NAG Library Function Document

nag_dgbsvx (f07bbc)

1 Purpose

nag_dgbsvx (f07bbc) uses the LU factorization to compute the solution to a real system of linear equations

$$AX = B \quad \text{or} \quad A^{\mathrm{T}}X = B,$$

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. Error bounds on the solution and a condition estimate are also provided.

2 Specification

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

```
void nag_dgbsvx (Nag_OrderType order, Nag_FactoredFormType fact,
     Nag_TransType trans, Integer n, Integer kl, Integer ku, Integer nrhs,
     double ab[], Integer pdab, double afb[], Integer pdafb, Integer ipiv[],
     Nag_EquilibrationType *equed, double r[], double c[], double b[],
     Integer pdb, double x[], Integer pdx, double *rcond, double ferr[],
     double berr[], double *recip_growth_factor, NagError *fail)
```

3 Description

nag_dgbsvx (f07bbc) performs the following steps:

1. **Equilibration**

   The linear system to be solved may be badly scaled. However, the system can be equilibrated as a first stage by setting **fact** = Nag_EquilibrateAndFactor. In this case, real scaling factors are computed and these factors then determine whether the system is to be equilibrated. Equilibrated forms of the systems $AX = B$ and $A^{\mathrm{T}}X = B$ are

   $$(D_R A D_C)\left(D_C^{-1}X\right) = D_R B$$

   and

   $$(D_R A D_C)^{\mathrm{T}}\left(D_R^{-1}X\right) = D_C B,$$

   respectively, where D_R and D_C are diagonal matrices, with positive diagonal elements, formed from the computed scaling factors.

   When equilibration is used, A will be overwritten by $D_R A D_C$ and B will be overwritten by $D_R B$ (or $D_C B$ when the solution of $A^{\mathrm{T}}X = B$ is sought).

2. **Factorization**

   The matrix A, or its scaled form, is copied and factored using the LU decomposition

   $$A = PLU,$$

   where P is a permutation matrix, L is a unit lower triangular matrix, and U is upper triangular.

   This stage can be by-passed when a factored matrix (with scaled matrices and scaling factors) are supplied; for example, as provided by a previous call to nag_dgbsvx (f07bbc) with the same matrix A.

3. **Condition Number Estimation**

   The LU factorization of A determines whether a solution to the linear system exists. If some diagonal element of U is zero, then U is exactly singular, no solution exists and the function returns with a failure. Otherwise the factorized 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*** then a warning code is returned on final exit.

4. **Solution**

   The (equilibrated) system is solved for X ($D_C^{-1}X$ or $D_R^{-1}X$) using the factored form of A ($D_R A D_C$).

5. **Iterative Refinement**

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

6. **Construct Solution Matrix *X***

   If equilibration was used, the matrix X is premultiplied by D_C (if **trans** $=$ Nag_NoTrans) or D_R (if **trans** $=$ Nag_Trans or Nag_ConjTrans) so that it solves the original system before equilibration.

4 References

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A and Sorensen D (1999) *LAPACK Users' Guide* (3rd Edition) SIAM, Philadelphia http://www.netlib.org/lapack/lug

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

Higham N J (2002) *Accuracy and Stability of Numerical Algorithms* (2nd Edition) SIAM, Philadelphia

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: **fact** – Nag_FactoredFormType *Input*

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.

fact $=$ Nag_Factored
: **afb** and **ipiv** contain the factorized form of A. If **equed** $\neq$ Nag_NoEquilibration, the matrix A has been equilibrated with scaling factors given by **r** and **c**. **ab**, **afb** and **ipiv** are not modified.

fact $=$ Nag_NotFactored
: The matrix A will be copied to **afb** and factorized.

fact $=$ Nag_EquilibrateAndFactor
: The matrix A will be equilibrated if necessary, then copied to **afb** and factorized.

Constraint: **fact** $=$ Nag_Factored, Nag_NotFactored or Nag_EquilibrateAndFactor.

3: **trans** – Nag_TransType *Input*

On entry: specifies the form of the system of equations.

trans = Nag_NoTrans
$AX = B$ (No transpose).

trans = Nag_Trans or Nag_ConjTrans
$A^{\mathrm{T}}X = B$ (Transpose).

