1 Purpose

nag_real_cholesky_skyline_solve (f04mcc) computes the approximate solution of a system of real linear equations with multiple right-hand sides, \( AX = B \), where \( A \) is a symmetric positive definite variable-bandwidth matrix, which has previously been factorized by nag_real_cholesky_skyline (f01mcc). Related systems may also be solved.

2 Specification

```c
#include <nag.h>
#include <nagf04.h>

void nag_real_cholesky_skyline_solve (Nag_SolveSystem selct, Integer n,
 Integer nrhs, const double al[], Integer lal, const double d[],
 const Integer row[], const double b[], Integer tdb, double x[],
 Integer tdx, NagError *fail)
```

3 Description

The normal use of nag_real_cholesky_skyline_solve (f04mcc) is the solution of the systems \( AX = B \), following a call of nag_real_cholesky_skyline (f01mcc) to determine the Cholesky factorization \( A = LDL^T \) of the symmetric positive definite variable-bandwidth matrix \( A \).

However, the function may be used to solve any one of the following systems of linear algebraic equations:

\[
LDL^T X = B \quad \text{(usual system)} \quad (1)
\]

\[
LDX = B \quad \text{(lower triangular system)} \quad (2)
\]

\[
DL^TX = B \quad \text{(upper triangular system)} \quad (3)
\]

\[
LL^TX = B \quad \text{(4)}
\]

\[
LX = B \quad \text{(unit lower triangular system)} \quad (5)
\]

\[
L^TX = B \quad \text{(unit upper triangular system)} \quad (6)
\]

\( L \) denotes a unit lower triangular variable-bandwidth matrix of order \( n \), \( D \) a diagonal matrix of order \( n \), and \( B \) a set of right-hand sides.

The matrix \( L \) is represented by the elements lying within its envelope, i.e., between the first nonzero of each row and the diagonal (see Section 10 for an example). The width \( \text{row}[i] \) of the \( i \)th row is the number of elements between the first nonzero element and the element on the diagonal inclusive.
4 References

5 Arguments

1: \( \text{select} \) – Nag_SolveSystem

*Input*

On entry: \( \text{select} \) must specify the type of system to be solved, as follows:

- if \( \text{select} = \text{Nag_LDLTX} \): solve \( LDL^TX = B \);
- if \( \text{select} = \text{Nag_LDX} \): solve \( LDX = B \);
- if \( \text{select} = \text{Nag_DLTX} \): solve \( DL^TX = B \);
- if \( \text{select} = \text{Nag_LLTX} \): solve \( LL^TX = B \);
- if \( \text{select} = \text{Nag_LX} \): solve \( LX = B \);
- if \( \text{select} = \text{Nag_LTX} \): solve \( L^TX = B \).

*Constraint:* \( \text{select} = \text{Nag_LDLTX, Nag_LDX, Nag_DLTX, Nag_LLTX, Nag_LX or Nag_LTX} \).

2: \( n \) – Integer

*Input*

On entry: \( n \), the order of the matrix \( L \).

*Constraint:* \( n \geq 1 \).

3: \( \text{nrhs} \) – Integer

*Input*

On entry: \( r \), the number of right-hand sides.

*Constraint:* \( \text{nrhs} \geq 1 \).

4: \( \text{al}[\cdot] \) – const double

*Input*

On entry: the elements within the envelope of the lower triangular matrix \( L \), taken in row by row order, as returned by nag_real_chol_esky (f01mcc). The unit diagonal elements of \( L \) must be stored explicitly.

5: \( \text{la} \) – Integer

*Input*

On entry: the dimension of the array \( \text{al} \).

*Constraint:* \( \text{la} \geq \text{row}[0] + \text{row}[1] + \cdots + \text{row}[n-1] \).

6: \( \text{d}[\cdot] \) – const double

*Input*

On entry: the diagonal elements of the diagonal matrix \( D \). \( \text{d} \) is not referenced if \( \text{select} = \text{Nag_LLTX, Nag_LX or Nag_LTX} \).

7: \( \text{row}[\cdot] \) – const Integer

*Input*

On entry: \( \text{row}[i] \) must contain the width of row \( i \) of \( L \), i.e., the number of elements between the first (left-most) nonzero element and the element on the diagonal, inclusive.

*Constraint:* \( 1 \leq \text{row}[i] \leq i+1 \) for \( i = 0, 1, \ldots, n-1 \).

8: \( \text{b}[\cdot] \) – const double

*Input*

*Note:* the \((i,j)\)th element of the matrix \( B \) is stored in \( B[(i-1) \times \text{tdb} + j - 1] \).

