NAG Library Function Document
nag_complex_band_lin_solve (f04cbc)

1 Purpose
nag_complex_band_lin_solve (f04cbc) computes the solution to a complex system of linear equations
\( AX = B \), where \( A \) is an \( n \) by \( n \) band matrix, with \( k_l \) subdiagonals and \( k_u \) superdiagonals, 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
#include <nag.h>
#include <nagf04.h>
void nag_complex_band_lin_solve (Nag_OrderType order, Integer n, Integer kl,
                  Integer ku, Integer nrhs, Complex ab[], Integer pdab, Integer ipiv[],
                  Complex b[], Integer pdb, double *rcond, double *errbnd, NagError *fail)

3 Description
The \( LU \) decomposition with partial pivoting and row interchanges is used to factor \( A \) as \( A = PLU \), where \( P \) is a permutation matrix, \( L \) is the product of permutation matrices and unit lower triangular matrices with \( k_l \) subdiagonals, and \( U \) is upper triangular with \( (k_l + k_u) \) superdiagonals. 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
\( \text{Input} \)
\( \text{On entry:} \) 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 \( \text{order} = \text{Nag_RowMajor} \). See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.
\( \text{Constraint:} \ \text{order} = \text{Nag_RowMajor} \) or \( \text{Nag_ColMajor} \).

2: \( n \) – Integer
\( \text{Input} \)
\( \text{On entry:} \) the number of linear equations \( n \), i.e., the order of the matrix \( A \).
\( \text{Constraint:} \ n \geq 0. \)

3: \( kl \) – Integer
\( \text{Input} \)
\( \text{On entry:} \) the number of subdiagonals \( k_l \), within the band of \( A \).
\( \text{Constraint:} \ kl \geq 0. \)
4:  \( \textbf{ku} \) – Integer  

\textit{Input}

\textit{On entry}: the number of superdiagonals \( k_u \), within the band of \( A \).

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

5:  \( \textbf{nrhs} \) – Integer  

\textit{Input}

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

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

6:  \( \textbf{ab}[\textbf{dim}] \) – Complex  

\textit{Input/Output}

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

\textit{On entry}: the \( n \) by \( n \) matrix \( A \).

This is stored as a notional two-dimensional array with row elements or column elements stored contiguously. The storage of elements \( A_{ij} \), for row \( i = 1, \ldots, n \) and column \( j = \max(1, i - k_l), \ldots, \min(n, i + k_u) \), depends on the \textit{order} argument as follows:

- if \( \textit{order} = \text{Nag\_ColMajor} \), \( A_{ij} \) is stored as \( \textbf{ab}[(j - 1) \times \textbf{pdab} + \textbf{kl} + \textbf{ku} + i - j] \);
- if \( \textit{order} = \text{Nag\_RowMajor} \), \( A_{ij} \) is stored as \( \textbf{ab}[(i - 1) \times \textbf{pdab} + \textbf{kl} + j - i] \).

See Section 9 for further details.

\textit{On exit}: \( \textbf{ab} \) is overwritten by details of the factorization.

The elements, \( u_{ij} \), of the upper triangular band factor \( U \) with \( k_l + k_u \) super-diagonals, and the multipliers, \( l_{ij} \), used to form the lower triangular factor \( L \) are stored. The elements \( u_{ij} \), for \( i = 1, \ldots, n \) and \( j = \max(1, i - k_l), \ldots, i \), are stored where \( A_{ij} \) is stored on entry.

7:  \( \textbf{pdab} \) – Integer  

\textit{Input}

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

\textit{Constraint}: \( \textbf{pdab} \geq 2 \times \textbf{kl} + \textbf{ku} + 1 \).

8:  \( \textbf{ipiv}[\textbf{n}] \) – Integer  

\textit{Output}

\textit{On exit}: if \( \textbf{fail\_code} = \text{NE\_NOERROR} \), the pivot indices that define the permutation matrix \( P \); at the \( i \)th step row \( i \) of the matrix was interchanged with row \( \textbf{ipiv}[i - 1] \). \( \textbf{ipiv}[i - 1] = i \) indicates a row interchange was not required.

9:  \( \textbf{b}[\textbf{dim}] \) – Complex  

\textit{Input/Output}

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

\[ \max(1, \textbf{pdab} \times \textbf{nrhs}) \quad \text{when} \quad \textit{order} = \text{Nag\_ColMajor}; \]
\[ \max(1, \textbf{n} \times \textbf{pdab}) \quad \text{when} \quad \textit{order} = \text{Nag\_RowMajor}. \]

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

- \( \textbf{b}[(j - 1) \times \textbf{pdab} + i - 1] \) when \( \textit{order} = \text{Nag\_ColMajor} \);
- \( \textbf{b}[(i - 1) \times \textbf{pdab} + j - 1] \) when \( \textit{order} = \text{Nag\_RowMajor} \).

