NAG Library Function Document
nag_dsbgvd (f08ucc)

1 Purpose
nag_dsbgvd (f08ucc) 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. If eigenvectors are desired, it uses a divide-and-conquer algorithm.

2 Specification

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

void nag_dsbgvd (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

\[ Z^T AZ = A \quad \text{and} \quad Z^T BZ = I, \]

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

4 References


Mark 25
5 Arguments

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

\textit{On entry:} the \textit{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 \textit{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:} order = Nag_RowMajor or Nag_ColMajor.

2: \textbf{job} – Nag_JobType \hspace{1cm} \textit{Input}

\textit{On entry:} indicates whether eigenvectors are computed.

\textit{job} = Nag_EigVals

Only eigenvalues are computed.

\textit{job} = Nag_DoBoth

Eigenvalues and eigenvectors are computed.

\textit{Constraint:} \textit{job} = Nag_EigVals or Nag_DoBoth.

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

\textit{On entry:} if \textbf{uplo} = Nag_Upper, the upper triangles of \(A\) and \(B\) are stored.

If \textbf{uplo} = Nag_Lower, the lower triangles of \(A\) and \(B\) are stored.

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

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

\textit{On entry:} \(n\), the order of the matrices \(A\) and \(B\).

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

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

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

If \textbf{uplo} = Nag_Lower, the number of subdiagonals, \(k_a\), of the matrix \(A\).

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

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

\textit{On entry:} if \textbf{uplo} = Nag_Upper, the number of superdiagonals, \(k_b\), of the matrix \(B\).

If \textbf{uplo} = Nag_Lower, the number of subdiagonals, \(k_b\), of the matrix \(B\).

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

7: \textbf{ab}[\textit{dim}] – double \hspace{1cm} \textit{Input/Output}

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

\textit{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 \textbf{order} and \textbf{uplo} arguments as follows:

if \textbf{order} = Nag_ColMajor and \textbf{uplo} = Nag_Upper,

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

if \textbf{order} = Nag_ColMajor and \textbf{uplo} = Nag_Lower,

\(A_{ij}\) is stored in \(ab[i - j + (j - 1) \times \text{pdab}]\), for \(j = 1, \ldots, n\) and \(i = j, \ldots, \min(n, j + k_a)\);
if \( \text{order} = \text{Nag}_\text{RowMajor} \) and \( \text{uplo} = \text{Nag}_\text{Upper} \),
\[
\begin{align*}
A_{ij} \text{ is stored in } & ab[j - i + (i - 1) \times \text{pdab}], \text{ for } i = 1, \ldots, n \text{ and } \nonumber \\
& j = i, \ldots, \min(n, i + \text{ka}); 
\end{align*}
\]
if \( \text{order} = \text{Nag}_\text{RowMajor} \) and \( \text{uplo} = \text{Nag}_\text{Lower} \),
\[
\begin{align*}
A_{ij} \text{ is stored in } & ab[k_a + j - i + (i - 1) \times \text{pdab}], \text{ for } i = 1, \ldots, n \text{ and } \nonumber \\
& j = \max(1, i - \text{ka}), \ldots, i. 
\end{align*}
\]

On exit: the contents of \( ab \) are overwritten.

8: \( \text{pdab} \) – Integer

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

\( Constraint: \text{pdab} \geq \text{ka} + 1. \)

9: \( \text{bb}[\text{dim}] \) – double

\( Input/Output \)

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

\( On entry: \) the upper or lower triangle of the \( n \times n \) 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 \( \text{order} \) and \( \text{uplo} \) arguments as follows:

