NAG Library Function Document

nag_dpbsvx (f07hbc)

1 Purpose

nag_dpbsvx (f07hbc) uses the Cholesky factorization

\[ A = U^T U \quad \text{or} \quad A = LL^T \]

to compute the solution to a real system of linear equations

\[ AX = B, \]

where \( A \) is an \( n \) by \( n \) symmetric positive definite band matrix of bandwidth \((2kd + 1)\) and \( X \) and \( B \) are \( n \) by \( r \) matrices. Error bounds on the solution and a condition estimate are also provided.

2 Specification

```c
#include <nag.h>
#include <nagf07.h>

void nag_dpbsvx (Nag_OrderType order, Nag_FactoredFormType fact,
    Nag_UploType uplo, Integer n, Integer kd, Integer nrhs,
    double ab[], Integer pdab, double afb[], Integer pdafb,
    Nag_EquilibrationType *equed, double s[], double b[],
    Integer pdb, double x[], Integer pdx, double *rcond,
    double ferr[], double berr[], NagError *fail)
```

3 Description

nag_dpbsvx (f07hbc) performs the following steps:

1. If `fact` = Nag_EquilibrateAndFactor, real diagonal scaling factors, \( D_S \), are computed to equilibrate the system:

\[ (D_S AD_S)(D_S^{-1} X) = D_S B. \]

Whether or not the system will be equilibrated depends on the scaling of the matrix \( A \), but if equilibration is used, \( A \) is overwritten by \( D_S AD_S \) and \( B \) by \( D_S B \).

2. If `fact` = Nag_NotFactored or Nag_EquilibrateAndFactor, the Cholesky decomposition is used to factor the matrix \( A \) (after equilibration if `fact` = Nag_EquilibrateAndFactor) as \( A = U^T U \) if `uplo` = Nag_Upper or \( A = LL^T \) if `uplo` = Nag_Lower, where \( U \) is an upper triangular matrix and \( L \) is a lower triangular matrix.

3. If the leading \( i \) by \( i \) principal minor of \( A \) is not positive definite, then the function returns with `fail.errnum` = \( i \) and `fail.code` = NE_MAT_NOT_POS_DEF. Otherwise, the factored form of \( A \) is used to estimate the condition number of the matrix \( A \). If the reciprocal of the condition number is less than `machine precision`, `fail.code` = NE_SINGULAR_WP is returned as a warning, but the function still goes on to solve for \( X \) and compute error bounds as described below.

4. The system of equations is solved for \( X \) using the factored form of \( A \).

5. Iterative refinement is applied to improve the computed solution matrix and to calculate error bounds and backward error estimates for it.

6. If equilibration was used, the matrix \( X \) is premultiplied by \( D_S \) so that it solves the original system before equilibration.
5 Arguments

1: \textbf{order} – Nag_OrderType \hspace{1cm} \textit{Input}

\textit{On entry}: the \textbf{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 \textbf{order} = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

\textit{Constraint}: \textbf{order} = Nag_RowMajor or Nag_ColMajor.

2: \textbf{fact} – Nag_FactoredFormType \hspace{1cm} \textit{Input}

\textit{On entry}: specifies whether or not the factorized form of the matrix \(A\) is supplied on entry, and if not, whether the matrix \(A\) should be equilibrated before it is factorized.

\textbf{fact} = Nag_Factored
\hspace{1.5cm} \textbf{afb} contains the factorized form of \(A\). If \textbf{equed} = Nag_Equilibrated, the matrix \(A\) has been equilibrated with scaling factors given by \textbf{s}. \textbf{ab} and \textbf{afb} will not be modified.

\textbf{fact} = Nag_NotFactored
\hspace{1.5cm} The matrix \(A\) will be copied to \textbf{afb} and factorized.

\textbf{fact} = Nag_EquilibrateAndFactor
\hspace{1.5cm} The matrix \(A\) will be equilibrated if necessary, then copied to \textbf{afb} and factorized.

\textit{Constraint}: \textbf{fact} = Nag_Factored, Nag_NotFactored or Nag_EquilibrateAndFactor.

