

NAG C Library Function Document

nag_zhbgst (f08usc)

1 Purpose

nag_zhbgst (f08usc) reduces a complex Hermitian-definite generalized eigenproblem $Az = \lambda Bz$ to the standard form $Cy = \lambda y$, where A and B are band matrices, A is a complex Hermitian matrix, and B has been factorized by nag_zpbstf (f08utc).

2 Specification

```
void nag_zhbgst (Nag_OrderType order, Nag_VectType vect, Nag_UploType uplo,
                Integer n, Integer ka, Integer kb, Complex ab[], Integer pdab,
                const Complex bb[], Integer pdbb, Complex x[], Integer pdx, NagError *fail)
```

3 Description

To reduce the complex Hermitian-definite generalized eigenproblem $Az = \lambda Bz$ to the standard form $Cy = \lambda y$, where A , B and C are banded, this function must be preceded by a call to nag_zpbstf (f08utc) which computes the split Cholesky factorization of the positive-definite matrix B : $B = S^H S$. The split Cholesky factorization, compared with the ordinary Cholesky factorization, allows the work to be approximately halved.

This function overwrites A with $C = X^H A X$, where $X = S^{-1} Q$ and Q is a unitary 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 , Xz is an eigenvector of the original system.

4 References

Crawford C R (1973) Reduction of a band-symmetric generalized eigenvalue problem *Comm. ACM* **16** 41–44

Kaufman L (1984) Banded eigenvalue solvers on vector machines *ACM Trans. Math. Software* **10** 73–86

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: $n \geq 0$.

5: **ka** – Integer *Input*

On entry: k_A , the number of super-diagonals of the matrix A if **uplo** = **Nag_Upper**, or the number of sub-diagonals if **uplo** = **Nag_Lower**.

Constraint: $ka \geq 0$.

6: **kb** – Integer *Input*

On entry: k_B , the number of super-diagonals of the matrix B if **uplo** = **Nag_Upper**, or the number of sub-diagonals if **uplo** = **Nag_Lower**.

Constraint: $ka \geq kb \geq 0$.

7: **ab**[*dim*] – Complex *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 Hermitian band 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} depends on the **order** and **uplo** parameters as follows:

if **order** = **Nag_ColMajor** and **uplo** = **Nag_Upper**,

a_{ij} is stored in **ab**[$k_A + i - j + (j - 1) \times \mathbf{pdab}$], for $i = 1, \dots, n$ and $j = i, \dots, \min(n, i + k_A)$;

if **order** = **Nag_ColMajor** and **uplo** = **Nag_Lower**,

a_{ij} is stored in **ab**[$i - j + (j - 1) \times \mathbf{pdab}$], for $i = 1, \dots, n$ and $j = \max(1, i - k_A), \dots, i$;

if **order** = **Nag_RowMajor** and **uplo** = **Nag_Upper**,

a_{ij} is stored in **ab**[$j - i + (i - 1) \times \mathbf{pdab}$], for $i = 1, \dots, n$ and $j = i, \dots, \min(n, i + k_A)$;

if **order** = **Nag_RowMajor** and **uplo** = **Nag_Lower**,

a_{ij} is stored in **ab**[$k_A + j - i + (i - 1) \times \mathbf{pdab}$], for $i = 1, \dots, n$ and $j = \max(1, i - k_A), \dots, i$.

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: $\mathbf{pdab} \geq \mathbf{ka} + 1$.

9: **bb**[*dim*] – const Complex *Input*

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

On entry: the banded split Cholesky factor of B as specified by **uplo**, **n** and **kb** and returned by nag_zpbstf (f08utc).

