1 Purpose

nag_dposv (f07fac) computes the solution to a real system of linear equations

\[ AX = B, \]

where \( A \) is an \( n \) by \( n \) symmetric positive definite matrix and \( X \) and \( B \) are \( n \) by \( r \) matrices.

2 Specification

```c
#include <nag.h>
#include <nagf07.h>
void nag_dposv (Nag_OrderType order, Nag_UploType uplo, Integer n,
                 Integer nrhs, double a[], Integer pda, double b[],
                 Integer pdb, NagError *fail)
```

3 Description

nag_dposv (f07fac) uses the Cholesky decomposition to factor \( A \) as \( A = U^T U \) if \( \text{uplo} = \text{Nag}_\text{Upper} \) or \( A = LL^T \) if \( \text{uplo} = \text{Nag}_\text{Lower} \), where \( U \) is an upper triangular matrix and \( L \) is a lower triangular matrix. The factored form of \( A \) is then used to solve the system of equations \( AX = B \).

4 References


5 Arguments

1: \( \text{order} \) – Nag_OrderType

   *Input*

   On entry: the \text{order} argument specifies the two-dimensional storage scheme being used, i.e., row-major ordering or column-major ordering. C language defined storage is specified by \text{order} = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

   Constraint: \text{order} = Nag_RowMajor or Nag-ColMajor.

2: \( \text{uplo} \) – Nag_UploType

   *Input*

   On entry: if \text{uplo} = Nag_Upper, the upper triangle of \( A \) is stored.

   If \text{uplo} = Nag_Lower, the lower triangle of \( A \) is stored.

   Constraint: \text{uplo} = Nag_Upper or Nag_Lower.

3: \( n \) – Integer

   *Input*

   On entry: \( n \), the number of linear equations, i.e., the order of the matrix \( A \).

   Constraint: \( n \geq 0 \).
4:  \( \textbf{nrhs} \) – Integer  
*Input*

On entry: \( r \), the number of right-hand sides, i.e., the number of columns of the matrix \( B \).

Constraint: \( \textbf{nrhs} \geq 0 \).

5:  \( \textbf{a}[\text{\textit{dim}}] \) – double  
*Input/Output*

Note: the dimension, \( \text{\textit{dim}} \), of the array \( \textbf{a} \) must be at least \( \max(1, \textbf{pda} \times n) \).

On entry: the \( n \) by \( n \) symmetric matrix \( A \).

If \( \textbf{order} = \text{Nag\_ColMajor} \), \( A_{ij} \) is stored in \( \textbf{a}[(j - 1) \times \textbf{pda} + i - 1] \).

If \( \textbf{order} = \text{Nag\_RowMajor} \), \( A_{ij} \) is stored in \( \textbf{a}[(i - 1) \times \textbf{pda} + j - 1] \).

If \( \textbf{uplo} = \text{Nag\_Upper} \), the upper triangular part of \( A \) must be stored and the elements of the array below the diagonal are not referenced.

If \( \textbf{uplo} = \text{Nag\_Lower} \), the lower triangular part of \( A \) must be stored and the elements of the array above the diagonal are not referenced.

On exit: if \( \textbf{fail.code} = \text{NE\_NOERROR} \), the factor \( U \) or \( L \) from the Cholesky factorization \( A = U^T U \) or \( A = LL^T \).

6:  \( \textbf{pda} \) – Integer  
*Input*

On entry: the stride separating row or column elements (depending on the value of \( \textbf{order} \)) of the matrix \( A \) in the array \( \textbf{a} \).

Constraint: \( \textbf{pda} \geq \max(1, n) \).

7:  \( \textbf{b}[\text{\textit{dim}}] \) – double  
*Input/Output*

Note: the dimension, \( \text{\textit{dim}} \), of the array \( \textbf{b} \) must be at least \( \max(1, \text{\textit{pdb}} \times \text{\textit{nrhs}}) \) when \( \textbf{order} = \text{Nag\_ColMajor} \); \( \max(1, n \times \text{\textit{pdb}}) \) when \( \textbf{order} = \text{Nag\_RowMajor} \).

