LAIPE

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

List of Contents

About This Manual

About

More than 90% of scientific and engineering problems are formulated into a system of equations. LAIPE has the most useful and highly efficient solvers for scientific and engineering computing. 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.

LAIPE is written in MTASK, and LAIPE2 is programmed in neuLoop, which are parallel programming language. MTASK is based on multitasking technology, while neuLoop is based on soft core computing. There are two types of soft core, homogenous and heterogeneous.

When building an application with LAIPE direct solvers, a copy of MTASK is necessary. For example with gfortran, application can be built as:

```
gfortran your_program -llaipe -lmtask
```

When linking against LAIPE2, application can be built with gfortran as:

```
gfortran your_program -llaipe2 -lneuloop4
gfortran your_program -llaipe2 -lneuloop6
```

where neuloop4 is a library for homogeneous soft cores, and neuloop6 is a library for heterogeneous soft cores. There are three ways to build an application with LAIPE.

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 system equations 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 system equations 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 system equations 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 system equations 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 system equations 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 system equations 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 system equations 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 system equations 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 system equations 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 system equations 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 system equations 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 system equations 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 system equations 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 system equations in the category, with the definition of profile, data storage scheme, and example.

Chapter 1. Introduction

Parallel computing especially benefits to large-scaled problems, that distributes computing loads among employed cores 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 cores may produce better improvement.

LAIPE has high performance parallel solvers. On unicore environments, LAIPE run as usual. When multicore presents, LAIPE may split itself to fit the multicore environment. Users just link LAIPE to their applications. It is unnecessary for users to distribute computing instructions onto multicores. LAIPE is a package for both small and large-scaled problems, including solvers for the following:

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) includes 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 formulated 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]. 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 solution of dense system. If a system is sparse in nature, it is always better to apply sparse solvers.

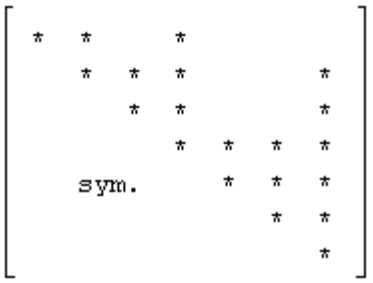
1.4 Profile

Profile is a contiguous space to store a matrix. For a dense matrix that is the simplest example, the profile is a space for the entire matrix, 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**. If compiler does not allocate a 2-dimensional array in a contiguous space, that may create problems for LAIPE. It is always safe to declare [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,



is a sparse matrix of order (7x7). If the matrix is stored as a dense matrix, it requires a space of 49 fill-ins; While when applying sparse storage scheme, the sparse matrix only requires 20 fill-ins, e. g., A_{11} , A_{12} , A_{22} , A_{23} , A_{33} , A_{14} , A_{24} , A_{34} , A_{44} , A_{45} , A_{55} , A_{46} , A_{56} , A_{66} , A_{27} , A_{37} , A_{47} , A_{57} , A_{67} , and A_{77} .

Data access scheme has an address reference label. Profile and address reference label will be introduced with each individual solver.

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*. 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 concept of pivoting is well known. There are two kinds of pivoting; *partial pivoting* and *full pivoting*.

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 for 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 are identified by 5 symbols. The name convention is as:

laipe$(Symbol 1)_(Symbol 2)(Symbol 3) (Symbol 4)_(Symbol 5)

Each symbol is described as follows.

§ Symbol 1

The symbol is the purpose of 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

§ Symbol 2

This symbol indicates the type of sparsity, and may be one of the following:

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

§ Symbol 3

This symbol indicates if matrix is symmetric or asymmetric, and is one of the following:

S : symmetric matrix
A : asymmetric matrix

§ Symbol 4

This symbol indicates if the matrix is positive definite or indefinite, and is one of the following:

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

§ Symbol 5

This symbol is the type of floating-point variables declared in the subroutine, and is one of the following:

4 : single precision REAL (4 bytes)
8 : double precision REAL (8 bytes)
10 : extended precision REAL (10 bytes)
16 : quad precision REAL (16 bytes)
Z4 : single precision COMPLEX (8 bytes)
Z8 : double precision COMPLEX (16 bytes)
Z10 : extended precision COMPLEX (20 bytes)
Z16 : quad precision COMPLEX (32 bytes)

LAIPE subroutines are identified by the above five symbols. For example, the subroutine "laipe$decompose_VSG_8" decomposes a variable-bandwidth, symmetric, and indefinite matrix in 8-byte REAL variables, e.g., double precision.

In this manual, the dummy arguments passed to LAIPE functions are with a suffix "_i", "_o", "_io", or "_x". The suffix "_i" identifies an input argument. The suffix "_o" identifies an output argument. The suffix "_io" identifies the argument is an input and returns the result. The suffix "_x" identifies a working space for temporary uses. For example, in the LAIPE subroutine

```
laipe$decompose_CSP_4(A_io, N_i, LowerBandwidth_i, NoGood_o)
```

the arguments "A_io", "N_i", and "LowerBandwidth_i" are input to the subroutine, and the subroutine returns the results in "A_io" and "NoGood_o".

1.8 Data Storage Schemes

A data storage scheme is associated with profile, and has two components. The first one is declaration of the profile, and the second one is the way to access the profile. For example, a skyline matrix [A] is declared in a Fortran subroutine as

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

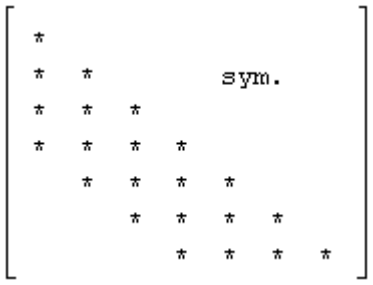
And, the coefficient A_{ij} is accessed in a Fortran program by A(I,Label(J)) where Label(J) is the address reference label for column J.

Data storage scheme is applied to dummy arguments, for example in a subroutine. 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 introduces subroutines for the solution of [A]{X}={B} where the left side matrix [A] is symmetric, and positive definite with a constant bandwidth. The non-zero fill-ins in the lower triangular part of matrix [A] are in a form, for example, as:



Three types of subroutine are included in LAIPE:

laipe$decompose_CSP_4
laipe$decompose_CSP_8
laipe$decompose_CSP_10
laipe$decompose_CSP_16
laipe$decompose_CSP_Z4
laipe$decompose_CSP_Z8
laipe$decompose_CSP_Z10
laipe$decompose_CSP_Z16

laipe$substitute_CSP_4
laipe$substitute_CSP_8
laipe$substitute_CSP_10
laipe$substitute_CSP_16
laipe$substitute_CSP_Z4
laipe$substitute_CSP_Z8
laipe$substitute_CSP_Z10
laipe$substitute_CSP_Z16

laipe$solution_CSP_4
laipe$solution_CSP_8
laipe$solution_CSP_10
laipe$solution_CSP_16
laipe$solution_CSP_Z4
laipe$solution_CSP_Z8
laipe$solution_CSP_Z10

laipe$solution_CSP_Z16

laipe$meSolution_CSP_4
laipe$meSolution_CSP_8
laipe$meSolution_CSP_10
laipe$meSolution_CSP_16
laipe$meSolution_CSP_Z4
laipe$meSolution_CSP_Z8
laipe$meSolution_CSP_Z10
laipe$meSolution_CSP_Z16

The subroutine with "me", i.e., laipe$meSolution_CSP_4, is a multiple-entry direct solver 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 $[A]=[L][L]^T$:

laipe$decompose_CSP_4 (A_io, N_i, LowerBandwidth_i, NoGood_o)
laipe$decompose_CSP_8 (A_io, N_i, LowerBandwidth_i, NoGood_o)
laipe$decompose_CSP_10 (A_io, N_i, LowerBandwidth_i, NoGood_o)
laipe$decompose_CSP_16 (A_io, N_i, LowerBandwidth_i, NoGood_o)
laipe$decompose_CSP_Z4 (A_io, N_i, LowerBandwidth_i, NoGood_o)
laipe$decompose_CSP_Z8 (A_io, N_i, LowerBandwidth_i, NoGood_o)
laipe$decompose_CSP_Z10 (A_io, N_i, LowerBandwidth_i, NoGood_o)
laipe$decompose_CSP_Z16 (A_io, N_i, LowerBandwidth_i, NoGood_o)

where

1. The argument A_io, array whose type 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=0. For the definition of profile, please see section 2.6.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument LowerBandwidth_i, a 4-byte INTEGER, is the lower bandwidth of matrix [A].
4. The argument NoGood_o, a 4-byte INTEGER, outputs a flag. If NoGood_o=1, the input matrix [A] is not positive definite and [A] returns an unpredicted result; otherwise the profile A_io returns the decomposed matrix [L]. For the situation where NoGood_o=1, please see section 2.8.

2.3 Fortran Syntax for Subroutine Substitute

The following subroutines perform forward and backward substitutions:

laipe$substitute_CSP_4 (A_i, N_i, LowerBandwidth_i, X_io)
laipe$substitute_CSP_8 (A_i, N_i, LowerBandwidth_i, X_io)
laipe$substitute_CSP_10 (A_i, N_i, LowerBandwidth_i, X_io)
laipe$substitute_CSP_16 (A_i, N_i, LowerBandwidth_i, X_io)
laipe$substitute_CSP_Z4 (A_i, N_i, LowerBandwidth_i, X_io)

laipe$substitute_CSP_Z8 (A_i, N_i, LowerBandwidth_i, X_io)
laipe$substitute_CSP_Z10 (A_i, N_i, LowerBandwidth_i, X_io)
laipe$substitute_CSP_Z16 (A_i, N_i, LowerBandwidth_i, X_io)

where

1. The argument A_i, array whose 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, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument LowerBandwidth_i, a 4-byte INTEGER, is the lower bandwidth of matrix [A].
4. The argument X_io, array whose type 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 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:

laipe$solution_CSP_4 (A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)
laipe$solution_CSP_8 (A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)
laipe$solution_CSP_10 (A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)
laipe$solution_CSP_16 (A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)
laipe$solution_CSP_Z4 (A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)
laipe$solution_CSP_Z8 (A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)
laipe$solution_CSP_Z10 (A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)
laipe$solution_CSP_Z16 (A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)

where

1. The argument A_io, array whose 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=0. For the definition of profile, please see section 2.6.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument LowerBandwidth_i, a 4-byte INTEGER, is the lower bandwidth of matrix [A].
4. The argument X_io, array whose type must be consistent with subroutine name convention, inputs the right side vector, and returns the solution if NoGood_o=0.
5. The argument NoGood_o, a 4-byte INTEGER, returns a flag. If NoGood_o=1, the input matrix [A] is not positive definite and [A] returns an unpredicted result; otherwise the profile A_io returns the decomposed matrix [L] and vector X_io returns the solution. For the situation where NoGood_o=1, please see section 2.8.

2.5 Fortran Syntax for meSolution

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

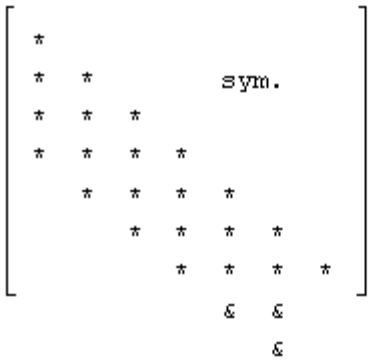
laipe$meSolution_CSP_4(A_io, N_i,LowerBandwidth_i, X_io, Nset_i, WorkingSpace_x, NoGood_o)
laipe$meSolution_CSP_8(A_io, N_i,LowerBandwidth_i, X_io, Nset_i, WorkingSpace_x, NoGood_o)
laipe$meSolution_CSP_10(A_io, N_i,LowerBandwidth_i, X_io, Nset_i, WorkingSpace_x, NoGood_o)
laipe$meSolution_CSP_16(A_io, N_i,LowerBandwidth_i, X_io, Nset_i, WorkingSpace_x, NoGood_o)
laipe$meSolution_CSP_Z4(A_io, N_i,LowerBandwidth_i, X_io, Nset_i, WorkingSpace_x, NoGood_o)
laipe$meSolution_CSP_Z8(A_io, N_i,LowerBandwidth_i, X_io, Nset_i, WorkingSpace_x, NoGood_o)
laipe$meSolution_CSP_Z10(A_io, N_i,LowerBandwidth_i, X_io, Nset_i,WorkingSpace_x, NoGood_o)
laipe$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 type 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 of [A] has been changed. For the definition of profile, please see section 2.6.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument LowerBandwidth_i, a 4-byte INTEGER, is the lower bandwidth of matrix [A].
4. The argument X_io, array whose type must be consistent with subroutine name convention, inputs the right side vector(s), and returns the solution if NoGood_o=0.
5. The argument Nset_i, a 4-byte INTEGER, is the number of right side vectors.
6. The argument WorkingSpace_x, array whose type 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 4-byte INTEGER, returns a flag. If NoGood_o=1, the input matrix [A] is not positive definite; otherwise the vector X_io returns the solution. For the situation where NoGood_o=1, please see section 2.8.