if \( \text{order} = \text{Nag}_\text{ColMajor} \) and \( \text{uplo} = \text{Nag}_\text{Upper} \),
\[
\begin{align*}
B_{ij} \text{ is stored in } & \text{bb}[k_b + i - j + (j - 1) \times \text{pdbb}], \text{ for } j = 1, \ldots, n \text{ and } \\
i = \max(1, j - k_b), \ldots, j; 
\end{align*}
\]
if \( \text{order} = \text{Nag}_\text{ColMajor} \) and \( \text{uplo} = \text{Nag}_\text{Lower} \),
\[
\begin{align*}
B_{ij} \text{ is stored in } & \text{bb}[i - j + (j - 1) \times \text{pdbb}], \text{ for } j = 1, \ldots, n \text{ and } \\
i = j, \ldots, \min(n, i + k_b); 
\end{align*}
\]
if \( \text{order} = \text{Nag}_\text{RowMajor} \) and \( \text{uplo} = \text{Nag}_\text{Upper} \),
\[
\begin{align*}
B_{ij} \text{ is stored in } & \text{bb}[j - i + (i - 1) \times \text{pdbb}], \text{ for } i = 1, \ldots, n \text{ and } \\
j = i, \ldots, \min(n, i + k_b); 
\end{align*}
\]
if \( \text{order} = \text{Nag}_\text{RowMajor} \) and \( \text{uplo} = \text{Nag}_\text{Lower} \),
\[
\begin{align*}
B_{ij} \text{ is stored in } & \text{bb}[k_b + j - i + (i - 1) \times \text{pdbb}], \text{ for } i = 1, \ldots, n \text{ and } \\
j = \max(1, i - \text{ka}), \ldots, i. 
\end{align*}
\]

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

10: \( \text{pdbb} \) – Integer

\( Input \)

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

\( Constraint: \text{pdbb} \geq \text{kb} + 1. \)

11: \( \text{w}[\text{n}] \) – double

\( Output \)

\( On exit: \) the eigenvalues in ascending order.

12: \( \text{z}[\text{dim}] \) – double

\( Output \)

\( Note: \) the dimension, \( \text{dim} \), of the array \( \text{z} \) must be at least \( \max(1, \text{pdz} \times n) \) when \( \text{job} = \text{Nag}_\text{DoBoth} \);

1 otherwise.

The \( (i, j) \)th element of the matrix \( Z \) is stored in
\[
\begin{align*}
\text{z}[(j - 1) \times \text{pdz} + i - 1] & \text{ when } \text{order} = \text{Nag}_\text{ColMajor}; \\
\text{z}[(i - 1) \times \text{pdz} + j - 1] & \text{ when } \text{order} = \text{Nag}_\text{RowMajor}. 
\end{align*}
\]
On exit: if job = Nag_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 job = Nag_EigVals, \( z \) is not referenced.

13: \( pdz \) – Integer

\( pdz \) is the stride separating row or column elements (depending on the value of \( order \)) in the array \( z \).

Constraints:
- if job = Nag_DoBoth, \( pdz \geq \max(1, n) \);
- otherwise \( pdz \geq 1 \).

14: \( fail \) – NagError

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 \( \langle value \rangle \) had an illegal value.

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

**NE_ENUM_INT_2**
On entry, \( \langle value \rangle \), \( pdz = \langle value \rangle \) and \( n = \langle value \rangle \).
Constraint: if job = Nag_DoBoth, \( pdz \geq \max(1, n) \);
otherwise \( pdz \geq 1 \).

**NE_INT**
On entry, \( ka = \langle value \rangle \).
Constraint: \( ka \geq 0 \).

On entry, \( n = \langle value \rangle \).
Constraint: \( n \geq 0 \).

On entry, \( pdb = \langle value \rangle \).
Constraint: \( pdb \geq 0 \).

On entry, \( pdbb = \langle value \rangle \).
Constraint: \( pdbb > 0 \).

On entry, \( pdz = \langle value \rangle \).
Constraint: \( pdz > 0 \).

**NE_INT_2**
On entry, \( ka = \langle value \rangle \) and \( kb = \langle value \rangle \).
Constraint: \( ka \geq kb \geq 0 \).

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

**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 $\text{fail.errnum} = \text{n} + \langle \text{value} \rangle$, for $1 \leq \langle \text{value} \rangle \leq \text{n}$, then 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
nag_dsbgv (f08ucc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_dsbgv (f08ucc) 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 nag_zhsbgvd (f08uqc).

