1 Purpose

nag_complex_sym_packed_lin_solve (f04djc) computes the solution to a complex system of linear equations $AX = B$, where $A$ is an $n$ by $n$ complex symmetric matrix, stored in packed format and $X$ and $B$ are $n$ by $r$ matrices. An estimate of the condition number of $A$ and an error bound for the computed solution are also returned.

2 Specification

```c
#include <nag.h>
#include <nagf04.h>
void nag_complex_sym_packed_lin_solve (Nag_OrderType order,
   Nag_UploType uplo, Integer n, Integer nrhs, Complex ap[],
   Integer ipiv[], Complex b[], Integer pdb, double *rcond, double *errbnd,
   NagError *fail)
```

3 Description

The diagonal pivoting method is used to factor $A$ as $A = UDU^T$, if uplo = Nag_Upper, or $A = LDL^T$, if uplo = Nag_Lower, where $U$ (or $L$) is a product of permutation and unit upper (lower) triangular matrices, and $D$ is symmetric and block diagonal with 1 by 1 and 2 by 2 diagonal blocks. The factored form of $A$ is then used to solve the system of equations $AX = B$.

4 References


5 Arguments

1: order – Nag_OrderType

*Input*

*On entry:* the 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 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:* order = Nag_RowMajor or Nag_ColMajor.

2: uplo – Nag_UploType

*Input*

*On entry:* if uplo = Nag_Upper, the upper triangle of the matrix $A$ is stored.

If uplo = Nag_Lower, the lower triangle of the matrix $A$ is stored.

*Constraint:* uplo = Nag_Upper or Nag_Lower.

3: n – Integer

*Input*

*On entry:* the number of linear equations $n$, i.e., the order of the matrix $A$.

*Constraint:* $n \geq 0$. 
4: \textbf{nrhs} – Integer \hspace{1cm} \textbf{Input}

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

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

5: \textbf{ap[dim]} – Complex \hspace{1cm} \textbf{Input/Output}

\textbf{Note:} the dimension, \(\text{dim}\), of the array \textbf{ap} must be at least \(\max(1,n \times (n + 1)/2)\).

\textit{On entry:} the \(n\) by \(n\) symmetric matrix \(A\), packed column-wise in a linear array. The \(j\)th column of the matrix \(A\) is stored in the array \textbf{ap} as follows:

The storage of elements \(A_{ij}\) depends on the \textbf{order} and \textbf{uplo} arguments as follows:

- if \textbf{order} = Nag_ColMajor and \textbf{uplo} = Nag_Upper, \(A_{ij}\) is stored in \textbf{ap}[(\(j - 1\) \times \(j/2\)) + \(i - 1\)], for \(i \leq j\);
- if \textbf{order} = Nag_ColMajor and \textbf{uplo} = Nag_Lower, \(A_{ij}\) is stored in \textbf{ap}[(\(2n - j\) \times \((j - 1)/2\)) + \(i - 1\)], for \(i \geq j\);
- if \textbf{order} = Nag_RowMajor and \textbf{uplo} = Nag_Upper, \(A_{ij}\) is stored in \textbf{ap}[(\(2n - i\) \times \((i - 1)/2\)) + \(j - 1\)], for \(i \leq j\);
- if \textbf{order} = Nag_RowMajor and \textbf{uplo} = Nag_Lower, \(A_{ij}\) is stored in \textbf{ap}[(\(i - 1\) \times \((i - 1)/2\)) + \(j - 1\)], for \(i \geq j\).

\textit{On exit:} if \textbf{fail.code} = NE_NOERROR, the block diagonal matrix \(D\) and the multipliers used to obtain the factor \(U\) or \(L\) from the factorization \(A = UDU^T\) or \(A = LDL^T\) as computed by \texttt{nag_zsptrf} (f07qrc), stored as a packed triangular matrix in the same storage format as \(A\).

6: \textbf{ipiv[n]} – Integer \hspace{1cm} \textbf{Output}

\textit{On exit:} if no constraints are violated, details of the interchanges and the block structure of \(D\), as determined by \texttt{nag_zsptrf} (f07qrc).

