrix [A] is suitable for decomposition. If NoGood_o=.True., the input matrix [A] cannot be decomposed and there is no output returned; otherwise the profile A_io returns the decomposed matrices [U] and [D]. For the situation where NoGood_o=.True., please see section 6.7.

6.3 Fortran Syntax for Subroutine Substitute

The following subroutines perform forward and backward substitutions. Syntax is as follows:

```
Substitute_VSG_4(A_i, N_i, Label_i, X_io)
Substitute_VSG_8(A_i, N_i, Label_i, X_io)
Substitute_VSG_10(A_i, N_i, Label_i, X_io)
Substitute_VSG_16(A_i, N_i, Label_i, X_io)
Substitute_VSG_Z4(A_i, N_i, Label_i, X_io)
Substitute_VSG_Z8(A_i, N_i, Label_i, X_io)
```

Substitute_VSG_Z10(A_i, N_i, Label_i, X_io)
Substitute_VSG_Z16(A_i, N_i, Label_i, X_io)

where

1. The argument A_i, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A] that inputs the result from decomposition.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument Label_i, an INTEGER(4) array, is the address reference label. For the definition of address reference label, please see section 6.6.
4. The argument X_io, array whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution.

6.4 Fortran Syntax for Subroutine Solution

The following subroutines first decompose matrix [A] into the product of $[U]^T[D][[U]$, and then perform forward and backward substitutions. Solve the system [A]{X}={B} in a single call. Syntax is as follows:

Solution_VSG_4(A_io, N_i, Label_i, X_io, NoGood_o)
Solution_VSG_8(A_io, N_i, Label_i, X_io, NoGood_o)
Solution_VSG_10(A_io, N_i, Label_i, X_io, NoGood_o)
Solution_VSG_16(A_io, N_i, Label_i, X_io, NoGood_o)
Solution_VSG_Z4(A_io, N_i, Label_i, X_io, NoGood_o)
Solution_VSG_Z8(A_io, N_i, Label_i, X_io, NoGood_o)
Solution_VSG_Z10(A_io, N_i, Label_i, X_io, NoGood_o)
Solution_VSG_Z16(A_io, N_i, Label_i, X_io, NoGood_o)

where

1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A], that inputs the original matrix and returns the decomposed result if the variable NoGood_o is false. For the definition of profile, please see section 6.5.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument Label_i, an INTEGER(4) array, is the address reference label. For the definition of address reference label, please see section 6.6.
4. The argument X_io, array whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution if NoGood_o is false.
5. The argument NoGood_o, a LOGICAL(4) variable, is a flag that indicates if the input system is suitable for the subroutine. If NoGood_o=.True., the input system cannot be solved by the subroutine and there is no output returned; otherwise the profile A_io returns the decomposed matrices [U] and [D], and vector X_io returns the solution. For the situation where NoGood_o=.True., please see section 6.7.

6.5 Profile

Profile for a variable-bandwidth and symmetric matrix is as:

$$
\begin{bmatrix}
* & * & & & & & \\
 & * & * & & * & & \\
 & & * & * & * & & \\
 & & & * & * & & * \\
 & \text{sym.} & & & * & * & * \\
 & & & & & * & * \\
 & & & & & & *
\end{bmatrix}
\qquad (6.1)
$$

where the symbol * represents non-zero fill-ins. Total length of profile is determined as

$$\text{profile size} = \text{Label(N)-1+ N} \qquad (6.2)$$

where N is the matrix order, and Label(N) is the address reference label for the N-th column. The address reference label is discussed in the next section.

6.6 Data Storage Scheme

Data storage scheme for a variable-bandwidth and symmetric matrix must be declared in a Fortran program, for example:

REAL (4) :: A(1,1)

where variable A here is a single precision profile for matrix [A]. For other kinds of variable, profile must be properly declared. Then, replace the column index, for example j, with the address reference label, for example Label(J). The coefficient A_{ij} in the upper triangular part of matrix [A] is programmed in a Fortran program as A(I,Label(J)). Address reference labels are defined by the following algorithm where N is the order of matrix [A]:

(1) Set Label(1) = 1
(2) For i = 2 to N, do the following
  Label(i) = Label(i-1) + [number of non-zero fill-ins
    above the diagonal in the i-th column] (6.3)

For the example in form (6.1), the address reference labels are 1, 2, 3, 4, 7, 8, and 11. Equation (6.2) computes 17 non-zero fill-ins that may be checked from the form (6.1). In the i-th column, the number of non-zero fill-ins above the diagonal is equal to the following:

i-[the row index of the first non-zero fill-in]

Therefore, the first non-zero fill-in in the i-th column is as:

$$\text{Label(i-1)-Label(i)+i} \qquad (6.4)$$

6.7 Failure of Calling Request

If a calling request fails, solving procedure meets a diagonal coefficient whose absolute value is very small and is negligible compared to unity.

The subroutines introduced in this chapter deal with symmetric systems without a consideration of pivoting. Failure of request does not mean that the input matrix is absolutely singular. A pivoting may continue execution. However, pivoting may destroy not only symmetric property but also sparsity. If a pivoting is necessary, try a constant-bandwidth solver with partial pivoting or a dense solver with pivoting.

6.8 Fortran Example

For a given system [A]{X}={B}, the left side matrix [A] and the right side vector {B} are defined as follows:

$$\begin{bmatrix} 1 & 4 & 72 & & & & \\ & 25 & 29 & 44 & & & \\ & & 14 & 34 & & & \\ & & & 19 & 23 & & 9 \\ & & & & 8 & 37 & 3 \\ & \text{sym.} & & & & 2 & 2 \\ & & & & & & 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 5 \\ 41 \\ 12 \\ 9 \\ 303 \\ 21 \\ 23 \end{bmatrix}$$

in which the order N=7. A Fortran program for decomposition and substitution is as follows. Subroutines "Input" and "Output" have data storage scheme. Subroutine "Decompose_VSG_4" decomposes matrix [A], and subroutine "Substitute_VSG_4" performs forward and backward substitutions.

```
!  *** Example program ***
!  define variables where the length of A is determined by equation (6.2)
!
      PARAMETER (N=7)
      REAL*4 A(17),X(N)
      INTEGER*4 Label(N)
      LOGICAL*4 NoGood
      DATA X/5.0,41.0,12.0,9.0,303.0,21.0,23.0/
      DATA Label/1,2,4,6,7,8,11/
!
!  input the upper triangular part of [A]
!
      CALL Input(A,Label)
!
!  decompose in parallel
```

```
!
      CALL Decompose_VSG_4(A,N,Label, NoGood)
!
!  stop if NoGood=.True.
!
      IF(NoGood) STOP 'Cannot be decomposed'
!
!  perform substitutions in parallel
!
      CALL Substitute_VSG_4(A,N,Label,X)
!
!  output decomposed matrix
!
      CALL Output(A,N,Label)
!
!  output the solution
!
      Write(*,'(" Solution is as:")')
      Write(*,*) X
!
!  laipe done
!
      call laipeDone
!
      STOP
      END

      SUBROUTINE Input(A,Label)
!
!
!  routine to demonstrate an application of data storage scheme
!  (A)FORTRAN CALL: CALL Input(A,Label)
!    1.A: <R4> profile of matrix [A], dimension(*)
!    2.Label: <I4> address reference labels, dimension(*)
!
!  dummy arguments
!
      INTEGER*4 Label(1)
      REAL*4 A(1,1)
!
!  input
!
      A(1,Label(1))= 1.0
      A(1,Label(2))= 4.0
      A(2,Label(2))=25.0
      A(1,Label(3))=72.0
      A(2,Label(3))=29.0
      A(3,Label(3))=14.0
      A(2,Label(4))=44.0
      A(3,Label(4))=34.0
      A(4,Label(4))=19.0
```

```
      A(4,Label(5))=23.0
      A(5,Label(5))= 8.0
      A(5,Label(6))=37.0
      A(6,Label(6))= 2.0
      A(4,Label(7))= 9.0
      A(5,Label(7))= 3.0
      A(6,Label(7))= 2.0
      A(7,Label(7))= 1.0
!
      RETURN
      END
      SUBROUTINE Output(A,N,Label)
!
!
! routine to output the decomposed matrix by data storage scheme
! (A)FORTRAN CALL: CALL Output(A,N,Label)
!   1.A: <R4> profile of matrix [A], dimension(*)
!   2.N: <I4> order of matrix [A]
!   3.Label: <I4> address reference labels, dimension(*)
!
! dummy arguments
!
      INTEGER*4 N,Label(1)
      REAL*4 A(1,1)
!
! local variables
!
      INTEGER*4 I4TEMP,Column,Row
!
! output the coefficients on non-zero fill-ins where the lower bound
! of "Row" is computed by equation (6.4)
!
      WRITE(*,'(" Row  Column  Coefficient")')
      WRITE(*,'(I4,I6,F9.3)') 1,1,A(1,1)
      DO I4TEMP=2,N
          Column=Label(I4TEMP)
          DO Row=Label(I4TEMP-1)-Column+I4TEMP, I4TEMP
               WRITE(*,'(I4,I6,F9.3)') Row,I4TEMP, A(Row,Column)
          END DO
      END DO
!
      RETURN
      END
```

Chapter 7. Dense and Symmetric Systems

7.1 Purpose

This chapter has subroutines for the solution of [A]{X}={B} where the left side matrix [A] is dense and symmetric. There is no consideration of definiteness of matrix [A]. The non-zero fill-ins in the lower triangular part of matrix [A] have a shape, for example, as:

$$\begin{bmatrix} * & & & & & & \\ * & * & & & \text{sym.} & & \\ * & * & * & & & & \\ * & * & * & * & & & \\ * & * & * & * & * & & \\ * & * & * & * & * & * & \\ * & * & * & * & * & * & * \end{bmatrix}$$

where the symbol * indicates non-zero fill-ins. Three types of subroutine are introduced in this chapter, which perform the following functions:

1. Decompose matrix [A] into the product of $[L][D][L]^T$ where matrix [L] is the lower triangular matrix and matrix [D] is the diagonal matrix.
2. Perform forward and backward substitutions.
3. Solve [A]{X}={B} in a single call.

Decomposition and substitution must be called in order, and work together as a pair. No pivoting is applied to the following subroutines:

Decompose_DSG_4
Decompose_DSG_8
Decompose_DSG_10
Decompose_DSG_16
Decompose_DSG_Z4
Decompose_DSG_Z8
Decompose_DSG_Z10
Decompose_DSG_Z16

Substitute_DSG_4
Substitute_DSG_8
Substitute_DSG_10
Substitute_DSG_16
Substitute_DSG_Z4
Substitute_DSG_Z8
Substitute_DSG_Z10

Substitute_DSG_Z16

Solution_DSG_4
Solution_DSG_8
Solution_DSG_10
Solution_DSG_16
Solution_DSG_Z4
Solution_DSG_Z8
Solution_DSG_Z10
Solution_DSG_Z16

7.2 Fortran Syntax for Subroutine Decompose

The following subroutines decompose matrix [A] into [A]= $[L][D][L]^T$. Syntax is as follows:

```
Decompose_DSG_4(A_io, N_i, Label_i, NoGood_o)
Decompose_DSG_8(A_io, N_i, Label_i, NoGood_o)
Decompose_DSG_10(A_io, N_i, Label_i, NoGood_o)
Decompose_DSG_16(A_io, N_i, Label_i, NoGood_o)
Decompose_DSG_Z4(A_io, N_i, Label_i, NoGood_o)
Decompose_DSG_Z8(A_io, N_i, Label_i, NoGood_o)
Decompose_DSG_Z10(A_io, N_i, Label_i, NoGood_o)
Decompose_DSG_Z16(A_io, N_i, Label_i, NoGood_o)
```

where

1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A] that inputs the original matrix and returns the result if the variable NoGood_o is false. For the definition of profile, please see section 7.5.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument Label_i, an INTEGER(4) array, is the address reference label. For the definition of address reference label, please see section 7.6.
4. The argument NoGood_o, a LOGICAL(4) variable, is a flag that indicates if the input matrix [A] is suitable for the subroutine. If NoGood_o=.True., the input matrix [A] cannot be decomposed and there is no output returned; otherwise the profile A_io returns the decomposed matrix [L]. For the situation where NoGood_o=.True., please see section 7.7.

7.3 Fortran Syntax for Subroutine Substitute

The following subroutines perform forward and backward substitutions. Syntax is as follows:

```
Substitute_DSG_4(A_i, N_i, Label_i, X_io)
Substitute_DSG_8(A_i, N_i, Label_i, X_io)
Substitute_DSG_10(A_i, N_i, Label_i, X_io)
Substitute_DSG_16(A_i, N_i, Label_i, X_io)
Substitute_DSG_Z4(A_i, N_i, Label_i, X_io)
Substitute_DSG_Z8(A_i, N_i, Label_i, X_io)
```

Substitute_DSG_Z10(A_i, N_i, Label_i, X_io)
Substitute_DSG_Z16(A_i, N_i, Label_i, X_io)

where

1. The argument A_i, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A] that inputs the result from decomposition.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument Label_i, an INTEGER(4) array, is the address reference label. For the definition of address reference label, please see section 7.6.
4. The argument X_io, array whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution.

7.4 Fortran Syntax for Subroutine Solution

The following subroutines first decompose matrix [A] into the product of $[L][D][L]^T$, and then perform forward and backward substitutions. Solve [A]{X}={B} in a single call. Syntax is as follows:

Solution_DSG_4(A_io, N_i, Label_i, X_io, NoGood_o)
Solution_DSG_8(A_io, N_i, Label_i, X_io, NoGood_o)
Solution_DSG_10(A_io, N_i, Label_i, X_io, NoGood_o)
Solution_DSG_16(A_io, N_i, Label_i, X_io, NoGood_o)
Solution_DSG_Z4(A_io, N_i, Label_i, X_io, NoGood_o)
Solution_DSG_Z8(A_io, N_i, Label_i, X_io, NoGood_o)
Solution_DSG_Z10(A_io, N_i, Label_i, X_io, NoGood_o)
Solution_DSG_Z16(A_io, N_i, Label_i, X_io, NoGood_o)

where

1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A], that inputs the original matrix and returns the decomposed result if the variable NoGood_o is false. For the definition of profile, please see section 7.5.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument Label_i, an INTEGER(4) array, is the address reference label. For the definition of address reference label, please see section 7.6.
4. The argument X_io, array whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution if NoGood_o is false.
5. The argument NoGood_o, a LOGICAL(4) variable, is a flag that indicates if the input system is suitable for the subroutine. If NoGood_o=.True., the input system cannot be solved by the subroutine and there is no output returned; otherwise the profile A_io returns the decomposed matrix [L], and vector X_io returns the solution. For the situation where NoGood_o=.True., please see section 7.7.

7.5 Profile

Profile for a dense and symmetric matrix is as:

$$
\begin{bmatrix}
* & & & & & & \\
* & * & & \text{sym.} & & & \\
* & * & * & & & & \\
* & * & * & * & & & \\
* & * & * & * & * & & \\
* & * & * & * & * & * & \\
* & * & * & * & * & * & *
\end{bmatrix}
\qquad (7.1)
$$

where the symbol * represents non-zero fill-ins. Total length of profile is determined as

$$\text{profile size} = ((N+1) * N) / 2 \qquad (7.2)$$

where N is the matrix order.

7.6 Data Storage Scheme

Data storage scheme for a dense and symmetric matrix must be declared in a Fortran program, for example:

```
REAL (4) :: A(1,1)
```

where variable A here is a single precision profile for matrix [A]. For other kinds of variable, profile must be properly declared. Then, replace the column index, for example j, with the address reference label, for example Label(J). The coefficient A_{ij} in the lower triangular part of matrix [A] is programmed in a Fortran program as A(I,Label(J)). The address reference labels are defined by the following algorithm where N is the order of matrix [A]:

(1) Set Label(1) = 1
(2) For i = 2 to N, do the following:
Label(i) = Label(i-1) + [number of non-zero fill-ins in the i-th column] (7.3)

For the example in form (7.1), the address reference labels are 1, 7, 12, 16, 19, 21, and 22. Equation (7.2) computes 28 non-zero fill-ins that may be checked from the form (7.1).

7.7 Failure of Calling Request

If a calling request fails, solving procedure meets a diagonal coefficient whose absolute value is very small and is negligible compared to unity.

The subroutines introduced in this chapter deal with symmetric systems without a consideration of pivoting. Failure of request does not mean that the input matrix is absolutely singular. A pivoting may continue execution. However, pivoting may destroy symmetry. A solver with a pivoting usually does not consider symmetry. If pivoting is necessary, try a dense solver with pivoting. A pivoting procedure always takes more time and is less efficient in parallel processing.