Constraint: **trans** = Nag_NoTrans, Nag_Trans or Nag_ConjTrans.

4: **n** – Integer *Input*

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

Constraint: **n** ≥ 0.

5: **kl** – Integer *Input*

On entry: k_l, the number of subdiagonals within the band of the matrix A.

Constraint: **kl** ≥ 0.

6: **ku** – Integer *Input*

On entry: k_u, the number of superdiagonals within the band of the matrix A.

Constraint: **ku** ≥ 0.

7: **nrhs** – Integer *Input*

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

Constraint: **nrhs** ≥ 0.

8: **ab**$[dim]$ – double *Input/Output*

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

On entry: the n by n coefficient 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 **order** argument as follows:

if **order** = 'Nag_ColMajor', A_{ij} is stored as $\mathbf{ab}[(j-1) \times \mathbf{pdab} + \mathbf{ku} + i - j]$;

if **order** = 'Nag_RowMajor', A_{ij} is stored as $\mathbf{ab}[(i-1) \times \mathbf{pdab} + \mathbf{kl} + j - i]$.

See Section 9 for further details.

If **fact** = Nag_Factored and **equed** $\neq$ Nag_NoEquilibration, A must have been equilibrated by the scaling factors in **r** and/or **c**.

On exit: if **fact** = Nag_Factored or Nag_NotFactored, or if **fact** = Nag_EquilibrateAndFactor and **equed** = Nag_NoEquilibration, **ab** is not modified.

If **equed** $\neq$ Nag_NoEquilibration then, if no constraints are violated, A is scaled as follows:

if **equed** = Nag_RowEquilibration, $A = D_rA$;

if **equed** = Nag_ColumnEquilibration, $A = AD_c$;

if **equed** = Nag_RowAndColumnEquilibration, $A = D_rAD_c$.

9: **pdab** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) of the matrix A in the array **ab**.

Constraint: $\mathbf{pdab} \geq \mathbf{kl} + \mathbf{ku} + 1$.

10: **afb**[dim] – double *Input/Output*

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

On entry: if **fact** = Nag_NotFactored or Nag_EquilibrateAndFactor, **afb** need not be set.

If **fact** = Nag_Factored, details of the LU factorization of the n by n band matrix A, as computed by nag_dgbtrf (f07bdc).

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 $j = i, \ldots, \min(n, i + k_l + k_u)$, and l_{ij}, for $i = 1, \ldots, n$ and $j = \max(1, i - k_l), \ldots, i$, are stored where A_{ij} is stored on entry.

If **equed** $\neq$ Nag_NoEquilibration, **afb** is the factorized form of the equilibrated matrix A.

On exit: if **fact** = Nag_Factored, **afb** is unchanged from entry.

Otherwise, if no constraints are violated, then if **fact** = Nag_NotFactored, **afb** returns details of the LU factorization of the band matrix A, and if **fact** = Nag_EquilibrateAndFactor, **afb** returns details of the LU factorization of the equilibrated band matrix A (see the description of **ab** for the form of the equilibrated matrix).

11: **pdafb** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) of the matrix A in the array **afb**.

Constraint: $\mathbf{pdafb} \geq 2 \times \mathbf{kl} + \mathbf{ku} + 1$.

12: **ipiv**[dim] – Integer *Input/Output*

Note: the dimension, *dim*, of the array **ipiv** must be at least $\max(1, \mathbf{n})$.

On entry: if **fact** = Nag_NotFactored or Nag_EquilibrateAndFactor, **ipiv** need not be set.

If **fact** = Nag_Factored, **ipiv** contains the pivot indices from the factorization $A = LU$, as computed by nag_dgbtrf (f07bdc); row i of the matrix was interchanged with row $\mathbf{ipiv}[i-1]$.

On exit: if **fact** = Nag_Factored, **ipiv** is unchanged from entry.