3: \textbf{uplo} – Nag_UploType \hspace{1cm} \textit{Input}

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

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

\textit{Constraint}: \textbf{uplo} = Nag_Upper or Nag_Lower.

4: \textbf{n} – Integer \hspace{1cm} \textit{Input}

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

\textit{Constraint}: \(n \geq 0\).

5: \textbf{kd} – Integer \hspace{1cm} \textit{Input}

\textit{On entry}: \(kd\), the number of superdiagonals of the matrix \(A\) if \textbf{uplo} = Nag_Upper, or the number of subdiagonals if \textbf{uplo} = Nag_Lower.

\textit{Constraint}: \(kd \geq 0\).

6: \textbf{nrhs} – Integer \hspace{1cm} \textit{Input}

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

\textit{Constraint}: \(nrhs \geq 0\).
On entry: the upper or lower triangle of the symmetric band matrix $A$, except if $\text{fact} = \text{Nag\_Factored}$ and $\text{equed} = \text{Nag\_Equilibrated}$, in which case $\text{ab}$ must contain the equilibrated matrix $D_S A D_S$.

This is stored as a notional two-dimensional array with row elements or column elements stored contiguously. The storage of elements of $A_{ij}$, depends on the order and uplo arguments as follows:

- If $\text{order} = \text{Nag\_ColMajor}$ and $\text{uplo} = \text{Nag\_Upper}$, $A_{ij}$ is stored in $\text{ab}[k_a + i - j + (j - 1) \times \text{pdab}]$, for $j = 1, \ldots, n$ and $i = \max(1,j - k_a), \ldots, j$;
- If $\text{order} = \text{Nag\_ColMajor}$ and $\text{uplo} = \text{Nag\_Lower}$, $A_{ij}$ is stored in $\text{ab}[i - j + (j - 1) \times \text{pdab}]$, for $j = 1, \ldots, n$ and $i = j, \ldots, \min(n, j + k_a)$;
- If $\text{order} = \text{Nag\_RowMajor}$ and $\text{uplo} = \text{Nag\_Upper}$, $A_{ij}$ is stored in $\text{ab}[j - i + (i - 1) \times \text{pdab}]$, for $i = 1, \ldots, n$ and $j = i, \ldots, \min(n, i + k_d)$;
- If $\text{order} = \text{Nag\_RowMajor}$ and $\text{uplo} = \text{Nag\_Lower}$, $A_{ij}$ is stored in $\text{ab}[k_d + j - i + (i - 1) \times \text{pdab}]$, for $i = 1, \ldots, n$ and $j = \max(1,i - k_d), \ldots, i$.

On exit: if $\text{fact} = \text{Nag\_EquilibrateAndFactor}$ and $\text{equed} = \text{Nag\_Equilibrated}$, $\text{ab}$ is overwritten by $D_S A D_S$.

8: \text{pdab} – Integer

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

Constraint: $\text{pdab} \geq kd + 1$.

9: \text{afb}[dim] – double

Note: the dimension, dim, of the array $\text{afb}$ must be at least $\max(1, \text{pdafb} \times n)$.

On entry: if $\text{fact} = \text{Nag\_Factored}$, $\text{afb}$ contains the triangular factor $U$ or $L$ from the Cholesky factorization $A = U^T U$ or $A = LL^T$ of the band matrix $A$, in the same storage format as $A$. If $\text{equed} = \text{Nag\_Equilibrated}$, $\text{afb}$ is the factorized form of the equilibrated matrix $A$.

On exit: if $\text{fact} = \text{Nag\_NotFactored}$, $\text{afb}$ returns the triangular factor $U$ or $L$ from the Cholesky factorization $A = U^T U$ or $A = LL^T$.

If $\text{fact} = \text{Nag\_EquilibrateAndFactor}$, $\text{afb}$ returns the triangular factor $U$ or $L$ from the Cholesky factorization $A = U^T U$ or $A = LL^T$ of the equilibrated matrix $A$ (see the description of $\text{ab}$ for the form of the equilibrated matrix).