7.8 Fortran Example

For a given system [A]{X}={B}, the left side matrix [A] and the right side vector {B} are defined as follows:

$$\begin{bmatrix} 1 & & & & & & \\ 4 & 5 & & & \text{sym.} & & \\ 2 & 29 & 4 & & & & \\ 3 & 9 & 34 & 8 & & & \\ 12 & 23 & 3 & 23 & 45 & & \\ 4 & 2 & 22 & 11 & 7 & 2 & \\ 2 & 27 & 3 & 49 & 33 & 12 & 9 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 21 \\ 141 \\ 2 \\ 9 \\ 333 \\ 1 \\ 3 \end{bmatrix}$$

in which the order N=7. A Fortran program for decomposition and substitution is as follows. Subroutines “Input” and “Output” have data storage scheme. Subroutine “DenseLabel” based on equation (7.3) generates address reference labels. Subroutine “Decompose_DSG_4” decomposes matrix [A], and subroutine “Substitute_DSG_4” performs forward and backward substitutions.

```
!  *** Example program ***
!  define variables where the length of A is determined by equation (7.2)
!
      PARAMETER (N=7)
      REAL*4 A(((N+1)*N)/2),X(N)
      INTEGER*4 Label(N)
      LOGICAL*4 NoGood
      DATA X/21.0,141.0,2.0,9.0,333.0,1.0,3.0/
!
!  generate address reference labels
!
      CALL DenseLabel(Label,N)
!
!  input the lower triangular part of [A]
!
      CALL Input(A,Label)
!
!  decompose in parallel
!
      CALL Decompose_DSG_4(A,N,Label,NoGood)
!
!  stop if NoGood=.True.
!
      IF(NoGood) STOP 'Cannot be decomposed'
!
!  perform substitutions in parallel
!
      CALL Substitute_DSG_4(A,N,Label,X)
!
```

```
!  output decomposed matrix
!
      CALL Output(A,N,Label)
!
!  output the solution
!
      Write(*,'(" Solution is as:")')
      Write(*,*) X
!
!  laipe done
!
      call  laipeDone
!
      STOP
      END
      SUBROUTINE DenseLabel(Label,N)
!
!
!  routine to generate address reference labels for a dense lower triangular matrix
!  (A)FORTRAN CALL: CALL DenseLabel(Label,N)
!     1.Label: <I4> return the address reference labels, dimension(N)
!     2.N: <I4> order of matrix
!
!  dummy arguments
!
      INTEGER*4 Label(1),N
!
!  local variables
!
      INTEGER*4 I4TEMP,J4TEMP
!
!  generate address label
!
      I4TEMP=N-1
      Label(1)=1
      DO J4TEMP=2,N
            Label(J4TEMP)=Label(J4TEMP-1)+I4TEMP
            I4TEMP=I4TEMP-1
      END DO
!
      RETURN
      END
      SUBROUTINE Input(A,Label)
!
!
!  routine to demonstrate an application of data storage scheme
!  (A)FORTRAN CALL: CALL Input(A,Label)
!     1.A: <R4> profile of matrix [A], dimension(*)
!     2.Label: <I4> the address reference labels, dimension(N)
!
!  dummy arguments
```

```
!
      INTEGER*4 Label(1)
      REAL*4 A(1,1)
!
! input
!
      A(1,Label(1))= 1.0
      A(2,Label(1))= 4.0
      A(3,Label(1))= 2.0
      A(4,Label(1))= 3.0
      A(5,Label(1))=12.0
      A(6,Label(1))= 4.0
      A(7,Label(1))= 2.0
      A(2,Label(2))= 5.0
      A(3,Label(2))=29.0
      A(4,Label(2))= 9.0
      A(5,Label(2))=23.0
      A(6,Label(2))= 2.0
      A(7,Label(2))=27.0
      A(3,Label(3))= 4.0
      A(4,Label(3))=34.0
      A(5,Label(3))= 3.0
      A(6,Label(3))=22.0
      A(7,Label(3))= 3.0
      A(4,Label(4))= 8.0
      A(5,Label(4))=23.0
      A(6,Label(4))=11.0
      A(7,Label(4))=49.0
      A(5,Label(5))=45.0
      A(6,Label(5))= 7.0
      A(7,Label(5))=33.0
      A(6,Label(6))= 2.0
      A(7,Label(6))=12.0
      A(7,Label(7))= 9.0
!
      RETURN
      END
      SUBROUTINE Output(A,N,Label)
!
!
! routine to output the decomposed matrix by data storage scheme
! (A)FORTRAN CALL: CALL Output(A,N,Label)
!   1.A: <R4> profile of matrix [A], dimension(*)
!   2.N: <I4> order of matrix [A]
!   3.Label: <I4> address reference labels, dimension(N)
!
! dummy arguments
!
      INTEGER*4 N,Label(1)
      REAL*4 A(1,1)
!
```

```
!  local variables
!
       INTEGER*4 Column,Row,I4TEMP
!
!  output the coefficients on non-zero fill-ins
!
       WRITE(*,'(" Row  Column  Coefficient")')
       DO I4TEMP=1,N
              Column=Label(I4TEMP)
              DO Row=I4TEMP,N
                     WRITE(*,'(I4,I6,F9.3)') Row, I4TEMP, A(Row,Column)
             END DO
       END DO
!
       RETURN
       END
```

Chapter 8. Constant-Bandwidth and Asymmetric Systems

8.1 Purpose

This chapter has subroutines for the solution of [A]{X}={B} where the left side matrix [A] is of constant bandwidth and asymmetric. There is no consideration of definiteness of matrix [A]. The non-zero fill-ins of matrix [A] have a shape, for example, as:

$$\begin{bmatrix} = & + & + & & & & \\ * & = & + & + & & & \\ * & * & = & + & + & & \\ * & * & * & = & + & + & \\ & * & * & * & = & + & + \\ & & * & * & * & = & + \\ & & & * & * & * & = \end{bmatrix}$$

where the symbol "+" indicates non-zero fill-ins in the upper triangular part, and the symbol "=" indicates non-zero fill-ins on the diagonal, and the symbol "*" indicates non-zero fill-ins in the lower triangular part. Matrix [A] has an upper bandwidth and a lower bandwidth. In this example, the upper bandwidth is 2 and the lower bandwidth is 3.

Three types of subroutine are introduced in this chapter, which perform the following functions:

1. Decompose matrix [A] into the product of [L][U] where matrix [L] is the lower triangular matrix and matrix [U] is the upper triangular matrix.
2. Perform forward and backward substitutions.
3. Solve [A]{X}={B} in a single call.

Decomposition and substitution must be called in order, and work together as a pair. No pivoting is applied to the subroutines, which are as follows:

Decompose_CAG_4
Decompose_CAG_8
Decompose_CAG_10
Decompose_CAG_16
Decompose_CAG_Z4
Decompose_CAG_Z8
Decompose_CAG_Z10
Decompose_CAG_Z16

Substitute_CAG_4
Substitute_CAG_8
Substitute_CAG_10

Substitute_CAG_16
Substitute_CAG_Z4
Substitute_CAG_Z8
Substitute_CAG_Z10
Substitute_CAG_Z16

Solution_CAG_4
Solution_CAG_8
Solution_CAG_10
Solution_CAG_16
Solution_CAG_Z4
Solution_CAG_Z8
Solution_CAG_Z10
Solution_CAG_Z16

meSolution_CAG_4
meSolution_CAG_8
meSolution_CAG_10
meSolution_CAG_16
meSolution_CAG_Z4
meSolution_CAG_Z8
meSolution_CAG_Z10
meSolution_CAG_Z16

The subroutines with a prefix "me", i.e., meSolution_CAG_4, are multiple-entry direct solvers that are most well suitable for systems with a small bandwidth. For more detailed discussions on multiple-entry direct solvers, please see section 1.7.

8.2 Fortran Syntax for Subroutine Decompose

The following subroutines decompose matrix [A] into [A]=[L][U]. Syntax is as follows:

```
Decompose_CAG_4(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i, NoGood_o)
Decompose_CAG_8(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i, NoGood_o)
Decompose_CAG_10(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i, NoGood_o)
Decompose_CAG_16(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i, NoGood_o)
Decompose_CAG_Z4(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i, NoGood_o)
Decompose_CAG_Z8(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i, NoGood_o)
Decompose_CAG_Z10(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i, NoGood_o)
Decompose_CAG_Z16(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i, NoGood_o)
```

where

1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A] that inputs the original matrix and returns the result if the variable NoGood_o is false. For the definition of profile, please see section 8.6.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument UpperBandwidth_i, an INTEGER(4) variable, is the upper bandwidth of matrix [A]. The upper bandwidth is the maximal number of non-zero fill-ins on the right side of diagonal in a row.

4. The argument LowerBandwidth_i, an INTEGER(4) variable, is the lower bandwidth of matrix [A]. The lower bandwidth is the maximal number of non-zero fill-ins below the diagonal in a column.
5. The argument NoGood_o, a LOGICAL(4) variable, is a flag that indicates if the input matrix [A] is suitable for decomposition. If NoGood_o=.True., the input matrix [A] cannot be decomposed and there is no output returned; otherwise the profile A_io returns the decomposed matrices [L] and [U]. For the situation where NoGood_o=.True., please see section 8.8.

8.3 Fortran Syntax for Subroutine Substitute

The following subroutines perform forward and backward substitutions. Syntax is as follows:

```
Substitute_CAG_4(A_i, N_i, UpperBandwidth_i, LowerBandwidth_i, X_io)
Substitute_CAG_8(A_i, N_i, UpperBandwidth_i, LowerBandwidth_i, X_io)
Substitute_CAG_10(A_i, N_i, UpperBandwidth_i, LowerBandwidth_i, X_io)
Substitute_CAG_16(A_i, N_i, UpperBandwidth_i, LowerBandwidth_i, X_io)
Substitute_CAG_Z4(A_i, N_i, UpperBandwidth_i, LowerBandwidth_i, X_io)
Substitute_CAG_Z8(A_i, N_i, UpperBandwidth_i, LowerBandwidth_i, X_io)
Substitute_CAG_Z10(A_i, N_i, UpperBandwidth_i, LowerBandwidth_i, X_io)
Substitute_CAG_Z16(A_i, N_i, UpperBandwidth_i, LowerBandwidth_i, X_io)
```

where

1. The argument A_i, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A] that inputs the result from decomposition.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument UpperBandwidth_i, an INTEGER(4) variable, is the upper bandwidth of matrix [A]. The upper bandwidth is the maximal number of non-zero fill-ins on the right side of diagonal in a row.
4. The argument LowerBandwidth_i, an INTEGER(4) variable, is the lower bandwidth of matrix [A]. The lower bandwidth is the maximal number of non-zero fill-ins below the diagonal in a column.
5. The argument X_io, array whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution.

8.4 Fortran Syntax for Subroutine Solution

The following subroutines decompose matrix [A] into the product of [L][U], and perform forward and backward substitutions. Solve [A]{X}={B} in a single call. The syntax is as follows:

```
Solution_CAG_4(A_io,N_i,UpperBandwidth_i,LowerBandwidth_i,X_io,NoGood_o)
Solution_CAG_8(A_io,N_i,UpperBandwidth_i,LowerBandwidth_i,X_io,NoGood_o)
Solution_CAG_10(A_io,N_i,UpperBandwidth_i,LowerBandwidth_i,X_io,NoGood_o)
Solution_CAG_16(A_io,N_i,UpperBandwidth_i,LowerBandwidth_i,X_io,NoGood_o)
Solution_CAG_Z4(A_io,N_i,UpperBandwidth_i,LowerBandwidth_i,X_io,NoGood_o)
Solution_CAG_Z8(A_io,N_i,UpperBandwidth_i,LowerBandwidth_i,X_io,NoGood_o)
```

Solution_CAG_Z10(A_io,N_i,UpperBandwidth_i,LowerBandwidth_i,X_io,NoGood_o)
Solution_CAG_Z16(A_io,N_i,UpperBandwidth_i,LowerBandwidth_i,X_io,NoGood_o)

where

1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A], that inputs the original matrix and returns the decomposed result if the variable NoGood_o is false. For the definition of profile, please see section 8.6.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument UpperBandwidth_i, an INTEGER(4) variable, is the upper bandwidth of matrix [A]. The upper bandwidth is the maximal number of non-zero fill-ins on the right side of diagonal in a row.
4. The argument LowerBandwidth_i, an INTEGER(4) variable, is the lower bandwidth of matrix [A]. The lower bandwidth is the maximal number of non-zero fill-ins below the diagonal in a column.
5. The argument X_io, array whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution if NoGood_o is false.
6. The argument NoGood_o, a LOGICAL(4) variable, is a flag that indicates if the input system is suitable for the subroutine. If NoGood_o=.True., the input system cannot be solved by the subroutine and there is no output returned; otherwise the profile A_io returns the decomposed matrices [L] and [U], and vector X_io returns the solution. For the situation where NoGood_o=.True., please see section 8.8.

8.5 Fortran Syntax for Subroutine meSolution

The following subroutines solve [A][X]=[B] by a multiple entry procedure, where [X] and [B] may be a matrix with multiple vectors, i.e., [X]=[{ X_1 } { X_2 } ...] and [B]=[{ B_1 } { B_2 } ...]. This subroutine is more efficient if the upper and lower bandwidths are small. The syntax is as follows:

```
meSolution_CAG_4(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,   &
                 X_io, Nset_i, WorkingSpace_x, NoGood_o)
meSolution_CAG_8(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,   &
                 X_io, Nset_i, WorkingSpace_x, NoGood_o)
meSolution_CAG_10(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,   &
                  X_io, Nset_i, WorkingSpace_x, NoGood_o)
meSolution_CAG_16(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,   &
                  X_io, Nset_i, WorkingSpace_x, NoGood_o)
meSolution_CAG_Z4(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,   &
                  X_io, Nset_i, WorkingSpace_x, NoGood_o)
meSolution_CAG_Z8(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,   &
                  X_io, Nset_i, WorkingSpace_x, NoGood_o)
meSolution_CAG_Z10(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,   &
                   X_io, Nset_i, WorkingSpace_x, NoGood_o)
meSolution_CAG_Z16(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,   &
                   X_io,Nset_i, WorkingSpace_x, NoGood_o)
```

where

1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A] that inputs the original matrix. After returning from this subroutine, the content in array A_io is destroyed. For the definition of profile, please see section 8.6.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument UpperBandwidth_i, an INTEGER(4) variable, is the upper bandwidth of matrix [A]. The upper bandwidth is the maximal number of non-zero fill-ins on the right side of the diagonal.
4. The argument LowerBandwidth_i, an INTEGER(4) variable, is the lower bandwidth of matrix [A]. The lower bandwidth is the maximal number of non-zero fill-ins below the diagonal.
5. The argument X_io, array whose kind must be consistent with subroutine name convention, inputs the right side vector(s), and returns the solution if NoGood_O is false.
6. The argument Nset_i, an INTEGER(4) variable, is the number of right side vectors.
7. The argument WorkingSpace_x, array whose kind must be consistent with subroutine name convention and providing a space of (N_i*(UpperBandwidth_i+LowerBandwidth_i)) elements, is a working space.
8. The argument NoGood_o, a LOGICAL(4) variable, is a flag that indicates if the input matrix [A] is suitable for the subroutine. If NoGood_o=.True., the input system cannot be solved and there is no output; otherwise the vector X_io returns the solution. For the situation where NoGood_o=.True., please see section 8.8.

8.6 Profile

Profile for a constant bandwidth and asymmetric matrix is as:

```
   &
 [ &  &                ]
 [ *  *  *             ]
 [ *  *  *  *          ]
 [ *  *  *  *  *       ]
 [ *  *  *  *  *  *    ]            (8.1)
 [    *  *  *  *  *  * ]
 [       *  *  *  *  * ]
 [          *  *  *  * ]
               &  &
                  &
```

where the symbol * represents non-zero fill-ins and the symbol & indicates an extra memory space whose content is never used. Total length of profile is determined as

profile size = N * (UpperBandwidth + LowerBandwidth + 1) –LowerBandwidth (8.2)

where N is the matrix order, and LowerBandwidth is the lower bandwidth, and UpperBandwidth is the upper bandwidth.

8.7 Data Storage Scheme

Data storage scheme for a constant bandwidth and asymmetric matrix must be declared in a Fortran program, for example:

```
INTEGER (4) :: UpperBandwidth,LowerBandwidth
REAL (4) :: A(1-UpperBandwidth:LowerBandwidth,1)
```

where variable A, in this example, is a single precision profile for matrix [A]. For other kinds of variable, profile must be properly declared. Then, the coefficient A_{ij} of matrix [A] is programmed in a Fortran program as A(I,J), no matter A_{ij} is in the upper triangular part or in the lower triangular part.

The non-zero fill-ins in the i-th column are from the beginning index as:

$$\text{Maximum of } (1, i - \text{UpperBandwidth}) \tag{8.3}$$

to the ending index as:

$$\text{Minimum of } (N, i + \text{LowerBandwidth}) \tag{8.4}$$

where N is the order of matrix [A].

8. 8 Failure of Calling Request

If a calling request fails, solving procedure meets a diagonal coefficient whose absolute value is very small and is negligible compared to unity.

Since the subroutines introduced in this chapter do not consider pivoting, failure of request does not mean that the matrix is absolutely singular. A pivoting may continue execution. However, pivoting may take more time. If a pivoting is necessary, try a corresponding solver with partial pivoting.