2.6 Profile

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



(2.1)

where the symbol * represents non-zero fill-ins and the symbol & indicates an extra memory space whose content is ignored. Total length of profile is determined as

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

where N is the order of square matrix, 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 is a single precision profile. Other types of variable, profile could be similarly declared. Then, the coefficient A_{ij} in the lower triangular part of matrix [A] is accessed by A(I,J) in a Fortran program.

2.8 Failure of Calling Request

If a calling request fails (e.g., NoGood=1), the solver is not suitable for the problem.

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:

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

in which the order N=7 and the lower bandwidth is 2. In the following Fortran program, subroutines "Input" and "Output" demonstrate data storage scheme; Subroutine "laipe$decompose_CSP_4" decomposes matrix [A]; Subroutine "laipe$substitute_CSP_4" performs forward and backward substitutions to obtain {X}.

```
!  *** 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)
      integer*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 laipe$decompose_CSP_4(A,N,LowerBandwidth, NoGood)

!
!  stop if NoGood=1
!
      IF(NoGood.eq.1) STOP 'Cannot be decomposed'

!
!  perform substitutions in parallel
!
      CALL laipe$substitute_CSP_4(A,N,LowerBandwidth,sX)

!
!  output decomposed matrix
!
      CALL Output(A,N,LowerBandwidth)

!
!  output the solution
!
      Write(*,'(" Solution is as:")')
      Write(*,*) X

!
!  end of the program
!
      CALL laipe$done
      STOP
      END

      SUBROUTINE Input(A,LowerBandwidth)
!
!
!  routine to demonstrate 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 square 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 of decomposed matrix
!
      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 introduces subroutines for the solution of [A]{X}={B} where the left side matrix [A] has a variable bandwidth, and is also symmetric and positive definite. The non-zero fill-ins in the upper triangular part of matrix [A] have a form, for example, as:

which looks like a skyline in a city, and is sometimes called skyline solver. Three types of subroutine are included:

laipe$Decompose_VSP_4
laipe$Decompose_VSP_8
laipe$Decompose_VSP_10
laipe$Decompose_VSP_16
laipe$Decompose_VSP_Z4
laipe$Decompose_VSP_Z8
laipe$Decompose_VSP_Z10
laipe$Decompose_VSP_Z16

laipe$Substitute_VSP_4
laipe$Substitute_VSP_8
laipe$Substitute_VSP_10
laipe$Substitute_VSP_16
laipe$Substitute_VSP_Z4
laipe$Substitute_VSP_Z8
laipe$Substitute_VSP_Z10
laipe$Substitute_VSP_Z16

laipe$Solution_VSP_4
laipe$Solution_VSP_8
laipe$Solution_VSP_10
laipe$Solution_VSP_16
laipe$Solution_VSP_Z4
laipe$Solution_VSP_Z8

laipe$Solution_VSP_Z10
laipe$Solution_VSP_Z16

3.2 Fortran Syntax for Subroutine Decompose

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

laipe$Decompose_VSP_4(A_io, N_i, Label_i, NoGood_o)
laipe$Decompose_VSP_8(A_io, N_i, Label_i, NoGood_o)
laipe$Decompose_VSP_10(A_io, N_i, Label_i, NoGood_o)
laipe$Decompose_VSP_16(A_io, N_i, Label_i, NoGood_o)
laipe$Decompose_VSP_Z4(A_io, N_i, Label_i, NoGood_o)
laipe$Decompose_VSP_Z8(A_io, N_i, Label_i, NoGood_o)
laipe$Decompose_VSP_Z10(A_io, N_i, Label_i, NoGood_o)
laipe$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=0. For the definition of profile, please see section 3.5.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument Label_i, a 4-byte INTEGER array of dimension(N_i), is the address reference label. For the definition of address reference label, please see section 3.6.
4. The argument NoGood_o, a 4-byte INTEGER, returns a flag. If NoGood_o=1, the input matrix [A] is not a well-defined positive definite and [A] returns an unpredicted result; otherwise the profile A_io returns the decomposed matrix [U]. For the situation where NoGood_o=1, please see section 3.7.

3.3 Fortran Syntax for Subroutine Substitute

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

laipe$Substitute_VSP_4(A_i, N_i, Label_i, X_io)
laipe$Substitute_VSP_8(A_i, N_i, Label_i, X_io)
laipe$Substitute_VSP_10(A_i, N_i, Label_i, X_io)
laipe$Substitute_VSP_16(A_i, N_i, Label_i, X_io)
laipe$Substitute_VSP_Z4(A_i, N_i, Label_i, X_io)
laipe$Substitute_VSP_Z8(A_i, N_i, Label_i, X_io)
laipe$Substitute_VSP_Z10(A_i, N_i, Label_i, X_io)
laipe$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, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument Label_i, a 4-byte INTEGER array of dimension(N_i), 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:

```
laipe$Solution_VSP_4 ( A_io, N_i, Label_i, X_io, NoGood_o)
laipe$Solution_VSP_8 ( A_io, N_i, Label_i, X_io, NoGood_o)
laipe$Solution_VSP_10 ( A_io, N_i, Label_i, X_io, NoGood_o)
laipe$Solution_VSP_16 ( A_io, N_i, Label_i, X_io, NoGood_o)
laipe$Solution_VSP_Z4 ( A_io, N_i, Label_i, X_io, NoGood_o)
laipe$Solution_VSP_Z8 ( A_io, N_i, Label_i, X_io, NoGood_o)
laipe$Solution_VSP_Z10 ( A_io, N_i, Label_i, X_io, NoGood_o)
laipe$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=0. For the definition of profile, please see section 3.5.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument Label_i, a 4-byte INTEGER array of dimension(N_i), 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=0.
5. The argument NoGood_o, a 4-byte INTEGER, returns a flag. If NoGood_o=1, the input matrix [A] is not positive definite and [A] returns an unpredicted result; otherwise the profile A_io returns the decomposed matrix [U] and vector X_io returns the solution. For the situation where NoGood_o=1, please see section 3.7.

3.5 Profile

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

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

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

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

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

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:

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

3.7 Failure of Calling Request

If a calling request fails (e. g., NoGood=1), the subroutine is not suitable for the problem.

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. In the following Fortran program, subroutines “Input” and “Output” demonstrate data storage scheme; Subroutine “laipe$Decompose_VSP_4” decomposes matrix [A]; Subroutine “laipe$Substitute_VSP_4” performs forward and backward substitutions to get {X}.

```
!  *** 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)
      INTEGER*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 laipe$decompose_VSP_4(A,N,Label,NoGood)

!
!  stop if NoGood=1
!
      IF(NoGood.eq.1) STOP 'Cannot be decomposed'

!
!  perform substitutions in parallel
!
      CALL laipe$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
```

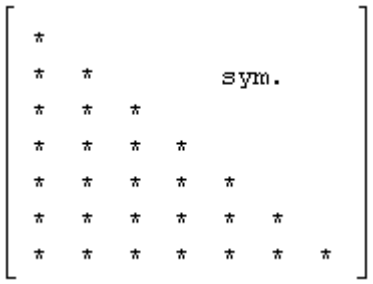
```
!
      call laipe$done

!
      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 square 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:



where the symbol * indicates non-zero fill-ins. Three types of subroutine are included:

laipe$Decompose_DSP_4
laipe$Decompose_DSP_8
laipe$Decompose_DSP_10
laipe$Decompose_DSP_16
laipe$Decompose_DSP_Z4
laipe$Decompose_DSP_Z8
laipe$Decompose_DSP_Z10
laipe$Decompose_DSP_Z16

laipe$Substitute_DSP_4
laipe$Substitute_DSP_8
laipe$Substitute_DSP_10
laipe$Substitute_DSP_16
laipe$Substitute_DSP_Z4
laipe$Substitute_DSP_Z8
laipe$Substitute_DSP_Z10
laipe$Substitute_DSP_Z16

laipe$Solution_DSP_4
laipe$Solution_DSP_8
laipe$Solution_DSP_10
laipe$Solution_DSP_16
laipe$Solution_DSP_Z4
laipe$Solution_DSP_Z8
laipe$Solution_DSP_Z10
laipe$Solution_DSP_Z16

4.2 Fortran Syntax for Subroutine Decompose

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

```
laipe$Decompose_DSP_4(A_io, N_i, Label_i, NoGood_o)
laipe$Decompose_DSP_8(A_io, N_i, Label_i, NoGood_o)
laipe$Decompose_DSP_10(A_io, N_i, Label_i, NoGood_o)
laipe$Decompose_DSP_16(A_io, N_i, Label_i, NoGood_o)
laipe$Decompose_DSP_Z4(A_io, N_i, Label_i, NoGood_o)
laipe$Decompose_DSP_Z8(A_io, N_i, Label_i, NoGood_o)
laipe$Decompose_DSP_Z10(A_io, N_i, Label_i, NoGood_o)
laipe$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=0. For the definition of profile, please see section 4.5.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument Label_i, a 4-byte INTEGER array of dimension(N_i), is the address reference label. For the definition of address reference label, please see section 4.6.
4. The argument NoGood_o, a 4-byte INTEGER, returns a flag. If NoGood_o=1, the input matrix [A] cannot be decomposed by the subroutine and [A] returns an unpredicted result; otherwise the profile A_io returns the decomposed matrix [L]. For the situation where NoGood_o=1, please see section 4.7.

4.3 Fortran Syntax for Subroutine Substitute

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

```
laipe$Substitute_DSP_4(A_i, N_i, Label_i, sX_io)
laipe$Substitute_DSP_8(A_i, N_i, Label_i, sX_io)
laipe$Substitute_DSP_10(A_i, N_i, Label_i, sX_io)
laipe$Substitute_DSP_16(A_i, N_i, Label_i, sX_io)
laipe$Substitute_DSP_Z4(A_i, N_i, Label_i, sX_io)
laipe$Substitute_DSP_Z8(A_i, N_i, Label_i, sX_io)
laipe$Substitute_DSP_Z10(A_i, N_i, Label_i, sX_io)
laipe$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, a 4-byte INTEGER, is the order of square matrix [A].

3. The argument Label_i, a 4-byte INTEGER array of dimension(N_i), 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 decompose [A] into the product of $[L][L]^T$, and then perform substitutions. Solve [A]{X}={B} in a single call. The syntax is as follows:

```
laipe$Solution_DSP_4(A_io, N_i, Label_i, X_io, NoGood_o)
laipe$Solution_DSP_8(A_io, N_i, Label_i, X_io, NoGood_o)
laipe$Solution_DSP_10(A_io, N_i, Label_i, X_io, NoGood_o)
laipe$Solution_DSP_16(A_io, N_i, Label_i, X_io, NoGood_o)
laipe$Solution_DSP_Z4(A_io, N_i, Label_i, X_io, NoGood_o)
laipe$Solution_DSP_Z8(A_io, N_i, Label_i, X_io, NoGood_o)
laipe$Solution_DSP_Z10(A_io, N_i, Label_i, X_io, NoGood_o)
laipe$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=0. For the definition of profile, please see section 4.5.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument Label_i, a 4-byte INTEGER array of dimension(N_i), 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=0.
5. The argument NoGood_o, a 4-byte INTEGER, returns a flag. If NoGood_o=1, the input system cannot be solved by the subroutine and [A] returns an unpredicted result; otherwise the profile A_io returns the decomposed matrix [L] and vector X_io returns the solution. For the situation where NoGood_o=1, 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 is a single precision profile for a matrix [A]. For other kinds of variable, profile must be properly declared. 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 (e. g., NoGood=1), the solver is not suitable for the problem.

