NAG Library Function Document
nag_dsbgv (f08uac)

1 Purpose

nag_dsbgv (f08uac) computes all the eigenvalues and, optionally, the eigenvectors of a real generalized symmetric-definite banded eigenproblem, of the form

\[ Az = \lambda Bz, \]

where \( A \) and \( B \) are symmetric and banded, and \( B \) is also positive definite.

2 Specification

```c
#include <nag.h>
#include <nagf08.h>

void nag_dsbgv (Nag_OrderType order, Nag_JobType job, Nag_UploType uplo,
                Integer n, Integer ka, Integer kb, double ab[], Integer pdab,
                double bb[], Integer pdbb, double w[], double z[], Integer pdz,
                NagError *fail)
```

3 Description

The generalized symmetric-definite band problem

\[ Az = \lambda Bz \]

is first reduced to a standard band symmetric problem

\[ Cx = \lambda x, \]

where \( C \) is a symmetric band matrix, using Wilkinson’s modification to Crawford’s algorithm (see Crawford (1973) and Wilkinson (1977)). The symmetric eigenvalue problem is then solved for the eigenvalues and the eigenvectors, if required, which are then backtransformed to the eigenvectors of the original problem.

The eigenvectors are normalized so that the matrix of eigenvectors, \( Z \), satisfies

\[ Z^T A Z = \Lambda \quad \text{and} \quad Z^T B Z = I, \]

where \( \Lambda \) is the diagonal matrix whose diagonal elements are the eigenvalues.

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. job – Nag_JobType

*Input*

On entry: indicates whether eigenvectors are computed.

job = Nag_EigVals
	Only eigenvalues are computed.

job = Nag_DoBoth
	Eigenvalues and eigenvectors are computed.

*Constraint:* job = Nag_EigVals or Nag_DoBoth.

3. uplo – Nag_UploType

*Input*

On entry: if uplo = Nag_Upper, the upper triangles of A and B are stored. If uplo = Nag_Lower, the lower triangles of A and B are stored.

*Constraint:* uplo = Nag_Upper or Nag_Lower.

4. n – Integer

*Input*

On entry: n, the order of the matrices A and B.

*Constraint:* n ≥ 0.

5. ka – Integer

*Input*

On entry: if uplo = Nag_Upper, the number of superdiagonals, k_a, of the matrix A. If uplo = Nag_Lower, the number of subdiagonals, k_a, of the matrix A.

*Constraint:* ka ≥ 0.

6. kb – Integer

*Input*

On entry: if uplo = Nag_Upper, the number of superdiagonals, k_b, of the matrix B. If uplo = Nag_Lower, the number of subdiagonals, k_b, of the matrix B.

*Constraint:* ka ≥ kb ≥ 0.

7. ab[dom] – double

*Input/Output*

Note: the dimension, dim, of the array ab must be at least max(1, pdab × n).

On entry: the upper or lower triangle of the n by n symmetric band matrix A.

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 order = Nag_ColMajor and uplo = Nag_Upper,

A_{ij} is stored in \( ab[k_a + i - j + (j - 1) \times pdab] \), for \( j = 1, \ldots, n \) and \( i = \max(1, j - k_a), \ldots, j \);

if order = Nag_ColMajor and uplo = Nag_Lower,

A_{ij} is stored in \( ab[i - j + (j - 1) \times pdab] \), for \( j = 1, \ldots, n \) and \( i = j, \ldots, \min(n, j + k_a) \);
if order = Nag_RowMajor and uplo = Nag_Upper,
    $A_{ij}$ is stored in $ab[j - i + (i - 1) \times pdab]$, for $i = 1, \ldots, n$ and $j = i, \ldots, \min(n, i + k_a)$;

if order = Nag_RowMajor and uplo = Nag_Lower,
    $A_{ij}$ is stored in $ab[k_a + j - i + (i - 1) \times pdab]$, for $i = 1, \ldots, n$ and $j = \max(1, i - k_a), \ldots, i$.

On exit: the contents of $ab$ are overwritten.

8: **pdab** – Integer

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

Constraint: $pdab \geq ka + 1$.

9: **bb[dim]** – double

Input/Output

Note: the dimension, dim, of the array $bb$ must be at least $\max(1, pdbb \times n)$.