If \textbf{ipiv}[\(k - 1\)] > 0, then rows and columns \(k\) and \textbf{ipiv}[\(k - 1\)] were interchanged, and \(d_{kk}\) is a 1 by 1 diagonal block;
- if \textbf{uplo} = Nag_Upper and \textbf{ipiv}[\(k - 1\)] = \textbf{ipiv}[\(k - 2\)] < 0, then rows and columns \(k - 1\) and \(-\textbf{ipiv}[\(k - 1\)]\) were interchanged and \(d_{k-1,k-1}\) is a 2 by 2 diagonal block;
- if \textbf{uplo} = Nag_Lower and \textbf{ipiv}[\(k - 1\)] = \textbf{ipiv}[\(k\)] < 0, then rows and columns \(k + 1\) and \(-\textbf{ipiv}[\(k - 1\)]\) were interchanged and \(d_{k,k+1,k,k+1}\) is a 2 by 2 diagonal block.

7: \textbf{b[dim]} – Complex \hspace{1cm} \textbf{Input/Output}

\textbf{Note:} the dimension, \(\text{dim}\), of the array \textbf{b} must be at least

\[
\max(1, \text{pdb} \times \text{nrhs}) \quad \text{when} \quad \textbf{order} = \text{Nag}_\text{ColMajor}; \\
\max(1, n \times \text{pdb}) \quad \text{when} \quad \textbf{order} = \text{Nag}_\text{RowMajor}.
\]

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

- \textbf{b}[(\(j - 1\) \times \text{pdb}) + \(i - 1\)] when \textbf{order} = Nag_ColMajor;
- \textbf{b}[(\(i - 1\) \times \text{pdb}) + \(j - 1\)] when \textbf{order} = Nag_RowMajor.

\textit{On entry:} the \(n\) by \(r\) matrix of right-hand sides \(B\).

\textit{On exit:} if \textbf{fail.code} = NE_NOERROR or NE_RCOND, the \(n\) by \(r\) solution matrix \(X\).

8: \textbf{pdb} – Integer \hspace{1cm} \textbf{Input}

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

\textbf{Constraints:}

- if \textbf{order} = Nag_ColMajor, \(\textbf{pdb} \geq \max(1,n)\);
- if \textbf{order} = Nag_RowMajor, \(\textbf{pdb} \geq \max(1,\text{nrhs})\).
9: rcond – double*
   On exit: if no constraints are violated, an estimate of the reciprocal of the condition number of the matrix A, computed as \( rcond = 1/(\|A\|_1 \|A^{-1}\|_1) \).

10: errbnd – double*
    On exit: if fail.code = NE_NOERROR or NE_RCOND, an estimate of the forward error bound for a computed solution \( \hat{x} \), such that \( \|\hat{x} - x\|_1 / \|x\|_1 \leq \text{errbnd} \), where \( \hat{x} \) is a column of the computed solution returned in the array b and x is the corresponding column of the exact solution X. If rcond is less than machine precision, then errbnd is returned as unity.

11: 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 = \( \langle \text{value} \rangle \).
Constraint: \( n \geq 0 \).

On entry, nrhs = \( \langle \text{value} \rangle \).
Constraint: \( nrhs \geq 0 \).

On entry, pdb = \( \langle \text{value} \rangle \).
Constraint: \( pdb > 0 \).

NE_INT_2
On entry, pdb = \( \langle \text{value} \rangle \) and n = \( \langle \text{value} \rangle \).
Constraint: \( pdb \geq \text{max}(1, n) \).

On entry, pdb = \( \langle \text{value} \rangle \) and nrhs = \( \langle \text{value} \rangle \).
Constraint: \( pdb \geq \text{max}(1, nrhs) \).

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_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.

NE_RCOND
A solution has been computed, but rcond is less than machine precision so that the matrix A is numerically singular.
NE_SINGULAR

Diagonal block \( \langle \text{value} \rangle \) of the block diagonal matrix is zero. The factorization has been completed, but the solution could not be computed.

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. nag_complex_sym_packed_lin_solve (f04djc) uses the approximation \( \|E\|_1 = \epsilon\|A\|_1 \) to estimate errbnd. See Section 4.4 of Anderson et al. (1999) for further details.

8 Parallelism and Performance

nag_complex_sym_packed_lin_solve (f04djc) is not threaded by NAG in any implementation.

nag_complex_sym_packed_lin_solve (f04djc) 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 packed storage scheme is illustrated by the following example when \( n = 4 \) and uplo = Nag_Upper.