10: \text{pdafb} – Integer

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

Constraint: $\text{pdafb} \geq kd + 1$.

11: \text{equed} – Nag\_EquilibrationType *

On entry: if $\text{fact} = \text{Nag\_NotFactored}$ or $\text{Nag\_EquilibrateAndFactor}$, $\text{equed}$ need not be set.

If $\text{fact} = \text{Nag\_Factored}$, $\text{equed}$ must specify the form of the equilibration that was performed as follows:
if \texttt{equed} = \texttt{Nag\_NoEquilibration}, no equilibration;

if \texttt{equed} = \texttt{Nag\_Equilibrated}, equilibration was performed, i.e., \( A \) has been replaced by \( D_S A D_S \).

On exit: if \texttt{fact} = \texttt{Nag\_Factored}, \texttt{equed} is unchanged from entry.

Otherwise, if no constraints are violated, \texttt{equed} specifies the form of the equilibration that was performed as specified above.

Constraint: if \texttt{fact} = \texttt{Nag\_Factored}, \texttt{equed} = \texttt{Nag\_NoEquilibration} or \texttt{Nag\_Equilibrated}.

12: \( s[\text{dim}] \) – double \hspace{2cm} \textit{Input/Output}

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

On entry: if \texttt{fact} = \texttt{Nag\_NotFactored} or \texttt{Nag\_EquilibrateAndFactor}, \( s \) need not be set.

If \texttt{fact} = \texttt{Nag\_Factored} and \texttt{equed} = \texttt{Nag\_Equilibrated}, \( s \) must contain the scale factors, \( D_S \), for \( A \); each element of \( s \) must be positive.

On exit: if \texttt{fact} = \texttt{Nag\_Factored}, \( s \) is unchanged from entry.

Otherwise, if no constraints are violated and \texttt{equed} = \texttt{Nag\_Equilibrated}, \( s \) contains the scale factors, \( D_S \), for \( A \); each element of \( s \) is positive.

13: \( b[\text{dim}] \) – double \hspace{2cm} \textit{Input/Output}

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

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

\[ b[(j - 1) \times \texttt{pdb} + i - 1] \] when \texttt{order} = \texttt{Nag\_ColMajor};

\[ b[(i - 1) \times \texttt{pdb} + j - 1] \] when \texttt{order} = \texttt{Nag\_RowMajor}.

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

On exit: if \texttt{equed} = \texttt{Nag\_NoEquilibration}, \( b \) is not modified.

If \texttt{equed} = \texttt{Nag\_Equilibrated}, \( b \) is overwritten by \( D_S B \).

14: \( \texttt{pdb} \) – Integer \hspace{2cm} \textit{Input}

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

\textbf{Constraints:}

if \texttt{order} = \texttt{Nag\_ColMajor}, \( \texttt{pdb} \geq \max(1,n) \);

if \texttt{order} = \texttt{Nag\_RowMajor}, \( \texttt{pdb} \geq \max(1,nrhs) \).

15: \( x[\text{dim}] \) – double \hspace{2cm} \textit{Output}

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

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

\[ x[(j - 1) \times \texttt{pdx} + i - 1] \] when \texttt{order} = \texttt{Nag\_ColMajor};

\[ x[(i - 1) \times \texttt{pdx} + j - 1] \] when \texttt{order} = \texttt{Nag\_RowMajor}.

On exit: if \texttt{fail.code} = \texttt{NE\_NOERROR} or \texttt{NE\_SINGULAR\_WP}, the \( n \) by \( r \) solution matrix \( X \) to the original system of equations. Note that the arrays \( A \) and \( B \) are modified on exit if \texttt{equed} = \texttt{Nag\_Equilibrated}, and the solution to the equilibrated system is \( D_S^{-1} X \).
pdx – Integer  
*Input*

On entry: the stride separating row or column elements (depending on the value of `order`) in the array x.

