**NAG C Library Function Document**

**nag_dsbgst (f08uec)**

1 **Purpose**

`nag_dsbgst (f08uec)` reduces a real symmetric-definite generalized eigenproblem $A z = \lambda B z$ to the standard form $C y = \lambda y$, where $A$ and $B$ are band matrices, $A$ is a real symmetric matrix, and $B$ has been factorized by `nag_dpbstf (f08ufc)`.

2 **Specification**

```c
void nag_dsbgst (Nag_OrderType order, Nag_VectType vect, Nag_UploType uplo,
                Integer n, Integer ka, Integer kb, double ab[], Integer pdab,
                const double bb[], Integer pddb, double x[], Integer pdx, NagError *fail)
```

3 **Description**

To reduce the real symmetric-definite generalized eigenproblem $A z = \lambda B z$ to the standard form $C y = \lambda y$, where $A$, $B$ and $C$ are banded, this function must be preceded by a call to `nag_dpbstf (f08ufc)` which computes the split Cholesky factorization of the positive-definite matrix $B$: $B = S^T S$. The split Cholesky factorization, compared with the ordinary Cholesky factorization, allows the work to be approximately halved.

This function overwrites $A$ with $C = X^T A X$, where $X = S^{-1} Q$ and $Q$ is a orthogonal matrix chosen (implicitly) to preserve the bandwidth of $A$. The function also has an option to allow the accumulation of $X$, and then, if $z$ is an eigenvector of $C$, $X z$ is an eigenvector of the original system.

4 **References**


5 **Parameters**

1: **order** – Nag_OrderType

   *Input*

   On entry: the `order` parameter 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 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this parameter.

   **Constraint:** `order = Nag_RowMajor` or `Nag_ColMajor`.

2: **vect** – Nag_VectType

   *Input*

   On entry: indicates whether $X$ is to be returned as follows:

   - if `vect = Nag_DoNotForm`, $X$ is not returned;
   - if `vect = Nag_FormX`, $X$ is returned.

   **Constraint:** `vect = Nag_DoNotForm` or `Nag_FormX`.

3: **uplo** – Nag_UploType

   *Input*

   On entry: indicates whether the upper or lower triangular part of $A$ is stored as follows:
if uplo = Nag_Upper, the upper triangular part of A is stored;
if uplo = Nag_Lower, the lower triangular part of A is 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: kA, the number of super-diagonals of the matrix A if uplo = Nag_Upper, or the number
   of sub-diagonals if uplo = Nag_Lower.
   Constraint: ka ≥ 0.

6: kb – Integer  
   Input
   On entry: kB, the number of super-diagonals of the matrix B if uplo = Nag_Upper, or the number
   of sub-diagonals if uplo = Nag_Lower.
   Constraint: ka ≥ kb ≥ 0.

7: ab[dim] – double  
   Input/Output
   Note: the dimension, dim, of the array ab must be at least max(1, pdab × n).
   On entry: 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 aij depends on
   the order and uplo parameters as follows:
   if order = Nag_ColMajor and uplo = Nag_Upper,
       aij is stored in ab[kA + i - j + (j - 1) × pdab], for i = 1,...,n and
       j = i,...,min(n,i+kA);
   if order = Nag_ColMajor and uplo = Nag_Lower,
       aij is stored in ab[i - j + (i - 1) × pdab], for i = 1,...,n and
       j = max(1,i-kA),...,i;
   if order = Nag_RowMajor and uplo = Nag_Upper,
       aij is stored in ab[j - i + (i - 1) × pdab], for i = 1,...,n and
       j = i,...,min(n,i+kA);
   if order = Nag_RowMajor and uplo = Nag_Lower,
       aij is stored in ab[kA + j - i + (i - 1) × pdab], for i = 1,...,n and
       j = max(1,i-kA),...,i.
   On exit: the upper or lower triangle of A is overwritten by the corresponding upper or lower triangle
   of C as specified by uplo.

8: 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: pdab ≥ ka + 1.

9: bb[dim] – const double  
   Input
   Note: the dimension, dim, of the array bb must be at least max(1, pdbb × n).
   On entry: the banded split Cholesky factor of B as specified by uplo, n and kb and returned by
   nag_dpbstf (f08ufc).
10:  pdbb – Integer

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

   Constraint: pdbb ≥ kb + 1.

11:  x[dim] – double

   Note: the dimension, dim, of the array x must be at least
       max(1, pdx × n) when vect = Nag_FormX;
       1 when vect = Nag_DoNotForm.