Otherwise, if no constraints are violated, **ipiv** contains the pivot indices that define the permutation matrix P; at the ith step row i of the matrix was interchanged with row $\mathbf{ipiv}[i-1]$. $\mathbf{ipiv}[i-1] = i$ indicates a row interchange was not required.

If **fact** = Nag_NotFactored, the pivot indices are those corresponding to the factorization $A = LU$ of the original matrix A.

If **fact** = Nag_EquilibrateAndFactor, the pivot indices are those corresponding to the factorization of $A = LU$ of the equilibrated matrix A.

13: **equed** – Nag_EquilibrationType * *Input/Output*

On entry: if **fact** = Nag_NotFactored or Nag_EquilibrateAndFactor, **equed** need not be set.

If **fact** = Nag_Factored, **equed** must specify the form of the equilibration that was performed as follows:

if **equed** = Nag_NoEquilibration, no equilibration;

if **equed** = Nag_RowEquilibration, row equilibration, i.e., A has been premultiplied by D_R;

if **equed** = Nag_ColumnEquilibration, column equilibration, i.e., A has been postmultiplied by D_C;

if **equed** = Nag_RowAndColumnEquilibration, both row and column equilibration, i.e., A has been replaced by $D_R A D_C$.

On exit: if **fact** = Nag_Factored, **equed** is unchanged from entry.

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

Constraint: if **fact** = Nag_Factored, **equed** = Nag_NoEquilibration, Nag_RowEquilibration, Nag_ColumnEquilibration or Nag_RowAndColumnEquilibration.

14: **r**[dim] – double *Input/Output*

Note: the dimension, *dim*, of the array **r** must be at least $\max(1, \mathbf{n})$.

On entry: if **fact** = Nag_NotFactored or Nag_EquilibrateAndFactor, **r** need not be set.

If **fact** = Nag_Factored and **equed** = Nag_RowEquilibration or Nag_RowAndColumnEquilibration, **r** must contain the row scale factors for A, D_R; each element of **r** must be positive.

On exit: if **fact** = Nag_Factored, **r** is unchanged from entry.

Otherwise, if no constraints are violated and **equed** = Nag_RowEquilibration or Nag_RowAndColumnEquilibration, **r** contains the row scale factors for A, D_R, such that A is multiplied on the left by D_R; each element of **r** is positive.

15: **c**[dim] – double *Input/Output*

Note: the dimension, *dim*, of the array **c** must be at least $\max(1, \mathbf{n})$.

On entry: if **fact** = Nag_NotFactored or Nag_EquilibrateAndFactor, **c** need not be set.

If **fact** = Nag_Factored or **equed** = Nag_ColumnEquilibration or Nag_RowAndColumnEquilibration, **c** must contain the column scale factors for A, D_C; each element of **c** must be positive.

On exit: if **fact** = Nag_Factored, **c** is unchanged from entry.

Otherwise, if no constraints are violated and **equed** = Nag_ColumnEquilibration or Nag_RowAndColumnEquilibration, **c** contains the row scale factors for A, D_C; each element of **c** is positive.

16: **b**[dim] – double *Input/Output*

Note: the dimension, *dim*, of the array **b** must be at least

$\max(1, \mathbf{pdb} \times \mathbf{nrhs})$ when **order** = Nag_ColMajor;
$\max(1, \mathbf{n} \times \mathbf{pdb})$ when **order** = Nag_RowMajor.

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

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

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

On exit: if **equed** = Nag_NoEquilibration, **b** is not modified.

If **trans** = Nag_NoTrans and **equed** = Nag_RowEquilibration or Nag_RowAndColumnEquilibration, **b** is overwritten by $D_R B$.

If **trans** = Nag_Trans or Nag_ConjTrans and **equed** = Nag_ColumnEquilibration or Nag_RowAndColumnEquilibration, **b** is overwritten by $D_C B$.

17: **pdb** – Integer *Input*

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

Constraints:

if **order** = Nag_ColMajor, **pdb** $\geq \max(1, \mathbf{n})$;
if **order** = Nag_RowMajor, **pdb** $\geq \max(1, \mathbf{nrhs})$.