Constraints:

if `order` = Nag_ColMajor, `pdx` ≥ max(1, n);
if `order` = Nag_RowMajor, `pdx` ≥ max(1, `nrhs`).

rcond – double *  
*Output*

On exit: if no constraints are violated, an estimate of the reciprocal condition number of the matrix A (after equilibration if that is performed), computed as `rcond` = 1.0/(∥A∥/∥A⁻¹∥).  

ferr[`nrhs`] – double  
*Output*

On exit: if `fail.code` = NE_NOERROR or NE_SINGULAR_WP, an estimate of the component-wise relative backward error of each computed solution vector `x̂` (i.e., the smallest relative change in any element of A or B that makes `x̂` an exact solution).

berr[`nrhs`] – double  
*Output*

On exit: if `fail.code` = NE_NOERROR or NE_SINGULAR_WP, an estimate of the component-wise relative backward error of each computed solution vector `x̂` (i.e., the smallest relative change in any element of A or B that makes `x̂` an exact solution).

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, `kd` = ⟨value⟩.
Constraint: `kd` ≥ 0.

On entry, `n` = ⟨value⟩.
Constraint: `n` ≥ 0.

On entry, `nrhs` = ⟨value⟩.
Constraint: `nrhs` ≥ 0.

On entry, `pdab` = ⟨value⟩.
Constraint: `pdab` > 0.

On entry, `pdafb` = ⟨value⟩.
Constraint: `pdafb` > 0.

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

**NE_INT_2**

On entry, \( pdab = \langle \text{value} \rangle \) and \( kd = \langle \text{value} \rangle \).
Constraint: \( pdab \geq kd + 1 \).

On entry, \( pdafb = \langle \text{value} \rangle \) and \( kd = \langle \text{value} \rangle \).
Constraint: \( pdafb \geq kd + 1 \).

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

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

**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 \( \langle \text{value} \rangle \) of \( A \) is not positive definite, so the factorization could not be completed, and the solution has not been computed. \( rcond = 0.0 \) is returned.

**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_SINGULAR_WP**

\( U \) (or \( L \)) is nonsingular, but \( rcond \) is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of \( rcond \) would suggest.

### 7 Accuracy

For each right-hand side vector \( b \), the computed solution \( x \) is the exact solution of a perturbed system of equations \((A + E)x = b\), where

- if \( uplo = \text{Nag\_Upper} \), \( |E| \leq c(n)\epsilon |U^T||U| \);
- if \( uplo = \text{Nag\_Lower} \), \( |E| \leq c(n)\epsilon |L^T||L| \),

\( c(n) \) is a modest linear function of \( n \), and \( \epsilon \) is the machine precision. See Section 10.1 of Higham (2002) for further details.

If \( \hat{x} \) is the true solution, then the computed solution \( x \) satisfies a forward error bound of the form

\[
\frac{||x - \hat{x}||_\infty}{||\hat{x}||_\infty} \leq \omega_\epsilon \text{cond}(A, \hat{x}, b)
\]

where \( \text{cond}(A, \hat{x}, b) = \left( ||A^{-1}|| \right) \left( ||A|| \epsilon + ||b|| \right) ||\hat{x}||_\infty \leq \text{cond}(A) = \left( ||A^{-1}|| \right) ||A||_\infty \leq \kappa_\infty(A) \). If \( \hat{x} \) is the
\( j \)th column of \( X \), then \( w_c \) is returned in \( \text{berr}[j-1] \) and a bound on \( \| x - \hat{x} \|_\infty / \| \hat{x} \|_\infty \) is returned in \( \text{ferr}[j-1] \). See Section 4.4 of Anderson et al. (1999) for further details.

8 Parallelism and Performance

\textbf{nag_dpbsvx (f07hbc)} is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

\textbf{nag_dpbsvx (f07hbc)} 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

When \( n \gg k \), the factorization of \( A \) requires approximately \( n(k+1)^2 \) floating-point operations, where \( k \) is the number of superdiagonals.