On entry: the \( n \) by \( r \) right-hand side matrix \( B \). See also Section 9.
9:  tdb – Integer  
    
    On entry: the stride separating matrix column elements in the array b.
    
    Constraint: tdb ≥ nrhs.

10:  x[n × tdx] – double
    
    Note: the $(i, j)\text{th}$ element of the matrix $X$ is stored in 
    $x[(i - 1) \times tdx + j - 1]$. 
    
    On exit: the $n$ by $r$ solution matrix $X$. See also Section 9.

11:  tdx – Integer
    
    On entry: the stride separating matrix column elements in the array x.
    
    Constraint: tdx ≥ nrhs.

12:  fail – NagError*
    
    The NAG error argument (see Section 3.6 in the Essential Introduction).

6  Error Indicators and Warnings

NE_2_INT_ARG_GT
    On entry, row[i] = ⟨value⟩ while $i = ⟨value⟩$. These arguments must satisfy 
    row[i] ≤ $i + 1$.

NE_2_INT_ARG_LT
    On entry, lal = ⟨value⟩ while $\text{row}[0] + \cdots + \text{row}[n - 1] = ⟨value⟩$. These arguments must satisfy 
    lal ≥ $\text{row}[0] + \cdots + \text{row}[n - 1]$.
    
    On entry, tdb = ⟨value⟩ while nrhs = ⟨value⟩. These arguments must satisfy tdb ≥ nrhs.
    
    On entry, tdx = ⟨value⟩ while nrhs = ⟨value⟩. These arguments must satisfy tdx ≥ nrhs.

NE_BAD_PARAM
    On entry, argument select had an illegal value.

NE_INT_ARG_LT
    On entry, n = ⟨value⟩.
    
    Constraint: $n ≥ 1$.
    
    On entry, nrhs = ⟨value⟩.
    
    Constraint: nrhs ≥ 1.
    
    On entry, row[⟨value⟩] must not be less than 1: row[⟨value⟩] = ⟨value⟩.

NE_NOT_UNIT_DIAG
    The lower triangular matrix $L$ has at least one diagonal element which is not equal to unity. The first non-unit element has been located in the array al[⟨value⟩].

NE_ZERO_DIAG
    The diagonal matrix $D$ is singular as it has at least one zero element. The first zero element has been located in the array d[⟨value⟩].

7  Accuracy

The usual backward error analysis of the solution of triangular system applies: each computed solution 
vector is exact for slightly perturbed matrices $L$ and $D$, as appropriate (see pages 25-27 and 54-55 of 
Wilkinson and Reinsch (1971)).
8 Parallelism and Performance

Not applicable.

9 Further Comments

The time taken by nag_real_cholesky_skyline_solve (f04mcc) is approximately proportional to $pr$, where $p = \text{row}[0] + \text{row}[1] + \cdots + \text{row}[n-1]$.

The function may be called with the same actual array supplied for the arguments $b$ and $x$, in which case the solution matrix will overwrite the right-hand side matrix.

10 Example

To solve the system of equations $AX = B$, where

\[
A = \begin{pmatrix}
1 & 2 & 0 & 0 & 5 & 0 \\
2 & 5 & 3 & 0 & 14 & 0 \\
0 & 3 & 13 & 0 & 18 & 0 \\
0 & 0 & 0 & 16 & 8 & 24 \\
5 & 14 & 18 & 8 & 55 & 17 \\
0 & 0 & 0 & 24 & 17 & 77 \\
\end{pmatrix}
\quad \text{and} \quad
B = \begin{pmatrix}
6 & -10 \\
15 & -21 \\
11 & -3 \\
0 & 24 \\
51 & -39 \\
46 & 67 \\
\end{pmatrix}.
\]

Here $A$ is symmetric and positive definite and must first be factorized by nag_real_cholesky_skyline (f01mcc).