On entry: the upper or lower triangle of the $n$ by $n$ symmetric band matrix $B$.

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

if order = Nag_ColMajor and uplo = Nag_Upper,
    $B_{ij}$ is stored in $bb[k_b + i - j + (j - 1) \times pdbb]$, for $j = 1, \ldots, n$ and $i = \max(1, j - k_b), \ldots, j$;

if order = Nag_ColMajor and uplo = Nag_Lower,
    $B_{ij}$ is stored in $bb[i - j + (j - 1) \times pdbb]$, for $j = 1, \ldots, n$ and $i = j, \ldots, \min(n, i + k_b)$;

if order = Nag_RowMajor and uplo = Nag_Upper,
    $B_{ij}$ is stored in $bb[j - i + (i - 1) \times pdbb]$, for $i = 1, \ldots, n$ and $j = i, \ldots, \min(n, i + k_b)$;

if order = Nag_RowMajor and uplo = Nag_Lower,
    $B_{ij}$ is stored in $bb[k_b + j - i + (i - 1) \times pdbb]$, for $i = 1, \ldots, n$ and $j = \max(1, i - k_b), \ldots, i$.

On exit: the factor $S$ from the split Cholesky factorization $B = S^T S$, as returned by nag_dpbstf (f08ufc).

10: **pdbb** – Integer

Input

On entry: the stride separating row or column elements (depending on the value of order) of the matrix $B$ in the array $bb$.

Constraint: $pdbb \geq kb + 1$.

11: **w[n]** – double

Output

On exit: the eigenvalues in ascending order.

12: **z[dim]** – double

Output

Note: the dimension, dim, of the array $z$ must be at least

$\max(1, pdx \times n)$ when job = Nag_DoBoth;

1 otherwise.

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

$z[(j - 1) \times pdx + i - 1]$ when order = Nag_ColMajor;

$z[(i - 1) \times pdx + j - 1]$ when order = Nag_RowMajor.
On exit: if \( \text{job} = \text{Nag}_\text{DoBoth} \), \( z \) contains the matrix \( Z \) of eigenvectors, with the \( i \)th column of \( Z \) holding the eigenvector associated with \( w[i - 1] \). The eigenvectors are normalized so that \( Z^T B Z = I \).

If \( \text{job} = \text{Nag}_\text{EigVals} \), \( z \) is not referenced.

13: \[ \text{pdz} \quad \text{Integer} \]

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

\[ \text{Constraints:} \]
\[ \quad \text{if } \text{job} = \text{Nag}_\text{DoBoth}, \text{pdz} \geq \max(1, n); \]
\[ \quad \text{otherwise pdz} \geq 1. \]

14: \[ \text{fail} \quad \text{NagError*} \]

\( \text{Input/Output} \)

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

6 Error Indicators and Warnings

\textbf{NEALLOC_FAIL}

Dynamic memory allocation failed.

See Section 3.2.1.2 in the Essential Introduction for further information.

\textbf{NE_BAD_PARAM}

On entry, argument \( \langle \text{value} \rangle \) had an illegal value.

\textbf{NE_CONVERGENCE}

The algorithm failed to converge; \( \langle \text{value} \rangle \) off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

\textbf{NE_ENUM_INT_2}

On entry, \( \text{job} = \langle \text{value} \rangle \), \( \text{pdz} = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).

Constraint: if \( \text{job} = \text{Nag}_\text{DoBoth}, \text{pdz} \geq \max(1, n); \)
\[ \quad \text{otherwise pdz} \geq 1. \]

\textbf{NE_INT}

On entry, \( \text{ka} = \langle \text{value} \rangle \).

Constraint: \( \text{ka} \geq 0. \)

On entry, \( n = \langle \text{value} \rangle \).

Constraint: \( n \geq 0. \)

On entry, \( \text{pdab} = \langle \text{value} \rangle \).

Constraint: \( \text{pdab} > 0. \)

On entry, \( \text{pdbb} = \langle \text{value} \rangle \).

Constraint: \( \text{pdbb} > 0. \)

On entry, \( \text{pdz} = \langle \text{value} \rangle \).

Constraint: \( \text{pdz} > 0. \)

\textbf{NE_INT_2}

On entry, \( \text{ka} = \langle \text{value} \rangle \) and \( \text{kb} = \langle \text{value} \rangle \).