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

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

10:  \( \textbf{pdab} \) – Integer  

\textit{Input}

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

If `order = Nag_ColMajor`, `pdb ≥ max(1, n)`;
If `order = Nag_RowMajor`, `pdb ≥ max(1, nrhs)`.

11: `rcond` – double *

Output

On exit: if `fail.code = NE_NOERROR`, an estimate of the reciprocal of the condition number of the matrix `A`, computed as `rcond = (\|A\|_i \|A^{-1}\|_i)`.

12: `errbnd` – double *

Output

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 \( \|\hat{x} - x\|_1 / \|x\|_1 \leq errbnd \), 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.

13: `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, `kl = ⟨value⟩`.

Constraint: `kl ≥ 0`.

On entry, `ku = ⟨value⟩`.

Constraint: `ku ≥ 0`.

On entry, `n = ⟨value⟩`.

Constraint: `n ≥ 0`.

On entry, `nrhs = ⟨value⟩`.

Constraint: `nrhs ≥ 0`.

On entry, `pdab = ⟨value⟩`.

Constraint: `pdab > 0`.

On entry, `pdb = ⟨value⟩`.

Constraint: `pdb > 0`.

**NE_INT_2**

On entry, `pdab = ⟨value⟩`, `kl = ⟨value⟩` and `ku = ⟨value⟩`.

Constraint: `pdab ≥ 2 × kl + ku + 1`.

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_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 \( \text{rcond} \) is less than machine precision so that the matrix \( A \) is numerically singular.

**NE_SINGULAR**

Diagonal element (value) of the upper triangular factor 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. \( \text{nag_complex_band_lin_solve (f04cbc)} \) 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**

\( \text{nag_complex_band_lin_solve (f04cbc)} \) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

\( \text{nag_complex_band_lin_solve (f04cbc)} \) 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 double allocatable memory required is \( n \), and the Complex allocatable memory required is \( 2 \times n \). In this case the factorization and the solution \( X \) have been computed, but \( \text{rcond} \) and \( \text{errbnd} \) have not been computed.

The band storage scheme for the array \( ab \) is illustrated by the following example, when \( n = 5, k_l = 2, \) and \( k_u = 1 \). Storage of the band matrix \( A \) in the array \( ab \):
Array elements marked * need not be set and are not referenced by the function. Array elements marked + need not be set, but are defined on exit from the function and contain the elements $u_{13}$, $u_{14}$, $u_{24}$, $u_{25}$ and $u_{35}$. In this example when order = Nag_ColMajor the first referenced element of ab is ab[3] = $a_{11}$; while for order = Nag_RowMajor the first referenced element is ab[2] = $a_{11}$.

In general, elements $a_{ij}$ are stored as follows:

if order = Nag_ColMajor, $a_{ij}$ are stored in $ab[(j - 1) \times pdab + kl + ku + i - j]$

if order = Nag_RowMajor, $a_{ij}$ are stored in $ab[(i - 1) \times pdab + kl + j - i]$

where $\max(1, i - kl) \leq j \leq \min(n, i + ku)$.

The total number of floating-point operations required to solve the equations $AX = B$ depends upon the pivoting required, but if $n \gg kl + ku$ then it is approximately bounded by $O(nk_l(k_l + k_u))$ for the factorization and $O(n(2k_l + k_u)r)$ for the solution following the factorization. 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.

The real analogue of nag_complex_band_lin_solve (f04cbc) is nag_real_band_lin_solve (f04bbc).

10 Example

This example solves the equations

$$AX = B,$$

where $A$ is the band matrix

$$A = \begin{pmatrix}
-1.65 + 2.26i & -2.05 - 0.85i & 0.97 - 2.84i & 0 \\
0.00 + 6.30i & -1.48 - 1.75i & -3.99 + 4.01i & 0.59 - 0.48i \\
0 & -0.77 + 2.83i & -1.06 + 1.94i & 3.33 - 1.04i \\
0 & 0 & 4.48 - 1.09i & -0.46 - 1.72i
\end{pmatrix}$$

and

$$B = \begin{pmatrix}
-1.06 + 21.50i & 12.85 + 2.84i \\
-22.72 - 53.90i & -70.22 + 21.57i \\
28.24 - 38.60i & -20.73 - 1.23i \\
-34.56 + 16.73i & 26.01 + 31.97i
\end{pmatrix}.$$
10.1 Program Text

