# nag_real_cholesky_skyline_solve (f04mcc)

## 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

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

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

## 3. Description

The normal use of this function 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:

$$LDL^TX = B \text{ (usual system)} \quad (1)$$
$$LDX = B \text{ (lower triangular system)} \quad (2)$$
$$DL^TX = B \text{ (upper triangular system)} \quad (3)$$
$$LL^TX = B \quad (4)$$
$$LX = B \text{ (unit lower triangular system)} \quad (5)$$
$$L^TX = B \text{ (unit upper triangular system).} \quad (6)$$

$L$ 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 non-zero of each row and the diagonal (see Section 8 for an example). The width **row**$[i]$ of the $i$th row is the number of elements between the first non-zero element and the element on the diagonal inclusive.

## 4. Parameters

**selct**

Input: **selct** must specify the type of system to be solved, as follows:
**selct** = **Nag_LDLTX**: solve $LDL^TX = B$
**selct** = **Nag_LDX**: solve $LDX = B$
**selct** = **Nag_DLTX**: solve $DL^TX = B$
**selct** = **Nag_LLTX**: solve $LL^TX = B$
**selct** = **Nag_LX**: solve $LX = B$
**selct** = **Nag_LTX**: solve $L^TX = B$.
Constraint: **selct** must be one of **Nag_LDLTX**, **Nag_LDX**, **Nag_DLTX**, **Nag_LLTX**, **Nag_LX**, **Nag_LTX**.

**n**

Input: $n$, the order of the matrix $L$.
Constraint: $\mathbf{n} \geq \mathbf{1}$.

**nrhs**

Input: $r$, the number of right-hand sides.
Constraint: $\mathbf{nrhs} \geq \mathbf{1}$.

**al[lal]**
Input: the elements within the envelope of the lower triangular matrix $L$, taken in row by row order, as returned by nag_real_cholesky_skyline (f01mcc). The unit diagonal elements of $L$ must be stored explicitly.

**lal**
Input: the dimension of the array **al** as declared in the function from which nag_real_cholesky_skyline_solve is called.
Constraint: **lal** $\geq$ **row**$[0]$ + **row**$[1]$ + $\ldots$ + **row**$[n-1]$.

**d[n]**
Input: the diagonal elements of the diagonal matrix $D$. **d** is not referenced if **selct** = **Nag_LLTX**, **Nag_LX** or **Nag_LTX**

**row[n]**
Input: **row**$[i]$ must contain the width of row $i$ of $L$, i.e., the number of elements between the first (left-most) non-zero element and the element on the diagonal, inclusive.
Constraint: $1 \leq$ **row**$[i] \leq i+1$ for $i = 0, 1, \ldots, n-1$.

**b[n][tdb]**
Input: the $n$ by $r$ right-hand side matrix $B$. See also Section 6.

**tdb**
Input: the second dimension of the array **b** as declared in the function from which nag_real_cholesky_skyline_solve is called.
Constraint: **tdb** $\geq$ **nrhs**.

**x[n][tdx]**
Output: the $n$ by $r$ solution matrix $X$. See also Section 6.

**tdx**
Input: the second dimension of the array **x** as declared in the function from which nag_real_cholesky_skyline_solve is called.
Constraint: **tdx** $\geq$ **nrhs**.

**fail**
The NAG error parameter, see the Essential Introduction to the NAG C Library.

## 5. Error Indications and Warnings

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

**NE_2_INT_ARG_GT**
On entry, **row**$[i]$ = ⟨*value*⟩ while $i$ = ⟨*value*⟩. These parameters must satisfy **row**$[i] \leq i+1$.

**NE_2_INT_ARG_LT**
On entry, **lal** = ⟨*value*⟩ while **row**$[0] + \ldots +$**row**$[n-1]$ = ⟨*value*⟩. These parameters must satisfy **lal** $\geq$ **row**$[0] + \ldots +$**row**$[n-1]$.
On entry, **tdb** = ⟨*value*⟩ while **nrhs** = ⟨*value*⟩. These parameters must satisfy **tdb** $\geq$ **nrhs**.
On entry, **tdx** = ⟨*value*⟩ while **nrhs** = ⟨*value*⟩. These parameters must satisfy **tdx** $\geq$ **nrhs**.