8.9 Fortran Example

For a given system [A]{X}={B}, the left side matrix [A] and the right side vector {B} are defined as follows:

$$\begin{bmatrix} 1 & 2 & & & & & \\ 4 & 25 & 4 & & & & \\ 2 & 29 & 14 & 9 & & & \\ & 99 & 34 & 19 & 71 & & \\ & & 3 & 23 & 5 & 93 & \\ & & & 11 & 7 & 22 & 4 \\ & & & & 3 & 2 & 9 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 21 \\ 11 \\ 122 \\ 19 \\ 333 \\ 1 \\ 3 \end{bmatrix}$$

in which the order N=7, and the lower bandwidth LowerBandwidth=2, and the UpperBandwidth=1. A Fortran program for decomposition and substitution is as follows. Subroutines “Input” and “Output” have data storage scheme. Subroutine “Decompose_CAG_4” decomposes matrix [A], and subroutine “Substitute_CAG_4” performs forward and backward substitutions.

```
!  *** Example program ***
!  define variables where the length of A is determined by equation (8.2)
!
     PARAMETER (N=7)
     INTEGER*4  UpperBandwidth
     PARAMETER (UpperBandwidth=1)
     PARAMETER (LowerBandwidth=2)
     REAL*4 A(N*(UpperBandwidth+LowerBandwidth+1)- LowerBandwidth)
     REAL*4 X(N)
     LOGICAL*4 NoGood
     DATA X/21.0,11.0,122.0,19.0,333.0,1.0,3.0/
!
!  input the non-zero fill-ins of matrix [A]
!
     CALL Input(A,UpperBandwidth,LowerBandwidth)
!
!  decompose in parallel
!
     CALL Decompose_CAG_4(A,N,UpperBandwidth, LowerBandwidth, NoGood)
!
!  stop if NoGood=.True.
!
     IF(NoGood) STOP 'Cannot be decomposed'
!
!  perform substitutions in parallel
!
     CALL Substitute_CAG_4(A,N,UpperBandwidth, LowerBandwidth,X)
!
!  output decomposed matrix
!
     CALL Output(A,N,UpperBandwidth,LowerBandwidth)
!
!  output the solution
!
     Write(*,'(" Solution is as:")')
     Write(*,*) X
!
!  laipe done
!
     call laipeDone
!
     STOP
```

```
      END
      SUBROUTINE Input(A,UpperBandwidth,LowerBandwidth)
!
!
! routine to demonstrate an application of data storage scheme
! (A)FORTRAN CALL: CALL Input(A,UpperBandwidth,LowerBandwidth)
!   1.A: <R4> profile of matrix [A], dimension(*)
!   2.UpperBandwidth: <I4> upper bandwidth
!   3.LowerBandwidth: <I4> lower bandwidth
!
! dummy arguments
!
      INTEGER*4 UpperBandwidth,LowerBandwidth
      REAL*4 A(1-UpperBandwidth:LowerBandwidth,1)
!
! input
!
      A(1,1)= 1.0
      A(2,1)= 4.0
      A(3,1)= 2.0
      A(1,2)= 2.0
      A(2,2)=25.0
      A(3,2)=29.0
      A(4,2)=99.0
      A(2,3)= 4.0
      A(3,3)=14.0
      A(4,3)=34.0
      A(5,3)= 3.0
      A(3,4)= 9.0
      A(4,4)=19.0
      A(5,4)=23.0
      A(6,4)=11.0
      A(4,5)=71.0
      A(5,5)= 5.0
      A(6,5)= 7.0
      A(7,5)= 3.0
      A(5,6)=93.0
      A(6,6)=22.0
      A(7,6)= 2.0
      A(6,7)= 4.0
      A(7,7)= 9.0
!
      RETURN
      END

      SUBROUTINE Output(A,N,UpperBandwidth, LowerBandwidth)
!
!
! routine to output the decomposed matrix by data storage scheme
! (A)FORTRAN CALL: CALL Output(A,N,UpperBandwidth,LowerBandwidth)
!   1.A: <R4> profile of matrix [A], dimension(*)
```

```
!    2.N: <I4> order of matrix [A]
!    3.UpperBandwidth: <I4> upper bandwidth
!    4.LowerBandwidth: <I4> lower bandwidth
!
!  dummy arguments
!
      INTEGER*4 N,UpperBandwidth,LowerBandwidth
      REAL*4 A(1-UpperBandwidth:LowerBandwidth,1)
!
!  local variables
!
      INTEGER*4 Column,Row
!
!  output the coefficients on non-zero fill-ins. The beginning and ending row indices for each
!  column are defined in equation (8.3) and equation (8.4)
!
      WRITE(*,'(" Row  Column  Coefficient")')
      DO Column=1,N
          DO Row=MAX0(1,Column-UpperBandwidth), MIN0(N,Column+LowerBandwidth)
              WRITE(*,'(I4,I6,F9.3)') Row, Column, A(Row,Column)
          END DO
      END DO
!
      RETURN
      END
```

Chapter 9. Variable-Bandwidth and Asymmetric Systems

9.1 Purpose

This chapter has subroutines for the solution of [A]{X}={B} where the left side matrix [A] is of variable bandwidth and asymmetric. There is no consideration of definiteness of matrix [A]. The non-zero fill-ins in the left side matrix [A] have a shape, for example, as:

$$\begin{bmatrix} * & * & & & & & \\ * & * & * & & * & & \\ * & * & * & * & * & & \\ & * & * & * & * & & * \\ & & * & * & * & * & * \\ & & & * & * & * & * \\ & & & * & & * & * \end{bmatrix}$$

Three types of subroutine are introduced in the chapter, which perform the following functions:

1. Decompose matrix [A] into the product of [L][U] where matrix [L] is the lower triangular matrix and matrix [U] is the upper triangular matrix.
2. Perform forward and backward substitutions.
3. Solve [A]{X}={B} in a single call.

Decomposition and substitution must be called in order, and work together as a pair. No pivoting is applied to the subroutines, which are as:

Decompose_VAG_4
Decompose_VAG_8
Decompose_VAG_10
Decompose_VAG_16
Decompose_VAG_Z4
Decompose_VAG_Z8
Decompose_VAG_Z10
Decompose_VAG_Z16

Substitute_VAG_4
Substitute_VAG_8
Substitute_VAG_10
Substitute_VAG_16
Substitute_VAG_Z4
Substitute_VAG_Z8
Substitute_VAG_Z10
Substitute_VAG_Z16

Solution_VAG_4
Solution_VAG_8
Solution_VAG_10
Solution_VAG_16
Solution_VAG_Z4
Solution_VAG_Z8
Solution_VAG_Z10
Solution_VAG_Z16

9.2 Fortran Syntax for Subroutine Decompose

The following subroutines decompose matrix [A] into [A]=[L][U]. Syntax is as follows:

```
Decompose_VAG_4(A_io, N_i, Label_i, Last_i, NoGood_o)
Decompose_VAG_8(A_io, N_i, Label_i, Last_i, NoGood_o)
Decompose_VAG_10(A_io, N_i, Label_i, Last_i, NoGood_o)
Decompose_VAG_16(A_io, N_i, Label_i, Last_i, NoGood_o)
Decompose_VAG_Z4(A_io, N_i, Label_i, Last_i, NoGood_o)
Decompose_VAG_Z8(A_io, N_i, Label_i, Last_i, NoGood_o)
Decompose_VAG_Z10(A_io, N_i, Label_i, Last_i, NoGood_o)
Decompose_VAG_Z16(A_io, N_i, Label_i, Last_i, NoGood_o)
```

where

1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A] that inputs the original matrix and returns the result if the variable NoGood_o is false. For the definition of profile, please see section 9.5.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument Label_i, an INTEGER(4) array, is the address reference label. For the definition of address reference label, please see section 9.6.
4. The argument Last_i, an INTEGER(4) array, is the last entry to each column in the profile. For the definition of the last entry, please see section 9.6.
5. The argument NoGood_o, a LOGICAL(4) variable, is a flag that indicates if the input matrix [A] is suitable for decomposition. If NoGood_o=.True., the input matrix [A] cannot be decomposed and there is no output returned; otherwise the profile A_io returns the decomposed matrices [L] and [U]. For the situation where NoGood_o=.True., please see section 9.7.

9.3 Fortran Syntax for Subroutine Substitute

The following subroutines perform forward and backward substitutions. Syntax is as follows:

```
Substitute_VAG_4(A_i, N_i, Label_i, Last_i, X_io)
Substitute_VAG_8(A_i, N_i, Label_i, Last_i, X_io)
Substitute_VAG_10(A_i, N_i, Label_i, Last_i, X_io)
Substitute_VAG_16(A_i, N_i, Label_i, Last_i, X_io)
Substitute_VAG_Z4(A_i, N_i, Label_i, Last_i, X_io)
```

Substitute_VAG_Z8(A_i, N_i, Label_i, Last_i, X_io)
Substitute_VAG_Z10(A_i, N_i, Label_i, Last_i, X_io)
Substitute_VAG_Z16(A_i, N_i, Label_i, Last_i, X_io)

where

1. The argument A_i, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A] that inputs the result from decomposition.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument Label_i, an INTEGER(4) array, is the address reference label. For the definition of address reference label, please see section 9.6.
4. The argument Last_i, an INTEGER(4) array, is the last entry of each column. For the definition of the last entry, please see section 9.6.
5. The argument X_io, array whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution.

9.4 Fortran Syntax for Subroutine Solution

The following subroutines first decompose matrix [A] into the product of [L][U], and then perform forward and backward substitutions. Solve the system [A]{X}={B} in a single call. Syntax is as follows:

Solution_VAG_4(A_io, N_i, Label_i, Last_i, X_io, NoGood_o)
Solution_VAG_8(A_io, N_i, Label_i, Last_i, X_io, NoGood_o)
Solution_VAG_10(A_io, N_i, Label_i, Last_i, X_io, NoGood_o)
Solution_VAG_16(A_io, N_i, Label_i, Last_i, X_io, NoGood_o)
Solution_VAG_Z4(A_io, N_i, Label_i, Last_i, X_io, NoGood_o)
Solution_VAG_Z8(A_io, N_i, Label_i, Last_i, X_io, NoGood_o)
Solution_VAG_Z10(A_io, N_i, Label_i, Last_i, X_io, NoGood_o)
Solution_VAG_Z16(A_io, N_i, Label_i, Last_i, X_io, NoGood_o)

where

1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A], that inputs the original matrix and returns the decomposed result if the variable NoGood_o is false. For the definition of profile, please see section 9.5.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument Label_i, an INTEGER(4) array, is the address reference label. For the definition of address reference label, please see section 9.6.
4. The argument Last_i, an INTEGER(4) array, is the last entry of column. For the definition of the last entry, please see section 9.6.
5. The argument X_io, array whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution if NoGood_o is false.
6. The argument NoGood_o, a LOGICAL(4) variable, is a flag that indicates if the input system is suitable for the subroutine. If NoGood_o=.True., the input system cannot be solved and there is no output returned; otherwise the profile A_io returns the decomposed matrices [L] and [U], and vector X_io returns the solution. For the situation where NoGood_o=.True., please see section 9.7.

9.5 Profile

Profile for variable bandwidth and asymmetric matrix is more complex than the other ones discussed in the previous chapters, and requires some extra memory spaces in the lower triangular part. The profile for the upper triangular part simply hinges on the non-zero fill-ins. Before discussing profile for the lower triangular part of matrix [A], let us examine two variables, *Beginning(I)* and *Ending(I)*. *Beginning(I)* is the row index of the first non-zero fill-in in the i-th column and *Ending(I)* is the row index of the last non-zero fill-in in the i-th column. Then, the last entry, denoted by *Last*, is defined as:

1 Set Last(1) = Ending(1)
2. For I = 2 to N, do the following
   Last(I) = Maximum of (Last(I-1),Ending(I)) (9.1)

The *Beginning* and *Last* indices define the profile of an asymmetric and variable bandwidth matrix. The address reference label is then defined as:

1. Set Label(1) = 1
2. For I = 2 to N, do the following
   Label(I) = Label(I-1) + Last(I-1) -Beginning(I) + 1 (9.2)

The required length of profile is written as:

profile size = Label(N)-1+ N (9.3)

where N is the matrix order, and Label(N) is the address reference label for the N-th column. For example, if a sparse matrix is written as follows.

$$\begin{bmatrix} * & * & & & & & \\ * & * & * & & * & & \\ * & * & * & * & * & & \\ & * & * & * & * & & * \\ & & * & * & * & * & * \\ & & * & * & * & * & * \\ & & * & & & * & * \end{bmatrix} \qquad (9.4)$$

where the symbol * represents a non-zero fill-in. Then, the beginning indices are 1, 1, 2, 3, 2, 5, and 4, and the ending indices are 3, 4, 7, 6, 6, 7, and 7. Then, the last entries determined by equation (9.1) are 3, 4, 7, 7, 7, 7, and 7. The beginning and last indices define the profile which may be written as

$$\begin{bmatrix} = & = & & & & & \\ = & = & = & & = & & \\ = & = & = & = & = & & \\ & = & = & = & = & & = \\ & & = & = & = & = & = \\ & & = & = & = & = & = \\ & & = & = & = & = & = \end{bmatrix} \qquad (9.5)$$

where the symbol = indicates an entry to the profile. The address reference labels are 1, 4, 7, 12, 18, 21, and 25. Equation (9.3) computes that the profile size is 31, which may be checked from the form (9.5).

For a variable-bandwidth and asymmetric matrix, the profile size is usually greater than the number of non-zero fill-ins. Comparing form (9.4) with form (9.5) finds that the profile has two more elements, A(7,4) and A(7,5). It must initialize the extra memory space in the profile, i.e., A(7,4)=0 and A(7,5)=0, before calling any of the following subroutines:

Decompose_VAG_4
Decompose_VAG_8
Decompose_VAG_10
Decompose_VAG_16
Decompose_VAG_Z4
Decompose_VAG_Z8
Decompose_VAG_Z10
Decompose_VAG_Z16

Solution_VAG_4
Solution_VAG_8
Solution_VAG_10
Solution_VAG_16
Solution_VAG_Z4
Solution_VAG_Z8
Solution_VAG_Z10
Solution_VAG_Z16

9.6 Data Storage Scheme

Data storage scheme for a variable-bandwidth and asymmetric matrix must be declared in a Fortran program, for example:

```
REAL (4) :: A(1,1)
```

where variable A, in this example, is a single precision profile for matrix [A]. For other kinds of variable, profile must be properly declared. Then, replace the column index, for example j, with the address reference label, for example Label(J). The coefficient A_{ij} of matrix [A] is programmed in a Fortran program as A(I,Label(J)).

The previous section introduces the *beginning* and *ending* indices, the address reference label, and the last entry for a profile. In practical calling convention, only the address reference label and the *last* entry are required. The *address reference label* and *last entry* then determine the *beginning index*. In the i-th column, from equation (9.2) the *beginning index* is determined as:

$$\text{Label(I-1) + Last(I-1) - Label(I) + 1} \qquad (9.6)$$

9.7 Failure of Calling Request

If a calling request fails, solving procedure meets a diagonal coefficient whose absolute value is very small and is negligible compared to unity.

Since the subroutines introduced in this chapter do not consider pivoting, failure of request does not mean that the input matrix is absolutely singular. A pivoting may continue execution. However, a pivoting may destroy sparsity. If a pivoting is necessary, try a constant bandwidth solver with partial pivoting or a dense solver with pivoting.

9.8 Fortran Example

For a given system [A]{X}={B}, the left side matrix [A] and the right side vector {B} are defined as follows:

$$\begin{bmatrix} 1 & 4 & & & & & \\ 5 & 25 & 29 & & 32 & & \\ 9 & 13 & 1 & 34 & 17 & & \\ & 4 & 5 & 9 & 23 & & 9 \\ & & 7 & 3 & 8 & 37 & 3 \\ & & 2 & 22 & 6 & 2 & 2 \\ & & 11 & & & 1 & 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 5 \\ 41 \\ 12 \\ 9 \\ 303 \\ 21 \\ 23 \end{bmatrix}$$

in which the order N=7. A Fortran program for decomposition and substitution is as follows. Subroutines “Input” and “Output” have data storage scheme. Subroutine “Decompose_VAG_4” decomposes matrix [A], and subroutine “Substitute_VAG_4” performs forward and backward substitutions.

```
! *** Example program ***
! define variables where the length of A is determined by equation (9.3),
! Equation (9.1), and the address reference define the last entry
! label is defined by equation(9.2)
!
      PARAMETER (N=7)
      REAL*4 A(31),X(N)
      INTEGER*4 Label(N),Last(N)
      LOGICAL*4 NoGood
      DATA X/5.0,41.0,12.0,9.0,303.0,21.0,23.0/
      DATA Label/1,4,7,12,18,21,25/
      DATA Last/3,4,7,7,7,7,7/
!
! input matrix [A]
!
      CALL Input(A,Label,Last,N)
!
! decompose in parallel
!
      CALL Decompose_VAG_4(A,N,Label,Last,NoGood)
!
```

```
!  stop if NoGood=.True.
!
      IF(NoGood) STOP 'Cannot be decomposed'
!
!  perform substitutions in parallel
!
      CALL Substitute_VAG_4(A,N,Label,Last,X)
!
!  output decomposed matrix
!
      CALL Output(A,N,Label,Last)
!
!  output the solution
!
      Write(*,'(" Solution is as:")')
      Write(*,*) X
!
!  laipe done
!
      call laipeDone
!
      STOP
      END
      SUBROUTINE Input(A,Label,Last,N)
!
!
!  routine to demonstrate an application of data storage scheme
!  (A)FORTRAN CALL: CALL Input(A,Label,Last,N)
!     1.A: <R4> profile of matrix [A], dimension(*)
!     2.Label: <I4> address reference labels, dimension(*)
!     3.Last: <I4> the last entry to each column, dimension(*)
!     4.N: <I4> order of matrix [A]
!
!  dummy arguments
!
      INTEGER*4 Label(1),Last(1),N
      REAL*4 A(1,1)
!
!  local variable
!
      INTEGER*4 I4TEMP
!
!  initialization where the length of profile is determined by equation (9.3)
!
      DO I4TEMP=1,Label(N)-1+N
         A(I4TEMP,1)=0.0
      END DO
!
!  input
!
      A(1,Label(1))= 1.0
```

```
      A(2,Label(1))= 5.0
      A(3,Label(1))= 9.0
      A(1,Label(2))= 4.0
      A(2,Label(2))=25.0
      A(3,Label(2))=13.0
      A(4,Label(2))= 4.0
      A(2,Label(3))=29.0
      A(3,Label(3))= 1.0
      A(4,Label(3))= 5.0
      A(5,Label(3))= 7.0
      A(6,Label(3))= 2.0
      A(7,Label(3))=11.0
      A(3,Label(4))=34.0
      A(4,Label(4))= 9.0
      A(5,Label(4))= 3.0
      A(6,Label(4))=22.0
      A(2,Label(5))=32.0
      A(3,Label(5))=17.0
      A(4,Label(5))=23.0
      A(5,Label(5))= 8.0
      A(6,Label(5))= 6.0
      A(5,Label(6))=37.0
      A(6,Label(6))= 2.0
      A(7,Label(6))= 1.0
      A(4,Label(7))= 9.0
      A(5,Label(7))= 3.0
      A(6,Label(7))= 2.0
      A(7,Label(7))= 1.0
!
      RETURN
      END
      SUBROUTINE Output(A,N,Label,Last)
!
!
! routine to output the decomposed matrix by data storage scheme
! (A)FORTRAN CALL: CALL Output(A,N,Label,Last)
!   1.A: <R4> profile of matrix [A], dimension(*)
!   2.N: <I4> order of matrix [A]
!   3.Label: <I4> address reference labels, dimension(*)
!   4.Last: <I4> the last entry to each column, dimension(*)
!
! dummy arguments
!
      INTEGER*4 N,Label(1),Last(1)
      REAL*4 A(1,1)
!
! local variables
!
      INTEGER*4 I4TEMP,Column,Row
!
! output the coefficients on non-zero fill-ins where the beginning index is
```

```
!  computed by equation (9.6)
!
      WRITE(*,'(" Row  Column  Coefficient")')
      DO I4TEMP=1,N
            Column=Label(I4TEMP)
            DO Row=Label(I4TEMP-1)+Last(I4TEMP-1)- Column+1, Last(I4TEMP)
                  WRITE(*,'(I4,I6,F9.3)') Row,I4TEMP, A(Row,Column)
            END DO
      END DO
!
      RETURN
      END
```

Chapter 10. Dense and Asymmetric Systems

10.1 Purpose

This chapter has subroutines for the solution of [A]{X}={B} where the left side matrix [A] is dense and asymmetric. There is no consideration of definiteness of matrix [A]. The non-zero fill-ins of matrix [A] have a simple shape, for example, as:

$$\begin{bmatrix} * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \end{bmatrix}$$

where the symbol * indicates non-zero fill-ins. Three types of subroutine are introduced in the chapter, which perform the following functions:

1. Decompose matrix [A] into the product of [L][U] where matrix [L] is the lower triangular matrix and matrix [U] is the upper triangular matrix.
2. Perform forward and backward substitutions.
3. Solve [A]{X}={B} in a single call.