Two-dimensional storage of the symmetric matrix \( A \):

\[
\begin{array}{cccc}
  a_{11} & a_{12} & a_{13} & a_{14} \\
  a_{22} & a_{23} & a_{24} & \\
  a_{33} & a_{34} & & \\
  & & & a_{44}
\end{array}
\]

\( a_{ij} = a_{ji} \)

Packed storage of the upper triangle of \( A \):

\[
ap = [a_{11}, \ a_{12}, \ a_{22}, \ a_{13}, \ a_{23}, \ a_{33}, \ a_{14}, \ a_{24}, \ a_{34}, \ a_{44}]
\]

The total number of floating-point operations required to solve the equations \( AX = B \) is proportional to \( (\frac{1}{3}n^3 + 2n^2r) \). The condition number estimation typically requires between four and five solves and never more than eleven solves, following the factorization.

In practice the condition number estimator is very reliable, but it can underestimate the true condition number; see Section 15.3 of Higham (2002) for further details.

Function nag_herm_packed_lin_solve (f04jec) is for complex Hermitian matrices, and the real analogue of nag_complex_sym_packed_lin_solve (f04djc) is nag_real_sym_packed_lin_solve (f04bjc).
10 Example

This example solves the equations

\[ AX = B, \]

where \( A \) is the symmetric indefinite matrix

\[
A = \begin{pmatrix}
-0.56 + 0.12i & -1.54 - 2.86i & 5.32 - 1.59i & 3.80 + 0.92i \\
-1.54 - 2.86i & -2.83 - 0.03i & -3.52 + 0.58i & -7.86 - 2.96i \\
5.32 - 1.59i & -3.52 + 0.58i & 8.86 + 1.81i & 5.14 - 0.64i \\
3.80 + 0.92i & -7.86 - 2.96i & 5.14 - 0.64i & -0.39 - 0.71i
\end{pmatrix}
\]

and

\[
B = \begin{pmatrix}
-6.43 + 19.24i & -4.59 - 35.53i \\
-0.49 - 1.47i & 6.95 + 20.49i \\
-48.18 + 66.00i & -12.08 - 27.02i \\
-55.64 + 41.22i & -19.09 - 35.97i
\end{pmatrix}
\]

An estimate of the condition number of \( A \) and an approximate error bound for the computed solutions are also printed.

10.1 Program Text

/* nag_complex_sym_packed_lin_solve (f04djc) Example Program. */
* Copyright 2014 Numerical Algorithms Group.
* /

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf04.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    double errbnd, rcond;
    Integer exit_status, i, j, n, nrhs, pdb;

    /* Arrays */
    char nag_enum_arg[40];
    char *clabs = 0, *rlabs = 0;
    Complex *ap = 0, *b = 0;
    Integer *ipiv = 0;

    /* Nag types */
    NagError fail;
    Nag_OrderType order;
    Nag_UploType uplo;

    #ifdef NAG_COLUMN_MAJOR
    /*
    #define A(I,J) a[(J-1)*pda + I - 1]
    #define A_UPPER(I, J) ap[J*(J-1)/2 + I - 1]
    #define A_LOWER(I, J) ap[(2*n-J)*(J-1)/2 + I - 1]
    #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 A_UPPER(I, J) ap[I*(I-1)/2 + J - 1]
    #define A_LOWER(I, J) ap[(2*n-I)*(I-1)/2 + J - 1]
    #define B(I, J) b[(I-1)*pdb + J - 1]
    
    order = Nag_RowMajor;
    */
    #endif
}
exit_status = 0;
INIT_FAIL(fail);

printf("nag_complex_sym_packed_lin_solve (f04djc) 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)
    {
        /* Allocate memory */
        if (!(clabs = NAG_ALLOC(2, char)) ||
            !(rlabs = NAG_ALLOC(2, char)) ||
            !(ap = NAG_ALLOC(n*(n+1)/2, Complex)) ||
            !(b = NAG_ALLOC(n*nrhs, Complex)) ||
            !(ipiv = NAG_ALLOC(n, Integer)))
        {
            printf("Allocation failure\n");
            exit_status = -1;
            goto END;
        }
        #ifdef NAG_COLUMN_MAJOR
            pdb = n;
        #else
            pdb = nrhs;
        #endif
    }
    else
        {
            printf("%s\n", "n and/or nrhs too small");
            exit_status = 1;
            return exit_status;
        }