/* nag_complex_band_lin_solve (f04cbc) Example Program. *
 * Copyright 2014 Numerical Algorithms Group. *
 * Mark 8, 2004. *
 */

#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, kl, ku, n, nrhs, pdab, pdb;

    /* Arrays */
    char *clabs = 0, *rlabs = 0;
    Complex *ab = 0, *b = 0;
    Integer *ipiv = 0;

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

    #ifdef NAG_COLUMN_MAJOR
    #define AB(I, J) ab[(J-1)*pdab + kl + ku + I - J]
    #define B(I, J) b[(J-1)*pdb + I - 1]
    #else
    #define AB(I, J) ab[(I-1)*pdab + kl + J - I]
    #define B(I, J) b[(I-1)*pdb + J - 1]
    #endif

    exit_status = 0;
    INIT_FAIL(fail);

    printf("nag_complex_band_lin_solve (f04cbc)"
            " Example Program Results\n\n");

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

    #ifdef _WIN32
    scanf_s("%d%*d%*d%*d%*[^\n] ",
            &n, &kl, &ku, &nrhs);
    #else
    scanf("%d%*d%*d%*d%*[^\n] ",
            &n, &kl, &ku, &nrhs);
    #endif

    if (n > 0 && kl > 0 && ku > 0 && nrhs > 0)
    {
        /* Allocate memory */
        if (! (clabs = NAG_ALLOC(2, char)) ||
            ! (rlabs = NAG_ALLOC(2, char)) ||
            ! (ab = NAG_ALLOC((2*kl+ku+1)*n, Complex)) ||
            ! (b = NAG_ALLOC(n*nrhs, Complex)) ||
            ! (ipiv = NAG_ALLOC(n, Integer)))
            {
                printf("Allocation failure\n");
            }
    }

    /* Compute the solution and estimate the condition number */
    if (! (exit_status = nag_complex_band_lin_solve(n, kl, ku, n, nrhs,
                                                     &errbnd, &rcond,
                                                     &fail,
                                                     &order,
                                                     &ordjs)) ||
        exit_status)
        {
            printf("Error %d
", exit_status);
        }
}

f04cbc

NAG Library Manual

f04cbc.6 Mark 25
exit_status = -1;
go to END;
}
pdab = 2*kl+ku+1;
#endif NAG_COLUMN_MAJOR
pdb = n;
#else
pdb = nrhs;
#endif
} else {
    printf("%s\n", "One or more of NMAX, KLMAX, KUMAX or NRHSMX is"
    " too small");
    exit_status = 1;
    return exit_status;
}

/* Read A and B from data file */
for (i = 1; i <= n; ++i) {
    for (j = MAX(i - kl, 1); j <= MIN(i + ku, n); ++j) {
        #ifdef _WIN32
            scanf_s(" ( %lf , %lf )", &AB(i, j).re, &AB(i, j).im);
        #else
            scanf(" ( %lf , %lf )", &AB(i, j).re, &AB(i, j).im);
        #endif
    }
} #ifdef _WIN32
    scanf_s("%*[\n ] ");
#else
    scanf("%*[\n ] ");
#endif
for (i = 1; i <= n; ++i) {
    for (j = 1; j <= nrhs; ++j) {
        #ifdef _WIN32
            scanf_s(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
        #else
            scanf(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
        #endif
    }
} #ifdef _WIN32
    scanf_s("%*[\n ] ");
#else
    scanf("%*[\n ] ");
#endif

/* Solve the equations AX = B for X */
/* nag_complex_band_lin_solve (f04cbc). */
* Computes the solution and error-bound to a complex banded *
* system of linear equations */
# if (fail.code == NE_NOERROR)
    nag_complex_band_lin_solve(order, n, kl, ku, nrhs, ab, pdab, ipiv, b,
        pdb, &rcond, &errbnd, &fail);
#endif