Decomposition and substitution must be called in order, and work together as a pair. No pivoting is applied to the subroutines introduced in this chapter. The subroutines are as follows:

Decompose_DAG_4
Decompose_DAG_8
Decompose_DAG_10
Decompose_DAG_16
Decompose_DAG_Z4
Decompose_DAG_Z8
Decompose_DAG_Z10
Decompose_DAG_Z16

Substitute_DAG_4
Substitute_DAG_8
Substitute_DAG_10
Substitute_DAG_16
Substitute_DAG_Z4
Substitute_DAG_Z8
Substitute_DAG_Z10
Substitute_DAG_Z16

Solution_DAG_4
Solution_DAG_8
Solution_DAG_10
Solution_DAG_16
Solution_DAG_Z4
Solution_DAG_Z8
Solution_DAG_Z10
Solution_DAG_Z16

10.2 Fortran Syntax for Subroutine Decompose

The following subroutines decompose matrix [A] into [A]=[L][U]. Syntax is as follows:

```
Decompose_DAG_4(A_io, N_i, NoGood_o)
Decompose_DAG_8(A_io, N_i, NoGood_o)
Decompose_DAG_10(A_io, N_i, NoGood_o)
Decompose_DAG_16(A_io, N_i, NoGood_o)
Decompose_DAG_Z4(A_io, N_i, NoGood_o)
Decompose_DAG_Z8(A_io, N_i, NoGood_o)
Decompose_DAG_Z10(A_io, N_i, NoGood_o)
Decompose_DAG_Z16(A_io, N_i, NoGood_o)
```

where

1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A] that inputs the original matrix and returns the result if the variable NoGood_o is false. For the definition of profile, please see section 10.5.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument NoGood_o, a LOGICAL(4) variable, is a flag that indicates if the input matrix [A] is suitable for the subroutine. If NoGood_o=.True., the input matrix [A] cannot be decomposed and there is no output returned; otherwise the profile A_io returns the decomposed matrices [L] and [U]. For the situation where NoGood_o=.True., please see section 10.7.

10.3 Fortran Syntax for Subroutine Substitute

The following subroutines perform forward and backward substitutions. Syntax is as follows:

```
Substitute_DAG_4(A_i, N_i, X_io)
Substitute_DAG_8(A_i, N_i, X_io)
Substitute_DAG_10(A_i, N_i, X_io)
Substitute_DAG_16(A_i, N_i, X_io)
Substitute_DAG_Z4(A_i, N_i, X_io)
Substitute_DAG_Z8(A_i, N_i, X_io)
Substitute_DAG_Z10(A_i, N_i, X_io)
Substitute_DAG_Z16(A_i, N_i, X_io)
```

where

1. The argument A_i, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A] that inputs the result from decomposition.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument X_io, array whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution.

10.4 Fortran Syntax for Subroutine Solution

The following subroutines first decompose matrix [A] into the product of [L][U], and then perform forward and backward substitutions. Solve [A]{X}={B} in a single call. The syntax is as follows:

```
Solution_DAG_4(A_io, N_i, X_io, NoGood_o)
Solution_DAG_8(A_io, N_i, X_io, NoGood_o)
Solution_DAG_10(A_io, N_i, X_io, NoGood_o)
Solution_DAG_16(A_io, N_i, X_io, NoGood_o)
Solution_DAG_Z4(A_io, N_i, X_io, NoGood_o)
Solution_DAG_Z8(A_io, N_i, X_io, NoGood_o)
Solution_DAG_Z10(A_io, N_i, X_io, NoGood_o)
Solution_DAG_Z16(A_io, N_i, X_io, NoGood_o)
```

where

1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A], that inputs the original matrix and returns the decomposed result if the variable NoGood_o is false. For the definition of profile, please see section 10.5.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument X_io, array whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution if NoGood_o is false.
4. The argument NoGood_o, a LOGICAL(4) variable, is a flag that indicates if the input system is suitable for the subroutine. If NoGood_o=.True., the input system cannot be solved by the subroutine and there is no output returned; otherwise the profile A_io returns the decomposed matrices [L] and [U], and vector X_io returns the solution. For the situation where NoGood_o=.True., please see section 10.7.

10.5 Profile

Profile for a dense and asymmetric matrix is the simplest as:

$$\begin{bmatrix} * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \end{bmatrix} \tag{10.1}$$

where the symbol · represents non-zero fill-ins. Total length of profile is determined as

$$\text{profile size} = N * N \tag{10.2}$$

where N is the matrix order.

10.6 Data Storage Scheme

Data storage scheme for a dense and asymmetric matrix must be declared in a Fortran program, for example:

```
REAL (4) :: A(N,N)
```

where variable A here is a single precision profile for matrix [A], and N is the matrix order. For other kinds of variable, profile must be properly declared. Then, the coefficient A_{ij} of matrix [A] is simply programmed in a Fortran program as A(I,J).

10.7 Failure of Calling Request

If a calling request fails, solving procedure meets a diagonal coefficient whose absolute value is very small and is negligible compared to unity.

Since the subroutines introduced in this chapter do not consider pivoting, failure of request does not mean that the input matrix is absolutely singular. A pivoting may continue execution. However, pivoting always takes more time. If a pivoting is necessary, try a dense solver with partial or full pivoting.

10.8 Fortran Example

For a given system [A]{X}={B}, the left side matrix [A] and the right side vector {B} are defined as follows:

$$\begin{bmatrix} 1 & 2 & 13 & 17 & 32 & 47 & 6 \\ 4 & 5 & 3 & 5 & 0 & 0 & 6 \\ 2 & 29 & 4 & 7 & 11 & 5 & 4 \\ 3 & 9 & 34 & 8 & 33 & 14 & 3 \\ 12 & 23 & 3 & 23 & 45 & -1 & 2 \\ 4 & 2 & 22 & 11 & 7 & 2 & 1 \\ 2 & 27 & 3 & 49 & 33 & 12 & 9 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 21 \\ 141 \\ 2 \\ 9 \\ 333 \\ 1 \\ 3 \end{bmatrix}$$

in which the order N=7. A Fortran program for decomposition and substitution is as follows. Subroutines “Input” and “Output” have data storage scheme. Subroutine “Decompose_DAG_4” decomposes matrix [A], and subroutine “Substitute_DAG_4” performs forward and backward substitutions.

```
! *** Example program ***
! define variables where the length of A is determined by equation (10.2)
!
    PARAMETER (N=7)
    REAL*4 A(N,N),X(N)
    LOGICAL*4 NoGood
    DATA X/21.0,141.0,2.0,9.0,333.0,1.0,3.0/
!
! input matrix [A]
!
    CALL Input(A,N)
!
! decompose in parallel
!
    CALL Decompose_DAG_4(A,N,NoGood)
!
! stop if NoGood=.True.
!
    IF(NoGood) STOP 'Cannot be decomposed'
!
! perform substitutions in parallel
!
    CALL Substitute_DAG_4(A,N,X)
!
! output decomposed matrix
!
    CALL Output(A,N)
!
! output the solution
!
    Write(*,'(" Solution is as:")')
    Write(*,*) X
!
! laipe done
!
    call laipeDone
!
    STOP
    END
    SUBROUTINE Input(A,N)
!
!
! routine to demonstrate an application of data storage scheme
! (A)FORTRAN CALL: CALL Input(A,N)
!   1.A: <R4> profile of matrix [A], dimension(N,N)
!   2.N: <I4> the order of matrix [A]
!
! dummy arguments
!
    INTEGER*4 N
```

```
      REAL*4 A(N,N)
!
! first column
!
      A(1,1)= 1.0
      A(2,1)= 4.0
      A(3,1)= 2.0
      A(4,1)= 3.0
      A(5,1)=12.0
      A(6,1)= 4.0
      A(7,1)= 2.0
!
! second column
!
      A(1,2)= 2.0
      A(2,2)= 5.0
      A(3,2)=29.0
      A(4,2)= 9.0
      A(5,2)=23.0
      A(6,2)= 2.0
      A(7,2)=27.0
!
! third column
!
      A(1,3)=13.0
      A(2,3)= 3.0
      A(3,3)= 4.0
      A(4,3)=34.0
      A(5,3)= 3.0
      A(6,3)=22.0
      A(7,3)= 3.0
!
! fourth column
!
      A(1,4)=17.0
      A(2,4)= 5.0
      A(3,4)= 7.0
      A(4,4)= 8.0
      A(5,4)=23.0
      A(6,4)=11.0
      A(7,4)=49.0
!
! fifth column
!
      A(1,5)=32.0
      A(2,5)= 0.0
      A(3,5)=11.0
      A(4,5)=33.0
      A(5,5)=45.0
      A(6,5)= 7.0
      A(7,5)=33.0
```

```
!
! sixth column
!
      A(1,6)=47.0
      A(2,6)= 0.0
      A(3,6)= 5.0
      A(4,6)=14.0
      A(5,6)=-1.0
      A(6,6)= 2.0
      A(7,6)=12.0
!
! seventh column
!
      A(1,7)= 6.0
      A(2,7)= 6.0
      A(3,7)= 4.0
      A(4,7)= 3.0
      A(5,7)= 2.0
      A(6,7)= 1.0
      A(7,7)= 9.0
!
      RETURN
      END
      SUBROUTINE Output(A,N)
!
!
! routine to output the decomposed matrix by data storage scheme
! (A)FORTRAN CALL: CALL Output(A,N)
!    1.A: <R4> profile of matrix [A], dimension(*)
!    2.N: <I4> order of matrix [A]
!
! dummy arguments
!
      INTEGER*4 N
      REAL*4 A(N,N)
!
! local variables
!
      INTEGER*4 Column,Row
!
! output the coefficients on non-zero fill-ins
!
      WRITE(*,'(" Row  Column  Coefficient")')
      DO Column=1,N
         DO Row=1,N
            WRITE(*,'(I4,I6,F9.3)') Row,Column, A(Row,Column)
         END DO
      END DO
!
      RETURN
      END
```

Chapter 11. Constant-Bandwidth and Asymmetric Solvers with Partial Pivoting

11.1 Purpose

This chapter has subroutines for the solution of [A]{X}={B} with partial pivoting where the left side matrix [A] has a constant bandwidth and is asymmetric. There is no consideration of definiteness of matrix [A]. The non-zero fill-ins of matrix [A] have a shape, for example, as:

$$\begin{bmatrix} = & + & + & & & & \\ * & = & + & + & & & \\ * & * & = & + & + & & \\ * & * & * & = & + & + & \\ & * & * & * & = & + & + \\ & & * & * & * & = & + \\ & & & * & * & * & = \end{bmatrix}$$

where the symbol "+" indicates non-zero fill-ins in the upper triangular part, and the symbol "=" indicates non-zero fill-ins on the diagonal, and the symbol "*" indicates non-zero fill-ins in the lower triangular part. Matrix [A] has an upper bandwidth and a lower bandwidth. In the above example, the upper bandwidth is two and the lower bandwidth is three.

Three types of subroutine are introduced in this chapter, which perform the following functions:

1. Decompose matrix [A] into the product of [L][U] where matrix [L] is the lower triangular matrix and matrix [U] is the upper triangular matrix.
2. Perform forward and backward substitutions.
3. Solve [A]{X}={B} in a single call.

Decomposition and substitution must be called in order, and work together as a pair. The subroutines are as:

ppDecompose_CAG_4
ppDecompose_CAG_8
ppDecompose_CAG_10
ppDecompose_CAG_16
ppDecompose_CAG_Z4
ppDecompose_CAG_Z8
ppDecompose_CAG_Z10
ppDecompose_CAG_Z16

ppSubstitute_CAG_4
ppSubstitute_CAG_8

ppSubstitute_CAG_10
ppSubstitute_CAG_16
ppSubstitute_CAG_Z4
ppSubstitute_CAG_Z8
ppSubstitute_CAG_Z10
ppSubstitute_CAG_Z16

ppSolution_CAG_4
ppSolution_CAG_8
ppSolution_CAG_10
ppSolution_CAG_16
ppSolution_CAG_Z4
ppSolution_CAG_Z8
ppSolution_CAG_Z10
ppSolution_CAG_Z16

11.2 Fortran Syntax for Subroutine ppDecompose

The following subroutines decompose matrix [A] into [A]=[L][U] with partial pivoting. Syntax is as follows:

```
ppDecompose_CAG_4(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,    &
                  From_o, First_o, NoGood_o)
ppDecompose_CAG_8(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,    &
                  From_o, First_o, NoGood_o)
ppDecompose_CAG_10(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,    &
                  From_o, First_o, NoGood_o)
ppDecompose_CAG_16(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,    &
                  From_o, First_o, NoGood_o)
ppDecompose_CAG_Z4(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,    &
                  From_o, First_o, NoGood_o)
ppDecompose_CAG_Z8(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,    &
                  From_o, First_o, NoGood_o)
ppDecompose_CAG_Z10(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,    &
                  From_o, First_o, NoGood_o)
ppDecompose_CAG_Z16(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,    &
                  From_o, First_o, NoGood_o)
```

where

1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A] that inputs the original matrix and returns the result if the variable NoGood_o is false. For the definition of profile, please see section 11.5.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument UpperBandwidth_i, an INTEGER(4) variable, is the upper bandwidth of matrix [A]. The upper bandwidth is the maximal number of non-zero fill-ins on the right side of diagonal in a row.
4. The argument LowerBandwidth_i, an INTEGER(4) variable, is the lower bandwidth of matrix [A]. The lower bandwidth is the maximal number of non-zero fill-ins below the diagonal in a column.

5. The argument From_o, an INTEGER(4) array having N_i elements, returns the row index where the remaining elements in a row are from if NoGood_o is false.
6. The argument First_o, an INTEGER(4) array having N_i elements, returns the index of the first non-zero fill-in on each column if NoGood_o is false.
7. The argument NoGood_o, a LOGICAL(4) variable, is a flag that indicates if the input matrix [A] is suitable for the subroutine. If NoGood_o=.True., the input matrix [A] cannot be decomposed and there is no output returned; otherwise the profile A_io returns the decomposed matrices [L] and [U]. For the situation where NoGood_o=.True., please see section 11.7.

11.3 Fortran Syntax for Subroutine ppSubstitute

This subroutine performs forward and backward substitutions. Syntax is as follows:

```
ppSubstitute_CAG_4(A_i, N_i, UpperBandwidth_i, LowerBandwidth_i,    &
                   From_i, First_i, X_io)
ppSubstitute_CAG_8(A_i, N_i, UpperBandwidth_i, LowerBandwidth_i,    &
                   From_i, First_i ,X_io)
ppSubstitute_CAG_10(A_i, N_i, UpperBandwidth_i, LowerBandwidth_i,    &
                    From_i, First_i, X_io)
ppSubstitute_CAG_16(A_i, N_i, UpperBandwidth_i, LowerBandwidth_i,    &
                    From_i, First_i, X_io)
ppSubstitute_CAG_Z4(A_i, N_i, UpperBandwidth_i, LowerBandwidth_i,    &
                    From_i, First_i, X_io)
ppSubstitute_CAG_Z8(A_i, N_i, UpperBandwidth_i, LowerBandwidth_i,    &
                    From_i, First_i, X_io)
ppSubstitute_CAG_Z10(A_i, N_i, UpperBandwidth_i, LowerBandwidth_i,    &
                     From_i, First_i, X_io)
ppSubstitute_CAG_Z16(A_i, N_i, UpperBandwidth_i, LowerBandwidth_i,    &
                     From_i, First_i, X_io)
```

where

1. The argument A_i, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A] that inputs the result from decomposition.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument UpperBandwidth_i, an INTEGER(4) variable, is the upper bandwidth of matrix [A]. The upper bandwidth is the maximal number of non-zero fill-ins on the right side of diagonal in a row.
4. The argument LowerBandwidth_i, an INTEGER(4) variable, is the lower bandwidth of matrix [A]. The lower bandwidth is the maximal number of non-zero fill-ins below the diagonal in a column.
5. The argument From_i, an INTEGER(4) array having N_i elements, inputs the row index where the remaining coefficients on a row are from.
6. The argument First_i, an INTEGER(4) array having N_i elements, inputs the index of the first nonzero fill-in on each column from.
7. The argument X_io, array whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution.