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. In the following Fortran program, subroutines “Input” and “Output” demonstrate data storage scheme; Subroutine “DenseLabel” demonstrates equation (4.3) for address reference labels; Subroutine “laipe$Decompose_DSP_4” decomposes matrix [A]; Subroutine “laipe$Substitute_DSP_4” performs forward and backward substitutions to get {X}.

```
!  *** 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)
      INTEGER*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 laipe$Decompose_DSP_4(A,N,Label,NoGood)
!
!  stop if NoGood=1
!
      IF(NoGood.eq.1) STOP 'Cannot be decomposed'
!
!  perform substitutions in parallel
!
      CALL laipe$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 laipe$Done
```

```
!
      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 square 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
```

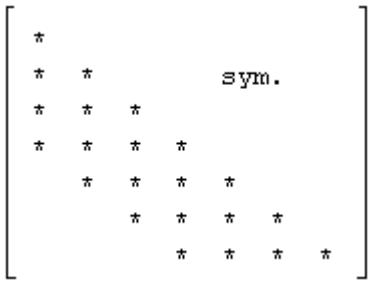
```
      A(3,Label(1))= 2.0
      A(4,Label(1))= 3.0
      A(5,Label(1))= 1.0
      A(6,Label(1))= 4.0
      A(7,Label(1))= 2.0
      A(2,Label(2))=25.0
      A(3,Label(2))=19.0
      A(4,Label(2))= 9.0
      A(5,Label(2))=-2.0
      A(6,Label(2))= 2.0
      A(7,Label(2))= 7.0
      A(3,Label(3))=44.0
      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 square 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:



Three types of subroutine are included:

laipe$Decompose_CSG_4
laipe$Decompose_CSG_8
laipe$Decompose_CSG_10
laipe$Decompose_CSG_16
laipe$Decompose_CSG_Z4
laipe$Decompose_CSG_Z8
laipe$Decompose_CSG_Z10
laipe$Decompose_CSG_Z16

laipe$Substitute_CSG_4
laipe$Substitute_CSG_8
laipe$Substitute_CSG_10
laipe$Substitute_CSG_16
laipe$Substitute_CSG_Z4
laipe$Substitute_CSG_Z8
laipe$Substitute_CSG_Z10
laipe$Substitute_CSG_Z16

laipe$Solution_CSG_4
laipe$Solution_CSG_8
laipe$Solution_CSG_10
laipe$Solution_CSG_16
laipe$Solution_CSG_Z4
laipe$Solution_CSG_Z8
laipe$Solution_CSG_Z10
laipe$Solution_CSG_Z16

laipe$meSolution_CSG_4
laipe$meSolution_CSG_8
laipe$meSolution_CSG_10
laipe$meSolution_CSG_16
laipe$meSolution_CSG_Z4
laipe$meSolution_CSG_Z8
laipe$meSolution_CSG_Z10
laipe$meSolution_CSG_Z16

The subroutines with "me", i.e., laipe$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:

laipe$Decompose_CSG_4(A_io, N_i, LowerBandwidth_i, NoGood_o)
laipe$Decompose_CSG_8(A_io, N_i, LowerBandwidth_i, NoGood_o)
laipe$Decompose_CSG_10(A_io, N_i, LowerBandwidth_i, NoGood_o)
laipe$Decompose_CSG_16(A_io, N_i, LowerBandwidth_i, NoGood_o)
laipe$Decompose_CSG_Z4(A_io, N_i, LowerBandwidth_i, NoGood_o)
laipe$Decompose_CSG_Z8(A_io, N_i, LowerBandwidth_i, NoGood_o)
laipe$Decompose_CSG_Z10(A_io, N_i, LowerBandwidth_i, NoGood_o)
laipe$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=0. For the definition of profile, please see section 5.6.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument LowerBandwidth_i, a 4-byte INTEGER, is the lower bandwidth of matrix [A].
4. The argument NoGood_o, a 4-byte INTEGER, returns a flag. If NoGood_o=1, the input matrix [A] cannot be decomposed and [A] returns an unpredicted result; Otherwise the profile A_io returns the decomposed matrices [L] and [D]. For the situation where NoGood_o=1, please see section 5.8.

5.3 Fortran Syntax for Subroutine Substitute

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

laipe$Substitute_CSG_4(A_i, N_i, LowerBandwidth_i, X_io)
laipe$Substitute_CSG_8(A_i, N_i, LowerBandwidth_i, X_io)
laipe$Substitute_CSG_10(A_i, N_i, LowerBandwidth_i, X_io)
laipe$Substitute_CSG_16(A_i, N_i, LowerBandwidth_i, X_io)
laipe$Substitute_CSG_Z4(A_i, N_i, LowerBandwidth_i, X_io)
laipe$Substitute_CSG_Z8(A_i, N_i, LowerBandwidth_i, X_io)

laipe$Substitute_CSG_Z10(A_i, N_i, LowerBandwidth_i, X_io)
laipe$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, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument LowerBandwidth_i, a 4-byte INTEGER, is the lower bandwidth of matrix [A].
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 decompose [A] into the product of $[L][D][L]^T$, and then perform substitutions. Solve [A]{X}={B} in a single call. The syntax is as follows:

laipe$Solution_CSG_4(A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)
laipe$Solution_CSG_8(A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)
laipe$Solution_CSG_10(A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)
laipe$Solution_CSG_16(A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)
laipe$Solution_CSG_Z4(A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)
laipe$Solution_CSG_Z8(A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)
laipe$Solution_CSG_Z10(A_io, N_i, LowerBandwidth_i, X_io, NoGood_o)
laipe$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=0. For the definition of profile, please see section 5.6.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument LowerBandwidth_i, a 4-byte INTEGER, is the lower bandwidth of matrix [A].
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=0.
5. The argument NoGood_o, a 4-byte INTEGER, returns a flag. If NoGood_o=1, the input system cannot be solved by the subroutine and [A] returns an unpredicted result; 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=1, 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:

laipe$meSolution_CSG_4(A_io,N_i,LowerBandwidth_i,X_io,Nset_i,WorkingSpace_x,NoGood_o)
laipe$meSolution_CSG_8(A_io,N_i,LowerBandwidth_i,X_io,Nset_i,WorkingSpace_x,NoGood_o)
laipe$meSolution_CSG_10(A_io,N_i,LowerBandwidth_i,X_io,Nset_i,WorkingSpace_x,NoGood_o)
laipe$meSolution_CSG_16(A_io,N_i,LowerBandwidth_i,X_io,Nset_i,WorkingSpace_x, NoGood_o)
laipe$meSolution_CSG_Z4(A_io,N_i,LowerBandwidth_i,X_io,Nset_i,WorkingSpace_x, NoGood_o)
laipe$meSolution_CSG_Z8(A_io,N_i,LowerBandwidth_i,X_io,Nset_i,WorkingSpace_x, NoGood_o)
laipe$meSolution_CSG_Z10(A_io,N_i,LowerBandwidth_i,X_io,Nset_i,WorkingSpace_x,NoGood_o)
laipe$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 unpredictable. For the definition of profile, please see section 5.6.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument LowerBandwidth_i, a 4-byte INTEGER, is the lower bandwidth of matrix [A].
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=0.
5. The argument Nset_i, a 4-byte INTEGER, 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 4-byte INTEGER, returns a flag. If NoGood_o=1, the input system cannot be solved by this function; otherwise the vector "X_io" returns the solution. For the situation NoGood_o=1, please see section 5.8.

5.6 Profile

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

$$\begin{bmatrix} * & & & & & & \\ * & * & & \text{sym.} & & & \\ * & * & * & & & & \\ * & * & * & * & & & \\ & * & * & * & * & & \\ & & * & * & * & * & \\ & & & * & * & * & * \\ & & & & \& & \& \\ & & & & & \& & \end{bmatrix} \qquad (5.1)$$

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 order of square matrix, 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 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 (e. g., NoGood=1), the solver is not suitable for the the problem.

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. In the following Fortran program, subroutines “Input” and “Output” demonstrate data storage scheme; Subroutine “laipe$Decompose_CSG_4” decomposes matrix [A]; Subroutine “laipe$Substitute_CSG_4” performs forward and backward substitutions to get {X}.

```
!  *** 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)
      INTEGER*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 laipe$Decompose_CSG_4(A,N,LowerBandwidth,NoGood)
!
!  stop if NoGood=1
!
      IF(NoGood.eq.1) STOP 'Cannot be decomposed'
!
!  perform substitutions in parallel
!
      CALL laipe$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 laipe$Done
!
      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 square 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 introduces 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:

which looks like a skyline in a city, and is sometimes called *skyline solver*. Three types of subroutine are included:

laipe$Decompose_VSG_4
laipe$Decompose_VSG_8
laipe$Decompose_VSG_10
laipe$Decompose_VSG_16
laipe$Decompose_VSG_Z4
laipe$Decompose_VSG_Z8
laipe$Decompose_VSG_Z10
laipe$Decompose_VSG_Z16

laipe$Substitute_VSG_4
laipe$Substitute_VSG_8
laipe$Substitute_VSG_10
laipe$Substitute_VSG_16
laipe$Substitute_VSG_Z4
laipe$Substitute_VSG_Z8
laipe$Substitute_VSG_Z10
laipe$Substitute_VSG_Z16

laipe$Solution_VSG_4
laipe$Solution_VSG_8
laipe$Solution_VSG_10
laipe$Solution_VSG_16
laipe$Solution_VSG_Z4
laipe$Solution_VSG_Z8
laipe$Solution_VSG_Z10
laipe$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:

```
laipe$Decompose_VSG_4(A_io, N_i, Label_i, NoGood_o)
laipe$Decompose_VSG_8(A_io, N_i, Label_i, NoGood_o)
laipe$Decompose_VSG_10(A_io, N_i, Label_i, NoGood_o)
laipe$Decompose_VSG_16(A_io, N_i, Label_i, NoGood_o)
laipe$Decompose_VSG_Z4(A_io, N_i, Label_i, NoGood_o)
laipe$Decompose_VSG_Z8(A_io, N_i, Label_i, NoGood_o)
laipe$Decompose_VSG_Z10(A_io, N_i, Label_i, NoGood_o)
laipe$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=0. For the definition of profile, please see section 6.5.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument Label_i, a 4-byte INTEGER array of dimension (N_i), is the address reference label. For the definition of address reference label, please see section 6.6.
4. The argument NoGood_o, a 4-byte INTEGER, returns a flag. If NoGood_o=1, the input matrix [A] cannot be decomposed and [A] returns an unpredicted result; otherwise the profile A_io returns the decomposed matrices [U] and [D]. For the situation where NoGood_o=1, please see section 6.7.

6.3 Fortran Syntax for Subroutine Substitute

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

```
laipe$Substitute_VSG_4(A_i, N_i, Label_i, X_io)
laipe$Substitute_VSG_8(A_i, N_i, Label_i, X_io)
laipe$Substitute_VSG_10(A_i, N_i, Label_i, X_io)
laipe$Substitute_VSG_16(A_i, N_i, Label_i, X_io)
laipe$Substitute_VSG_Z4(A_i, N_i, Label_i, X_io)
laipe$Substitute_VSG_Z8(A_i, N_i, Label_i, X_io)
laipe$Substitute_VSG_Z10(A_i, N_i, Label_i, X_io)
laipe$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, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument Label_i, a 4-byte INTEGER array of dimension(N_i), 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:

```
laipe$Solution_VSG_4(A_io, N_i, Label_i, X_io, NoGood_o)
laipe$Solution_VSG_8(A_io, N_i, Label_i, X_io, NoGood_o)
laipe$Solution_VSG_10(A_io, N_i, Label_i, X_io, NoGood_o)
laipe$Solution_VSG_16(A_io, N_i, Label_i, X_io, NoGood_o)
laipe$Solution_VSG_Z4(A_io, N_i, Label_i, X_io, NoGood_o)
laipe$Solution_VSG_Z8(A_io, N_i, Label_i, X_io, NoGood_o)
laipe$Solution_VSG_Z10(A_io, N_i, Label_i, X_io, NoGood_o)
laipe$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 NoGood_o=0. For the definition of profile, please see section 6.5.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument Label_i, a 4-byte INTEGER array of dimension (N_i), 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=0.
5. The argument NoGood_o, a 4-byte INTEGER, returns a flag. If NoGood_o=1, the input system cannot be solved by the subroutine and [A] returns an unpredicted result; 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=1, 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

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

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

(1) Set Label(1) = 1
(2) For i = 2 to N, do the following

$$\text{Label}(i) = \text{Label}(i-1) + [\text{ number of non-zero fill-ins above the diagonal in the i-th column }] \tag{6.3}$$

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:

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

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

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

6.7 Failure of Calling Request

If a calling request fails, the solver is not suitable for the problem.

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:

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

in which the order N=7. The following Fortran program, subroutines "Input" and "Output" demonstrate data storage scheme; Subroutine "laipe$Decompose_VSG_4" decomposes matrix [A]; The subroutine "laipe$Substitute_VSG_4" performs forward and backward substitutions to obtain {X}.