10 Example
This example finds all the eigenvalues of the generalized band symmetric eigenproblem $Az = \lambda Bz$, where

$$A = \begin{pmatrix} 0.24 & 0.39 & 0.42 & 0 \\ 0.39 & -0.11 & 0.79 & 0.63 \\ 0.42 & -0.25 & 0.48 & 0 \\ 0 & 0.63 & 0.48 & -0.03 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 2.07 & 0.95 & 0 & 0 \\ 0.95 & 1.69 & -0.29 & 0 \\ 0 & -0.29 & 0.65 & -0.33 \\ 0 & 0 & -0.33 & 1.17 \end{pmatrix}.$$
10.1 Program Text

/* nag_dsbgvd (f08ucc) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 23, 2011. */

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

int main(void)
{
    /* Scalars */
    Integer exit_status = 0, i, j, ka, kb, n, pdab, pdbb, pdz, zsize;
    /* Arrays */
    double *ab = 0, *bb = 0, *w = 0, *z = 0;
    char nag_enum_arg[40];
    /* Nag Types */
    NagError fail;
    Nag_OrderType order;
    Nag_UploType uplo;
    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]
    order = Nag_ColMajor;
    #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 + J - I]
    #define BB_LOWER(I, J) bb[(I-1)*pdbb + kb + J - I]
    order = Nag_RowMajor;
    #endif

    INIT_FAIL(fail);
    printf("nag_dsbgvd (f08ucc) Example Program Results\n\n");

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

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

    INIT_FAIL(fail);
    printf("%s\n", nag_enum_arg);

    /* Input data */
    #ifdef _WIN32
    scanf_s(" %39s%*
\n");
    #else
    scanf(" %39s%*
\n");
    #endif

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

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

dpab = ka + 1;
dpdb = kb + 1;
/* 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");
extit_status = -1;
goto END;
}
#endif _WIN32
/* Read the triangular parts of the matrices A and B from data file */
if (uplo == Nag_Upper) {
    #ifdef _WIN32
        for (i = 1; i <= n; ++i)
            for (j = i; j <= MIN(i+ka, n); ++j) scanf_s("%lf", &AB_UPPER(i, j));
    #else
        for (i = 1; i <= n; ++i)
            for (j = i; j <= MIN(i+ka, n); ++j) scanf("%lf", &AB_UPPER(i, j));
    #endif _WIN32
    scanf_s("%*[\n]");
#else
    for (i = 1; i <= n; ++i)
        for (j = i; j <= MIN(i+kb, n); ++j) scanf("%lf", &BB_UPPER(i, j));
    #endif _WIN32
    scanf("%*[\n]");
#else
    for (i = 1; i <= n; ++i)
        for (j = MAX(1, i-ka); j <= i; ++j) scanf_s("%lf", &AB_LOWER(i, j));
    #else
        for (i = 1; i <= n; ++i)
            for (j = MAX(1, i-kb); j <= i; ++j) scanf("%lf", &BB_LOWER(i, j));
    #endif _WIN32
    scanf_s("%*[\n]");
#else
    for (i = 1; i <= n; ++i)
        for (j = MAX(1, i-kb); j <= i; ++j) scanf("%lf", &BB_LOWER(i, j));
    #else
        for (i = 1; i <= n; ++i)
            for (j = MAX(1, i-ka); j <= i; ++j) scanf("%lf", &AB_LOWER(i, j));
    #endif _WIN32
    scanf("%*[\n]");
#endif _WIN32

Mark 25

f08ucc.7
/* Solve the generalized symmetric band eigenvalue problem \( A \mathbf{x} = \lambda B \mathbf{x} \) using nag_dsbgvd (f08ucc).
*/

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

if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dsbgvd (f08ucc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print eigensolution */
printf(" Eigenvalues\n   ");
for (j = 0; j < n; ++j) printf(" %9.4f%s", 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%s\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_dsbgvd (f08ucc) 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
### 10.3 Program Results

**nag_dsbgvd (f08ucc) Example Program Results**

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