18: $\mathbf{x}[dim]$ – double *Output*

Note: the dimension, *dim*, of the array **x** must be at least

$\max(1, \mathbf{pdx} \times \mathbf{nrhs})$ when **order** = Nag_ColMajor;
$\max(1, \mathbf{n} \times \mathbf{pdx})$ when **order** = Nag_RowMajor.

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

$\mathbf{x}[(j-1) \times \mathbf{pdx} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{x}[(i-1) \times \mathbf{pdx} + j - 1]$ when **order** = Nag_RowMajor.

On exit: if **fail.code** = NE_NOERROR or 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 **equed** $\neq$ Nag_NoEquilibration, and the solution to the equilibrated system is $D_C^{-1}X$ if **trans** = Nag_NoTrans and **equed** = Nag_ColumnEquilibration or Nag_RowAndColumnEquilibration, or $D_R^{-1}X$ if **trans** = Nag_Trans or Nag_ConjTrans and **equed** = Nag_RowEquilibration or Nag_RowAndColumnEquilibration.

19: **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** $\geq \max(1, \mathbf{n})$;
if **order** = Nag_RowMajor, **pdx** $\geq \max(1, \mathbf{nrhs})$.

20: **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 $\mathbf{rcond} = 1.0/\left(\|A\|_1\|A^{-1}\|_1\right)$.

21: **ferr**[**nrhs**] – double *Output*

On exit: if **fail.code** = NE_NOERROR or NE_SINGULAR_WP, an estimate of the forward error bound for each computed solution vector, such that $\|\hat{x}_j - x_j\|_\infty / \|x_j\|_\infty \leq \mathbf{ferr}[j-1]$ where $\hat{x}_j$ is the jth column of the computed solution returned in the array **x** and x_j is the corresponding column of the exact solution X. The estimate is as reliable as the estimate for **rcond**, and is almost always a slight overestimate of the true error.

22: **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 $\hat{x}_j$ (i.e., the smallest relative change in any element of A or B that makes $\hat{x}_j$ an exact solution).

23: **recip_growth_factor** – double * *Output*

On exit: if **fail.code** = NE_NOERROR, the reciprocal pivot growth factor $\|A\|/\|U\|$, where $\|.\|$ denotes the maximum absolute element norm. If **recip_growth_factor** $\ll 1$, the stability of the LU factorization of (equilibrated) A could be poor. This also means that the solution **x**, condition estimate **rcond**, and forward error bound **ferr** could be unreliable. If the factorization fails with

fail.code = NE_SINGULAR, then **recip_growth_factor** contains the reciprocal pivot growth factor for the leading **fail.errnum** columns of A.

24: **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.

NE_BAD_PARAM

On entry, argument $\langle value \rangle$ had an illegal value.

NE_INT

On entry, **kl** $= \langle value \rangle$.
Constraint: **kl** ≥ 0.

On entry, **ku** $= \langle value \rangle$.
Constraint: **ku** ≥ 0.

On entry, **n** $= \langle value \rangle$.
Constraint: **n** ≥ 0.

On entry, **nrhs** $= \langle value \rangle$.
Constraint: **nrhs** ≥ 0.

On entry, **pdab** $= \langle value \rangle$.
Constraint: **pdab** > 0.

On entry, **pdafb** $= \langle value \rangle$.
Constraint: **pdafb** > 0.

On entry, **pdb** $= \langle value \rangle$.
Constraint: **pdb** > 0.

On entry, **pdx** $= \langle value \rangle$.
Constraint: **pdx** > 0.

NE_INT_2

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

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

On entry, **pdx** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: **pdx** $\geq \max(1, \mathbf{n})$.

On entry, **pdx** $= \langle value \rangle$ and **nrhs** $= \langle value \rangle$.
Constraint: **pdx** $\geq \max(1, \mathbf{nrhs})$.

NE_INT_3

On entry, **pdab** $= \langle value \rangle$, **kl** $= \langle value \rangle$ and **ku** $= \langle value \rangle$.
Constraint: **pdab** $\geq$ **kl** + **ku** + 1.

On entry, **pdafb** $= \langle value \rangle$, **kl** $= \langle value \rangle$ and **ku** $= \langle value \rangle$.
Constraint: **pdafb** $\geq 2 \times$ **kl** + **ku** + 1.