11.4 Fortran Syntax for Subroutine ppSolution

The following subroutines first decompose matrix [A] into the product of [L][U] with partial pivoting, and then perform forward and backward substitutions. Solve [A]{X}={B} in a single call. Syntax is as follows:

```
ppSolution_CAG_4(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,    &
                 From_x, First_x, X_io, NoGood_o)
ppSolution_CAG_8(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,    &
                 From_x,  First_x, X_io, NoGood_o)
ppSolution_CAG_10(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,    &
                 From_x, First_x, X_io, NoGood_o)
ppSolution_CAG_16(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,    &
                 From_x, First_x, X_io, NoGood_o)
ppSolution_CAG_Z4(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,    &
                 From_x, First_x, X_io, NoGood_o)
ppSolution_CAG_Z8(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,     &
                  From_x, First_x, X_io, NoGood_o)
ppSolution_CAG_Z10(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,    &
                   From_x, First_x, X_io, NoGood_o)
ppSolution_CAG_Z16(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,    &
                   From_x, First_x, X_io, NoGood_o)
```

where

1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A], that inputs the original matrix and returns the decomposed result if the variable NoGood_o is false. For the definition of profile, please see section 11.5.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument UpperBandwidth_i, an INTEGER(4) variable, is the upper bandwidth of matrix [A]. The upper bandwidth is the maximal number of non-zero fill-ins on the right side of diagonal in a row.
4. The argument LowerBandwidth_i, an INTEGER(4) variable, is the lower bandwidth of matrix [A]. The lower bandwidth is the maximal number of non-zero fill-ins below the diagonal in a column.
5. The argument From_x, an INTEGER(4) array having N_i elements, is a working array.
6. The argument First_x, an INTEGER(4) array having N_i elements, is a working array.
7. The argument X_io, array whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution if NoGood_o is false.
8. The argument NoGood_o, a LOGICAL(4) variable, is a flag indicating if the input system is suitable for the subroutine. If NoGood_o=.True., the input system cannot be solved by the subroutine and there is no output returned; otherwise the profile A_io returns the decomposed matrices [L] and [U], and vector X_io returns the solution. For the situation where NoGood_o=.True., please see section 11.7.

11.5 Profile

Similar to profile of variable-bandwidth and asymmetric solver, profile for constant-bandwidth and asymmetric solver with partial pivoting requires extra memory spaces for decomposition. Consider a constant-bandwidth and asymmetric matrix as follows:

$$
\begin{bmatrix}
= & + & & & & & \\
\ast & = & + & & & & \\
\ast & \ast & = & + & & & \\
 & \ast & \ast & = & + & & \\
 & & \ast & \ast & = & + & \\
 & & & \ast & \ast & = & + \\
 & & & & \ast & \ast & =
\end{bmatrix}
\qquad (11.1)
$$

where the symbol "+" indicates non-zero fill-ins in the upper triangular part, and the symbol "=" indicates non-zero fill-ins on the diagonal, and the symbol "*" indicates non-zero fill-ins in the lower triangular part. For the matrix in the form of (11.1), the upper bandwidth=1, and the lower bandwidth is 2. The profile for the lower triangular part is defined by the non-zero fill-ins in the lower triangular part, but the profile for the upper triangular part requires extra memory spaces. The upper bandwidth enlarges by adding the lower bandwidth, and the profile for the form (11.1) is written as follows:

$$
\begin{array}{ccccccc}
\& & & & & & & \\
\& & \& & & & & \\
\& & \& & \& & & & \\
= & + & \% & \% & & & \\
\ast & = & + & \% & \% & & \\
\ast & \ast & = & + & \% & \% & \\
 & \ast & \ast & = & + & \% & \% \\
 & & \ast & \ast & = & + & \% \\
 & & & \ast & \ast & = & + \\
 & & & & \ast & \ast & = \\
 & & & & & \& &
\end{array}
\qquad (11.2)
$$

There are five symbols in the profile, each of which is discussed in the following:

1. The symbol "+" represents non-zero fill-ins in the upper triangular part of the original matrix.
2. The symbol "=" represents non-zero fill-ins on the diagonal of the original matrix.
3. The symbol "*" represents non-zero fill-ins in the lower triangular part of the original matrix.
4. The symbol % represents extra memory space in the profile. All the extra space must be initialized to zero before calling any of the following subroutines

ppDecompose_CAG_4
ppDecompose_CAG_8
ppDecompose_CAG_10
ppDecompose_CAG_16
ppDecompose_CAG_Z4
ppDecompose_CAG_Z8
ppDecompose_CAG_Z10
ppDecompose_CAG_Z16

ppSolution_CAG_4
ppSolution_CAG_8
ppSolution_CAG_10
ppSolution_CAG_16

ppSolution_CAG_Z4
ppSolution_CAG_Z8
ppSolution_CAG_Z10
ppSolution_CAG_Z16

Each extra space denoted by the symbol % returns a coefficient after decomposition.

5. The symbol & indicates an extra memory space whose content is never used.

Total length of profile is determined as

profile size = N * (UpperBandwidth + LowerBandwidth * 2 + 1) – LowerBandwidth (11.3)

where N is the matrix order, and the variable *LowerBandwidth* is the lower bandwidth of the original matrix before decomposition, and *UpperBandwidth* is the upper bandwidth of the original matrix before decomposition.

11.6 Data Storage Scheme

Data storage scheme for a constant-bandwidth and asymmetric solver with partial pivoting must be declared in a Fortran program, for example:

```
INTEGER (4) :: Upper,Lower
REAL (4) :: A(1-Upper-Lower:Lower,1)
```

where variable A here is a single precision profile for matrix [A], and variable "Upper" is the upper bandwidth of the original matrix, and variable "Lower" is the lower bandwidth of the original matrix. For other kinds of variable, profile must be properly declared. Then, the coefficient A_{ij} of matrix [A] is programmed in a Fortran program as A(I,J), no matter A_{ij} is in the upper triangular part or in the lower triangular part

"Before decomposition", the non-zero fill-ins in the i-th column are from the beginning index:

Maximum of (1,i-Upper) (11.4)

to the ending index:

Minimum of (N,i+Lower) (11.5)

where N is the order of matrix [A]. After decomposition, the bandwidth in the upper triangular part has enlarged, and the beginning index in the i-th column becomes

Maximum of (1,i-Upper-Lower). (11.6)

In equations (11.4), (11.5), and (11.6), the variable "Upper" is the upper bandwidth of the original matrix before decomposition, and the variable "Lower" is the lower bandwidth of the original matrix before decomposition.

11.7 Failure of Calling Request

If the calling request fails, solving procedure cannot find a pivoting row such that the absolute value of the diagonal element is not negligible compared to unity.

11.8 Fortran Example

For a given system [A]{X}={B}, the left side matrix [A] and the right side vector {B} are defined as follows:

$$\begin{bmatrix} 1 & 2 & & & & & \\ 4 & 25 & 4 & & & & \\ 2 & 29 & 14 & 9 & & & \\ & 99 & 34 & 19 & 71 & & \\ & & 3 & 23 & 5 & 93 & \\ & & & 11 & 7 & 22 & 4 \\ & & & & 3 & 2 & 9 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 21 \\ 11 \\ 122 \\ 19 \\ 333 \\ 1 \\ 3 \end{bmatrix}$$

in which the order N=7, and the lower bandwidth LowerBandwidth=2, and the UpperBandwidth=1. A Fortran program for decomposition and substitution is as follows. There are four subroutines in the example: subroutines "Input" and "Output" have data storage scheme; subroutine "ppDecompose_CAG_4" decomposes matrix [A] with partial pivoting; subroutine "ppSubstitute_CAG_4" performs forward and backward substitutions.

```
!  *** Example program ***
!  define variables where the length of A is determined by equation (11.3)
!
      PARAMETER (N=7)
      INTEGER*4  UpperBandwidth
      PARAMETER (UpperBandwidth=1)
      PARAMETER (LowerBandwidth=2)
      REAL*4 A (N*(UpperBandwidth+LowerBandwidth*2+1)- LowerBandwidth )
      REAL*4 X(N)
      LOGICAL*4 NoGood
      INTEGER*4 From(N)
      INTEGER*4 First(N)
      DATA X/21.0,11.0,122.0,19.0,333.0,1.0,3.0/
!
!  input the non-zero fill-ins of matrix [A]
!
      CALL Input(A,UpperBandwidth, LowerBandwidth,N)
!
!  decompose in parallel
!
      CALL ppDecompose_CAG_4(A,N,UpperBandwidth, LowerBandwidth,  &
                                          From, First, NoGood)
!
!  stop if NoGood=.True.
```

```
!
      IF(NoGood) STOP 'Cannot be decomposed'
!
!  perform substitutions in parallel
!
      CALL ppSubstitute_CAG_4(A,N,UpperBandwidth, LowerBandwidth, From, First, X)
!
!  output decomposed matrix
!
      CALL Output(A,N,UpperBandwidth, LowerBandwidth)
!
!  output the solution
!
      Write(*,'(" Solution is as:")')
      Write(*,*) X
!
!  laipe done
!
      call laipeDone
!
      STOP
      END
      SUBROUTINE Input(A,Upper,Lower,N)
!
!
!  routine to demonstrate an application of data storage scheme
!  (A)FORTRAN CALL: CALL Input(A,Upper,Lower,N)
!     1.A: <R4> profile of matrix [A], dimension(*)
!     2.Upper: <I4> upper bandwidth
!     3.Lower: <I4> lower bandwidth
!     4.N: <I4> order of matrix
!
!  dummy arguments
!
      INTEGER*4 Upper,Lower,N
      REAL*4 A(1-Upper-Lower:Lower,1)
!
!  initialize
!     The ending bound of I4TEMP is determined by equation (11.3)
!
      DO I4TEMP=1,N*(Upper+Lower*2+1)-Lower
         A(I4TEMP,1)=0.0
      END DO
!
!  input
!
      A(1,1)= 1.0
      A(2,1)= 4.0
      A(3,1)= 2.0
      A(1,2)= 2.0
      A(2,2)=25.0
```

```
      A(3,2)=29.0
      A(4,2)=99.0
      A(2,3)= 4.0
      A(3,3)=14.0
      A(4,3)=34.0
      A(5,3)= 3.0
      A(3,4)= 9.0
      A(4,4)=19.0
      A(5,4)=23.0
      A(6,4)=11.0
      A(4,5)=71.0
      A(5,5)= 5.0
      A(6,5)= 7.0
      A(7,5)= 3.0
      A(5,6)=93.0
      A(6,6)=22.0
      A(7,6)= 2.0
      A(6,7)= 4.0
      A(7,7)= 9.0
!
      RETURN
      END
      SUBROUTINE Output(A,N,Upper,Lower)
!
!
!  routine to output the decomposed matrix by data storage scheme
!  (A)FORTRAN CALL: CALL Output(A,N,Upper,Lower)
!     1.A: <R4> profile of matrix [A], dimension(*)
!     2.N: <I4> order of matrix [A]
!     3.Upper: <I4> upper bandwidth
!     4.Lower: <I4> lower bandwidth
!
!  dummy arguments
!
      INTEGER*4 N,Upper,Lower
      REAL*4 A(1-Upper-Lower:Lower,1)
!
!  local variables
!
      INTEGER*4 Column,Row
!
!  output the coefficients on non-zero fill-ins. The beginning and ending indices for each
!  column are defined in equation (11.6) and equation (11.5)
!
      WRITE(*,'(" Row  Column  Coefficient")')
      DO Column=1,N
         DO Row=MAX0(1,Column-Upper-Lower), MIN0(N,Column+Lower)
            WRITE(*,'(I4,I6,F9.3)') Row,Column,A(Row,Column)
         END DO
      END DO
!
```

```
RETURN
END
```

Chapter 12. Constant-Bandwidth, Symmetric, and Positive Definite Solvers with Partial Pivoting

12.1 Purpose

This chapter has subroutines for the solution of [A]{X}={B} with partial pivoting where the left side matrix [A] is constant-bandwidth, symmetric, and positive definite. The non-zero fill-ins of matrix [A] have a shape, for example, as:

```
=
*   =      sym.
*   *   =
*   *   *   =
    *   *   *   =
        *   *   *   =
            *   *   *   =
```

where the symbol "=" indicates non-zero fill-ins on the diagonal, and the symbol "*" indicates non-zero fill-ins in the lower triangular part. Since the matrix [A] is symmetric, the upper bandwidth is equal to the lower bandwidth before decomposition. A partial pivoting generally disturbs symmetry. A decomposed result is not symmetric, such that the upper triangular part is different from the lower triangular part on the decomposed result. When applying the subroutines, just input the lower triangular part of the original matrix, and LAIPE solvers output the lower and upper triangular matrices after decomposition.

Three types of subroutine are introduced in this chapter, which perform the following functions:

1. Decompose matrix [A] into the product of [L][U] where matrix [L] is the lower triangular matrix and matrix [U] is the upper triangular matrix.
2. Perform forward and backward substitutions.
3. Solve [A]{X}={B} in a single call.

Decomposition and substitution must be called in order, and work together as a pair. Subroutines are as follows:

ppDecompose_CSP_4
ppDecompose_CSP_8
ppDecompose_CSP_10
ppDecompose_CSP_16
ppDecompose_CSP_Z4
ppDecompose_CSP_Z8
ppDecompose_CSP_Z10

ppDecompose_CSP_Z16

ppSubstitute_CSP_4
ppSubstitute_CSP_8
ppSubstitute_CSP_10
ppSubstitute_CSP_16
ppSubstitute_CSP_Z4
ppSubstitute_CSP_Z8
ppSubstitute_CSP_Z10
ppSubstitute_CSP_Z16

ppSolution_CSP_4
ppSolution_CSP_8
ppSolution_CSP_10
ppSolution_CSP_16
ppSolution_CSP_Z4
ppSolution_CSP_Z8
ppSolution_CSP_Z10
ppSolution_CSP_Z16

12.2 Fortran Syntax for Subroutine ppDecompose

The following subroutines decompose matrix [A] into [A]=[L][U] with partial pivoting. Syntax is as follows:

```
ppDecompose_CSP_4(A_io,N_i,LowerBandwidth_i,From_o, First_o,NoGood_o)
ppDecompose_CSP_8(A_io,N_i,LowerBandwidth_i,From_o,First_o, NoGood_o)
ppDecompose_CSP_10(A_io,N_i,LowerBandwidth_i,From_o,First_o, NoGood_o)
ppDecompose_CSP_16(A_io,N_i,LowerBandwidth_i,From_o,First_o, NoGood_o)
ppDecompose_CSP_Z4(A_io,N_i,LowerBandwidth_i,From_o,First_o, NoGood_o)
ppDecompose_CSP_Z8(A_io,N_i,LowerBandwidth_i,From_o,First_o, NoGood_o)
ppDecompose_CSP_Z10(A_io,N_i,LowerBandwidth_i,From_o,First_o, NoGood_o)
ppDecompose_CSP_Z16(A_io,N_i,LowerBandwidth_i,From_o,First_o, NoGood_o)
```

where

1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A] that inputs the original matrix and returns the result if the variable NoGood_o is false. For the definition of profile, please see section 12.5.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument LowerBandwidth_i, an INTEGER(4) variable, is the lower bandwidth of matrix [A]. The lower bandwidth is the maximal number of non-zero fill-ins below the diagonal in a column.
4. The argument From_o, an INTEGER(4) array having N_i elements, returns the row index where the remaining elements are from if NoGood_o is false.
5. The argument First_o, an INTEGER(4) array having N_i elements, returns the index of the first nonzero fill-in on each column if NoGood_o is false.
6. The argument NoGood_o, a LOGICAL(4) variable, is a flag that indicates if the input matrix [A] is suitable for the subroutine. If NoGood_o=.True., the input matrix [A] cannot be decomposed and there is no output returned; otherwise the profile A_io returns the

decomposed matrices [L] and [U]. For the situation where NoGood_o=.True., please see section 12.7.

12.3 Fortran Syntax for Subroutine ppSubstitute

The following subroutines perform forward and backward substitutions. Syntax is as follows:

```
ppSubstitute_CSP_4(A_i, N_i, LowerBandwidth_i, From_i, First_i, X_io)
ppSubstitute_CSP_8(A_i, N_i, LowerBandwidth_i, From_i, First_i, X_io)
ppSubstitute_CSP_10(A_i, N_i, LowerBandwidth_i, From_i, First_i, X_io)
ppSubstitute_CSP_16(A_i, N_i, LowerBandwidth_i, From_i, First_i, X_io)
ppSubstitute_CSP_Z4(A_i, N_i, LowerBandwidth_i, From_i, First_i, X_io)
ppSubstitute_CSP_Z8(A_i, N_i, LowerBandwidth_i, From_i, First_i, X_io)
ppSubstitute_CSP_Z10(A_i, N_i, LowerBandwidth_i, From_i, First_i, X_io)
ppSubstitute_CSP_Z16(A_i, N_i, LowerBandwidth_i, From_i, First_i, X_io)
```

where

1. The argument A_i, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A] that inputs the result from decomposition.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument LowerBandwidth_i, an INTEGER(4) variable, is the lower bandwidth of matrix [A]. The lower bandwidth is the maximal number of non-zero fill-ins below the diagonal in a column.
4. The argument From_i, an INTEGER(4) array having N_i elements, inputs the row index where the remaining elements are from.
5. The argument First_i, an INTEGER(4) array having N_i elements, inputs the index of the first non-zero fill-in on each column.
6. The argument X_io, array whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution.