- 10: **pdbb** – Integer *Input*
On entry: the stride separating row or column elements (depending on the value of **order**) of the matrix in the array **bb**.
Constraint: **pdbb** $\geq$ **kb** + 1.
- 11: **x[dim]** – Complex *Output*
Note: the dimension, *dim*, of the array **x** must be at least
 $\max(1, \mathbf{pdx} \times \mathbf{n})$ 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 $\mathbf{x}[(j-1) \times \mathbf{pdx} + i - 1]$ and
if **order** = **Nag_RowMajor**, the (i, j) th element of the matrix X is stored in $\mathbf{x}[(i-1) \times \mathbf{pdx} + j - 1]$.
On exit: the n by n matrix $X = S^{-1}Q$, if **vect** = **Nag_FormX**.
x is not referenced if **vect** = **Nag_DoNotForm**.
- 12: **pdx** – Integer *Input*
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** $\geq$ $\max(1, \mathbf{n})$;
if **vect** = **Nag_DoNotForm**, **pdx** $\geq$ 1.
- 13: **fail** – NagError * *Output*
The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

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

On entry, **ka** = $\langle value \rangle$.
Constraint: **ka** $\geq$ 0.

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

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

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

NE_INT_2

On entry, **ka** = $\langle value \rangle$, **kb** = $\langle value \rangle$.
Constraint: **ka** $\geq$ **kb** $\geq$ 0.

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

On entry, **pdbb** = $\langle value \rangle$, **kb** = $\langle value \rangle$.
Constraint: **pdbb** $\geq$ **kb** + 1.

NE_ENUM_INT_2

On entry, **vect** = $\langle value \rangle$, **n** = $\langle value \rangle$, **pdx** = $\langle value \rangle$.
 Constraint: if **vect** = **Nag_FormX**, **pdx** $\geq \max(1, \mathbf{n})$;
 if **vect** = **Nag_DoNotForm**, **pdx** ≥ 1 .

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

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

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 consult NAG for assistance.

7 Accuracy

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.

8 Further Comments

The total number of real floating-point operations is approximately $20n^2k_B$, when **vect** = **Nag_DoNotForm**, assuming $n \gg k_A, k_B$; there are an additional $5n^3(k_B/k_A)$ operations when **vect** = **Nag_FormX**.

The real analogue of this function is nag_dsbgst (f08uec).

9 Example

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

$$A = \begin{pmatrix} -1.13 + 0.00i & 1.94 - 2.10i & -1.40 + 0.25i & 0.00 + 0.00i \\ 1.94 + 2.10i & -1.91 + 0.00i & -0.82 - 0.89i & -0.67 + 0.34i \\ -1.40 - 0.25i & -0.82 + 0.89i & -1.87 + 0.00i & -1.10 - 0.16i \\ 0.00 + 0.00i & -0.67 - 0.34i & -1.10 + 0.16i & 0.50 + 0.00i \end{pmatrix}$$

and

$$B = \begin{pmatrix} 9.89 + 0.00i & 1.08 - 1.73i & 0.00 + 0.00i & 0.00 + 0.00i \\ 1.08 + 1.73i & 1.69 + 0.00i & -0.04 + 0.29i & 0.00 + 0.00i \\ 0.00 + 0.00i & -0.04 - 0.29i & 2.65 + 0.00i & -0.33 + 2.24i \\ 0.00 + 0.00i & 0.00 + 0.00i & -0.33 - 2.24i & 2.17 + 0.00i \end{pmatrix}.$$

Here A is Hermitian, B is Hermitian positive-definite, and A and B are treated as band matrices. B must first be factorized by nag_zpbstf (f08utc). The program calls nag_zhbgst (f08usc) to reduce the problem to the standard form $Cy = \lambda y$, then nag_zhbtrd (f08hsc) to reduce C to tridiagonal form, and nag_dsterf (f08jfc) to compute the eigenvalues.