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.

NE_SINGULAR

$U(\langle value\rangle, \langle value\rangle)$ is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. **rcond** $= 0.0$ is returned.

NE_SINGULAR_WP

U 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 $\hat{x}$ is the exact solution of a perturbed system of equations $(A+E)\hat{x} = b$, where

$$|E| \leq c(n)\epsilon P|L||U|,$$

$c(n)$ is a modest linear function of n, and ϵ is the ***machine precision***. See Section 9.3 of Higham (2002) for further details.

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

$$\frac{\|x-\hat{x}\|_\infty}{\|\hat{x}\|_\infty} \leq w_c \operatorname{cond}(A, \hat{x}, b)$$

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

8 Parallelism and Performance

nag_dgbsvx (f07bbc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_dgbsvx (f07bbc) 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 Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

The band storage scheme for the array **ab** is illustrated by the following example, when $n = 6$, $k_l = 1$, and $k_u = 2$. Storage of the band matrix A in the array **ab**:

order = Nag_ColMajor

$$\begin{array}{cccccc} * & * & a_{13} & a_{24} & a_{35} & a_{46} \\ * & a_{12} & a_{23} & a_{34} & a_{45} & a_{56} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} & a_{66} \\ a_{21} & a_{32} & a_{43} & a_{54} & a_{65} & * \end{array}$$

order = Nag_RowMajor

$$\begin{array}{cccc} * & a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{32} & a_{33} & a_{34} & a_{35} \\ a_{43} & a_{44} & a_{45} & a_{46} \\ a_{54} & a_{55} & a_{56} & * \\ a_{65} & a_{66} & * & * \end{array}$$

The total number of floating-point operations required to solve the equations $AX = B$ depends upon the pivoting required, but if $n \gg k_l + k_u$ 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. The solution is then refined, and the errors estimated, using iterative refinement; see nag_dgbrfs (f07bhc) for information on the floating-point operations required.

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 complex analogue of this function is nag_zgbsvx (f07bpc).

10 Example

This example solves the equations

$$AX = B,$$

where A is the band matrix

$$A = \begin{pmatrix} -0.23 & 2.54 & -3.66 & 0 \\ -6.98 & 2.46 & -2.73 & -2.13 \\ 0 & 2.56 & 2.46 & 4.07 \\ 0 & 0 & -4.78 & -3.82 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 4.42 & -36.01 \\ 27.13 & -31.67 \\ -6.14 & -1.16 \\ 10.50 & -25.82 \end{pmatrix}.$$

Estimates for the backward errors, forward errors, condition number and pivot growth are also output, together with information on the equilibration of A.

10.1 Program Text

```
/* nag_dgbsvx (f07bbc) Example Program.
 *
 * Copyright 2011 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                 growth_factor, rcond;
  Integer                exit_status = 0, i, j, kl, ku, n, nrhs, pdab,
                         pdafb, pdb, pdx;

  /* Arrays */
  double                 *ab = 0, *afb = 0, *b = 0, *berr = 0, *c = 0;
  double                 *ferr = 0, *r = 0, *x = 0;
  Integer                *ipiv = 0;

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

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

```
  INIT_FAIL(fail);

  printf("nag_dgbsvx (f07bbc) Example Program Results\n\n");

  /* Skip heading in data file */
  scanf("%*[^\n] ");
  scanf("%ld%ld%ld%ld%*[^\n]", &n, &nrhs,
        &kl, &ku);
  if (n < 0 || kl < 0 || ku < 0 || nrhs < 0)
    {
      printf("Invalid n, kl, ku or nrhs\n");
      exit_status = 1;
      goto END;
    }

  pdab = kl+ku+1;
  pdafb = 2*kl+ku+1;
#ifdef NAG_COLUMN_MAJOR
  pdx = n;
  pdb = n;
#else
  pdx = nrhs;
  pdb = nrhs;
#endif