   If order = Nag_ColMajor, the (i, j)th element of the matrix X is stored in x[(j − 1) × pdx + i − 1] and
   if order = Nag_RowMajor, the (i, j)th element of the matrix X is stored in x[(i − 1) × pdx + j − 1].

   On exit: the n by n matrix X = S⁻¹Q, if vect = Nag_FormX.

   x is not referenced if vect = Nag_DoNotForm.

12:  pdx – Integer

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

   Constraints:
   
   if vect = Nag_FormX, pdx ≥ max(1, n);
   if vect = Nag_DoNotForm, pdx ≥ 1.

13:  fail – NagError *

   The NAG error parameter (see the Essential Introduction).

6   Error Indicators and Warnings

NE_INT

   On entry, n = ⟨value⟩.

   Constraint: n ≥ 0.

   On entry, ka = ⟨value⟩.

   Constraint: ka ≥ 0.

   On entry, pdab = ⟨value⟩.

   Constraint: pdab > 0.

   On entry, pdbb = ⟨value⟩.

   Constraint: pdbb > 0.

   On entry, pdx = ⟨value⟩.

   Constraint: pdx > 0.

NE_INT_2

   On entry, ka = ⟨value⟩, kb = ⟨value⟩.

   Constraint: ka ≥ kb ≥ 0.

   On entry, pdab = ⟨value⟩, ka = ⟨value⟩.

   Constraint: pdab ≥ ka + 1.

   On entry, pdbb = ⟨value⟩, kb = ⟨value⟩.

   Constraint: pdbb ≥ kb + 1.
On entry, vect = \langle value\rangle, n = \langle value\rangle, pdx = \langle value\rangle.
Constraint: if vect = Nag_FormX, pdx ≥ max(1, n);
if vect = Nag_DoNotForm, pdx ≥ 1.

Memory allocation failed.

On entry, parameter \langle value\rangle had an illegal value.

An internal error has occurred in this function. Check the function call and any array sizes. If the
call is correct then please consult NAG for assistance.

Forming the reduced matrix C is a stable procedure. However it involves implicit multiplication by \(B^{-1}\).
When the function is used as a step in the computation of eigenvalues and eigenvectors of the original
problem, there may be a significant loss of accuracy if \(B\) is ill-conditioned with respect to inversion.

The total number of floating-point operations is approximately \(6n^2k_B\), when vect = Nag_DoNotForm,
assuming \(n \gg k_A, k_B\); there are an additional \((3/2)n^3(k_B/k_A)\) operations when vect = Nag_FormX.
The complex analogue of this function is nag_zhbgst (f08usc).

To compute all the eigenvalues of \(Az = \lambda Bz\), where

\[
A = \begin{pmatrix}
0.24 & 0.39 & 0.42 & 0.00 \\
0.39 & -0.11 & 0.79 & 0.63 \\
0.42 & 0.79 & -0.25 & 0.48 \\
0.00 & 0.63 & 0.48 & -0.03
\end{pmatrix}
\text{ and } B = \begin{pmatrix}
2.07 & 0.95 & 0.00 & 0.00 \\
0.95 & 1.69 & -0.29 & 0.00 \\
0.00 & -0.29 & 0.65 & -0.33 \\
0.00 & 0.00 & -0.33 & 1.17
\end{pmatrix}.
\]

Here \(A\) is symmetric, \(B\) is symmetric positive-definite, and \(A\) and \(B\) are treated as band matrices. \(B\) must
first be factorized by nag_dpbstf (f08ufc). The program calls nag_dsbgst (f08uec) to reduce the problem to
the standard form \(Cy = \lambda y\), then nag_dsbrtrd (f08hec) to reduce \(C\) to tridiagonal form, and nag_dsterf
(f08jfc) to compute the eigenvalues.