```
!  *** 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)
      INTEGER*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 laipe$Decompose_VSG_4(A,N,Label, NoGood)
!
!  stop if NoGood=1
!
      IF(NoGood.eq.1) STOP 'Cannot be decomposed'
!
!  perform substitutions in parallel
!
      CALL laipe$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 laipe$Done
!
      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 square 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:

where the symbol * indicates non-zero fill-ins. Three types of subroutine are included:

laipe$Decompose_DSG_4
laipe$Decompose_DSG_8
laipe$Decompose_DSG_10
laipe$Decompose_DSG_16
laipe$Decompose_DSG_Z4
laipe$Decompose_DSG_Z8
laipe$Decompose_DSG_Z10
laipe$Decompose_DSG_Z16

laipe$Substitute_DSG_4
laipe$Substitute_DSG_8
laipe$Substitute_DSG_10
laipe$Substitute_DSG_16
laipe$Substitute_DSG_Z4
laipe$Substitute_DSG_Z8
laipe$Substitute_DSG_Z10
laipe$Substitute_DSG_Z16

laipe$Solution_DSG_4
laipe$Solution_DSG_8
laipe$Solution_DSG_10
laipe$Solution_DSG_16
laipe$Solution_DSG_Z4
laipe$Solution_DSG_Z8
laipe$Solution_DSG_Z10
laipe$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:

```
laipe$Decompose_DSG_4(A_io, N_i, Label_i, NoGood_o)
laipe$Decompose_DSG_8(A_io, N_i, Label_i, NoGood_o)
laipe$Decompose_DSG_10(A_io, N_i, Label_i, NoGood_o)
laipe$Decompose_DSG_16(A_io, N_i, Label_i, NoGood_o)
laipe$Decompose_DSG_Z4(A_io, N_i, Label_i, NoGood_o)
laipe$Decompose_DSG_Z8(A_io, N_i, Label_i, NoGood_o)
laipe$Decompose_DSG_Z10(A_io, N_i, Label_i, NoGood_o)
laipe$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=0. For the definition of profile, please see section 7.5.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument Label_i, a 4-byte INTEGER array of dimension(N_i), is the address reference label. For the definition of address reference label, please see section 7.6.
4. The argument NoGood_o, a 4-byte INTEGER, returns a flag. If NoGood_o=1, the input matrix [A] cannot be decomposed and [A] returns an unpredicted result; otherwise the profile A_io returns the decomposed matrix [L]. For the situation where NoGood_o=1, please see section 7.7.

7.3 Fortran Syntax for Subroutine Substitute

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

```
laipe$Substitute_DSG_4(A_i, N_i, Label_i, X_io)
laipe$Substitute_DSG_8(A_i, N_i, Label_i, X_io)
laipe$Substitute_DSG_10(A_i, N_i, Label_i, X_io)
laipe$Substitute_DSG_16(A_i, N_i, Label_i, X_io)
laipe$Substitute_DSG_Z4(A_i, N_i, Label_i, X_io)
laipe$Substitute_DSG_Z8(A_i, N_i, Label_i, X_io)
laipe$Substitute_DSG_Z10(A_i, N_i, Label_i, X_io)
laipe$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, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument Label_i, a 4-byte INTEGER array of dimension(N_i), 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 decompose matrix [A] into the product of $[L][D][L]^T$, and then perform substitutions. Solve [A]{X}={B} in a single call. Syntax is as follows:

```
laipe$Solution_DSG_4(A_io, N_i, Label_i, X_io, NoGood_o)
laipe$Solution_DSG_8(A_io, N_i, Label_i, X_io, NoGood_o)
laipe$Solution_DSG_10(A_io, N_i, Label_i, X_io, NoGood_o)
laipe$Solution_DSG_16(A_io, N_i, Label_i, X_io, NoGood_o)
laipe$Solution_DSG_Z4(A_io, N_i, Label_i, X_io, NoGood_o)
laipe$Solution_DSG_Z8(A_io, N_i, Label_i, X_io, NoGood_o)
laipe$Solution_DSG_Z10(A_io, N_i, Label_i, X_io, NoGood_o)
laipe$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 NoGood_o=0. For the definition of profile, please see section 7.5.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument Label_i, a 4-byte INTEGER array of dimension (N_i), 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=0.
5. The argument NoGood_o, a 4-byte INTEGER, returns a flag. If NoGood_o=1, the input system cannot be solved by the subroutine and [A] returns an unpredicted result; otherwise the profile A_io returns the decomposed matrix [L], and vector X_io returns the solution. For the situation where NoGood_o=1, 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 order of square matrix.

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. 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 square 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 (e. g., NoGood=1), the solver is not suitable for the problem.

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. In the following Fortran program, subroutines “Input” and “Output” demonstrate data storage scheme; Subroutine “DenseLabel” demonstrates equation (7.3) for address reference labels; The subroutine “laipe$Decompose_DSG_4” decomposes matrix [A]; The subroutine “laipe$Substitute_DSG_4” performs forward and backward substitutions to get {X}.

```
!  *** 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)
      INTEGER*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 laipe$Decompose_DSG_4(A,N,Label,NoGood)
!
!  stop if NoGood=1
!
      IF(NoGood.eq.1) STOP 'Cannot be decomposed'
!
!  perform substitutions in parallel
!
      CALL laipe$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  laipe$Done
!
      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 square 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
```

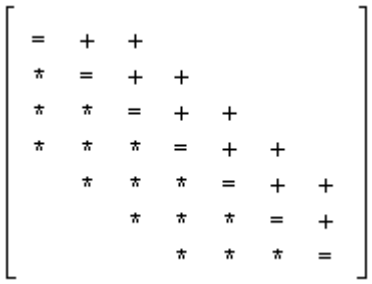
```
      A(3,Label(2))=29.0
      A(4,Label(2))= 9.0
      A(5,Label(2))=23.0
      A(6,Label(2))= 2.0
      A(7,Label(2))=27.0
      A(3,Label(3))= 4.0
      A(4,Label(3))=34.0
      A(5,Label(3))= 3.0
      A(6,Label(3))=22.0
      A(7,Label(3))= 3.0
      A(4,Label(4))= 8.0
      A(5,Label(4))=23.0
      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 square 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 introduces subroutines for the solution of [A]{X}={B} where the left side matrix [A] is asymmetric and of constant bandwidth. There is no consideration of definiteness of matrix [A]. The non-zero fill-ins of matrix [A] have a form, for example, as:



where the symbol "+" represents non-zero fill-ins in the upper triangular part, and the symbol "=" represents non-zero fill-ins on the diagonal, and the symbol "*" represents 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 included:

laipe$Decompose_CAG_4
laipe$Decompose_CAG_8
laipe$Decompose_CAG_10
laipe$Decompose_CAG_16
laipe$Decompose_CAG_Z4
laipe$Decompose_CAG_Z8
laipe$Decompose_CAG_Z10
laipe$Decompose_CAG_Z16

laipe$Substitute_CAG_4
laipe$Substitute_CAG_8
laipe$Substitute_CAG_10
laipe$Substitute_CAG_16
laipe$Substitute_CAG_Z4
laipe$Substitute_CAG_Z8
laipe$Substitute_CAG_Z10
laipe$Substitute_CAG_Z16

laipe$Solution_CAG_4
laipe$Solution_CAG_8
laipe$Solution_CAG_10
laipe$Solution_CAG_16

laipe$Solution_CAG_Z4
laipe$Solution_CAG_Z8
laipe$Solution_CAG_Z10
laipe$Solution_CAG_Z16

laipe$meSolution_CAG_4
laipe$meSolution_CAG_8
laipe$meSolution_CAG_10
laipe$meSolution_CAG_16
laipe$meSolution_CAG_Z4
laipe$meSolution_CAG_Z8
laipe$meSolution_CAG_Z10
laipe$meSolution_CAG_Z16

The subroutine names with "me", i.e., laipe$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:

laipe$Decompose_CAG_4(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i, NoGood_o)
laipe$Decompose_CAG_8(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i, NoGood_o)
laipe$Decompose_CAG_10(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i, NoGood_o)
laipe$Decompose_CAG_16(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i, NoGood_o)
laipe$Decompose_CAG_Z4(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i, NoGood_o)
laipe$Decompose_CAG_Z8(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i, NoGood_o)
laipe$Decompose_CAG_Z10(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i, NoGood_o)
laipe$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 NoGood_o=0. For the definition of profile, please see section 8.6.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument UpperBandwidth_i, a 4-byte INTEGER, is the upper bandwidth of matrix [A].
4. The argument LowerBandwidth_i, a 4-byte INTEGER, is the lower bandwidth of matrix [A].
5. The argument NoGood_o, a 4-byte INTEGER, returns a flag. If NoGood_o=1, the input matrix [A] cannot be decomposed and [A] returns an unpredicted result; otherwise the profile A_io returns the decomposed matrices [L] and [U]. For the situation where NoGood_o=1, please see section 8.8.

8.3 Fortran Syntax for Subroutine Substitute

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

```
laipe$Substitute_CAG_4(A_i, N_i, UpperBandwidth_i, LowerBandwidth_i, X_io)
laipe$Substitute_CAG_8(A_i, N_i, UpperBandwidth_i, LowerBandwidth_i, X_io)
laipe$Substitute_CAG_10(A_i, N_i, UpperBandwidth_i, LowerBandwidth_i, X_io)
laipe$Substitute_CAG_16(A_i, N_i, UpperBandwidth_i, LowerBandwidth_i, X_io)
laipe$Substitute_CAG_Z4(A_i, N_i, UpperBandwidth_i, LowerBandwidth_i, X_io)
laipe$Substitute_CAG_Z8(A_i, N_i, UpperBandwidth_i, LowerBandwidth_i, X_io)
laipe$Substitute_CAG_Z10(A_i, N_i, UpperBandwidth_i, LowerBandwidth_i, X_io)
laipe$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, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument UpperBandwidth_i, a 4-byte INTEGER, is the upper bandwidth of matrix [A].
4. The argument LowerBandwidth_i, a 4-byte INTEGER, is the lower bandwidth of matrix [A].
5. The argument X_io, array of dimension (N_i) 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:

```
laipe$Solution_CAG_4(A_io,N_i,UpperBandwidth_i,LowerBandwidth_i,X_io,NoGood_o)
laipe$Solution_CAG_8(A_io,N_i,UpperBandwidth_i,LowerBandwidth_i,X_io,NoGood_o)
laipe$Solution_CAG_10(A_io,N_i,UpperBandwidth_i,LowerBandwidth_i,X_io,NoGood_o)
laipe$Solution_CAG_16(A_io,N_i,UpperBandwidth_i,LowerBandwidth_i,X_io,NoGood_o)
laipe$Solution_CAG_Z4(A_io,N_i,UpperBandwidth_i,LowerBandwidth_i,X_io,NoGood_o)
laipe$Solution_CAG_Z8(A_io,N_i,UpperBandwidth_i,LowerBandwidth_i,X_io,NoGood_o)
laipe$Solution_CAG_Z10(A_io,N_i,UpperBandwidth_i,LowerBandwidth_i,X_io,NoGood_o)
laipe$Solution_CAG_Z16(A_io,N_i,UpperBandwidth_i,LowerBandwidth_i,X_io,NoGood_o)
```

where

1. The argument A_io, an 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 NoGood_o=0. For the definition of profile, please see section 8.6.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument UpperBandwidth_i, a 4-byte INTEGER, is the upper bandwidth of matrix [A].
4. The argument LowerBandwidth_i, a 4-byte INTEGER, is the lower bandwidth of matrix [A].
5. The argument X_io, an array of dimension (N_i) whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution if NoGood_o=0.
6. The argument NoGood_o, a 4-byte INTEGER, returns a flag. If NoGood_o=1, the input system cannot be solved by the subroutine and [A] returns an unpredicted result; 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=1, 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:

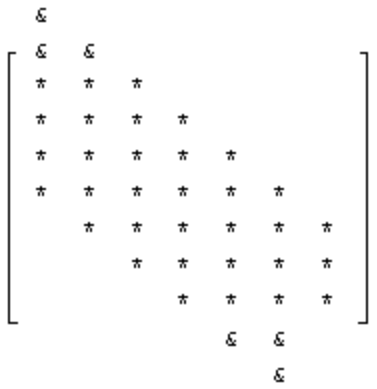
```
laipe$meSolution_CAG_4(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,  &
                       &  X_io, Nset_i, WorkingSpace_x, NoGood_o)
laipe$meSolution_CAG_8(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,  &
                       &  X_io, Nset_i, WorkingSpace_x, NoGood_o)
laipe$meSolution_CAG_10(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,  &
                       &  X_io, Nset_i, WorkingSpace_x, NoGood_o)
laipe$meSolution_CAG_16(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,  &
                       &  X_io, Nset_i, WorkingSpace_x, NoGood_o)
laipe$meSolution_CAG_Z4(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,  &
                       &  X_io, Nset_i, WorkingSpace_x, NoGood_o)
laipe$meSolution_CAG_Z8(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,  &
                       &  X_io, Nset_i, WorkingSpace_x, NoGood_o)
laipe$meSolution_CAG_Z10(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,  &
                       &  X_io, Nset_i, WorkingSpace_x, NoGood_o)
laipe$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, an 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 changed. For the definition of profile, please see section 8.6.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument UpperBandwidth_i, a 4-byte INTEGER, is the upper bandwidth of matrix [A].
4. The argument LowerBandwidth_i, a 4-byte INTEGER, is the lower bandwidth of matrix [A].
5. The argument X_io, an array of dimension (N_i) whose kind must be consistent with subroutine name convention, inputs the right side vector(s), and returns the solution if NoGood_o=0.
6. The argument Nset_i, a 4-byte INTEGER, is the number of right side vectors.
7. WorkingSpace_x, an array of dimension (N_i*(UpperBandwidth_i+LowerBandwidth_i)) whose kind must be consistent with subroutine name convention, is a working space.
8. The argument NoGood_o, a 4-byte INTEGER, returns a flag. If NoGood_o=1, the input system cannot be solved; otherwise the vector X_io returns the solution. For the situation where NoGood_o=1, 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 ignored. Total length of profile is determined as

$$\text{profile size} = N * (\text{UpperBandwidth} + \text{LowerBandwidth} + 1) - \text{LowerBandwidth} \tag{8.2}$$

where N is the order of square matrix, 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:

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

to the ending index as:

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

where N is the order of square matrix [A].

8. 8 Failure of Calling Request

If a calling request fails (e. g., NoGood=1), the solver is not suitable for the problem.

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. In the following Fortran program, subroutines "Input" and "Output" demonstrate data storage scheme; Subroutine "laipe$Decompose_CAG_4" decomposes matrix [A]; Subroutine "laipe$Substitute_CAG_4" performs forward and backward substitutions to solve {X}.

```
!  *** 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)
      INTEGER*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 laipe$Decompose_CAG_4(A,N,UpperBandwidth, LowerBandwidth, NoGood)
!
!  stop if NoGood=1
!
      IF(NoGood.eq.1) STOP 'Cannot be decomposed'
!
```

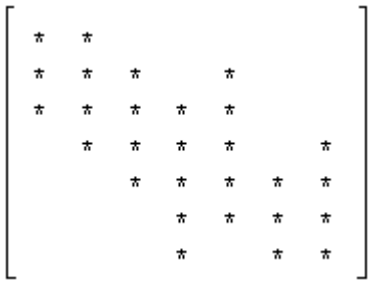
```
!  perform substitutions in parallel
!
      CALL laipe$Substitute_CAG_4(A,N,UpperBandwidth, LowerBandwidth,X)
!
!  output decomposed matrix
!
      CALL Output(A,N,UpperBandwidth,LowerBandwidth)
!
!  output the solution
!
      Write(*,'(" Solution is as:")')
      Write(*,*) X
!
!  laipe done
!
      call laipe$Done
!
      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 square 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 introduces subroutines for the solution of [A]{X}={B} where the left side matrix [A] is asymmetric and of variable bandwidth. There is no consideration of definiteness of matrix [A]. The non-zero fill-ins in the left side matrix [A] have a form, for example, as:



Three types of subroutine are included:

laipe$Decompose_VAG_4
laipe$Decompose_VAG_8
laipe$Decompose_VAG_10
laipe4Decompose_VAG_16
laipe$Decompose_VAG_Z4
laipe$Decompose_VAG_Z8
laipe$Decompose_VAG_Z10
laipe$Decompose_VAG_Z16

laipe$Substitute_VAG_4
laipe$Substitute_VAG_8
laipe$Substitute_VAG_10
laipe$Substitute_VAG_16
laipe$Substitute_VAG_Z4
laipe$Substitute_VAG_Z8
laipe$Substitute_VAG_Z10
laipe$Substitute_VAG_Z16

laipe$Solution_VAG_4
laipe$Solution_VAG_8
laipe$Solution_VAG_10
laipe$Solution_VAG_16
laipe$Solution_VAG_Z4
laipe$Solution_VAG_Z8
laipe$Solution_VAG_Z10
laipe$Solution_VAG_Z16

9.2 Fortran Syntax for Subroutine Decompose

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

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

where

1. The argument A_io, an 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 NoGood_o=0. For the definition of profile, please see section 9.5.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument Label_i, a 4-byte INTEGER array of dimension (N_i), is the address reference label. For the definition of address reference label, please see section 9.6.
4. The argument Last_i, a 4-byte INTEGER array of dimension(N_i), is the last entry to each column. For the definition of the last entry, please see section 9.6.
5. The argument NoGood_o, a 4-byte INTEGER, returns a flag. If NoGood_o=1, the input matrix [A] cannot be decomposed and [A] returns an unpredicted result; otherwise the profile A_io returns the decomposed matrices [L] and [U]. For the situation where NoGood_o=1, please see section 9.7.

9.3 Fortran Syntax for Subroutine Substitute

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

```
laipe$Substitute_VAG_4(A_i, N_i, Label_i, Last_i, X_io)
laipe$Substitute_VAG_8(A_i, N_i, Label_i, Last_i, X_io)
laipe$Substitute_VAG_10(A_i, N_i, Label_i, Last_i, X_io)
laipe$Substitute_VAG_16(A_i, N_i, Label_i, Last_i, X_io)
laipe$Substitute_VAG_Z4(A_i, N_i, Label_i, Last_i, X_io)
laipe$Substitute_VAG_Z8(A_i, N_i, Label_i, Last_i, X_io)
laipe$Substitute_VAG_Z10(A_i, N_i, Label_i, Last_i, X_io)
laipe$Substitute_VAG_Z16(A_i, N_i, Label_i, Last_i, X_io)
```

where

1. The argument A_i, an 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, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument Label_i, a 4-byte INTEGER array of dimension (N_i), is the address reference label. For the definition of address reference label, please see section 9.6.

4. The argument Last_i, a 4-byte INTEGER array of dimension (N_i), is the last entry of each column. For the definition of the last entry, please see section 9.6.
5. The argument X_io, an array of dimension (N_i) 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 decompose matrix [A] into the product of [L][U], and then perform substitutions. Solve the system [A]{X}={B} in a single call. Syntax is as follows:

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

where

1. The argument A_io, an 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=0. For the definition of profile, please see section 9.5.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument Label_i, a 4-byte INTEGER array of dimension (N_i), is the address reference label. For the definition of address reference label, please see section 9.6.
4. The argument Last_i, a 4-byte INTEGER array of dimension (N_i), is the last entry of column. For the definition of the last entry, please see section 9.6.
5. The argument X_io, an array of dimension (N_i) whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution if NoGood_o=0.
6. The argument NoGood_o, a 4-byte INTEGER, returns a flag. If NoGood_o=1, the input system cannot be solved; 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=1, please see section 9.7.

9.5 Profile

Profile for variable bandwidth and asymmetric matrix is more complex than the other profiles, and requires extra memory spaces in the lower triangular part. The profile for the upper triangular part simply hinges on the non-zero fill-ins. There are two variables for a determination of profile: *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

$$\text{Last}(I) = \text{Maximum of } (\text{Last}(I-1), \text{Ending}(I)) \tag{9.1}$$

The address reference label is then defined as:

1. Set Label(1) = 1
2. For I = 2 to N, do the following

$$\text{Label}(I) = \text{Label}(I-1) + \text{Last}(I-1) - \text{Beginning}(I) + 1 \tag{9.2}$$

The required length of profile is written as:

$$\text{profile size} = \text{Label}(N) - 1 + N \tag{9.3}$$

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

$$\begin{bmatrix} * & * & & & & & \\ * & * & * & & * & & \\ * & * & * & * & * & & \\ & * & * & * & * & & * \\ & & * & * & * & * & * \\ & & * & * & * & * & * \\ & & * & & & * & * \end{bmatrix} \tag{9.4}$$

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

$$\begin{bmatrix} = & = & & & & & \\ = & = & = & & = & & \\ = & = & = & = & = & & \\ & = & = & = & = & & = \\ & & = & = & = & = & = \\ & & = & = & = & = & = \\ & & = & = & = & = & = \end{bmatrix} \tag{9.5}$$

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).

Profile size for the solver is usually greater than the number of non-zero fill-ins. It can be seen, from (9.4) and (9.5), that the profile has two additional elements, A(7,4) and A(7,5). The additional elements must be initialized to zero, i.e., A(7,4)=0 and A(7,5)=0, before calling any of the following subroutines:

laipe$Decompose_VAG_4
laipe$Decompose_VAG_8
laipe$Decompose_VAG_10
laipe$Decompose_VAG_16
laipe$Decompose_VAG_Z4
laipe$Decompose_VAG_Z8

laipe$Decompose_VAG_Z10
laipe$Decompose_VAG_Z16

laipe$Solution_VAG_4
laipe$Solution_VAG_8
laipe$Solution_VAG_10
laipe$Solution_VAG_16
laipe$Solution_VAG_Z4
laipe$Solution_VAG_Z8
laipe$Solution_VAG_Z10
laipe$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 is a single precision profile. For other kinds of variable, profile must be properly declared. The coefficient A_{ij} of matrix [A] is programmed in a Fortran program as A(I,Label(J)).

Determination of profile considers the two variables: *beginning* and *ending*. The *beginning index* also can be determined as:

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

9.7 Failure of Calling Request

If a calling request fails (e. g., NoGood=1), the solver is not suitable for the problem.

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. In the following Fortran program, subroutines “Input” and “Output” demonstrate data storage scheme; Subroutine “laipe$Decompose_VAG_4” decomposes matrix [A]; Subroutine “laipe$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)
      INTEGER*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 laipe$Decompose_VAG_4(A,N,Label,Last,NoGood)
!
!  stop if NoGood=1
!
      IF(NoGood.eq.1) STOP 'Cannot be decomposed'
!
!  perform substitutions in parallel
!
      CALL laipe$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 laipe$Done
!
      STOP
      END
```