For each right-hand side, computation of the backward error involves a minimum of \( 8nk \) floating-point operations. Each step of iterative refinement involves an additional \( 12nk \) operations. At most five steps of iterative refinement are performed, but usually only one or two steps are required. Estimating the forward error involves solving a number of systems of equations of the form \( Ax = b \); the number is usually 4 or 5 and never more than 11. Each solution involves approximately \( 4nk \) operations.

The complex analogue of this function is \textbf{nag_zpbsvx (f07hpc)}.

10 Example

This example solves the equations

\[ AX = B, \]

where \( A \) is the symmetric positive definite band matrix

\[
A = \begin{pmatrix}
5.49 & 2.68 & 0 & 0 \\
2.68 & 5.63 & -2.39 & 0 \\
0 & -2.39 & 2.60 & -2.22 \\
0 & 0 & -2.22 & 5.17
\end{pmatrix}
\]

and

\[
B = \begin{pmatrix}
22.09 & 5.10 \\
9.31 & 30.81 \\
-5.24 & -25.82 \\
11.83 & 22.90
\end{pmatrix}
\]

Error estimates for the solutions, information on equilibration and an estimate of the reciprocal of the condition number of the scaled matrix \( A \) are also output.

10.1 Program Text

/ * nag_dpbsvx (f07hbc) 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 */
    double rcond;
    Integer exit_status = 0;
    Integer i, j, kd, n, nrhs, pdab, pdafb, pdb, pdx;

    /* Arrays */
    double *ab = 0, *afb = 0, *b = 0, *berr = 0, *ferr = 0;
    double *s = 0, *x = 0;
    char nag_enum_arg[40];

    /* Nag Types */
    NagError fail;
    Nag_UploType uplo;
    Nag_OrderType order;
    Nag_EquilibrationType equed;

    #ifdef NAG_COLUMN_MAJOR
    #define AB_UPPER(I, J) ab[(J-1)*pdab + kd + I - J]
    #define AB_LOWER(I, J) ab[(J-1)*pdab + I - J]
    #define B(I, J) b[(J-1)*pdb + I - 1]
    order = Nag_ColMajor;
    #else
    #define AB_UPPER(I, J) ab[(I-1)*pdab + J - I]
    #define AB_LOWER(I, J) ab[(I-1)*pdab + kd + J - I]
    #define B(I, J) b[(I-1)*pdb + J - 1]
    order = Nag_RowMajor;
    #endif

    INIT_FAIL(fail);

    printf("nag_dpbsvx (f07hbc) Example Program Results\n\n");
    /* Skip heading in data file */
    #ifdef _WIN32
    scanf_s("%*[\n]");
    #else
    scanf("%*[\n]");
    #endif
    #ifdef _WIN32
    scanf_s(" %39s%*[\n]", nag_enum_arg, _countof(nag_enum_arg));
    #else
    scanf(" %39s%*[\n]", nag_enum_arg);
    #endif

    if (n < 0 || nrhs < 0 || kd < 0)
    {
        printf("Invalid n, kd or nrhs\n");
        exit_status = 1;
        goto END;
    }

    if (n < 0 || nrhs < 0 || kd < 0)
    {
        printf("Invalid n, kd or nrhs\n");
        exit_status = 1;
        goto END;
    }

    uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);

    #ifdef _WIN32
    scanf_s(" %39s%*[\n]", nag_enum_arg, _countof(nag_enum_arg));
    #else
    scanf(" %39s%*[\n]", nag_enum_arg);
    #endif

    /* Allocate memory */
    if (! (ab = NAG_ALLOC((kd+1) * n, double)) ||
        ! (afb = NAG_ALLOC((kd+1) * n, double)) ||
        ! (b = NAG_ALLOC(n * nrhs, double)) ||
        ! (berr = NAG_ALLOC(nrhs, double)) ||
        ! (ferr = NAG_ALLOC(nrhs, double)) ||
        ! (s = NAG_ALLOC(n, double)) ||
    }
!(x = NAG_ALLOC(n * nrhs, double)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

pdab = kd+1;
pdafb = kd+1;
#ifdef NAG_COLUMN_MAJOR
    pdb = n;
pdx = n;
#else
    pdb = nrhs;
pdx = nrhs;
#endif