12.4 Fortran Syntax for Subroutine ppSolution

The following subroutines first decompose matrix [A] into the product of [L][U] with partial pivoting, and then perform forward and backward substitutions. Solve [A]{X}={B} in a single call. Syntax is as follows:

```
ppSolution_CSP_4(A_io,N_i,LowerBandwidth_i,From_x,First_x,X_io,NoGood_o)
ppSolution_CSP_8(A_io,N_i,LowerBandwidth_i,From_x,First_x,X_io,NoGood_o)
ppSolution_CSP_10(A_io,N_i,LowerBandwidth_i,From_x,First_x,X_io,NoGood_o)
ppSolution_CSP_16(A_io,N_i,LowerBandwidth_i,From_x,First_x,X_io,NoGood_o)
ppSolution_CSP_Z4(A_io,N_i,LowerBandwidth_i,From_x,First_x,X_io,NoGood_o)
ppSolution_CSP_Z8(A_io,N_i,LowerBandwidth_i,From_x,First_x,X_io,NoGood_o)
ppSolution_CSP_Z10(A_io,N_i,LowerBandwidth_i,From_x,First_x,X_io,NoGood_o)
ppSolution_CSP_Z16(A_io,N_i,LowerBandwidth_i,From_x,First_x,X_io,NoGood_o)
```

where

1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A], that inputs the original matrix and returns the decomposed result if the variable NoGood_o is false. For the definition of profile, please see section 12.5.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument LowerBandwidth_i, an INTEGER(4) variable, is the lower bandwidth of matrix [A]. The lower bandwidth is the maximal number of non-zero fill-ins below the diagonal in a column.
4. The argument From_x, an INTEGER(4) array having N_i elements, is a working array.
5. The argument First_x, an INTEGER(4) array having N_i elements, is a working array.
6. The argument X_io, array whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution if NoGood_o is false.
7. The argument NoGood_o, a LOGICAL(4) variable, is a flag indicating if the input system is suitable for the subroutine. If NoGood_o=.True., the input system cannot be solved by the subroutine and there is no output returned; otherwise the profile A_io returns the decomposed matrices [L] and [U], and vector X_io returns the solution. For the situation where NoGood_o=.True., please see section 12.7.

12.5 Profile

Profile for a constant-bandwidth, symmetric, and positive definite solver with partial pivoting always requires extra memory spaces for the upper triangular part. There are two reasons for the extra memory space. The first one is that pivoting disturbs symmetry, such that the upper triangular part is not the transport of lower triangular part and the upper triangular part has to be completely saved. The second reason is that pivoting may enlarge the bandwidth of an upper triangular part.

Consider a constant-bandwidth and symmetric matrix as follows.

$$\begin{bmatrix} = & & & & & & \\ * & = & & \text{sym.} & & & \\ * & * & = & & & & \\ & * & * & = & & & \\ & & * & * & = & & \\ & & & * & * & = & \\ & & & & * & * & = \end{bmatrix} \tag{12.1}$$

where the symbol "=" indicates non-zero fill-ins on the diagonal, and the symbol "*" indicates non-zero fill-ins in the lower triangular part. For the matrix in the form of (12.1), the lower bandwidth is 2. Since the example matrix is symmetric, the upper bandwidth is 2. The profile for the lower triangular part is defined by the non-zero fill-ins in the lower triangular part, but the profile for the upper triangular part enlarges by adding the lower bandwidth. The profile for the example in form (12.1) is then written as follows

```
    &
    &   &
    &   &   &
 ┌  &   &   &   &                ┐
 │  =   %   %   %   %            │
 │  *   =   %   %   %   %        │
 │  *   *   =   %   %   %   %    │
 │      *   *   =   %   %   %    │        (12.2)
 │          *   *   =   %   %    │
 │              *   *   =   %    │
 │                  *   *   =    │
 └                               ┘
                        &
```

1. The symbol "=" represents non-zero fill-ins on the diagonal of the original matrix.
2. The symbol "*" represents non-zero fill-ins in the lower triangular part of the original matrix.
3. The symbol "%" represents an extra memory space in the profile. The space returns the upper triangular part of the decomposed matrix. It is unnecessary to initialize the space denoted by the symbol "%".
4. The symbol "&"indicates an extra memory space whose content is never used.

Total length of profile is determined as

profile size = N * (LowerBandwidth * 3 + 1) – LowerBandwidth (12.3)

where N is the matrix order, and the variable LowerBandwidth is the lower bandwidth.

12.6 Data Storage Scheme

Data storage scheme for a constant-bandwidth and symmetric solver with partial pivoting must be declared in a Fortran program, for example:

```
INTEGER (4) :: LowerBandwidth
REAL (4) :: A(1-LowerBandwidth*2:LowerBandwidth,1)
```

where variable A here is a single precision profile for a matrix [A], and the variable "LowerBandwidth" is the lower bandwidth of the matrix. For other kinds of variable, profile must be properly declared. Then, the coefficient A_{ij} of matrix [A] is programmed in a Fortran program as A(I,J), no matter A_{ij} is in the upper triangular part or in the lower triangular part.

"Before decomposition", the non-zero fill-ins in the i-th column are from the beginning index:

Maximum of (1,i-LowerBandwidth) (12.4)

to the ending index:

Minimum of (N,i+LowerBandwidth) (12.5)

where N is the order of matrix [A]. "After decomposition", the bandwidth in the upper triangular part has enlarged, and the beginning index in the i-th column becomes

$$\text{Maximum of } (1, i\text{-LowerBandwidth*2}) \qquad (12.6)$$

12.7 Failure of Calling Request

If the calling request fails, solving procedure cannot find a pivoting row such that the absolute value of diagonal element is not negligible compared to unity.

12.8 Fortran Example

For a given system [A]{X}={B}, the left side matrix [A] and the right side vector {B} are defined as:

$$\begin{bmatrix} 6 & & & & & & \\ 4 & 55 & & & \text{sym.} & & \\ 2 & 29 & 44 & & & & \\ & 9 & 34 & 91 & & & \\ & & 3 & 2 & 15 & & \\ & & & 11 & 7 & 22 & \\ & & & & 3 & 2 & 9 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 21 \\ 11 \\ 122 \\ 19 \\ 333 \\ 1 \\ 3 \end{bmatrix}$$

in which the order N=7, and the lower bandwidth LowerBandwidth=2. A Fortran program for decomposition and substitution is as follows. There are four subroutines in the example: subroutines “Input” and “Output” have data storage scheme; subroutine “ppDecompose_CSP_4” decomposes matrix [A]; subroutine “ppSubstitute_CSP_4” performs substitutions.

```
!  *** Example program ***
!  define variables where the length of A is determined by equation (12.3)
!
      PARAMETER (N=7)
      INTEGER*4  LowerBandwidth
      PARAMETER (LowerBandwidth=2)
      REAL*4 A(N*(LowerBandwidth*3+1)-LowerBandwidth)
      REAL*4 X(N)
      LOGICAL*4 NoGood
      INTEGER*4 From(N)
      INTEGER*4 First(N)
      DATA X/21.0,11.0,122.0,19.0,333.0,1.0,3.0/
!
!  input the non-zero fill-ins of matrix [A]
!
      CALL Input(A,LowerBandwidth,N)
!
```

```
!  decompose in parallel
!
      CALL ppDecompose_CSP_4(A,N,LowerBandwidth, From, First, NoGood)
!
!  stop if NoGood=.True.
!
      IF(NoGood) STOP 'Cannot be decomposed'
!
!  perform substitutions in parallel
!
      CALL ppSubstitute_CSP_4(A,N,LowerBandwidth,From,First, X)
!
!  output decomposed matrix
!
       CALL Output(A,N,LowerBandwidth)
!
!  output the solution
!
      Write(*,'(" Solution is as:")')
      Write(*,*) X
!
!  laipe done
!
      call laipeDone
!
      STOP
      END
      SUBROUTINE Input(A,Lower,N)
!
!
!  routine to demonstrate an application of data storage scheme
!  (A)FORTRAN CALL: CALL Input(A,Lower,N)
!    1.A: <R4> profile of matrix [A], dimension(*)
!    2.Lower: <I4> lower bandwidth
!    3.N: <I4> order of matrix
!
!  dummy arguments
!
      INTEGER*4 Lower,N
      REAL*4 A(1-Lower*2:Lower,1)
!
!  input
!
      A(1,1)= 6.0
      A(2,1)= 4.0
      A(3,1)= 2.0
      A(2,2)=55.0
      A(3,2)=29.0
      A(4,2)= 9.0
      A(3,3)=44.0
      A(4,3)=34.0
```

```
      A(5,3)= 3.0
      A(4,4)=91.0
      A(5,4)= 2.0
      A(6,4)=11.0
      A(5,5)=15.0
      A(6,5)= 7.0
      A(7,5)= 3.0
      A(6,6)=22.0
      A(7,6)= 2.0
      A(7,7)= 9.0
!
      RETURN
      END
      SUBROUTINE Output(A,N,Lower)
!
!
! routine to output the decomposed matrix by data storage scheme
! (A)FORTRAN CALL: CALL Output(A,N,Lower)
!   1.A: <R4> profile of matrix [A], dimension(*)
!   2.N: <I4> order of matrix [A]
!   3.Lower: <I4> lower bandwidth
!
! dummy arguments
!
      INTEGER*4 N,Lower
      REAL*4 A(1-Lower*2:Lower,1)
!
! local variables
!
      INTEGER*4 Column,Row
!
! output the coefficients on non-zero fill-ins
! The beginning and ending indices for each column are defined in
! equation (12.6) and equation (12.5)
!
      WRITE(*,'(" Row  Column  Coefficient")')
      DO Column=1,N
         DO Row=MAX0(1,Column-Lower*2), MIN0(N,Column+Lower)
            WRITE(*,'(I4,I6,F9.3)') Row,Column,A(Row,Column)
         END DO
      END DO
!
      RETURN
      END
```

Chapter 13. Constant-Bandwidth and Symmetric Solvers with Partial Pivoting

13.1 Purpose

This chapter has subroutines for the solution of [A]{X}={B} with partial pivoting where the left side matrix [A] has a constant bandwidth, and is symmetric. There is no consideration of definiteness of matrix [A]. The non-zero fill-ins of matrix [A] have a shape, for example, as:

$$\begin{bmatrix} = & & & & & & \\ * & = & & \text{sym.} & & & \\ * & * & = & & & & \\ * & * & * & = & & & \\ & * & * & * & = & & \\ & & * & * & * & = & \\ & & & * & * & * & = \end{bmatrix}$$

where the symbol "=" indicates non-zero fill-ins on the diagonal, and the symbol "*" indicates non-zero fill-ins in the lower triangular part. Since the matrix [A] is symmetric, the upper bandwidth is equal to the lower bandwidth before decomposition. A partial pivoting generally disturbs symmetry. A decomposed result is not symmetric, such that the upper triangular part is different from the lower triangular part. When applying the subroutines, just input the lower triangular part of the original matrix, and LAIPE solvers output the lower and upper triangular matrices after decomposition.

Three types of subroutine are introduced in this chapter, which perform the following functions:

1. Decompose matrix [A] into the product of [L][U] where matrix [L] is the lower triangular matrix and matrix [U] is the upper triangular matrix.
2. Perform forward and backward substitutions.
3. Solve [A]{X}={B} in a single call.

Decomposition and substitution must be called in order, and work together as a pair. Subroutines are as:

ppDecompose_CSG_4
ppDecompose_CSG_8
ppDecompose_CSG_10
ppDecompose_CSG_16
ppDecompose_CSG_Z4
ppDecompose_CSG_Z8
ppDecompose_CSG_Z10

ppDecompose_CSG_Z16

ppSubstitute_CSG_4
ppSubstitute_CSG_8
ppSubstitute_CSG_10
ppSubstitute_CSG_16
ppSubstitute_CSG_Z4
ppSubstitute_CSG_Z8
ppSubstitute_CSG_Z10
ppSubstitute_CSG_Z16

ppSolution_CSG_4
ppSolution_CSG_8
ppSolution_CSG_10
ppSolution_CSG_16
ppSolution_CSG_Z4
ppSolution_CSG_Z8
ppSolution_CSG_Z10
ppSolution_CSG_Z16

13.2 Fortran Syntax for Subroutine ppDecompose

The following subroutines decompose matrix [A] into [A]=[L][U] with partial pivoting. Syntax is as follows:

```
ppDecompose_CSG_4(A_io,N_i,LowerBandwidth_i,From_o,First_o,NoGood_o)
ppDecompose_CSG_8(A_io,N_i,LowerBandwidth_i,From_o,First_o,NoGood_o)
ppDecompose_CSG_10(A_io,N_i,LowerBandwidth_i,From_o,First_o,NoGood_o)
ppDecompose_CSG_16(A_io,N_i,LowerBandwidth_i,From_o,First_o,NoGood_o)
ppDecompose_CSG_Z4(A_io,N_i,LowerBandwidth_i,From_o,First_o,NoGood_o)
ppDecompose_CSG_Z8(A_io,N_i,LowerBandwidth_i,From_o,First_o,NoGood_o)
ppDecompose_CSG_Z10(A_io,N_i,LowerBandwidth_i,From_o,First_o,NoGood_o)
ppDecompose_CSG_Z16(A_io,N_i,LowerBandwidth_i,From_o,First_o,NoGood_o)
```

where

1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A] that inputs the original matrix and returns the result if the variable NoGood_o is false. For the definition of profile, please see section 13.5.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument LowerBandwidth_i, an INTEGER(4) variable, is the lower bandwidth of matrix [A]. The lower bandwidth is the maximal number of non-zero fill-ins below the diagonal in a column.
4. The argument From_o, an INTEGER(4) array having N_i elements, returns the row index where the remaining elements are from if NoGood_o is false.
5. The argument First_o, an INTEGER(4) array having N_i elements, returns the index of the first nonzero fill-in on each column if NoGood_o is false.
6. The argument NoGood_o, a LOGICAL(4) variable, is a flag that indicates if the input matrix [A] is suitable for the subroutine. If NoGood_o=.True., the input matrix [A] cannot be decomposed and there is no output returned; otherwise the profile A_io returns the

decomposed matrices [L] and [U]. For the situation where NoGood_o=.True., please see section 13.7.

13.3 Fortran Syntax for Subroutine ppSubstitute

The following subroutines perform forward and backward substitutions. Syntax is as follows:

```
ppSubstitute_CSG_4(A_i,N_i,LowerBandwidth_i,From,_i,First_i,X_io)
ppSubstitute_CSG_8(A_i,N_i,LowerBandwidth_i,From_i,First_i,X_io)
ppSubstitute_CSG_10(A_i,N_i,LowerBandwidth_i,From_i,First_i,X_io)
ppSubstitute_CSG_16(A_i,N_i,LowerBandwidth_i,From_i,First_i,X_io)
ppSubstitute_CSG_Z4(A_i,N_i,LowerBandwidth_I,From_i,First_i,X_io)
ppSubstitute_CSG_Z8(A_i,N_i,LowerBandwidth_i,From_i,First_i,X_io)
ppSubstitute_CSG_Z10(A_i,N_i,LowerBandwidth_i,From_i,First_i,X_io)
ppSubstitute_CSG_Z16(A_i,N_i,LowerBandwidth_i,From_i,First_i,X_io)
```

where

1. The argument A_i, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A] that inputs the result from decomposition.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument LowerBandwidth_i, an INTEGER(4) variable, is the lower bandwidth of matrix [A]. The lower bandwidth is the maximal number of non-zero fill-ins below the diagonal in a column.
4. The argument From_i, an INTEGER(4) array having N_i elements, inputs the row index where the remaining elements are from.
5. The argument First_i, an INTEGER(4) array having N_i elements, inputs the index of the first nonzero fill-in on each column.
6. The argument X_io, array whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution.

13.4 Fortran Syntax for Subroutine ppSolution

The following subroutines first decompose matrix [A] into the product of [L][U] with partial pivoting, and then perform forward and backward substitutions. Solve the system [A]{X}={B} in a single call. Syntax is as follows:

```
ppSolution_CSG_4(A_io, N_i, LowerBandwidth_i, From_x, First_x, X_io, NoGood_o)
ppSolution_CSG_8(A_io, N_i, LowerBandwidth_i, From_x, First_x, X_io, NoGood_o)
ppSolution_CSG_10(A_io, N_i, LowerBandwidth_i, From_x, First_x, X_io, NoGood_o)
ppSolution_CSG_16(A_io, N_i, LowerBandwidth_i, From_x, First_x, X_io, NoGood_o)
ppSolution_CSG_Z4(A_io, N_i, LowerBandwidth_i, From_x, First_x, X_io, NoGood_o)
ppSolution_CSG_Z8(A_io, N_i, LowerBandwidth_i, From_x, First_x, X_io, NoGood_o)
ppSolution_CSG_Z10(A_io, N_i, LowerBandwidth_i, From_x, First_x, X_io, NoGood_o)
ppSolution_CSG_Z16(A_io, N_i, LowerBandwidth_i, From_x, First_x, X_io, NoGood_o)
```

where

1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A], that inputs the original matrix and returns the decomposed result if the variable NoGood_o is false. For the definition of profile, please see section 13.5.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument LowerBandwidth_i, an INTEGER(4) variable, is the lower bandwidth of matrix [A]. The lower bandwidth is the maximal number of non-zero fill-ins below the diagonal in a column.
4. The argument From_x, an INTEGER(4) array having N_i elements, is a working array.
5. The argument First_x, an INTEGER(4) array having N_i elements, is a working array.
6. The argument X_io, array whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution if NoGood_o is false.
7. The argument NoGood_o, a LOGICAL(4) variable, is a flag indicating if the input system is suitable for the subroutine. If NoGood_o=.True., the input system cannot be solved and there is no output; otherwise the profile A_io returns the decomposed matrices [L] and [U], and vector X_io returns the solution. For the situation where NoGood_o=.True., please see section 13.7.