**NE_BAD_PARAM**
On entry, parameter **selct** had an illegal 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*⟩]

**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*⟩]

## 6. Further Comments

The time taken by the function is approximately proportional to $pr$, where $p = \mathbf{row}[0] + \mathbf{row}[1] + \ldots + \mathbf{row}[n-1]$.

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

### 6.1. 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 Wilkinson and Reinsch (1971) pp 25-27 and 54-55).

### 6.2. References

Wilkinson J H and Reinsch C (1971) *Handbook for Automatic Computation (Vol II, Linear Algebra)* Springer-Verlag.

## 7. See Also

nag_real_cholesky_skyline (f01mcc)

## 8. Example

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

$$A = \begin{pmatrix} 1 & 2 & 0 & 0 & 5 & 0 \\ 2 & 5 & 3 & 0 & 14 & 0 \\ 0 & 3 & 13 & 0 & 18 & 0 \\ 0 & 0 & 0 & 16 & 8 & 24 \\ 5 & 14 & 18 & 8 & 55 & 17 \\ 0 & 0 & 0 & 24 & 17 & 77 \end{pmatrix}$$

and

$$B = \begin{pmatrix} 6 & -10 \\ 15 & -21 \\ 11 & -3 \\ 0 & 24 \\ 51 & -39 \\ 46 & 67 \end{pmatrix}.$$

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

### 8.1. Program Text

```
/* nag_real_cholesky_skyline_solve(f04mcc) Example Program
 *
 * Copyright 1996 Numerical Algorithms Group.
 *
 * Mark 4, 1996.
 */

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

#define NMAX 6
#define NRHSMAX 2
#define TDB NRHSMAX
#define TDX NRHSMAX
#define LALMAX 14
```

```
main()
{
  Integer i, nrhs, k, k1, k2, lal, n;
  double a[LALMAX], al[LALMAX], b[NMAX][TDB], d[NMAX], x[NMAX][TDX];
  Integer row[NMAX];
  Nag_SolveSystem select;
  static NagError fail;

  Vprintf("f04mcc Example Program Results\n");
  /* Skip heading in data file */
  Vscanf("%*[^\n]");
  Vscanf("%ld",&n);
  if (n<1 || n>NMAX)
    {
      Vprintf("\n n is out of range: n = %ld\n", n);
      exit(EXIT_FAILURE);
    }
  for (i=0; i<n; ++i)
    Vscanf("%ld",&row[i]);
  k2 = 0;
  for (i=0; i<n; ++i)
    {
      k1 = k2;
      k2 = k2 + row[i];
      for (k=k1; k<k2; ++k)
        Vscanf("%lf",&a[k]);
    }
  lal = k2;
  if (lal > LALMAX)
    {
      Vprintf("\n lal is out of range: lal = %ld\n", lal);
      exit(EXIT_FAILURE);
    }
  Vscanf("%ld",&nrhs);
  if (nrhs<1 || nrhs>NRHSMAX)
    {
      Vprintf("\n nrhs is out of range: nrhs = %ld\n", nrhs);
      exit(EXIT_FAILURE);
    }
  for (i=0; i<n; ++i)
    for (k=0; k<nrhs; ++k)
      Vscanf("%lf",&b[i][k]);
  fail.print = TRUE;
  f01mcc(n, a, lal, row, al, d, &fail);
  if (fail.code != NE_NOERROR)
    exit(EXIT_FAILURE);
  select = Nag_LDLTX;
  f04mcc(select, n, nrhs, al, lal, d, row, (double *)b, (Integer)TDB,
         (double *)x, (Integer)TDX, &fail);
  if (fail.code != NE_NOERROR)
    exit(EXIT_FAILURE);
  Vprintf("\n Solution\n");
  for (i=0; i<n; ++i)
    {
      for (k=0; k<nrhs; ++k)
        Vprintf("%9.3f",x[i][k]);
      Vprintf("\n");
    }
  exit(EXIT_SUCCESS);
}
```

### 8.2. Program Data

```
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
```

### 8.3. Program Results

```
f04mcc Example Program Results

 Solution
   -3.000    4.000
    2.000   -2.000
   -1.000    3.000
   -2.000    1.000
    1.000   -2.000
    1.000    1.000
```