9.1 Program Text

```
/* nag_zhbgst (f08usc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */
```

```

#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, pddb, pdx, d_len, e_len;
    Integer exit_status=0;
    NagError fail;
    Nag_UploType uplo;
    Nag_OrderType order;
    /* Arrays */
    char uplo_char[2];
    Complex *ab=0, *bb=0, *x=0;
    double *d=0, *e=0;

#ifdef NAG_COLUMN_MAJOR
#define AB_UPPER(I,J) ab[(J-1)*pdab + k1 + I - J - 1]
#define AB_LOWER(I,J) ab[(J-1)*pdab + I - J]
#define BB_UPPER(I,J) bb[(J-1)*pddb + k2 + I - J - 1]
#define BB_LOWER(I,J) bb[(J-1)*pddb + 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 + k1 + J - I - 1]
#define BB_UPPER(I,J) bb[(I-1)*pddb + J - I]
#define BB_LOWER(I,J) bb[(I-1)*pddb + k2 + J - I - 1]
    order = Nag_RowMajor;
#endif

    INIT_FAIL(fail);
    Vprintf("f08usc Example Program Results\n\n");

    /* Skip heading in data file */
    Vscanf("%*[^\\n] ");
    Vscanf("%ld%ld%ld%*[^\\n] ", &n, &ka, &kb);
    pdab = ka + 1;
    pddb = kb + 1;
    pdx = n;
    d_len = n;
    e_len = n-1;

    /* Allocate memory */
    if ( !(ab = NAG_ALLOC(pdab * n, Complex)) ||
        !(bb = NAG_ALLOC(pddb * n, Complex)) ||
        !(d = NAG_ALLOC(d_len, double)) ||
        !(e = NAG_ALLOC(e_len, double)) ||
        !(x = NAG_ALLOC(n * n, Complex)) )
    {
        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 , %lf ) ", &AB_UPPER(i,j).re,
                    &AB_UPPER(i,j).im);
            }
        }
        Vscanf("%*[\n] ");
    }
else
    {
        for (i = 1; i <= n; ++i)
        {
            for (j = MAX(1,i-ka); j <= i; ++j)
            {
                Vscanf(" ( %lf , %lf ) ", &AB_LOWER(i,j).re,
                    &AB_LOWER(i,j).im);
            }
        }
        Vscanf("%*[\n] ");
    }
if (uplo == Nag_Upper)
    {
        for (i = 1; i <= n; ++i)
        {
            for (j = i; j <= MIN(i+kb,n); ++j)
            {
                Vscanf(" ( %lf , %lf ) ", &BB_UPPER(i,j).re,
                    &BB_UPPER(i,j).im);
            }
        }
        Vscanf("%*[\n] ");
    }
else
    {
        for (i = 1; i <= n; ++i)
        {
            for (j = MAX(1,i-kb); j <= i; ++j)
            {
                Vscanf(" ( %lf , %lf ) ", &BB_LOWER(i,j).re,
                    &BB_LOWER(i,j).im);
            }
        }
        Vscanf("%*[\n] ");
    }
/* Compute the split Cholesky factorization of B */
f08uttc(order, uplo, n, kb, bb, pddb, &fail);
if (fail.code != NE_NOERROR)
    {
        Vprintf("Error from f08uttc.\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 */
f08usc(order, Nag_DoNotForm, uplo, n, ka, kb, ab, pdab, bb, pddb,
    x, pdx, &fail);
if (fail.code != NE_NOERROR)
    {
        Vprintf("Error from f08usc.\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }
/* Reduce C to tridiagonal form T = (Q**T)*C*Q */
f08hsc(order, Nag_DoNotForm, uplo, n, ka, ab, pdab, d, e,
    x, pdx, &fail);
if (fail.code != NE_NOERROR)
    {
        Vprintf("Error from f08hsc.\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

f08usc Example Program Data

```

4 2 1                                :Values of N, KA and KB
'L'                                  :Value of UPLO
(-1.13, 0.00)
( 1.94, 2.10) (-1.91, 0.00)
(-1.40,-0.25) (-0.82, 0.89) (-1.87, 0.00)
                (-0.67,-0.34) (-1.10, 0.16) ( 0.50, 0.00)      :End of matrix A
( 9.89, 0.00)
( 1.08, 1.73) ( 1.69, 0.00)
                (-0.04,-0.29) ( 2.65, 0.00)
                        (-0.33,-2.24) ( 2.17, 0.00)      :End of matrix B

```

9.3 Program Results

f08usc Example Program Results

```

Eigenvalues
-6.6089  -2.0416   0.1603   1.7712

```