/* Read the upper or lower triangular part of the band matrix A from
 * data file.
*/
if (uplo == Nag_Upper)
    for (i = 1; i <= n; ++i)
#ifdef _WIN32
        for (j = i; j <= MIN(n, i + kd); ++j) scanf_s("%lf", &AB_UPPER(i, j));
#else
        for (j = i; j <= MIN(n, i + kd); ++j) scanf("%lf", &AB_UPPER(i, j));
#endif
#else
    for (i = 1; i <= n; ++i)
#ifdef _WIN32
        for (j = MAX(1, i - kd); j <= i; ++j) scanf_s("%lf", &AB_LOWER(i, j));
#else
        for (j = MAX(1, i - kd); j <= i; ++j) scanf("%lf", &AB_LOWER(i, j));
#endif
#endif
    scanf("%*[\n]");
/* 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
    scanf("%*[\n]");

/* Solve the equations AX = B for X using nag_dpbsvx (f07hbc). */
fflush(stdout);
nag_dpbsvx(order, Nag_EquilibrateAndFactor, uplo, n, kd, nrhs, ab, pdab, afb, pdafb, s, b, pdb, x, pdx, &rcond, ferr, berr, &fail);
if (fail.code != NE_NOERROR & fail.code != NE_SINGULAR)
{
    printf("Error from nag_dpbsvx (f07hbc).\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print solution using nag_gen_real_mat_print (x04cac). */
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, nrhs, x, pdx, "Solution(s)", 0, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_gen_real_mat_print (x04cac).\n", fail.message);
}
exit_status = 1;
goto END;
}
/* Print error bounds, condition number and the form of equilibration */
printf("\nBackward errors (machine-dependent)\n");
for (j = 0; j < nrhs; ++j) printf("%11.1e%s", berr[j], j%7 == 6?"\n":" ");
printf("\n\nEstimated forward error bounds (machine-dependent)\n");
for (j = 0; j < nrhs; ++j) printf("%11.1e%s", ferr[j], j%7 == 6?"\n":" ");
printf("\n\nEstimate of reciprocal condition number\n%11.1e\n\n", rcond);
if (equed == Nag_NoEquilibration)
    printf("A has not been equilibrated\n");
else if (equed == Nag_RowAndColumnEquilibration)
    printf("A has been row and column scaled as diag(S)*A*diag(S)\n");
if (fail.code == NE_SINGULAR)
    {
        printf("Error from nag_dpbsvx (f07hbc).\n%s\n", fail.message);
        exit_status = 1;
    }
END:
NAG_FREE(ab);
NAG_FREE(afb);
NAG_FREE(b);
NAG_FREE(berr);
NAG_FREE(ferr);
NAG_FREE(s);
NAG_FREE(x);
return exit_status;
}
#undef AB_UPPER
#undef AB_LOWER
#undef B

10.2 Program Data

nag_dpbsvx (f07hbc) Example Program Data
4 1 2 : n, kd and nrhs
Nag_Upper : uplo
5.49 2.68 5.63 -2.39 2.60 -2.22 5.17 : matrix A
22.09 5.10 9.31 30.81 -5.24 -25.82 11.83 22.90 : matrix B
10.3 Program Results

nag_dpbsvx (f07hbc) Example Program Results

Solution(s)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.0000</td>
<td>-2.0000</td>
</tr>
<tr>
<td>2</td>
<td>-2.0000</td>
<td>6.0000</td>
</tr>
<tr>
<td>3</td>
<td>-3.0000</td>
<td>-1.0000</td>
</tr>
<tr>
<td>4</td>
<td>1.0000</td>
<td>4.0000</td>
</tr>
</tbody>
</table>

Backward errors (machine-dependent)

1.1e-16 1.1e-16

Estimated forward error bounds (machine-dependent)

2.1e-14 3.0e-14

Estimate of reciprocal condition number

1.3e-02

A has not been equilibrated