  /* Allocate memory */
  if (!(ab   = NAG_ALLOC(pdab * n, double)) ||
      !(afb  = NAG_ALLOC(pdafb * n, double)) ||
      !(b    = NAG_ALLOC(n*nrhs, double)) ||
      !(berr = NAG_ALLOC(nrhs, double)) ||
      !(c    = NAG_ALLOC(n, double)) ||
      !(ferr = NAG_ALLOC(nrhs, double)) ||
      !(r    = NAG_ALLOC(n, double)) ||
      !(x    = NAG_ALLOC(n*nrhs, double)) ||
      !(ipiv = NAG_ALLOC(n, Integer)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read the band matrix A and B from data file */
  for (i = 1; i <= n; ++i)
    for (j = MAX(i - kl, 1); j <= MIN(i + ku, n); ++j)
      scanf("%lf", &AB(i, j));
  scanf("%*[^\n]");

  for (i = 1; i <= n; ++i)
    for (j = 1; j <= nrhs; ++j) scanf("%lf", &B(i, j));
  scanf("%*[^\n]");


  /* Solve the equations Ax = B for x using  nag_dgbsvx (f07bbc). */
  nag_dgbsvx(order, Nag_EquilibrateAndFactor, Nag_NoTrans, n, kl, ku, nrhs, ab,
             pdab, afb, pdafb, ipiv, &equed, r, c, b, pdb, x, pdx, &rcond,
             ferr, berr, &growth_factor, &fail);
  if (fail.code != NE_NOERROR && fail.code != NE_SINGULAR)
    {
      printf("Error from nag_dgbsvx (f07bbc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Print solution using nag_gen_real_mat_print (x04cac). */
  fflush(stdout);
  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%s\n",
```

```
              fail.message);
      exit_status = 1;
      goto END;
    }

  /* Print error bounds, condition number, the form of equilibration
   * and the pivot growth factor
   */
  printf("\nBackward errors (machine-dependent)\n");
  for (j = 1; j <= nrhs; ++j)
    printf("%11.1e%s", berr[j - 1], j%7 == 0?"\n":" ");

  printf("\n\nEstimated forward error bounds (machine-dependent)\n");
  for (j = 1; j <= nrhs; ++j)
    printf("%11.1e%s", ferr[j - 1], j%7 == 0?"\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_RowEquilibration)
    printf("A has been row scaled as diag(R)*A\n");
  else if (equed == Nag_ColumnEquilibration)
    printf("A has been column scaled as A*diag(C)\n");
  else if (equed == Nag_RowAndColumnEquilibration)
    printf("A has been row and column scaled as diag(R)*A*diag(C)\n");

  printf("\nEstimate of reciprocal pivot growth factor\n%11.1e\n",
          growth_factor);

  if (fail.code == NE_SINGULAR)
    printf("Error from nag_dgbsvx (f07bbc).\n%s\n", fail.message);
 END:
  NAG_FREE(ab);
  NAG_FREE(afb);
  NAG_FREE(b);
  NAG_FREE(berr);
  NAG_FREE(c);
  NAG_FREE(ferr);
  NAG_FREE(r);
  NAG_FREE(x);
  NAG_FREE(ipiv);

  return exit_status;
}

#undef AB
#undef B
```

10.2 Program Data

```
nag_dgbsvx (f07bbc) Example Program Data
  4      2      1      2      : n, nrhs, kl and ku
 -0.23   2.54  -3.66
 -6.98   2.46  -2.73  -2.13
         2.56   2.46   4.07
               -4.78  -3.82   : matrix A
  4.42 -36.01
 27.13 -31.67
 -6.14  -1.16
 10.50 -25.82                 : matrix B
```

10.3 Program Results

```
nag_dgbsvx (f07bbc) Example Program Results

 Solution(s)
            1          2
 1     -2.0000     1.0000
 2      3.0000    -4.0000
 3      1.0000     7.0000
 4     -4.0000    -2.0000

Backward errors (machine-dependent)
    1.1e-16     9.9e-17

Estimated forward error bounds (machine-dependent)
    1.6e-14     1.9e-14

Estimate of reciprocal condition number
    1.8e-02

A has not been equilibrated

Estimate of reciprocal pivot growth factor
    1.0e+00
```