13.5 Profile

Profile for a constant-bandwidth and symmetric solver with partial pivoting always requires extra memory spaces for the upper triangular part of the decomposed result. There are two reasons for the extra memory space. The first one is that pivoting disturbs symmetry, such that the upper triangular part is not the transport of lower triangular part and the upper triangular part has to be completely saved. The second reason is that pivoting may enlarge the bandwidth of an upper triangular part.

Consider a constant-bandwidth and symmetric matrix as follows.

$$
\begin{bmatrix}
= & & & & & & \\
* & = & & \mathrm{sym.} & & & \\
* & * & = & & & & \\
 & * & * & = & & & \\
 & & * & * & = & & \\
 & & & * & * & = & \\
 & & & & * & * & =
\end{bmatrix}
\tag{13.1}
$$

where the symbol "=" indicates non-zero fill-ins on the diagonal, and the symbol "*" indicates non-zero fill-ins in the lower triangular part. For the matrix in the form of (13.1), the lower bandwidth is 2. Since the example matrix is symmetric, the upper bandwidth is 2. The profile for the lower triangular part is defined by the non-zero fill-ins in the lower triangular part, but the profile for the upper triangular part enlarges by adding the lower bandwidth. The profile for the example in form (13.1) is then written as follows

```
&
&  &
&  &  &
&  &  &  &
=  %  %  %  %
*  =  %  %  %  %
*  *  =  %  %  %  %
   *  *  =  %  %  %        (13.2)
      *  *  =  %  %
         *  *  =  %
            *  *  =
               &
```

profile size = N * (LowerBandwidth * 3 + 1) – LowerBandwidth (13.3)

where N is the matrix order, and the variable LowerBandwidth is the lower bandwidth.

13.6 Data Storage Scheme

Data storage scheme for a constant-bandwidth and symmetric solver with partial pivoting must be declared in a Fortran program, for example:

```
INTEGER (4) :: LowerBandwidth
REAL (4) :: A(1-LowerBandwidth*2:LowerBandwidth,1)
```

where variable A here is a single precision profile for matrix [A], and the variable "LowerBandwidth" is the lower bandwidth of the matrix. For other kinds of variable, profile must be properly declared. Then, the coefficient A_{ij} of matrix [A] is programmed in a Fortran program as A(I,J), no matter A_{ij} is in the upper triangular part or in the lower triangular part.

"Before decomposition", the non-zero fill-ins in the i-th column are from the beginning index:

Maximum of (1,i-LowerBandwidth) (13.4)

to the ending index:

Minimum of (N,i+LowerBandwidth) (13.5)

where N is the order of matrix [A]. "After decomposition", the bandwidth in the upper triangular part has enlarged, and the beginning index in the i-th column becomes

Maximum of (1,i-LowerBandwidth*2). (13.6)

13.7 Failure of Calling Request

If the calling request fails, solving procedure cannot find a pivoting row such that the absolute value of diagonal element is not negligible compared to unity.

13.8 Fortran Example

For a given system [A]{X}={B}, the left side matrix [A] and the right side vector {B} are defined as:

$$\begin{bmatrix} 6 & & & & & & \\ 4 & 5 & & & \text{sym.} & & \\ 2 & 29 & 44 & & & & \\ & 9 & 34 & 1 & & & \\ & & 3 & 2 & 15 & & \\ & & & 11 & 7 & 22 & \\ & & & & 3 & 2 & 9 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 21 \\ 11 \\ 122 \\ 19 \\ 333 \\ 1 \\ 3 \end{bmatrix}$$

in which the order N=7, and the lower bandwidth LowerBandwidth=2. A Fortran program for decomposition and substitution is as follows. There are four subroutines in the example. Subroutines "Input" and "Output" have data storage scheme; subroutine "ppDecompose_CSG_4" decomposes matrix [A]; subroutine "ppSubstitute_CSG_4" performs substitutions.

```
!  *** Example program ***
!  define variables where the length of A is determined by equation (13.3)
!
      PARAMETER (N=7)
      INTEGER*4  LowerBandwidth
      PARAMETER (LowerBandwidth=2)
      REAL*4 A(N*(LowerBandwidth*3+1)-LowerBandwidth)
      REAL*4 X(N)
      LOGICAL*4 NoGood
      INTEGER*4 From(N)
      INTEGER*4 First(N)
      DATA X/21.0,11.0,122.0,19.0,333.0,1.0,3.0/
!
!  input the non-zero fill-ins of matrix [A]
!
      CALL Input(A,LowerBandwidth,N)
!
!  decompose in parallel
!
      CALL ppDecompose_CSG_4(A,N, LowerBandwidth, From, First, NoGood)
!
!  stop if NoGood=.True.
!
      IF(NoGood) STOP 'Cannot be decomposed'
!
!  perform substitutions in parallel
!
      CALL ppSubstitute_CSG_4(A,N, LowerBandwidth, From, First, X)
!
```

```
!  output decomposed matrix
!
      CALL Output(A,N,LowerBandwidth)
!
!  output the solution
!
      Write(*,'(" Solution is as:")')
      Write(*,*) X
!
!  laipe done
!
      call laipeDone
!
      STOP
      END
      SUBROUTINE Input(A,Lower,N)
!
!
!  routine to demonstrate an application of data storage scheme
!  (A)FORTRAN CALL: CALL Input(A,Lower,N)
!    1.A: <R4> profile of matrix [A], dimension(*)
!    2.Lower: <I4> lower bandwidth
!    3.N: <I4> order of matrix
!
!  dummy arguments
!
      INTEGER*4 Lower,N
      REAL*4 A(1-Lower*2:Lower,1)
!
!  input
!
      A(1,1)= 6.0
      A(2,1)= 4.0
      A(3,1)= 2.0
      A(2,2)= 5.0
      A(3,2)=29.0
      A(4,2)= 9.0
      A(3,3)=44.0
      A(4,3)=34.0
      A(5,3)= 3.0
      A(4,4)= 1.0
      A(5,4)= 2.0
      A(6,4)=11.0
      A(5,5)=15.0
      A(6,5)= 7.0
      A(7,5)= 3.0
      A(6,6)=22.0
      A(7,6)= 2.0
      A(7,7)= 9.0
!
      RETURN
```

```
      END
      SUBROUTINE Output(A,N,Lower)
!
!
! routine to output the decomposed matrix by data storage scheme
! (A)FORTRAN CALL: CALL Output(A,N,Lower)
!   1.A: <R4> profile of matrix [A], dimension(*)
!   2.N: <I4> order of matrix [A]
!   3.Lower: <I4> lower bandwidth
!
! dummy arguments
!
      INTEGER*4 N,Lower
      REAL*4 A(1-Lower*2:Lower,1)
!
! local variables
!
      INTEGER*4 Column,Row
!
! output the coefficients on non-zero fill-ins
! The beginning and ending indices for each column are defined in
! equation (13.6) and equation (13.5)
!
      WRITE(*,'(" Row  Column  Coefficient")')
      DO Column=1,N
         DO Row=MAX0(1,Column-Lower*2), MIN0(N,Column+Lower)
            WRITE(*,'(I4,I6,F9.3)') Row,Column,A(Row,Column)
         END DO
      END DO
!
      RETURN
      END
```

Chapter 14. Dense and Asymmetric Solvers with Partial Pivoting

14.1 Purpose

This chapter has subroutines for the solution of [A]{X}={B} with partial pivoting where the left side matrix [A] is dense and asymmetric. There is no consideration of definiteness of matrix [A]. The non-zero fill-ins of matrix [A] have a simple shape, for example, as:

$$\begin{bmatrix} * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \end{bmatrix}$$

where the symbol "*" indicates non-zero fill-ins. Three types of subroutine are introduced in this chapter, which perform the following functions:

1. Decompose matrix [A] into the product of [L][U] where matrix [L] is the lower triangular matrix and matrix [U] is the upper triangular matrix.
2. Perform forward and backward substitutions.
3. Solve [A]{X}={B} in a single call.

Decomposition and substitution must be called in order, and work together as a pair. Subroutines are as follows:

ppDecompose_DAG_4
ppDecompose_DAG_8
ppDecompose_DAG_10
ppDecompose_DAG_16
ppDecompose_DAG_Z4
ppDecompose_DAG_Z8
ppDecompose_DAG_Z10
ppDecompose_DAG_Z16

ppSubstitute_DAG_4
ppSubstitute_DAG_8
ppSubstitute_DAG_10
ppSubstitute_DAG_16
ppSubstitute_DAG_Z4
ppSubstitute_DAG_Z8
ppSubstitute_DAG_Z10
ppSubstitute_DAG_Z16

ppSolution_DAG_4
ppSolution_DAG_8
ppSolution_DAG_10
ppSolution_DAG_16
ppSolution_DAG_Z4
ppSolution_DAG_Z8
ppSolution_DAG_Z10
ppSolution_DAG_Z16

14.2 Fortran Syntax for Subroutine ppDecompose

The following subroutines decompose matrix [A] into [A]=[L][U] with partial pivoting. Syntax is as follows:

```
ppDecompose_DAG_4(A_io, N_i, RowOrder_io, NoGood_o)
ppDecompose_DAG_8(A_io, N_i, RowOrder_io, NoGood_o)
ppDecompose_DAG_10(A_io, N_i, RowOrder_io, NoGood_o)
ppDecompose_DAG_16(A_io, N_i, RowOrder_io, NoGood_o)
ppDecompose_DAG_Z4(A_io, N_i, RowOrder_io, NoGood_o)
ppDecompose_DAG_Z8(A_io, N_i, RowOrder_io, NoGood_o)
ppDecompose_DAG_Z10(A_io, N_i, RowOrder_io, NoGood_o)
ppDecompose_DAG_Z16(A_io, N_i, RowOrder_io, NoGood_o)
```

where

1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A] that inputs the original matrix and returns the result if the variable NoGood_o is false. For the definition of profile, please see section 14.5.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument RowOrder_io, an INTEGER(4) array having N_i elements, enters a sequence of consecutive numbers from one to N_i and returns the pivoting rows if NoGood_o is false.
4. The argument NoGood_o, a LOGICAL(4) variable, is a flag that indicates if the input matrix [A] is suitable for the subroutine. If NoGood_o=.True., the input matrix [A] cannot be decomposed and there is no output returned; otherwise the profile A_io returns the decomposed matrices [L] and [U]. For the situation where NoGood_o=.True., please see section 14.7.

14.3 Fortran Syntax for Subroutine ppSubstitute

The following subroutines perform forward and backward substitutions. Syntax is as follows:

```
ppSubstitute_DAG_4(A_i, N_i, From_i, X_io)
ppSubstitute_DAG_8(A_i, N_i, From_i, X_io)
ppSubstitute_DAG_10(A_i, N_i, From_i, X_io)
ppSubstitute_DAG_16(A_i, N_i, From_i, X_io)
ppSubstitute_DAG_Z4(A_i, N_i, From_i, X_io)
ppSubstitute_DAG_Z8(A_i, N_i, From_i, X_io)
```

ppSubstitute_DAG_Z10(A_i, N_i, From_i, X_io)
ppSubstitute_DAG_Z16(A_i, N_i, From_i, X_io)

where

1. The argument A_i, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A] that inputs the result from decomposition.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument From_i, an INTEGER(4) array having N_i elements, inputs the pivoting rows from decomposition.
4. The argument X_io, array whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution.

14.4 Fortran Syntax for Subroutine ppSolution

The subroutines first decompose matrix [A] into the product of [L][U] with partial pivoting, and then perform forward and backward substitutions. Solve [A]{X}={B} in a single call. Syntax is as follows:

ppSolution_DAG_4(A_io, N_i, RowOrder_io, X_io, NoGood_o)
ppSolution_DAG_8(A_io, N_i, RowOrder_io, X_io, NoGood_o)
ppSolution_DAG_10(A_io, N_i, RowOrder_io, X_io, NoGood_o)
ppSolution_DAG_16(A_io, N_i, RowOrder_io, X_io, NoGood_o)
ppSolution_DAG_Z4(A_io, N_i, RowOrder_io, X_io, NoGood_o)
ppSolution_DAG_Z8(A_io, N_i, RowOrder_io, X_io, NoGood_o)
ppSolution_DAG_Z10(A_io, N_i, RowOrder_io, X_io, NoGood_o)
ppSolution_DAG_Z16(A_io, N_i, RowOrder_io, X_io, NoGood_o)

where

1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A], that inputs the original matrix and returns the decomposed result if the variable NoGood_o is false. For the definition of profile, please see section 14.5.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument RowOrder_io, an INTEGER(4) array having N_i elements, enters a sequence of consecutive numbers from one to N_i and returns the pivoting rows if NoGood_o is false.
4. The argument X_io, array whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution if NoGood_o is false.
5. The argument NoGood_o, a LOGICAL(4) variable, is a flag that indicates if the input system is suitable for the subroutine. If NoGood_o=.True., the input system cannot be solved by the subroutine and there is no output returned; otherwise the profile A_io returns the decomposed matrices [L] and [U], and vector X_io returns the solution. For the situation where NoGood_o=.True., please see section 14.7.

14.5 Profile

Profile for a dense and asymmetric matrix is the simplest as:

$$\begin{bmatrix} * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \end{bmatrix} \qquad (14.1)$$

Data storage scheme for a dense and asymmetric matrix must be declared in Fortran program, for example:

REAL (4) :: A(N,N)

where variable A here is a single precision profile for matrix [A], and N is the matrix order. For other kinds of variable, profile must be properly declared.

14.7 Failure of Calling Request

If a calling request fails, solving procedure cannot find a pivoting row such that the absolute value of diagonal element is not negligible compared to unity.

14.8 Fortran Example

For a given system [A]{X}={B}, the left side matrix [A] and the right side vector {B} are defined as follows:

$$\begin{bmatrix} 1 & 2 & 13 & 17 & 32 & 47 & 6 \\ 4 & 5 & 3 & 5 & 0 & 0 & 6 \\ 2 & 29 & 4 & 7 & 11 & 5 & 4 \\ 3 & 9 & 34 & 8 & 33 & 14 & 3 \\ 12 & 23 & 3 & 23 & 45 & -1 & 2 \\ 4 & 2 & 22 & 11 & 7 & 2 & 1 \\ 2 & 27 & 3 & 49 & 33 & 12 & 9 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 21 \\ 141 \\ 2 \\ 9 \\ 333 \\ 1 \\ 3 \end{bmatrix}$$

in which the order N=7. A Fortran program for decomposition and substitution is as follows. Subroutines "Input" and "Output" have data storage scheme. Subroutine "ppDecompose_DAG_4" decomposes matrix [A] with partial pivoting, and subroutine "ppSubstitute_DAG_4" performs forward and backward substitutions.

```
!  *** Example program ***
!  define variables where the length of A is determined by equation (14.2)
!
      PARAMETER (N=7)
      REAL*4 A(N,N),X(N)
      LOGICAL*4 NoGood
```

```
      INTEGER*4 RowOrder(N)
      DATA X/21.0,141.0,2.0,9.0,333.0,1.0,3.0/
!
! input matrix [A]
!
      CALL Input(A,N,RowOrder)
!
! decompose in parallel with partial pivoting
!
      CALL ppDecompose_DAG_4(A,N,RowOrder,NoGood)
!
! stop if NoGood=.True.
!
      IF(NoGood) STOP 'Cannot be decomposed'
!
! perform substitutions in parallel
!
      CALL ppSubstitute_DAG_4(A,N,RowOrder,X)
!
! output decomposed matrix
!
      CALL Output(A,N)
!
! output the solution
!
      Write(*,'(" Solution is as:")')
      Write(*,*) X
!
! laipe done
!
      call laipeDone
!
      STOP
      END
      SUBROUTINE Input(A,N,RowOrder)
!
!
! routine to demonstrate an application of data storage scheme
! (A)FORTRAN CALL: CALL Input(A,N,RowOrder)
!    1.A: <R4> profile of matrix [A], dimension(N,N)
!    2.N: <I4> the order of matrix [A]
!    3.RowOrder: <I4> return a sequence of  consecutive numbers from one to N, dimension(N)
!
! dummy arguments
!
      INTEGER*4 N
      REAL*4 A(N,N),RowOrder(N)
!
! set consecutive numbers
!
      DO I=1,N
```

```
            RowOrder(I)=I
         END DO
!
!  first column
!
         A(1,1)= 1.0
         A(2,1)= 4.0
         A(3,1)= 2.0
         A(4,1)= 3.0
         A(5,1)=12.0
         A(6,1)= 4.0
         A(7,1)= 2.0
!
!  second column
!
         A(1,2)= 2.0
         A(2,2)= 5.0
         A(3,2)=29.0
         A(4,2)= 9.0
         A(5,2)=23.0
         A(6,2)= 2.0
         A(7,2)=27.0
!
!  third column
!
         A(1,3)=13.0
         A(2,3)= 3.0
         A(3,3)= 4.0
         A(4,3)=34.0
         A(5,3)= 3.0
         A(6,3)=22.0
         A(7,3)= 3.0
!
!  fourth column
!
         A(1,4)=17.0
         A(2,4)= 5.0
         A(3,4)= 7.0
         A(4,4)= 8.0
         A(5,4)=23.0
         A(6,4)=11.0
         A(7,4)=49.0
!
!  fifth column
!
         A(1,5)=32.0
         A(2,5)= 0.0
         A(3,5)=11.0
         A(4,5)=33.0
         A(5,5)=45.0
         A(6,5)= 7.0
```

```
      A(7,5)=33.0
!
! sixth column
!
      A(1,6)=47.0
      A(2,6)= 0.0
      A(3,6)= 5.0
      A(4,6)=14.0
      A(5,6)=-1.0
      A(6,6)= 2.0
      A(7,6)=12.0
!
! seventh column
!
      A(1,7)= 6.0
      A(2,7)= 6.0
      A(3,7)= 4.0
      A(4,7)= 3.0
      A(5,7)= 2.0
      A(6,7)= 1.0
      A(7,7)= 9.0
!
      RETURN
      END
      SUBROUTINE Output(A,N)
!
!
! routine to output the decomposed matrix by data storage scheme
! (A)FORTRAN CALL: CALL Output(A,N)
!   1.A: <R4> profile of matrix [A], dimension(*)
!   2.N: <I4> order of matrix [A]
!
! dummy arguments
!
      INTEGER*4 N
      REAL*4 A(N,N)
!
! local variables
!
      INTEGER*4 Column,Row
!
! output the coefficients on non-zero fill-ins
!
      WRITE(*,'(" Row  Column  Coefficient")')
      DO Column=1,N
          DO Row=1,N
              WRITE(*,'(I4,I6,F9.3)') Row,Column,A(Row,Column)
          END DO
      END DO
!
      RETURN
```

END

Chapter 15. Dense and Asymmetric Solvers with Full Pivoting

15.1 Purpose

This chapter has subroutines for the solution of [A]{X}={B} with full pivoting where the left side matrix [A] is dense and asymmetric. There is no consideration of definiteness of matrix [A]. The non-zero fill-ins of matrix [A] have a simple shape, for example, as:

$$\begin{bmatrix} * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \end{bmatrix}$$

where the symbol "*" indicates non-zero fill-ins. Three types of subroutine are introduced in this chapter, which perform the following functions:

1. Decompose matrix [A] into the product of [L][U] where matrix [L] is the lower triangular matrix and matrix [U] is the upper triangular matrix.
2. Perform forward and backward substitutions.
3. Solve [A]{X}={B} in a single call.