10.1 Program Text

/* nag_real_cholesky_skyline_solve (f04mcc) Example Program.  
* * Copyright 2014 Numerical Algorithms Group. 
* * Mark 4, 1996. 
* * Mark 8 revised, 2004. 
*/

#include <nag.h>
#include <math.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nagf01.h>
#include <nagf04.h>

#define B(I, J) b[(I) *tdb + J]
#define X(I, J) x[(I) *tdx + J]

int main(void)
{
    Integer exit_status = 0, i, k, k1, k2, lal, n, nrhs, *row = 0, tdb, tdx;
    Nag_SolveSystem select;
    double *a = 0, *al = 0, *b = 0, *d = 0, *x = 0;
    NagError fail;

    INIT_FAIL(fail);

    printf("nag_real_cholesky_skyline_solve (f04mcc) Example Program Results\n");
    /* Skip heading in data file */
    #ifdef _WIN32
    scanf_s("%*[\n]");
    #else
    scanf("%*[\n]");
    #endif
    #ifdef _WIN32
    scanf_s("%s"NAG_IFMT", &n);
    #endif

    return exit_status;
}
#else
    scanf("%"NAG_IFMT", &n);
#endif
if (n >= 1)
{
    if (!(row = NAG_ALLOC(n, Integer)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
}
else
{
    printf("Invalid n.\n");
    exit_status = 1;
    return exit_status;
}
lal = 0;
for (i = 0; i < n; ++i)
{
    #ifdef _WIN32
        scanf_s("%"NAG_IFMT", &row[i]);
    #else
        scanf("%"NAG_IFMT", &row[i]);
    #endif
    lal += row[i];
}
if (!(a = NAG_ALLOC(lal, double)) ||
    !(al = NAG_ALLOC(lal, double))}
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
} 
k2 = 0;
for (i = 0; i < n; ++i)
{
    k1 = k2;
    k2 = k2 + row[i];
    for (k = k1; k < k2; ++k)
        #ifdef _WIN32
            scanf_s("%lf", &a[k]);
        #else
            scanf("%lf", &a[k]);
        #endif
    }
    #ifdef _WIN32
        scanf_s("%"NAG_IFMT", &nrhs);
    #else
        scanf("%"NAG_IFMT", &nrhs);
    #endif
    if (nrhs >= 1)
    {
        if (!(b = NAG_ALLOC(n*nrhs, double)) ||
            !(d = NAG_ALLOC(n, double)) ||
            !(x = NAG_ALLOC(n*nrhs, double))
            printf("Allocation failure\n");
            exit_status = -1;
            goto END;
        }
        tdb = nrhs;
        tdx = nrhs;
    }
else
{
    printf("Invalid nrhs.\n");
    exit_status = 1;
    return exit_status;
}
for (i = 0; i < n; ++i)
    for (k = 0; k < nrhs; ++k)
#define _WIN32
    scanf_s("%lf", &B(i, k));
#else
    scanf("%lf", &B(i, k));
#endif

/* nag_real_cholesky_skyline (f01mcc).
** LDL^T factorization of real symmetric positive-definite
** variable-bandwidth (skyline) matrix
*/
nag_real_cholesky_skyline(n, a, lal, row, al, d, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_real_cholesky_skyline (f01mcc).\n%s
", fail.message);
    exit_status = 1;
    goto END;
}
select = Nag_LDLTX;
/* nag_real_cholesky_skyline_solve (f04mcc).
** Approximate solution of real symmetric positive-definite
** variable-bandwidth simultaneous linear equations
** (coefficient matrix already factorized by
** nag_real_cholesky_skyline (f01mcc))
*/
nag_real_cholesky_skyline_solve(select, n, nrhs, al, lal, d, row, b, tdb, x, tdx, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_real_cholesky_skyline_solve (f04mcc).\n%s
", fail.message);
    exit_status = 1;
    goto END;
}
printf("\n Solution\n");
for (i = 0; i < n; ++i)
{
    for (k = 0; k < nrhs; ++k)
    {
        printf("%9.3f", X(i, k));
    }
}
END:
NAG_FREE(row);
NAG_FREE(b);
NAG_FREE(al);
NAG_FREE(d);
NAG_FREE(x);
NAG_FREE(a);
NAG_FREE(al);
return exit_status;

10.2 Program Data

nag_real_cholesky_skyline_solve (f04mcc) Example Program Data
6
1 2 2 1 5 3
1.0
2.0 5.0
3.0 13.0
16.0
5.0 14.0 18.0 8.0 55.0
24.0 17.0 77.0
2
6.0 -10.0
15.0  -21.0
11.0  -3.0
 0.0  24.0
51.0  -39.0
46.0  67.0

10.3 Program Results

nag_real_cholesky_skyline_solve (f04mcc) Example Program Results

<table>
<thead>
<tr>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>-3.000 4.000</td>
</tr>
<tr>
<td>2.000 -2.000</td>
</tr>
<tr>
<td>-1.000 3.000</td>
</tr>
<tr>
<td>-2.000 1.000</td>
</tr>
<tr>
<td>1.000 -2.000</td>
</tr>
<tr>
<td>1.000 1.000</td>
</tr>
</tbody>
</table>