NAG 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

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

```
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, nag_zhbgst (f08usc) must be preceded by a call to nag_zpbstf (f08utc) which computes the split Cholesky factorization of the positive definite matrix B: $B = S^{\mathrm{H}}S$. The split Cholesky factorization, compared with the ordinary Cholesky factorization, allows the work to be approximately halved.

This function overwrites A with $C = X^{\mathrm{H}}AX$, 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 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: **vect** – Nag_VectType *Input*

On entry: indicates whether X is to be returned.

vect = Nag_DoNotForm
X is not returned.

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.

uplo $=$ Nag_Upper
The upper triangular part of A is stored.

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: $\mathbf{n} \geq 0$.

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: $\mathbf{ka} \geq 0$.

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: $\mathbf{ka} \geq \mathbf{kb} \geq 0$.

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 upper or lower triangle of 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 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 $\mathbf{ab}[k_a + i - j + (j - 1) \times \mathbf{pdab}]$, 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 $\mathbf{ab}[i - j + (j - 1) \times \mathbf{pdab}]$, 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 $\mathbf{ab}[j - i + (i - 1) \times \mathbf{pdab}]$, 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 $\mathbf{ab}[k_a + j - i + (i - 1) \times \mathbf{pdab}]$, for $i = 1, \ldots, n$ and $j = \max(1, i - k_a), \ldots, i$.

On exit: the upper or lower triangle of **ab** 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** $\geq$ **ka** $+ 1$.

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

Note: the dimension, *dim*, of the array **bb** must be at least max$(1,$ **pdbb** $\times$ **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$(1,$ **pdx** $\times$ **n**$)$ when **vect** = Nag_FormX;
1 when **vect** = Nag_DoNotForm.

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

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

On exit: the n by n matrix $X = S^{-1}Q$, if **vect** = Nag_FormX.

If **vect** = Nag_DoNotForm, **x** is not referenced.

12: **pdx** – Integer *Input*

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

Constraints:

if **vect** = Nag_FormX, **pdx** $\geq$ max$(1,$ **n**$)$;
if **vect** = Nag_DoNotForm, **pdx** ≥ 1.

13: **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 ⟨*value*⟩ had an illegal value.

NE_ENUM_INT_2

On entry, **vect** = ⟨*value*⟩, **pdx** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: if **vect** = Nag_FormX, **pdx** $\geq$ max$(1,$ **n**$)$;
if **vect** = Nag_DoNotForm, **pdx** ≥ 1.

NE_INT

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

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

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

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

7 Accuracy

Forming the reduced matrix C is a stable procedure. However it involves implicit multiplication by B^{-1}. When nag_zhbgst (f08usc) 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 Parallelism and Performance

nag_zhbgst (f08usc) is not threaded by NAG in any implementation.

nag_zhbgst (f08usc) 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 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).

10 Example

This example computes 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.

10.1 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, pdbb, pdx, d_len, e_len;
  Integer       exit_status = 0;
  NagError      fail;
  Nag_UploType  uplo;
  Nag_OrderType order;
  /* Arrays */
  char          nag_enum_arg[40];
  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)*pdbb + k2 + I - J - 1]
#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 + k1 + J - I - 1]
#define BB_UPPER(I, J) bb[(I-1)*pdbb + J - I]
#define BB_LOWER(I, J) bb[(I-1)*pdbb + k2 + J - I - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_zhbgst (f08usc) Example Program Results\n\n");

  /* Skip heading in data file */
  scanf("%*[^\n] ");
```

```
scanf("%ld%ld%ld%*[^\n] ", &n, &ka, &kb);
pdab = ka + 1;
pdbb = kb + 1;
pdx = n;
d_len = n;
e_len = n-1;

/* Allocate memory */
if (!(ab = NAG_ALLOC(pdab * n, Complex)) ||
    !(bb = NAG_ALLOC(pdbb * n, Complex)) ||
    !(d = NAG_ALLOC(d_len, double)) ||
    !(e = NAG_ALLOC(e_len, double)) ||
    !(x = NAG_ALLOC(n * n, Complex)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
/* Read whether Upper or Lower part of A is stored */
scanf("%39s%*[^\n] ", nag_enum_arg);
/* nag_enum_name_to_value (x04nac).
 * Converts NAG enum member name to value
 */
uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);
/* 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)
          {
            scanf(" ( %lf , %lf ) ", &AB_UPPER(i, j).re,
                   &AB_UPPER(i, j).im);
          }
      }
    scanf("%*[^\n] ");
  }
else
  {
    for (i = 1; i <= n; ++i)
      {
        for (j = MAX(1, i-ka); j <= i; ++j)
          {
            scanf(" ( %lf , %lf ) ", &AB_LOWER(i, j).re,
                   &AB_LOWER(i, j).im);
          }
      }
    scanf("%*[^\n] ");
  }
if (uplo == Nag_Upper)
  {
    for (i = 1; i <= n; ++i)
      {
        for (j = i; j <= MIN(i+kb, n); ++j)
          {
            scanf(" ( %lf, %lf ) ", &BB_UPPER(i, j).re,
                   &BB_UPPER(i, j).im);
          }
      }
    scanf("%*[^\n] ");
  }
else
  {
    for (i = 1; i <= n; ++i)
      {
        for (j = MAX(1, i-kb); j <= i; ++j)
          {
            scanf(" ( %lf, %lf ) ", &BB_LOWER(i, j).re,
                   &BB_LOWER(i, j).im);
```

```
            }
        }
      scanf("%*[^\n] ");
    }
  /* Compute the split Cholesky factorization of B */
  /* nag_zpbstf (f08utc).
   * Computes a split Cholesky factorization of complex
   * Hermitian positive-definite band matrix A
   */
  nag_zpbstf(order, uplo, n, kb, bb, pdbb, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zpbstf (f08utc).\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 */
  /* nag_zhbgst (f08usc).
   * Reduction of complex Hermitian-definite banded
   * generalized eigenproblem Ax = lambda Bx to standard form
   * Cy = lambda y, such that C has the same bandwidth as A
   */
  nag_zhbgst(order, Nag_DoNotForm, uplo, n, ka, kb, ab, pdab, bb, pdbb,
             x, pdx, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zhbgst (f08usc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Reduce C to tridiagonal form T = (Q**T)*C*Q */
  /* nag_zhbtrd (f08hsc).
   * Unitary reduction of complex Hermitian band matrix to
   * real symmetric tridiagonal form
   */
  nag_zhbtrd(order, Nag_DoNotForm, uplo, n, ka, ab, pdab, d, e,
             x, pdx, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zhbtrd (f08hsc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Calculate the eigenvalues of T (same as C) */
  /* nag_dsterf (f08jfc).
   * All eigenvalues of real symmetric tridiagonal matrix,
   * root-free variant of QL or QR
   */
  nag_dsterf(n, d, e, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dsterf (f08jfc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Print eigenvalues */
  printf(" Eigenvalues\n");
  for (i = 0; i < n; ++i)
    printf(" %8.4lf", d[i]);
  printf("\n");
 END:
  NAG_FREE(ab);
  NAG_FREE(bb);
  NAG_FREE(d);
  NAG_FREE(e);
  NAG_FREE(x);
  return exit_status;
}
```

10.2 Program Data

```
nag_zhbgst (f08usc) Example Program Data
  4  2  1                                                    :Values of n, ka and kb
  Nag_Lower                                                  :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
```

10.3 Program Results

```
nag_zhbgst (f08usc) Example Program Results

 Eigenvalues
  -6.6089  -2.0416   0.1603   1.7712
```