Decomposition and substitution must be called in order, and work together as a pair. Subroutines are as follows:

fpDecompose_DAG_4
fpDecompose_DAG_8
fpDecompose_DAG_10
fpDecompose_DAG_16
fpDecompose_DAG_Z4
fpDecompose_DAG_Z8
fpDecompose_DAG_Z10
fpDecompose_DAG_Z16

fpSubstitute_DAG_4
fpSubstitute_DAG_8
fpSubstitute_DAG_10
fpSubstitute_DAG_16
fpSubstitute_DAG_Z4
fpSubstitute_DAG_Z8
fpSubstitute_DAG_Z10
fpSubstitute_DAG_Z16

fpSolution_DAG_4
fpSolution_DAG_8
fpSolution_DAG_10
fpSolution_DAG_16
fpSolution_DAG_Z4
fpSolution_DAG_Z8
fpSolution_DAG_Z10
fpSolution_DAG_Z16

15.2 Fortran Syntax for Subroutine fpDecompose

This subroutine decomposes matrix [A] into [A]=[L][U] with full pivoting. Syntax is as follows:

```
fpDecompose_DAG_4(A_io,N_i,RowOrder_io,ColumnOrder_io,NoGood_o)
fpDecompose_DAG_8(A_io,N_i,RowOrder_io,ColumnOrder_io,NoGood_o)
fpDecompose_DAG_10(A_io,N_i,RowOrder_io,ColumnOrder_io,NoGood_o)
fpDecompose_DAG_16(A_io,N_i,RowOrder_io,ColumnOrder_io,NoGood_o)
fpDecompose_DAG_Z4(A_io,N_i,RowOrder_io,ColumnOrder_io,NoGood_o)
fpDecompose_DAG_Z8(A_io,N_i,RowOrder_io,ColumnOrder_io,NoGood_o)
fpDecompose_DAG_Z10(A_io,N_i,RowOrder_io,ColumnOrder_io,NoGood_o)
fpDecompose_DAG_Z16(A_io,N_i,RowOrder_io,ColumnOrder_io,NoGood_o)
```

where

1. The argument A_io, array whose kind must be consistent with subroutine name convention, is the profile of matrix [A] that inputs the original matrix and returns the result if the variable NoGood_o is false. For the definition of profile, please see section 15.5.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument RowOrder_io, an INTEGER(4) array having N_i elements, enters a sequence of consecutive numbers from one to N_i and returns the pivoting rows if NoGood_o is false.
4. The argument ColumnOrder_io, an INTEGER(4) array having N_i elements, enters a sequence of consecutive numbers from one to N_i and returns the pivoting columns if NoGood_o is false.
5. The argument NoGood_o, a LOGICAL(4) variable, is a flag that indicates if the input matrix [A] is suitable for the subroutine. If NoGood_o=.True., the input matrix [A] cannot be decomposed and there is no output returned; otherwise the profile A_io returns the decomposed matrices [L] and [U]. For the situation where NoGood_o=.True., please see section 15.7.

15.3 Fortran Syntax for Subroutine fpSubstitute

This subroutine performs forward and backward substitutions. Syntax is as follows:

```
fpSubstitute_DAG_4(A_i, N_i, RowOrder_i, ColumnOrder_i, X_io)
fpSubstitute_DAG_8(A_i, N_i, RowOrder_i, ColumnOrder_i, X_io)
fpSubstitute_DAG_10(A_i, N_i, RowOrder_i, ColumnOrder_i, X_io)
fpSubstitute_DAG_16(A_i, N_i, RowOrder_i, ColumnOrder_i, X_io)
```

```
fpSubstitute_DAG_Z4(A_i, N_i, RowOrder_i, ColumnOrder_i, X_io)
fpSubstitute_DAG_Z8(A_i, N_i, RowOrder_i, ColumnOrder_i, X_io)
fpSubstitute_DAG_Z10(A_i, N_i, RowOrder_i, ColumnOrder_i, X_io)
fpSubstitute_DAG_Z16(A_i, N_i, RowOrder_i, ColumnOrder_i, X_io)
```

where

1. The argument A_i, array which type must be consistent with subroutine name convention, is the profile of matrix [A] that inputs the result from decomposition.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument RowOrder_i, an INTEGER(4) array having N_i elements, inputs the pivoting rows from decomposition.
4. The argument ColumnOrder_i, an INTEGER(4) array having N_i elements, inputs the pivoting columns from decomposition.
5. The argument X_io, array which type must be consistent with subroutine name convention, inputs the right side vector, and returns the solution.

15.4 Fortran Syntax for Subroutine fpSolution

The following subroutines first decompose matrix [A] into the product of [L][U] with full pivoting, and then perform forward and backward substitutions. Solve [A]{X}={B} in a single call. Syntax is as follows:

```
fpSolution_DAG_4(A_io, N_i, RowOrder_io, ColumnOrder_io, X_io, NoGood_o)
fpSolution_DAG_8(A_io, N_i, RowOrder_io, ColumnOrder_io, X_io, NoGood_o)
fpSolution_DAG_10(A_io, N_i, RowOrder_io, ColumnOrder_io, X_io, NoGood_o)
fpSolution_DAG_16(A_io, N_i, RowOrder_io, ColumnOrder_io, X_io, NoGood_o)
fpSolution_DAG_Z4(A_io, N_i, RowOrder_io, ColumnOrder_io, X_io, NoGood_o)
fpSolution_DAG_Z8(A_io, N_i, RowOrder_io, ColumnOrder_io, X_io, NoGood_o)
fpSolution_DAG_Z10(A_io, N_i, RowOrder_io, ColumnOrder_io, X_io, NoGood_o)
fpSolution_DAG_Z16(A_io, N_i, RowOrder_io, ColumnOrder_io, X_io, NoGood_o)
```

where

1. The argument A_io, array which type must be consistent with subroutine name convention, is the profile of matrix [A], that inputs the original matrix and returns the decomposed result if the variable NoGood_o is false. For the definition of profile, please see section 15.5.
2. The argument N_i, an INTEGER(4) variable, is the order of matrix [A].
3. The argument RowOrder_io, an INTEGER(4) array having N_i elements, enters a sequence of consecutive numbers from one to N_i and returns the pivoting rows if NoGood_o is false.
4. The argument ColumnOrder_io, an INTEGER(4) array having N_i elements, enters a sequence of consecutive numbers from one to N_i and returns the pivoting columns if NoGood_o is false.
5. The argument X_io, array which type must be consistent with subroutine name convention, inputs the right side vector, and returns the solution if NoGood_o is false.
6. The argument NoGood_o, a LOGICAL(4) variable, is a flag that indicates if the input system is suitable for the subroutine If NoGood_o=.True., the input system cannot be solved by the subroutine and there is no output returned; otherwise the profile A_io returns the decomposed matrices [L] and [U], and vector X_io returns the solution. For the situation where NoGood_o=.True., please see section 15.7.

15.5 Profile

Profile for a dense and a

$$\begin{bmatrix} * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \end{bmatrix} \qquad (15.1)$$

where the symbol "*" represents non-zero fill-ins. Total length of profile is determined as

$$\text{profile size} = N * N \qquad (15.2)$$

where N is the matrix order.

15.6 Data Storage Scheme

Data storage scheme for a dense and asymmetric matrix must be declared in Fortran program, for example:

```
REAL (4) :: A(N,N)
```

where variable A here is a single precision profile for matrix [A], and N is the matrix order. For other kinds of variable, profile must be properly declared. Then, the coefficient A_{ij} of matrix [A] is programmed in a Fortran program as A(I,J).

15.7 Failure of Calling Request

If a calling request fails, solving procedure cannot find a pivoting row such that the absolute value of diagonal element is not negligible compared to unity.

15.8 Fortran Example

For a given system [A]{X}={B}, the left side matrix [A] and the right side vector {B} are defined as:

$$\begin{bmatrix} 1 & 2 & 13 & 17 & 32 & 47 & 6 \\ 4 & 5 & 3 & 5 & 0 & 0 & 6 \\ 2 & 29 & 4 & 7 & 11 & 5 & 4 \\ 3 & 9 & 34 & 8 & 33 & 14 & 3 \\ 12 & 23 & 3 & 23 & 45 & -1 & 2 \\ 4 & 2 & 22 & 11 & 7 & 2 & 1 \\ 2 & 27 & 3 & 49 & 33 & 12 & 9 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 21 \\ 141 \\ 2 \\ 9 \\ 333 \\ 1 \\ 3 \end{bmatrix}$$

in which the order N=7. A Fortran program for decomposition and substitution is as follows. Subroutines "Input" and "Output" have data storage scheme. Subroutine "fpDecompose_DAG_8" decomposes matrix [A] with full pivoting, and subroutine "fpSubstitute_DAG_8" performs forward and backward substitutions.

```
!  *** Example program ***
!  define variables where the length of A is determined by equation (15.2)
!
      PARAMETER (N=7)
      REAL*4 A(N,N),X(N)
      LOGICAL*4 NoGood
      INTEGER*4 RowOrder(N),ColumnOrder(N)
      DATA X/21.0,141.0,2.0,9.0,333.0,1.0,3.0/
!
!  input matrix [A]
!
      CALL Input(A,N,RowOrder,ColumnOrder)
!
!  decompose in parallel with full pivoting
!
      CALL fpDecompose_DAG_4(A,N,RowOrder, ColumnOrder, NoGood)
!
!  stop if NoGood=.True.
!
      IF(NoGood) STOP 'Cannot be decomposed'
!
!  perform substitutions in parallel
!
      CALL fpSubstitute_DAG_4(A,N,RowOrder,ColumnOrder,X)
!
!  output decomposed matrix
!
      CALL Output(A,N)
!
!  output the solution
!
      Write(*,'(" Solution is as:")')
      Write(*,*) X
!
!  laipe done
!
```

```
      call laipeDone
!
      STOP
      END
      SUBROUTINE Input(A,N,RowOrder,ColumnOrder)
!
!
!  routine to demonstrate an application of data storage scheme
!  (A)FORTRAN CALL: CALL Input(A,N,RowOrder,ColumnOrder)
!     1.A: <R4> profile of matrix [A], dimension(N,N)
!     2.N: <I4> the order of matrix [A]
!     3.RowOrder: <I4> return consecutive numbers from one to N
!     4.ColumnOrder: <I4> return consecutive numbers from one to N
!
!  dummy arguments
!
      INTEGER*4 N
      REAL*4 A(N,N),RowOrder(M),ColumnOrder(N)
!
! set consecutive numbers
!
      DO I=1,N
         RowOrder(I)=I
      END DO
      DO I=1,N
         ColumnOrder(I)=I
      END DO
!
!  first column
!
      A(1,1)= 1.0
      A(2,1)= 4.0
      A(3,1)= 2.0
      A(4,1)= 3.0
      A(5,1)=12.0
      A(6,1)= 4.0
      A(7,1)= 2.0
!
!  second column
!
      A(1,2)= 2.0
      A(2,2)= 5.0
      A(3,2)=29.0
      A(4,2)= 9.0
      A(5,2)=23.0
      A(6,2)= 2.0
      A(7,2)=27.0
!
!  third column
!
      A(1,3)=13.0
```

```
      A(2,3)= 3.0
      A(3,3)= 4.0
      A(4,3)=34.0
      A(5,3)= 3.0
      A(6,3)=22.0
      A(7,3)= 3.0
!
! fourth column
!
      A(1,4)=17.0
      A(2,4)= 5.0
      A(3,4)= 7.0
      A(4,4)= 8.0
      A(5,4)=23.0
      A(6,4)=11.0
      A(7,4)=49.0
!
! fifth column
!
      A(1,5)=32.0
      A(2,5)= 0.0
      A(3,5)=11.0
      A(4,5)=33.0
      A(5,5)=45.0
      A(6,5)= 7.0
      A(7,5)=33.0
!
! sixth column
!
      A(1,6)=47.0
      A(2,6)= 0.0
      A(3,6)= 5.0
      A(4,6)=14.0
      A(5,6)=-1.0
      A(6,6)= 2.0
      A(7,6)=12.0
!
! seventh column
!
      A(1,7)=6.0
      A(2,7)=6.0
      A(3,7)=4.0
      A(4,7)=3.0
      A(5,7)=2.0
      A(6,7)=1.0
      A(7,7)=9.0
!
      RETURN
      END
      SUBROUTINE Output(A,N)
!
```

```
!
!  routine to output the decomposed matrix by data storage scheme
!  (A)FORTRAN CALL: CALL Output(A,N)
!     1.A: <R4> profile of matrix [A], dimension(*)
!     2.N: <I4> order of matrix [A]
!
!  dummy arguments
!
      INTEGER*4 N
      REAL*4 A(N,N)
!
!  local variables
!
      INTEGER*4 Column,Row
!
!  output the coefficients on non-zero fill-ins
!
      WRITE(*,'(" Row  Column  Coefficient")')
      DO Column=1,N
          DO Row=1,N
              WRITE(*,'(I4,I6,F9.3)') Row,Column,A(Row,Column)
          END DO
      END DO
!
      RETURN
      END
```

Appendix A. Auxiliary Subroutine for Releasing System Resource

LAIPE is programmed in MTASK that allocates some system resource. The system resource allocated by MTASK may be automatically released when the system resource is unnecessary any more. LAIPE provides an auxiliary subroutine to immediately release system resource when LAIPE is no longer required in an application.

A.1 Fortran Syntax for Subroutine laipeDone

This subroutine has no arguments. Fortran syntax is as follow:

```
CALL laipeDone
```

Appendix B. Auxiliary Subroutines for Task Manipulations

This chapter has subroutines to set tasks for LAIPE solvers. Setting tasks for LAIPE solver is always necessary when monitoring the performance. That may allow the executing time to be collected with respect to a specified number of tasks. Then, speedup is obtained. This shows a situation to set tasks for LAIPE solver.

Another situation to set tasks for LAIPE solvers is to reduce overhead for small-size problems. By default, LAIPE solvers use all the available processors for computing. For example, if there are 4 processors available, LAIPE solvers automatically start 4 tasks for computing. It is not worth distributing small system onto multiprocessors. When applying LAIPE solvers to small problems, i.e. of order 50x50, set a single task for the solution. On a single processor computer, the default task is one. This chapter has three subroutines for task manipulations, which are as:

GetTasks
SetTasks
ResetTasks

B.1 Fortran Syntax for Subroutine GetTasks

This subroutine gets the number of tasks that are ready for LAIPE solvers. Fortran syntax is as follow:

```
CALL GetTasks(tasks_o)
```

where

1. The argument tasks_o, an INTEGER*4 variable, returns the number of tasks available for LAIPE solvers.

B.2 Fortran Syntax for Subroutine SetTasks

This subroutine sets tasks for LAIPE solvers. Fortran syntax is as follow:

```
CALL SetTasks(tasks_i)
```

where

1. The argument tasks_i, an INTEGER*4 variable, inputs the number of tasks for LAIPE solvers. The input tasks_i cannot be greater than the number of available processors. By default, the parameter is the number of processors available.

B.3 Fortran Syntax for Subroutine ResetTasks

This subroutine resets tasks to be the number of available processors. If an application never set tasks, it is unnecessary to call this subroutine to reset the parameter. Fortran syntax is as follow:

```
CALL ResetTasks
```

There is no argument required in the subroutine.