Constraint: \( \text{ka} \geq \text{kb} \geq 0. \)

On entry, \( \text{pdab} = \langle \text{value} \rangle \) and \( \text{ka} = \langle \text{value} \rangle \).

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

**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.
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**
If \( \text{fail.errnum} = n + \langle \text{value} \rangle \), for \( 1 \leq \langle \text{value} \rangle \leq n \), then \( 
\text{nag_dpbstf (f08ufc)} \) returned
\( \text{fail.errnum} = \langle \text{value} \rangle \): \( B \) is not positive definite. The factorization of \( B \) could not be completed
and no eigenvalues or eigenvectors were 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
If \( B \) is ill-conditioned with respect to inversion, then the error bounds for the computed eigenvalues and
vectors may be large, although when the diagonal elements of \( B \) differ widely in magnitude the
eigenvalues and eigenvectors may be less sensitive than the condition of \( B \) would suggest. See Section
4.10 of Anderson et al. (1999) for details of the error bounds.

8 Parallelism and Performance
\( \text{nag_dsbgv (f08uac)} \) is threaded by NAG for parallel execution in multithreaded implementations of the
NAG Library.
\( \text{nag_dsbgv (f08uac)} \) 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 proportional to \( n^3 \) if \( \text{job} = \text{Nag_DoBoth} \) and, assuming
that \( n \gg k_n \), is approximately proportional to \( n^2 k_n \) otherwise.
The complex analogue of this function is \( \text{nag_zhbgv (f08unc)} \).

10 Example
This example finds all the eigenvalues of the generalized band symmetric eigenproblem \( Az = \lambda Bz \),
where
\[
A = \begin{pmatrix}
0.24 & 0.39 & 0.42 & 0 \\
0.39 & -0.11 & 0.79 & 0.63 \\
0.42 & -0.25 & 0.48 & -0.03 \\
0 & 0.63 & 0.48 & -0.03
\end{pmatrix}
\quad \text{and} \quad
B = \begin{pmatrix}
2.07 & 0.95 & 0 & 0 \\
0.95 & 1.69 & -0.29 & 0 \\
0 & -0.29 & 0.65 & -0.33 \\
0 & 0 & -0.33 & 1.17
\end{pmatrix}.
\]
10.1 Program Text

/* nag_dsbgv (f08uac) Example Program. */
* Copyright 2014 Numerical Algorithms Group.
* Mark 23, 2011. */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf08.h>
#include <nagx04.h>

int main(void)
{
/* Scalars */
 Integer i, j, ka, kb, n, pdab, pdbb, pdz, zsize;
 Integer exit_status = 0;

/* Arrays */
 double *ab = 0, *bb = 0, *w = 0, *z = 0;
 char nag_enum_arg[40];

/* Nag Types */
 NagError fail;
 Nag_UploType uplo;
 Nag_OrderType order;
 Nag_JobType job;

#ifdef NAG_COLUMN_MAJOR
define AB_UPPER(I, J) ab[(J-1)*pdab + ka + I-J]
define AB_LOWER(I, J) ab[(J-1)*pdab + I-J]
define BB_UPPER(I, J) bb[(J-1)*pdbb + kb + I-J]
define BB_LOWER(I, J) bb[(J-1)*pdbb + I-J]
#else
define AB_UPPER(I, J) ab[(I-1)*pdab + J-I]
define AB_LOWER(I, J) ab[(I-1)*pdab + ka + J-I]
define BB_UPPER(I, J) bb[(I-1)*pdbb + kb + J-I]
define BB_LOWER(I, J) bb[(I-1)*pdbb + J-I]
#endif

order = Nag_ColMajor; 
#else 
order = Nag_RowMajor;
#endif

INIT_FAIL(fail);

printf("nag_dsbgv (f08uac) 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%NAG_IFMT%*[\n"]", &n, &ka, &kb);
#else
scanf("%NAG_IFMT%NAG_IFMT%NAG_IFMT%*[\n"]", &n, &ka, &kb);
#endif