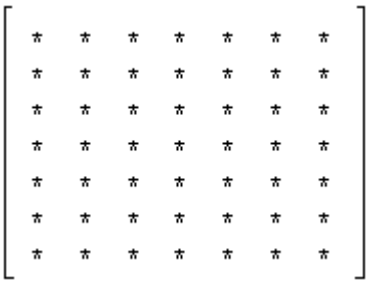
```
      SUBROUTINE Input(A,Label,Last,N)
!
!
!  routine to demonstrate an application of data storage scheme
!  (A)FORTRAN CALL: CALL Input(A,Label,Last,N)
!     1.A: <R4> profile of matrix [A], dimension(*)
!     2.Label: <I4> address reference labels, dimension(*)
!     3.Last: <I4> the last entry to each column, dimension(*)
!     4.N: <I4> order of square 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 square 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 introduces 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 form, for example, as:



where the symbol * indicates non-zero fill-ins. Three types of subroutine are included:

laipe$Decompose_DAG_4
laipe$Decompose_DAG_8
laipe$Decompose_DAG_10
laipe$Decompose_DAG_16
laipe$Decompose_DAG_Z4
laipe$Decompose_DAG_Z8
laipe$Decompose_DAG_Z10
laipe$Decompose_DAG_Z16

laipe$Substitute_DAG_4
laipe$Substitute_DAG_8
laipe$Substitute_DAG_10
laipe$Substitute_DAG_16
laipe$Substitute_DAG_Z4
laipe$Substitute_DAG_Z8
laipe$Substitute_DAG_Z10
laipe$Substitute_DAG_Z16

laipe$Solution_DAG_4
laipe$Solution_DAG_8
laipe$Solution_DAG_10
laipe$Solution_DAG_16
laipe$Solution_DAG_Z4
laipe$Solution_DAG_Z8
laipe$Solution_DAG_Z10
laipe$Solution_DAG_Z16

10.2 Fortran Syntax for Subroutine Decompose

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

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

where

1. The argument A_io, an 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=0. For the definition of profile, please see section 10.5.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument NoGood_o, a 4-byte INTEGER, returns a flag. If NoGood_o=1, the input matrix [A] cannot be decomposed and [A] returns an unpredicted result; otherwise the profile A_io returns the decomposed matrices [L] and [U]. For the situation where NoGood_o=1, please see section 10.7.

10.3 Fortran Syntax for Subroutine Substitute

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

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

where

1. The argument A_i, an 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, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument X_io, an array of dimension (N_i) 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 decompose matrix [A] into the product of [L][U], and then perform substitutions. Solve [A]{X}={B} in a single call. The syntax is as follows:

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

where

1. The argument A_io, an 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 NoGood_o=0. For the definition of profile, please see section 10.5.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument X_io, array of dimension (N_i) whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution if NoGood_o=0.
4. The argument NoGood_o, a 4-byte INTEGER, returns a flag. If NoGood_o=1, the input system cannot be solved by the subroutine and [A] returns an unpredicted result; 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=1, please see section 10.7.

10.5 Profile

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

$$\begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix} \qquad (10.1)$$

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

$$\text{profile size} = N * N \qquad (10.2)$$

where N is the order of square matrix.

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 is a single precision profile, and N is the order of square matrix. 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 (e. g., NoGood=1), the solver is not suitable for the problem.

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. In the following Fortran program, subroutines “Input” and “Output” demonstrate data storage scheme; Subroutine “laipe$Decompose_DAG_4” decomposes matrix [A]; Subroutine “laipe$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)
      INTEGER*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 laipe$Decompose_DAG_4(A,N,NoGood)
!
!  stop if NoGood=1
!
      IF(NoGood.eq.1) STOP 'Cannot be decomposed'
!
!  perform substitutions in parallel
!
      CALL laipe$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 laipe$Done
!
      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 square 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
```

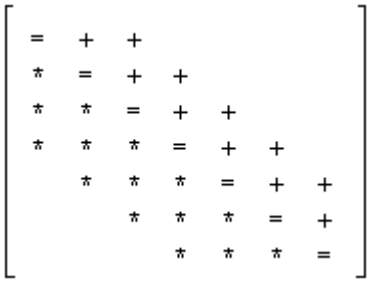
```
!
      A(1,2)= 2.0
      A(2,2)= 5.0
      A(3,2)=29.0
      A(4,2)= 9.0
      A(5,2)=23.0
      A(6,2)= 2.0
      A(7,2)=27.0
!
!  third column
!
      A(1,3)=13.0
      A(2,3)= 3.0
      A(3,3)= 4.0
      A(4,3)=34.0
      A(5,3)= 3.0
      A(6,3)=22.0
      A(7,3)= 3.0
!
!  fourth column
!
      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 square 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 introduces 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 form, for example, as:



where the symbol "+" represents non-zero fill-ins in the upper triangular part, and the symbol "=" represents non-zero fill-ins on the diagonal, and the symbol "*" represents 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 included:

laipe$ppDecompose_CAG_4
laipe$ppDecompose_CAG_8
laipe$ppDecompose_CAG_10
laipe$ppDecompose_CAG_16
laipe$ppDecompose_CAG_Z4
laipe$ppDecompose_CAG_Z8
laipe$ppDecompose_CAG_Z10
laipe$ppDecompose_CAG_Z16

laipe$ppSubstitute_CAG_4
laipe$ppSubstitute_CAG_8
laipe$ppSubstitute_CAG_10
laipe$ppSubstitute_CAG_16
laipe$ppSubstitute_CAG_Z4
laipe$ppSubstitute_CAG_Z8
laipe$ppSubstitute_CAG_Z10
laipe$ppSubstitute_CAG_Z16

laipe$ppSolution_CAG_4
laipe$ppSolution_CAG_8
laipe$ppSolution_CAG_10

```
laipe$ppSolution_CAG_16
laipe$ppSolution_CAG_Z4
laipe$ppSolution_CAG_Z8
laipe$ppSolution_CAG_Z10
laipe$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:

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

where

1. The argument A_io, an 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 NoGood_o=0. For the definition of profile, please see section 11.5.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument UpperBandwidth_i, a 4-byte INTEGER, 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, a 4-byte INTEGER, 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, a 4-byte INTEGER array of dimension (N_i), returns the row index where the remaining elements in a row are from if NoGood_o=0.
6. The argument First_o, a 4-byte INTEGER array of dimension (N_i), returns the index of the first non-zero fill-in on each column if NoGood_o=0.
7. The argument NoGood_o, a 4-byte INTEGER, returns a flag. If NoGood_o=1., the input matrix [A] cannot be decomposed; otherwise the profile A_io returns the decomposed matrices [L] and [U]. For the situation where NoGood_o=1, please see section 11.7.

11.3 Fortran Syntax for Subroutine ppSubstitute

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

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

where

1. The argument A_i, an 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, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument UpperBandwidth_i, a 4-byte INTEGER, 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, a 4-byte INTEGER, 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, a 4-byte INTEGER array of dimension (N_i), inputs the row index from decomposition.
6. The argument First_i, a 4-byte INTEGER array of dimension(N_i), inputs the index of the first nonzero fill-in from decomposition.
7. The argument X_io, an array of dimension (N_i) 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:

```
laipe$ppSolution_CAG_4(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,    &
                       &   From_x, First_x, X_io, NoGood_o)
laipe$ppSolution_CAG_8(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,    &
                       &   From_x,  First_x, X_io, NoGood_o)
laipe$ppSolution_CAG_10(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,    &
                       &   From_x, First_x, X_io, NoGood_o)
laipe$ppSolution_CAG_16(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,    &
                       &   From_x, First_x, X_io, NoGood_o)
laipe$ppSolution_CAG_Z4(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,    &
                       &   From_x, First_x, X_io, NoGood_o)
laipe$ppSolution_CAG_Z8(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,     &
                       &   From_x, First_x, X_io, NoGood_o)
laipe$ppSolution_CAG_Z10(A_io, N_i, UpperBandwidth_i, LowerBandwidth_i,    &
                       &   From_x, First_x, X_io, NoGood_o)
laipe$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, an 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 NoGood_o=0. For the definition of profile, please see section 11.5.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument UpperBandwidth_i, a 4-byte INTEGER, is the upper bandwidth of matrix [A].
4. The argument LowerBandwidth_i, a 4-byte INTEGER, is the lower bandwidth of matrix [A].
5. The argument From_x, a 4-byte INTEGER array of dimension (N_i), is a working array.
6. The argument First_x, a 4-byte INTEGER array of dimension (N_i), is a working array.
7. The argument X_io, an array of dimension (N_i) whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution if NoGood_o=0.
8. The argument NoGood_o, a 4-byte INTEGER, returns a flag. If NoGood_o=1, the input system cannot be solved by the subroutine; 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=1, please see section 11.7.

11.5 Profile

Non-zero fillins of a constant-bandwidth and asymmetric matrix is, for example, as:

```
 ┌                       ┐
 │ =  +                  │
 │ *  =  +               │
 │ *  *  =  +            │
 │    *  *  =  +         │     (11.1)
 │       *  *  =  +      │
 │          *  *  =  +   │
 │             *  *  =   │
 └                       ┘
```

where the symbol "+" represents non-zero fill-ins in the upper triangular part, and the symbol "=" represents non-zero fill-ins on the diagonal, and the symbol "*" represents 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 lower profile is defined by the non-zero fill-ins, but the upper profile extents the bandwidth of lower part. The profile for the form (11.1) is written as follows:

```
   &
   &  &
 ┌ &  &  &              ┐
 │ =  +  %  %           │
 │ *  =  +  %  %        │
 │ *  *  =  +  %  %     │
 │    *  *  =  +  %  %  │     (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.
2. The symbol "=" represents non-zero fill-ins on the diagonal.
3. The symbol "*" represents non-zero fill-ins in the lower triangular part.
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

```
laipe$ppDecompose_CAG_4
laipe$ppDecompose_CAG_8
laipe$ppDecompose_CAG_10
laipe$ppDecompose_CAG_16
laipe$ppDecompose_CAG_Z4
laipe$ppDecompose_CAG_Z8
laipe$ppDecompose_CAG_Z10
laipe$ppDecompose_CAG_Z16

laipe$ppSolution_CAG_4
laipe$ppSolution_CAG_8
laipe$ppSolution_CAG_10
laipe$ppSolution_CAG_16
laipe$ppSolution_CAG_Z4
laipe$ppSolution_CAG_Z8
laipe$ppSolution_CAG_Z10
laipe$ppSolution_CAG_Z16
```

5. The symbol & indicates an extra memory space whose content is ignored.

Total length of profile is determined as

$$\text{profile size} = N * (\text{UpperBandwidth} + \text{LowerBandwidth} * 2 + 1) - \text{LowerBandwidth} \qquad (11.3)$$

where N is the order of square matrix, 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 square 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 (e.g., NoGood=1), the solver is not suitable for the problem.

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. In the following Fortran program, subroutines “Input” and “Output” demonstrate data storage scheme; subroutine “laipe$ppDecompose_CAG_4” decomposes matrix [A] with partial pivoting; subroutine “laipe$ppSubstitute_CAG_4” performs forward and backward substitutions to solve {X}.

```
! *** 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)
      INTEGER*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 laipe$ppDecompose_CAG_4(A,N,UpperBandwidth, LowerBandwidth,  &
                                                  From, First, NoGood)
!
!  stop if NoGood=1
!
      IF(NoGood.eq.1) STOP 'Cannot be decomposed'
!
!  perform substitutions in parallel
!
      CALL laipe$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 laipe$Done
!
      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)
```

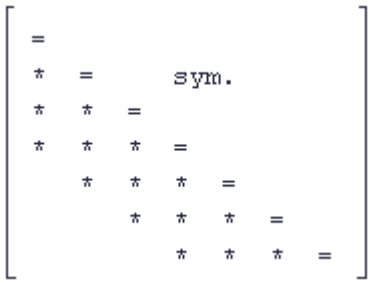
```
!    1.A: <R4> profile of matrix [A], dimension(*)
!    2.Upper: <I4> upper bandwidth
!    3.Lower: <I4> lower bandwidth
!    4.N: <I4> order of square matrix
!
! dummy arguments
!
      INTEGER*4 Upper,Lower,N
      REAL*4 A(1-Upper-Lower:Lower,1)
!
! initialize
!   The ending bound of I4TEMP is 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 square 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 introduces 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 form, for example, as:



where the symbol "=" represents non-zero fill-ins on the diagonal, and the symbol "*" represents non-zero fill-ins in the lower triangular part.

After decomposition, generally matrix [A] is not symmetric. When applying the subroutines in this chapter, just input the lower triangular part of the original matrix. Three types of subroutine are included:

laipe$ppDecompose_CSP_4
laipe$ppDecompose_CSP_8
laipe$ppDecompose_CSP_10
laipe$ppDecompose_CSP_16
laipe$ppDecompose_CSP_Z4
laipe$ppDecompose_CSP_Z8
laipe$ppDecompose_CSP_Z10
laipe$ppDecompose_CSP_Z16

laipe$ppSubstitute_CSP_4
laipe$ppSubstitute_CSP_8
laipe$ppSubstitute_CSP_10
laipe$ppSubstitute_CSP_16
laipe$ppSubstitute_CSP_Z4
laipe$ppSubstitute_CSP_Z8
laipe$ppSubstitute_CSP_Z10
laipe$ppSubstitute_CSP_Z16

laipe$ppSolution_CSP_4

laipe$ppSolution_CSP_8
laipe$ppSolution_CSP_10
laipe$ppSolution_CSP_16
laipe$ppSolution_CSP_Z4
laipe$ppSolution_CSP_Z8
laipe$ppSolution_CSP_Z10
laipe$ppSolution_CSP_Z16

The solvers with partial pivoting are not included in LAIPE2.

12.2 Fortran Syntax for Subroutine ppDecompose

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

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

where

1. The argument A_io, an 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 NoGood_o=0. For the definition of profile, please see section 12.5.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument LowerBandwidth_i, a 4-byte INTEGER, is the lower bandwidth of matrix [A].
4. The argument From_o, a 4-byte INTEGER array of dimension (N_i), returns the row index where the remaining elements are from if NoGood_o=0.
5. The argument First_o, a 4-byte INTEGER array of dimension (N_i), returns the index of the first nonzero fill-in on each column if NoGood_o=0.
6. The argument NoGood_o, a 4-byte INTEGER, returns a flag. If NoGood_o=1, the input matrix [A] cannot be decomposed; otherwise the profile A_io returns the decomposed matrices [L] and [U]. For the situation where NoGood_o=1, please see section 12.7.

12.3 Fortran Syntax for Subroutine ppSubstitute

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