/* Print solution, estimate of condition number and approximate */
/* error bound */
/* nag_gen_complx_mat_print_comp (x04dbc). */
* Print complex general matrix (comprehensive) */
fflush(stdout);
nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix,
        Nag_Gen_Complx_Mat_Print_Comp, n, nrhs, b, pdb,
        Nag_BracketForm, "%7.4f", "%7.4f", "%7.4f", "%7.4f", "%7.4f", "%7.4f", "%7.4f", "%7.4f", "%7.4f"
Solution, Nag_IntegerLabels, 0, Nag_IntegerLabels, 0, 80, 0, 0, &fail);

if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

printf("\n%s\n%4s%10.1e\n\n", "Estimate of condition number", "", 1.0/rcond);
printf("\n%s\n%4s%10.1e\n\n", "Estimate of error bound for computed solutions", "", errbnd);

else if (fail.code == NE_RCOND) {
    /* Matrix A is numerically singular. Print estimate of */
    /* reciprocal of condition number and solution */
    printf("\n%s\n%4s%10.1e\n\n", "Estimate of reciprocal of condition number", "", rcond);
    /* nag_gen_complx_mat_print_comp (x04dbc), see above. */
    fflush(stdout);
    nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix,
        Nag_NonUnitDiag, n, nrhs, b, pdb,
        Nag_BracketForm, "%7.4f",
        "Solution", Nag_IntegerLabels, 0,
        Nag_IntegerLabels, 0, 80, 0, 0,
        &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }
}

else if (fail.code == NE_SINGULAR) {
    /* The upper triangular matrix U is exactly singular. Print */
    /* details of factorization */
    printf("\n");
    /* nag_band_complx_mat_print_comp (x04dfc).
        Print complex packed banded matrix (comprehensive) */
    fflush(stdout);
    nag_band_complx_mat_print_comp(order, n, n, kl, kl + ku, ab, pdab,
        Nag_BracketForm, "%7.4f",
        "Details of factorization",
        Nag_IntegerLabels, 0,
        Nag_IntegerLabels, 0, 80, 0, 0,
        &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_band_complx_mat_print_comp (x04dfc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

    /* Print pivot indices */
    printf("\n%s\n", "Pivot indices");
    for (i = 1; i <= n; ++i) {

```
printf("%11NAG_IFMT"%s", ipiv[i - 1],
i%7 == 0 || i == n?"\n": "");
}
printf("\n");
} else {
printf("Error from nag_complex_band_lin_solve (f04cbc)\n", fail.message);
exit_status = 1;
goto END;
}
END:
NAG_FREE(clabs);
NAG_FREE(rlabs);
NAG_FREE(ab);
NAG_FREE(b);
NAG_FREE(ipiv);
return exit_status;
}

10.2 Program Data

nag_complex_band_lin_solve (f04cbc) Example Program Data

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1.65</td>
<td>2.26</td>
</tr>
<tr>
<td></td>
<td>-2.05</td>
<td>-0.85</td>
</tr>
<tr>
<td></td>
<td>0.97</td>
<td>-2.84</td>
</tr>
<tr>
<td>0.00</td>
<td>6.30</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-1.48</td>
<td>-1.75</td>
</tr>
<tr>
<td></td>
<td>-3.99</td>
<td>4.01</td>
</tr>
<tr>
<td></td>
<td>0.59</td>
<td>-0.48</td>
</tr>
<tr>
<td></td>
<td>-0.77</td>
<td>2.83</td>
</tr>
<tr>
<td></td>
<td>-1.06</td>
<td>1.94</td>
</tr>
<tr>
<td></td>
<td>3.33</td>
<td>-1.04</td>
</tr>
<tr>
<td></td>
<td>4.48</td>
<td>-1.09</td>
</tr>
<tr>
<td></td>
<td>-0.46</td>
<td>-1.72</td>
</tr>
</tbody>
</table>

End matrix A

|   | 1.06 | 21.50 |
|   | -22.72 | -53.90 |
|   | -70.22 | 21.57 |
|   | 28.24 | -38.60 |
|   | -20.73 | -1.23 |
|   | -34.56 | 16.73 |
|   | 26.01 | 31.97 |

End matrix B

10.3 Program Results

nag_complex_band_lin_solve (f04cbc) Example Program Results

Solution

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(-3.0000, 2.0000)</td>
<td>(1.0000, 6.0000)</td>
</tr>
<tr>
<td>2</td>
<td>(1.0000, -7.0000)</td>
<td>(-7.0000, -4.0000)</td>
</tr>
<tr>
<td>3</td>
<td>(-5.0000, 4.0000)</td>
<td>(3.0000, 5.0000)</td>
</tr>
<tr>
<td>4</td>
<td>(6.0000, -8.0000)</td>
<td>(-8.0000, 2.0000)</td>
</tr>
</tbody>
</table>

Estimate of condition number

1.0e+02

Estimate of error bound for computed solutions

1.2e-14