if (n < 0 || ka < kb || kb < 0)
{
 printf("Invalid n, ka or kb\n");
 exit_status = 1;
 goto END;
}
#ifdef _WIN32
scanf_s(" %39s*\n", nag_enum_arg, _countof(nag_enum_arg));
#else
scanf(" %39s*\n", nag_enum_arg);
}
uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
scanf_s("%39s*[
]*", nag_enum_arg, _countof(nag_enum_arg));
#else
scanf("%39s*[
]*", nag_enum_arg);
#endif
job = (Nag_JobType) nag_enum_name_to_value(nag_enum_arg);
if (job==Nag_EigVals) {
    zsize = 1;
    pdz = 1;
} else {
    zsize = n*n;
    pdz = n;
}
/* Allocate memory */
if (!(ab = NAG_ALLOC((ka+1) * n, double)) ||
! (bb = NAG_ALLOC((kb+1) * n, double)) ||
! (z = NAG_ALLOC(zsize, double)) ||
! (w = NAG_ALLOC(n, double))) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
} /* Read the triangular parts of the matrices A and B from data file */
if (uplo == Nag_Upper) {
    for (i = 1; i <= n; ++i)
    #ifdef _WIN32
        for (j = i; j <= MIN(i+ka, n); ++j) scanf_s("%lf", &AB_UPPER(i, j));
    #else
        for (j = i; j <= MIN(i+ka, n); ++j) scanf("%lf", &AB_UPPER(i, j));
    #endif
    #ifdef _WIN32
        scanf_s("%*[\n]");
    #else
        scanf("%*[\n]");
    #endif
    for (i = 1; i <= n; ++i)
    #ifdef _WIN32
        for (j = MAX(1, i-ka); j <= i; ++j) scanf_s("%lf", &AB_LOWER(i, j));
    #else
        for (j = MAX(1, i-ka); j <= i; ++j) scanf("%lf", &AB_LOWER(i, j));
    #endif
    #ifdef _WIN32
        scanf_s("%*[\n]");
    #else
        scanf("%*[\n]");
    #endif
} else {
    for (i = 1; i <= n; ++i)
    #ifdef _WIN32
        for (j = MAX(1, i-ka); j <= i; ++j) scanf_s("%lf", &AB_LOWER(i, j));
    #else
        for (j = MAX(1, i-ka); j <= i; ++j) scanf("%lf", &AB_LOWER(i, j));
    #endif
    #ifdef _WIN32
        scanf_s("%*[\n]");
    #else
        scanf("%*[\n]");
    #endif
}


#ifndef
#endif

#ifdef _WIN32
    scanf_s("%*[\n"]);
#else
    scanf("%*[\n"]);
#endif

/* Solve the generalized symmetric band eigenvalue problem A*x = lambda*B*x * using nag_dsbgv (f08uac).
*
*/

nag_dsbgv(order, job, uplo, n, ka, kb, ab, pdab, bb, pdbb, w, z, pdz, &fail);

if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dsbgv (f08uac).\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print eigensolution */
printf(" Eigenvalues\n ");
for (j = 0; j < n; ++j) printf(" %9.4f", w[j], j%6 == 5?"\n":" ");
printf("\n");

if (job==Nag_DoBoth) {
    /* nag_gen_real_mat_print (x04cac): Print Matrix of eigenvectors Z. */
    printf("\n");
    fflush(stdout);
    nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, 
z, pdz, "Eigenvectors (by Column)", 0, &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_gen_real_mat_print (x04cac).\n", fail.message);
        exit_status = 1;
    }
}

END:
NAG_FREE(ab);
NAG_FREE(bb);
NAG_FREE(z);
NAG_FREE(w);
return exit_status;

10.2 Program Data

nag_dsbgv (f08uac) Example Program Data

4 2 1 : n, ka and kb
Nag_Upper : uplo
Nag_EigVals : job (=Nag_DoBoth for eigenvectors)
0.24 0.39 0.42  
-0.11 0.79 0.63
-0.25 0.48
-0.03 : matrix A
2.07 0.95 1.69 -0.29
0.65 -0.33
1.17 : matrix B

f08uac.8 Mark 25
### 10.3 Program Results

**nag_dsbgv (f08uac) Example Program Results**

<table>
<thead>
<tr>
<th>Eigenvalues</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.8305</td>
<td>-0.6401</td>
<td>0.0992</td>
<td>1.8525</td>
</tr>
</tbody>
</table>