    /* Read A from data file */
    #ifdef _WIN32
        scanf_s("%39s%*[\n ]", nag_enum_arg, _countof(nag_enum_arg));
    #else
        scanf("%39s%*[\n ]", nag_enum_arg);
    #endif
    /* nag_enum_name_to_value (x04nac).
     * Converts NAG enum member name to value */
    uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);
    /* Read the upper or lower triangular part of the matrix A from */
    /* data file */
    if (uplo == Nag_Upper)
        {
            for (i = 1; i <= n; ++i)
                {
                    for (j = i; j <= n; ++j)
                        {
                            #ifdef _WIN32
                                scanf_s(" ( %lf , %lf )", &A_UPPER(i, j).re,
                                        &A_UPPER(i, j).im);
                            #else
                                scanf(" ( %lf , %lf )", &A_UPPER(i, j).re,
                                        &A_UPPER(i, j).im);
                            
                            #ifdef _WIN32
                                scanf_s("%*[\n ]");
                            #else
                                scanf("%*[\n ]");
                            #endif
                        }
                }
        }
if (fail.code == NE_NOERROR)
{
    /* Print solution, estimate of condition number and approximate */
    /* error bound */
    /* nag_gen_complex_mat_print_comp (f04djc). */
    /* Print complex general matrix (comprehensive) */
    fflush(stdout);
    nag_gen_complex_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
                                    n, nrhs, b, pdb, Nag Bracket Form, 0,
                                    "Solution", Nag_IntegerLabels, 0,
                                    Nag_IntegerLabels, 0, 80, 0, 0,
                                    &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_gen_complex_mat_print_comp (x04dbc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
printf("\n");
printf("%s\n%8s%10.1e\n", "Estimate of condition number", 
  1.0/rcond);
printf("\n\n");
printf("%s\n%8s%10.1e\n", "Estimate of error bound for computed solutions", 
  errbnd);
}
elif (fail.code == NE_RCOND)
{
/* Matrix A is numerically singular. Print estimate of */
/* reciprocal of condition number and solution */
printf("\n");
printf("%s\n%8s%10.1e\n", "Estimate of reciprocal of condition number", 
  rcond);
/* nag_gen_complex_mat_print_comp (x04dbc), see above. */
fflush(stdout);
nag_gen_complex_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
  n, nrhs, b, pdb, Nag_BracketForm, 0,
  "Solution", Nag_IntegerLabels, 0,
  Nag_IntegerLabels, 0, 80, 0, 0,
  &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_gen_complex_mat_print_comp (x04dbc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
}
elif (fail.code == NE_SINGULAR)
{
/* The upper triangular matrix U is exactly singular. Print */
/* details of factorization */
printf("\n");
/* nag_pack_complex_mat_print_comp (x04ddc). */
/* Print complex packed triangular matrix (comprehensive) */
fflush(stdout);
nag_pack_complex_mat_print_comp(order, Nag_Upper, Nag_NonUnitDiag, n, ap,
  Nag_BracketForm, 0,
  "Details of factorization", 
  Nag_IntegerLabels, 0, Nag_IntegerLabels,
  0, 80, 0, 0, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_pack_complex_mat_print_comp (x04ddc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Print pivot indices */
printf("\n");
printf("%s\n", "Pivot indices");
for (i = 1; i <= n; ++i)
{
printf("%11" NAG_FMT "s", ipiv[i - 1], i%7 == 0 || i == n?"\n":" ");
} else
{
  printf(
    "Error from nag_complex_sym_packed_lin_solve (f04djc).\n%s\n",
    fail.message);
  exit_status = 1;
  goto END;
}
END:
NAG_FREE(clabs);
NAG_FREE(rlabs);
NAG_FREE(ap);
NAG_FREE(b);
NAG_FREE(ipiv);
return exit_status;
}
#endif

10.2 Program Data

nag_complex_sym_packed_lin_solve (f04djc) Example Program Data

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4 2

Nag_Upper

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

10.3 Program Results

nag_complex_sym_packed_lin_solve (f04djc) Example Program Results

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Solution

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Estimate of condition number
2.1e+01

Estimate of error bound for computed solutions
2.3e-15

Mark 25