The \( (i, j) \)th element of the matrix \( B \) is stored in

\[
\begin{align*}
\textbf{b}[(j - 1) \times \text{\textit{pdb}} + i - 1] & \quad \text{when} \quad \textbf{order} = \text{Nag\_ColMajor}; \\
\textbf{b}[(i - 1) \times \text{\textit{pdb}} + j - 1] & \quad \text{when} \quad \textbf{order} = \text{Nag\_RowMajor}.
\end{align*}
\]

On entry: the \( n \) by \( r \) right-hand side matrix \( B \).

On exit: if \( \textbf{fail.code} = \text{NE\_NOERROR} \), the \( n \) by \( r \) solution matrix \( X \).

8:  \( \textbf{pdb} \) – Integer  
*Input*

On entry: the stride separating row or column elements (depending on the value of \( \textbf{order} \)) in the array \( \textbf{b} \).

Constraints:

- if \( \textbf{order} = \text{Nag\_ColMajor} \), \( \textbf{pdb} \geq \max(1, n) \);
- if \( \textbf{order} = \text{Nag\_RowMajor} \), \( \textbf{pdb} \geq \max(1, \text{\textit{nrhs}}) \).

9:  \( \textbf{fail} \) – NagError *  
*Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

6  **Error Indicators and Warnings**

**NE\_ALLOC\_FAIL**

Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.
NE_BAD_PARAM
On entry, argument <value> had an illegal value.

NE_INT
On entry, n = <value>.
Constraint: n ≥ 0.
On entry, nrhs = <value>.
Constraint: nrhs ≥ 0.
On entry, pda = <value>.
Constraint: pda > 0.
On entry, pdb = <value>.
Constraint: pdb > 0.

NE_INT_2
On entry, pda = <value> and n = <value>.
Constraint: pda ≥ max(1, n).
On entry, pdb = <value> and n = <value>.
Constraint: pdb ≥ max(1, n).
On entry, pdb = <value> and nrhs = <value>.
Constraint: pdb ≥ max(1, nrhs).

NE_INTERNAL_ERROR
An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG. See Section 3.6.6 in the Essential Introduction for further information.

NE_MAT_NOT_POS_DEF
The leading minor of order <value> of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

NE_NO_LICENCE
Your licence key may have expired or may not have been installed correctly. See Section 3.6.5 in the Essential Introduction for further information.

7 Accuracy
The computed solution for a single right-hand side, \( \hat{x} \), satisfies an equation of the form
\[
(A + E)\hat{x} = b,
\]
where
\[
\|E\|_1 = O(\epsilon)\|A\|_1
\]
and \( \epsilon \) is the machine precision. An approximate error bound for the computed solution is given by
\[
\frac{\|\hat{x} - x\|_1}{\|x\|_1} \leq \kappa(A) \frac{\|E\|_1}{\|A\|_1},
\]
where \( \kappa(A) = \|A^{-1}\|_1\|A\|_1 \), the condition number of A with respect to the solution of the linear equations. See Section 4.4 of Anderson et al. (1999) for further details.

nag_dposvx (f07fbc) is a comprehensive LAPACK driver that returns forward and backward error bounds and an estimate of the condition number. Alternatively, nag_real_sym_posdef_lin_solve (f04bdc)
solves $Ax = b$ and returns a forward error bound and condition estimate. `nag_real_sym_posdef_lin_solve (f04bdc)` calls `nag_dposv (f07fac)` to solve the equations.

8 Parallelism and Performance

`nag_dposv (f07fac)` is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

`nag_dposv (f07fac)` makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users’ Note for your implementation for any additional implementation-specific information.

9 Further Comments

The total number of floating-point operations is approximately $\frac{3}{2}n^3 + 2n^2r$, where $r$ is the number of right-hand sides.

The complex analogue of this function is `nag_zposv (f07fnc).

10 Example

This example solves the equations

$$Ax = b,$$

where $A$ is the symmetric positive definite matrix

$$A = \begin{pmatrix}
-4.16 & -3.12 & 0.56 & -0.10 \\
-3.12 & 5.03 & -0.83 & 1.18 \\
0.56 & -0.83 & 0.76 & 0.34 \\
-0.10 & 1.18 & 0.34 & 1.18
\end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix}
8.70 \\
-13.35 \\
1.89 \\
-4.14
\end{pmatrix}.$$  

Details of the Cholesky factorization of $A$ are also output.