```
laipe$ppSubstitute_CSP_4(A_i, N_i, LowerBandwidth_i, From_i, First_i, X_io)
laipe$ppSubstitute_CSP_8(A_i, N_i, LowerBandwidth_i, From_i, First_i, X_io)
laipe$ppSubstitute_CSP_10(A_i, N_i, LowerBandwidth_i, From_i, First_i, X_io)
```

```
laipe$ppSubstitute_CSP_16(A_i, N_i, LowerBandwidth_i, From_i, First_i, X_io)
laipe$ppSubstitute_CSP_Z4(A_i, N_i, LowerBandwidth_i, From_i, First_i, X_io)
laipe$ppSubstitute_CSP_Z8(A_i, N_i, LowerBandwidth_i, From_i, First_i, X_io)
laipe$ppSubstitute_CSP_Z10(A_i, N_i, LowerBandwidth_i, From_i, First_i, X_io)
laipe$ppSubstitute_CSP_Z16(A_i, N_i, LowerBandwidth_i, From_i, First_i, X_io)
```

where

1. The argument A_i, an 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, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument LowerBandwidth_i, a 4-byte INTEGER, is the lower bandwidth of matrix [A].
4. The argument From_i, a 4-byte INTEGER array of dimension (N_i), inputs the row index from decomposition.
5. The argument First_i, a 4-byte INTEGER array of dimension (N_i), inputs the index of the first non-zero fill-in from decomposition.
6. The argument X_io, an array of dimension (N_i) 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:

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

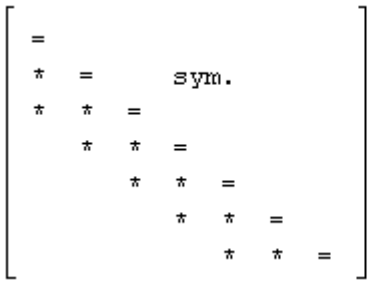
where

1. The argument A_io, an 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 NoGood_o=0. For the definition of profile, please see section 12.5.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument LowerBandwidth_i, a 4-byte INTEGER, is the lower bandwidth of matrix [A].
4. The argument From_x, a 4-byte INTEGER array of dimension (N_i), is a working array.
5. The argument First_x, a 4-byte INTEGER array of dimension (N_i), is a working array.
6. The argument X_io, an array of dimension (N-i) whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution if NoGood_o=0.
7. The argument NoGood_o, a 4-byte INTEGER, returns a flag. If NoGood_o=1, the input system cannot be solved by the subroutine; 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=1, please see section 12.7.

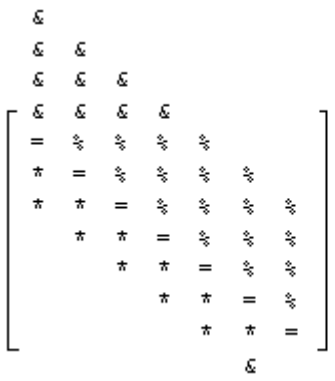
12.5 Profile

Non-zero fill-ins of a constant-bandwidth and symmetric matrix could be, for example, in the following form:



(12.1)

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



(12.2)

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

1. The symbol "=" represents non-zero fill-ins on the diagonal.
2. The symbol "*" represents non-zero fill-ins in the lower triangular part.
3. The symbol "%" represents profile of the upper part. It is unnecessary to initialize the space denoted by the symbol "%".
4. The symbol "&"indicates an extra memory space whose content is ignored.

Total length of profile is determined as

$$\text{profile size} = N * (\text{LowerBandwidth} * 3 + 1) - \text{LowerBandwidth} \qquad (12.3)$$

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

Maximum of (1,i-LowerBandwidth*2) (12.6)

12.7 Failure of Calling Request

If the calling request fails (e.g., NoGood=1), the solver is suitable for the problem.

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. In the following Fortran program, the subroutines "Input" and "Output" demonstrate data storage scheme; The subroutine "laipe$ppDecompose_CSP_4" decomposes matrix [A]; The subroutine "laipe$ppSubstitute_CSP_4" performs substitutions to solve {X}.

```
!  *** 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)
      INTEGER*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 laipe$ppDecompose_CSP_4(A,N,LowerBandwidth, From, First, NoGood)
!
!  stop if NoGood=1
!
      IF(NoGood.eq.1) STOP 'Cannot be decomposed'
!
!  perform substitutions in parallel
!
      CALL laipe$ppSubstitute_CSP_4(A,N,LowerBandwidth,From,First, X)
!
!  output decomposed matrix
!
       CALL Output(A,N,LowerBandwidth)
```

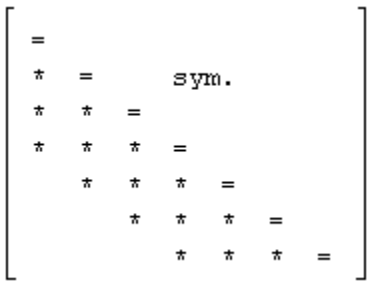
```
!
!  output the solution
!
      Write(*,'(" Solution is as:")')
      Write(*,*) X
!
!  laipe done
!
      call laipe$Done
!
      STOP
      END
      SUBROUTINE Input(A,Lower,N)
!
!
!  routine to demonstrate an application of data storage scheme
!  (A)FORTRAN CALL: CALL Input(A,Lower,N)
!     1.A: <R4> profile of matrix [A], dimension(*)
!     2.Lower: <I4> lower bandwidth
!     3.N: <I4> order of square 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 square 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 introduces 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 form, for example, as:



where the symbol "=" represents non-zero fill-ins on the diagonal, and the symbol "*" represents 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. After being decomposed, the upper triangular is different from the lower triangular. When applying the subroutines, just input the lower part of the original matrix. Three types of subroutine are included:

laipe$ppDecompose_CSG_4
laipe$ppDecompose_CSG_8
laipe$ppDecompose_CSG_10
laipe$ppDecompose_CSG_16
laipe$ppDecompose_CSG_Z4
laipe$ppDecompose_CSG_Z8
laipe$ppDecompose_CSG_Z10
laipe$ppDecompose_CSG_Z16

laipe$ppSubstitute_CSG_4
laipe$ppSubstitute_CSG_8
laipe$ppSubstitute_CSG_10
laipe$ppSubstitute_CSG_16
laipe$ppSubstitute_CSG_Z4
laipe$ppSubstitute_CSG_Z8
laipe$ppSubstitute_CSG_Z10
laipe$ppSubstitute_CSG_Z16

laipe$ppSolution_CSG_4

laipe$ppSolution_CSG_8
laipe$ppSolution_CSG_10
laipe$ppSolution_CSG_16
laipe$ppSolution_CSG_Z4
laipe$ppSolution_CSG_Z8
laipe$ppSolution_CSG_Z10
laipe$ppSolution_CSG_Z16

The solvers are not included in LAIPE2.

13.2 Fortran Syntax for Subroutine ppDecompose

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

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

where

1. The argument A_io, an 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 NoGood_o=0. For the definition of profile, please see section 13.5.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument LowerBandwidth_i, a 4-byte INTEGER, is the lower bandwidth of matrix [A].
4. The argument From_o, a 4-byte INTEGER array of dimension (N_i), returns the row index where the remaining elements are from if NoGood_o=0.
5. The argument First_o, a 4-byte INTEGER array of dimension (N_i), returns the index of the first nonzero fill-in on each column if NoGood_o=0.
6. The argument NoGood_o, a 4-byte INTEGER, returns a flag. If NoGood_o=1, the input matrix [A] cannot be decomposed; otherwise the profile A_io returns the decomposed matrices [L] and [U]. For the situation where NoGood_o=1, please see section 13.7.

13.3 Fortran Syntax for Subroutine ppSubstitute

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

laipe$ppSubstitute_CSG_4(A_i,N_i,LowerBandwidth_i,From,_i,First_i,X_io)
laipe$ppSubstitute_CSG_8(A_i,N_i,LowerBandwidth_i,From_i,First_i,X_io)
laipe$ppSubstitute_CSG_10(A_i,N_i,LowerBandwidth_i,From_i,First_i,X_io)
laipe$ppSubstitute_CSG_16(A_i,N_i,LowerBandwidth_i,From_i,First_i,X_io)

```
laipe$ppSubstitute_CSG_Z4(A_i,N_i,LowerBandwidth_I,From_i,First_i,X_io)
laipe$ppSubstitute_CSG_Z8(A_i,N_i,LowerBandwidth_i,From_i,First_i,X_io)
laipe$ppSubstitute_CSG_Z10(A_i,N_i,LowerBandwidth_i,From_i,First_i,X_io)
laipe$ppSubstitute_CSG_Z16(A_i,N_i,LowerBandwidth_i,From_i,First_i,X_io)
```

where

1. The argument A_i, an 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, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument LowerBandwidth_i, a 4-byte INTEGER, is the lower bandwidth of matrix [A].
4. The argument From_i, a 4-byte INTEGER array of dimension (N_i), inputs the row index from decomposition.
5. The argument First_i, a 4-byte INTEGER array of dimension (N_i), inputs the index of the first nonzero fill-in from decomposition.
6. The argument X_io, an array of dimension (N_i) 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:

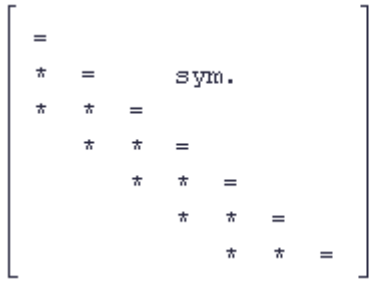
```
laipe$ppSolution_CSG_4(A_io, N_i, LowerBandwidth_i, From_x, First_x, X_io, NoGood_o)
laipe$ppSolution_CSG_8(A_io, N_i, LowerBandwidth_i, From_x, First_x, X_io, NoGood_o)
laipe$ppSolution_CSG_10(A_io, N_i, LowerBandwidth_i, From_x, First_x, X_io, NoGood_o)
laipe$ppSolution_CSG_16(A_io, N_i, LowerBandwidth_i,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

where

1. The argument A_io, an 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 NoGood_o=0. For the definition of profile, please see section 13.5.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument LowerBandwidth_i, a 4-byte INTEGER, is the lower bandwidth of matrix [A].
4. The argument From_x, a 4-byte INTEGER array of dimension (N_i), is a working array.
5. The argument First_x, a 4-byte INTEGER array of dimension (N_i), is a working array.
6. The argument X_io, an array of dimension (N_i) whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution if NoGood_o=0.
7. The argument NoGood_o, a 4-byte INTEGER, returns a flag. If NoGood_o=1, the input system cannot be solved; 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=1, please see section 13.7.

13.5 Profile

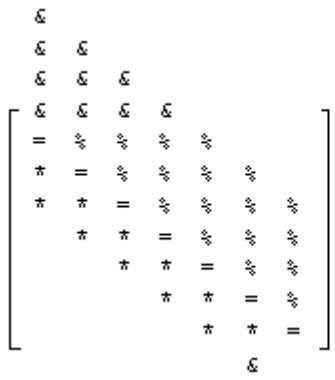
Non-zero fillings of a constant-bandwidth and symmetric matrix could be in a form, for example, as follows.



(13.1)

where the symbol "=" represents non-zero fill-ins on the diagonal, and the symbol "*" represents non-zero fill-ins in the lower triangular part. For the matrix in the form of (13.1), the lower bandwidth is 2, and the upper bandwidth is also 2.

The profile for the lower part is defined by the non-zero fill-ins, but the profile for the upper part increases by the lower bandwidth. The profile for the non-zero fillings in form (13.1) is defined as:



(13.2)

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

1. The symbol "=" represents non-zero fill-ins on the diagonal.
2. The symbol "*" represents non-zero fill-ins in the lower triangular part.
3. The symbol "%" represents the upper profile. It is unnecessary to initialize the space denoted by the symbol %.
4. The symbol "&" represents an extra memory space whose content is ignored.

Total length of profile is determined as

$$\text{profile size} = N * (\text{LowerBandwidth} * 3 + 1) - \text{LowerBandwidth} \qquad (13.3)$$

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

13.6 Data Storage Scheme

Data storage scheme for the solver 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 of matrix [A], and the variable "LowerBandwidth" is the lower bandwidth of the matrix. For other kinds of variable, profile must be properly declared. 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 square matrix [A]. "After decomposition", the beginning index in the i-th column is defend as

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

13.7 Failure of Calling Request

If the calling request fails (e. g., NoGood=1), the solver is not suitable for the problem.

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. In the following Fortran program, the subroutines "Input" and "Output" demonstrate data storage scheme; The subroutine "laipe$ppDecompose_CSG_4" decomposes matrix [A]; The subroutine "laipe$ppSubstitute_CSG_4" performs substitutions to solve {X}.

```
!  *** 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)
      INTEGER*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 laipe$ppDecompose_CSG_4(A,N, LowerBandwidth, From, First, NoGood)
!
!  stop if NoGood=1
!
      IF(NoGood.eq.1) STOP 'Cannot be decomposed'
!
!  perform substitutions in parallel
!
      CALL laipe$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 laipe$Done
!
      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 square 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 square 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 introduces 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 the simplest form:

where the symbol "*" represents non-zero fill-ins. Three types of subroutine are inckuded:

laipe$ppDecompose_DAG_4
laipe$ppDecompose_DAG_8
laipe$ppDecompose_DAG_10
laipe$ppDecompose_DAG_16
laipe$ppDecompose_DAG_Z4
laipe$ppDecompose_DAG_Z8
laipe$ppDecompose_DAG_Z10
laipe$ppDecompose_DAG_Z16

laipe$ppSubstitute_DAG_4
laipe$ppSubstitute_DAG_8
laipe$ppSubstitute_DAG_10
laipe$ppSubstitute_DAG_16
laipe$ppSubstitute_DAG_Z4
laipe$ppSubstitute_DAG_Z8
laipe$ppSubstitute_DAG_Z10
laipe$ppSubstitute_DAG_Z16

laipe$ppSolution_DAG_4
laipe$ppSolution_DAG_8
laipe$ppSolution_DAG_10
laipe$ppSolution_DAG_16
laipe$ppSolution_DAG_Z4
laipe$ppSolution_DAG_Z8
laipe$ppSolution_DAG_Z10
laipe$ppSolution_DAG_Z16

The solvers are not included in LAIPE2.

14.2 Fortran Syntax for Subroutine ppDecompose

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

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

where

1. The argument A_io, an 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 NoGood_o=0. For the definition of profile, please see section 14.5.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument RowOrder_io, a 4-byte INTEGER array of dimension (N_i), enters a sequence of consecutive numbers from one to N_i and returns the pivoting rows if NoGood_o=0.
4. The argument NoGood_o, a 4-byte INTEGER, returns a flag. If NoGood_o=1, the input matrix [A] cannot be decomposed; otherwise the profile A_io returns the decomposed matrices [L] and [U]. For the situation where NoGood_o=1, please see section 14.7.

14.3 Fortran Syntax for Subroutine ppSubstitute

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

```
laipe$ppSubstitute_DAG_4(A_i, N_i, From_i, X_io)
laipe$ppSubstitute_DAG_8(A_i, N_i, From_i, X_io)
laipe$ppSubstitute_DAG_10(A_i, N_i, From_i, X_io)
laipe$ppSubstitute_DAG_16(A_i, N_i, From_i, X_io)
laipe$ppSubstitute_DAG_Z4(A_i, N_i, From_i, X_io)
laipe$ppSubstitute_DAG_Z8(A_i, N_i, From_i, X_io)
laipe$ppSubstitute_DAG_Z10(A_i, N_i, From_i, X_io)
laipe$ppSubstitute_DAG_Z16(A_i, N_i, From_i, X_io)
```

where

1. The argument A_i, an 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, a 4-byte INTEGER, is the order of square matrix [A].

3. The argument From_i, a 4-byte INTEGER array of dimension (N_i), inputs the pivoting rows from decomposition.
4. The argument X_io, an array of dimension (N_i) 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:

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

where

1. The argument A_io, an 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 NoGood_o=0. For the definition of profile, please see section 14.5.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument RowOrder_io, a 4-byte INTEGER array of dimension (N_i), enters consecutive numbers from one to N_i and returns the pivoting rows if NoGood_o=0.
4. The argument X_io, an array of dimension (N_i) whose kind must be consistent with subroutine name convention, inputs the right side vector, and returns the solution if NoGood_o=0.
5. The argument NoGood_o, a 4-byte INTEGER, returns a flag. If NoGood_o=1, the input system cannot be solved by the subroutine; 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=1, please see section 14.7.

14.5 Profile

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

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

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

$$\text{profile size} = N * N \tag{14.2}$$

where N is the order of square matrix.

14.6 Data Storage Scheme

Data storage scheme for the solver 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 order of square matrix. For other kinds of variable, profile must be properly declared.

14.7 Failure of Calling Request

If a calling request fails (e. g. NoGood=1), the solver is not suitable for the problem.

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. In the following Fortran program, the subroutines “Input” and “Output” demonstrate data storage scheme; The subroutine “laipe$ppDecompose_DAG_4” decomposes matrix [A] with partial pivoting; The subroutine “laipe$ppSubstitute_DAG_4” performs forward and backward substitutions to solve {X}.

```
!  *** 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)
      INTEGER*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 laipe$ppDecompose_DAG_4(A,N,RowOrder,NoGood)
!
!  stop if NoGood=1
!
      IF(NoGood.eq.1) STOP 'Cannot be decomposed'
!
!  perform substitutions in parallel
!
      CALL laipe$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 laipe$Done
!
      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 square 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 square 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 introduces 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 the simplest form:

where the symbol "*" represents non-zero fill-ins. Three types of subroutine are included:

laipe$fpDecompose_DAG_4
laipe$fpDecompose_DAG_8
laipe$fpDecompose_DAG_10
laipe$fpDecompose_DAG_16
laipe$fpDecompose_DAG_Z4
laipe$fpDecompose_DAG_Z8
laipe$fpDecompose_DAG_Z10
laipe$fpDecompose_DAG_Z16

laipe$fpSubstitute_DAG_4
laipe$fpSubstitute_DAG_8
laipe$fpSubstitute_DAG_10
laipe$fpSubstitute_DAG_16
laipe$fpSubstitute_DAG_Z4
laipe$fpSubstitute_DAG_Z8
laipe$fpSubstitute_DAG_Z10
laipe$fpSubstitute_DAG_Z16

laipe$fpSolution_DAG_4
laipe$fpSolution_DAG_8
laipe$fpSolution_DAG_10
laipe$fpSolution_DAG_16
laipe$fpSolution_DAG_Z4
laipe$fpSolution_DAG_Z8
laipe$fpSolution_DAG_Z10
laipe$fpSolution_DAG_Z16

The solvers are not included in LAIPE2.

15.2 Fortran Syntax for Subroutine fpDecompose

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

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

where

1. The argument A_io, an 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 NoGood_o=0. For the definition of profile, please see section 15.5.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument RowOrder_io, a 4-byte INTEGER array of dimension (N_i), enters a sequence of consecutive numbers from one to N_i and returns the pivoting rows if NoGood_o=0.
4. The argument ColumnOrder_io, a 4-byte INTEGER array of dimension (N_i), enters a sequence of consecutive numbers from one to N_i and returns the pivoting columns if NoGood_o=0.
5. The argument NoGood_o, a 4-byte INTEGER, returns a flag. If NoGood_o=1, the input matrix [A] cannot be decomposed; otherwise the profile A_io returns the decomposed matrices [L] and [U]. For the situation where NoGood_o=1, please see section 15.7.

15.3 Fortran Syntax for Subroutine fpSubstitute

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

```
laipe$fpSubstitute_DAG_4(A_i, N_i, RowOrder_i, ColumnOrder_i, X_io)
laipe$fpSubstitute_DAG_8(A_i, N_i, RowOrder_i, ColumnOrder_i, X_io)
laipe$fpSubstitute_DAG_10(A_i, N_i, RowOrder_i, ColumnOrder_i, X_io)
laipe$fpSubstitute_DAG_16(A_i, N_i, RowOrder_i, ColumnOrder_i, X_io)
laipe$fpSubstitute_DAG_Z4(A_i, N_i, RowOrder_i, ColumnOrder_i, X_io)
laipe$fpSubstitute_DAG_Z8(A_i, N_i, RowOrder_i, ColumnOrder_i, X_io)
laipe$fpSubstitute_DAG_Z10(A_i, N_i, RowOrder_i, ColumnOrder_i, X_io)
laipe$fpSubstitute_DAG_Z16(A_i, N_i, RowOrder_i, ColumnOrder_i, X_io)
```

where

1. The argument A_i, an array which type must be consistent with subroutine name convention, is the the result from decomposition.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].

3. The argument RowOrder_i, a 4-byte INTEGER array of dimension (N_i), inputs the pivoting rows from decomposition.
4. The argument ColumnOrder_i, a 4-byte INTEGER array of dimension (N_i), inputs the pivoting columns from decomposition.
5. The argument X_io, an array of dimension (N_i) 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:

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

where

1. The argument A_io, an 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 NoGood_o=0. For the definition of profile, please see section 15.5.
2. The argument N_i, a 4-byte INTEGER, is the order of square matrix [A].
3. The argument RowOrder_io, a 4-byte INTEGER array of dimension (N_i), enters a sequence of consecutive numbers from one to N_i and returns the pivoting rows if NoGood_o=0.
4. The argument ColumnOrder_io, a 4-byte INTEGER array of dimension (N_i), enters a sequence of consecutive numbers from one to N_i and returns the pivoting columns if NoGood_o=0.
5. The argument X_io, an array of dimension (N_i) which type must be consistent with subroutine name convention, inputs the right side vector, and returns the solution if NoGood_o=0.
6. The argument NoGood_o, a 4-byte INTEGER, returns a flag. If NoGood_o=1, the input system cannot be solved by the subroutine; 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=1, please see section 15.7.

15.5 Profile

Profile for the solver is in the simplest form as:

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

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 order of square matrix.

15.6 Data Storage Scheme

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

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

where variable A is a single precision profile of matrix [A], and N is the order of square matrix. For other kinds of variable, profile must be properly declared. 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 (e. g., NpGood=1), the solver is not fit for the problem.

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. In the following Fortran program, the subroutines “Input” and “Output” demonstrate data storage scheme; the subroutine “laipe$fpDecompose_DAG_8” decomposes matrix [A]

with full pivoting; The subroutine “laipe$fpSubstitute_DAG_8” performs forward and backward substitutions to obtain {X}.

```
!  *** 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)
      INTEGER*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 laipe$fpDecompose_DAG_4(A,N,RowOrder, ColumnOrder, NoGood)
!
!  stop if NoGood=1
!
      IF(NoGood.eq.1) STOP 'Cannot be decomposed'
!
!  perform substitutions in parallel
!
      CALL laipe$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 laipe$Done
!
      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 square 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 square 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
```