*/
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf08.h>

int main(void)
{
    /* Scalars */
Integer i, j, k1, k2, ka, kb, n, pdab, pdbb, pdx, d_len, e_len;
Integer exit_status=0;
NagError fail;
Nag_UploType uplo;
Nag_OrderType order;
/* Arrays */
char uplo_char[2];
double *a0, *d=0, *e=0, *x=0;
#endif
#define AB_UPPER(I,J) a[(J-1)*pdab + k1 +I-J-1]
#define AB_LOWER(I,J) a[(J-1)*pdab + I - J]
#define BB_UPPER(I,J) b[(J-1)*pdbb + k2 +I-J-1]
#define BB_LOWER(I,J) b[(J-1)*pdbb + I - J]
order = Nag_ColMajor;
#else
#define AB_UPPER(I,J) a[(I-1)*pdab + J - I]
#define AB_LOWER(I,J) a[(I-1)*pdab + k1 +J-I-1]
#define BB_UPPER(I,J) b[(I-1)*pdbb + J - I]
#define BB_LOWER(I,J) b[(I-1)*pdbb + k2 +J-I-1]
order = Nag_RowMajor;
#endif
INIT_FAIL(fail);
Vprintf("f08uec Example Program Results\n\n");
/* Skip heading in data file */
Vscanf("%*
"]");
Vscanf("%ld%ld%ld%*
"]", &n, &ka, &kb);
pdab = ka + 1;
pdbb = kb + 1;
pdx = n;
d_len = n;
e_len = n-1;
/* Allocate memory */
if ( !(a = NAG_ALLOC(pdab * n, double)) ||
    !(b = NAG_ALLOC(pdbb * n, double)) ||
    !(d = NAG_ALLOC(d_len, double)) ||
    !(e = NAG_ALLOC(e_len, double)) ||
    !(x = NAG_ALLOC(n * n, double)) )
{
    Vprintf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
/* Read whether Upper or Lower part of A is stored */
Vscanf(" %ls %*[\n"]", uplo_char);
if (*(unsigned char *)uplo_char == 'L')
    uplo = Nag_Lower;
else if (*(unsigned char *)uplo_char == 'U')
    uplo = Nag_Upper;
else
{
    Vprintf("Unrecognised character for Nag_UploType type\n");
    exit_status = -1;
    goto END;
}
/* Read A and B from data file */
k1 = ka + 1;
k2 = kb + 1;
if (uplo == Nag_Upper)
{
    for (i = 1; i <= n; ++i)
    {
        for (j = i; j <= MIN(i+ka,n); ++j)
            Vscanf("%lf", &AB_UPPER(i,j));
    }
    Vscanf("%*[\n"]");
}
else
for (i = 1; i <= n; ++i)
{
    for (j = MAX(1,i-ka); j <= i; ++j)
        Vscanf("%lf", &AB_LOWER(i,j));
    Vscanf("%*[\n\"]");
}
if (uplo == Nag_Upper)
{
    for (i = 1; i <= n; ++i)
    {
        for (j = i; j <= MIN(i+kb,n); ++j)
            Vscanf("%lf", &BB_UPPER(i,j));
        Vscanf("%*[\n\"]");
    }
    /* Compute the split Cholesky factorization of B */
    f08ufc(order, uplo, n, kb, bb, pdbb, &fail);
    if (fail.code != NE_NOERROR)
    {
        Vprintf("Error from f08ufc.\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }
    /* Reduce the problem to standard form C\*y = lambda\*y, */
    /* storing the result in A */
    f08uec(order, Nag_DoNotForm, uplo, n, ka, kb, ab, pdab, bb, pdbb,
           x, pdx, &fail);
    if (fail.code != NE_NOERROR)
    {
        Vprintf("Error from f08uec.\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }
    /* Reduce C to tridiagonal form T = (Q\*T)*C\*Q */
    f08hec(order, Nag_DoNotForm, uplo, n, ka, ab, pdab, d, e,
           x, pdx, &fail);
    if (fail.code != NE_NOERROR)
    {
        Vprintf("Error from f08hec.\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }
    /* Calculate the eigenvalues of T (same as C) */
    f08jfc(n, d, e, &fail);
    if (fail.code != NE_NOERROR)
    {
        Vprintf("Error from f08jfc.\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }
    /* Print eigenvalues */
    Vprintf(" Eigenvalues\n");
    for (i = 0; i < n; ++i)
        Vprintf(" %8.4lf",d[i]);
    Vprintf("\n");
END:
if (ab) NAG_FREE(ab);
if (bb) NAG_FREE(bb);
if (d) NAG_FREE(d);
if (e) NAG_FREE(e);
if (x) NAG_FREE(x);
return exit_status;
}

9.2 Program Data

f08uec Example Program Data

<table>
<thead>
<tr>
<th>Values of N, KA and KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 2 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value of UPLO</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘L’</td>
</tr>
</tbody>
</table>

| 0.24 |
| 0.39  |
| 0.42  |
| 0.63  |

| 0.79 |
| 0.48  |
| -0.25 |
| -0.03 |

<table>
<thead>
<tr>
<th>End of matrix A</th>
</tr>
</thead>
</table>

| 2.07 |
| 0.95  |
| 0.69  |
| -0.29 |
| 0.65  |
| -0.33 |

| 1.17 |
| 1.8525 |

9.3 Program Results

f08uec Example Program Results

Eigenvalues

| -0.8305 |
| -0.6401 |
| 0.0992  |
| 1.8525  |