10.1 Program Text

/* `nag_dposv (f07fac)` Example Program.  
   * Copyright 2014 Numerical Algorithms Group.  
   * Mark 23, 2011.  
   */
#include <stdio.h>
#include <nag.h>
#include <nagx04.h>
#include <nag_stdlib.h>
#include <nagf07.h>

int main(void)
{
    /* Scalars */
    Integer exit_status = 0, i, j, n, nrhs, pda, pdb;
    /* Arrays */
    double *a = 0, *b = 0;
    /* Nag Types */
    NagError fail;
    Nag_OrderType order;
    #ifdef NAG_COLUMN_MAJOR
    #define A(I, J) a[(J-1)*pda + I - 1]
    #else
    #define A(I, J) a[(I-1)*pda + J - 1]
    #endif
    ...
#define B(I, J) b[(J-1)*pdb +I-1]
order = Nag_ColMajor;
#else
#define A(I, J) a[(I-1)*pda +J-1]
#define B(I, J) b[(I-1)*pdb +J-1]
#endif
order = Nag_RowMajor;
#endif

INIT_FAIL(fail);

printf("nag_dposv (f07fac) Example Program Results\n\n");

/* Skip heading in data file */
#ifdef _WIN32
scanf_s("%*[\n]"");
#else
scanf("%*[\n]"");
#endif

#ifdef _WIN32
scanf_s("%"NAG_IFMT"%"NAG_IFMT"%*[\n]", &n, &nrhs);
#else
scanf("%"NAG_IFMT"%"NAG_IFMT"%*[\n]", &n, &nrhs);
#endif
if (n < 0 || nrhs < 0)
{
    printf("Invalid n or nrhs\n");
    exit_status = 1;
    goto END;
}

/* Allocate memory */
if (!(a = NAG_ALLOC(n * n, double)) ||
    !(b = NAG_ALLOC(n *nrhs, double)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

pda = n;
#ifdef NAG_COLUMN_MAJOR
pdb = n;
#else
pdb = nrhs;
#endif

/* Read the upper triangular part of A from data file */
for (i = 1; i <= n; ++i)
#ifdef _WIN32
    for (j = i; j <= n; ++j) scanf_s("%lf", &A(i, j));
#else
    for (j = i; j <= n; ++j) scanf("%lf", &A(i, j));
#endif
#ifdef _WIN32
scanf_s("%*[\n]"");
#else
scanf("%*[\n]"");
#endif

/* Read b from data file */
for (i = 1; i <= n; ++i)
#ifdef _WIN32
    for (j = 1; j <= nrhs; ++j) scanf_s("%lf", &B(i, j));
#else
    for (j = 1; j <= nrhs; ++j) scanf("%lf", &B(i, j));
#endif
#ifdef _WIN32
scanf_s("%*[\n]"");
#else
scanf("%*[\n]"");
#endif
#endif

/* Solve the equations Ax = b for x using nag_dposv (f07fac). */

nag_dposv(order, Nag_Upper, n, nrhs, a, pda, b, pdb, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dposv (f07fac).\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print solution */
printf("Solution\n");
for (i = 1; i <= n; ++i)
{
    for (j = 1; j <= nrhs; ++j)
        printf("%11.4f%s", B(i, j), j%7 == 0?"\n":" ");
    printf("\n");
}

/* Print details of factorization using nag_gen_real_mat_print (x04cac). */
printf("\n");
fflush(stdout);
nag_gen_real_mat_print(order, Nag_UpperMatrix, Nag_NonUnitDiag, n, n, a, pda,
                        "Cholesky factor U", 0, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_gen_real_mat_print (x04cac).\n", fail.message);
    exit_status = 1;
    goto END;
}

END:
NAG_FREE(a);
NAG_FREE(b);
return exit_status;

#undef A
#undef B

10.2 Program Data

nag_dposv (f07fac) Example Program Data

4 1 : n, nrhs
 4.16 -3.12 0.56 -0.10
 5.03 -0.83 1.18
 0.76 0.34
 1.18 : matrix A
8.70 -13.35 1.89 -4.14 : vector b

10.3 Program Results

nag_dposv (f07fac) Example Program Results

Solution
 1.0000
 -1.0000
 2.0000
 -3.0000

Cholesky factor U
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.0396</td>
<td>-1.5297</td>
<td>0.2746</td>
<td>-0.0490</td>
</tr>
<tr>
<td>2</td>
<td>1.6401</td>
<td>-0.2500</td>
<td>0.6737</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.7887</td>
<td>0.6617</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>0.5347</td>
<td></td>
</tr>
</tbody>
</table>