

NAG C Library Function Document

nag_zhpgst (f08tsc)

1 Purpose

nag_zhpgst (f08tsc) reduces a complex Hermitian-definite generalized eigenproblem $Az = \lambda Bz$, $ABz = \lambda z$ or $BAz = \lambda z$ to the standard form $Cy = \lambda y$, where A is a complex Hermitian matrix and B has been factorized by nag_zpstrf (f07grc), using packed storage.

2 Specification

```
void nag_zhpgst (Nag_OrderType order, Nag_ComputeType comp_type,
                 Nag_UploType uplo, Integer n, Complex ap[], const Complex bp[], NagError *fail)
```

3 Description

To reduce the complex Hermitian-definite generalized eigenproblem $Az = \lambda Bz$, $ABz = \lambda z$ or $BAz = \lambda z$ to the standard form $Cy = \lambda y$ using packed storage, this function must be preceded by a call to nag_zpstrf (f07grc) which computes the Cholesky factorization of B ; B must be positive-definite.

The different problem types are specified by the parameter **comp_type**, as indicated in the table below. The table shows how C is computed by the function, and also how the eigenvectors z of the original problem can be recovered from the eigenvectors of the standard form.

comp_type	Problem	uplo	B	C	z
1	$Az = \lambda Bz$	Nag_Upper Nag_Lower	$U^H U$ LL^H	$U^{-H} AU^{-1}$ $L^{-1} AL^{-H}$	$U^{-1} y$ $L^{-H} y$
2	$ABz = \lambda z$	Nag_Upper Nag_Lower	$U^H U$ LL^H	UAU^H $L^H AL$	$U^{-1} y$ $L^{-H} y$
3	$BAz = \lambda z$	Nag_Upper Nag_Lower	$U^H U$ LL^H	UAU^H $L^H AL$	$U^H y$ Ly

4 References

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

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: **comp_type** – Nag_ComputeType Input

On entry: indicates how the standard form is computed as follows:

if **comp_type = Nag_Compute_1**,

if **uplo** = **Nag_Upper**, $C = U^{-H}AU^{-1}$;
 if **uplo** = **Nag_Lower**, $C = L^{-1}AL^{-H}$;
 if **comp_type** = **Nag_Compute_2** or **Nag_Compute_3**,
 if **uplo** = **Nag_Upper**, $C = UAU^H$;
 if **uplo** = **Nag_Lower**, $C = L^H AL$.

Constraint: **comp_type** = **Nag_Compute_1**, **Nag_Compute_2** or **Nag_Compute_3**.

3: **uplo** – Nag_UploType *Input*

On entry: indicates whether the upper or lower triangular part of A is stored and how B has been factorized, as follows:

if **uplo** = **Nag_Upper**, the upper triangular part of A is stored and $B = U^H U$;
 if **uplo** = **Nag_Lower**, the lower triangular part of A is stored and $B = LL^H$.

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: **ap**[*dim*] – Complex *Input/Output*

Note: the dimension, *dim*, of the array **ap** must be at least $\max(1, n \times (n + 1)/2)$.

On entry: the symmetric matrix A , packed by rows or columns. 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 **ap**[($j - 1$) $\times$ $j/2 + i - 1$], for $i \leq j$;
 if **order** = **Nag_ColMajor** and **uplo** = **Nag_Lower**,
 a_{ij} is stored in **ap**[($2n - j$) $\times$ ($j - 1$)/2 + $i - 1$], for $i \geq j$;
 if **order** = **Nag_RowMajor** and **uplo** = **Nag_Upper**,
 a_{ij} is stored in **ap**[($2n - i$) $\times$ ($i - 1$)/2 + $j - 1$], for $i \leq j$;
 if **order** = **Nag_RowMajor** and **uplo** = **Nag_Lower**,
 a_{ij} is stored in **ap**[($i - 1$) $\times$ $i/2 + j - 1$], for $i \geq j$.

On exit: the upper or lower triangle of A is overwritten by the corresponding upper or lower triangle of C as specified by **comp_type** and **uplo**, using the same packed storage format as described above.

6: **bp**[*dim*] – const Complex *Input*

Note: the dimension, *dim*, of the array **bp** must be at least $\max(1, n \times (n + 1)/2)$.

On entry: the Cholesky factor of B as specified by **uplo** and returned by nag_zpptrf (f07grc).

7: **fail** – NagError * *Output*

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

On entry, **n** = *<value>*.

Constraint: $n \geq 0$.

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} if (**comp_type** = **Nag_Compute_1**) or B (if **comp_type** = **Nag_Compute_2** or **Nag_Compute_3**). 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. See the document for F02HDF for further details.

8 Further Comments

The total number of real floating-point operations is approximately $4n^3$.

The real analogue of this function is nag_dspgst (f08tec).

9 Example

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

$$A = \begin{pmatrix} -7.36 + 0.00i & 0.77 - 0.43i & -0.64 - 0.92i & 3.01 - 6.97i \\ 0.77 + 0.43i & 3.49 + 0.00i & 2.19 + 4.45i & 1.90 + 3.73i \\ -0.64 + 0.92i & 2.19 - 4.45i & 0.12 + 0.00i & 2.88 - 3.17i \\ 3.01 + 6.97i & 1.90 - 3.73i & 2.88 + 3.17i & -2.54 + 0.00i \end{pmatrix}$$

and

$$B = \begin{pmatrix} 3.23 + 0.00i & 1.51 - 1.92i & 1.90 + 0.84i & 0.42 + 2.50i \\ 1.51 + 1.92i & 3.58 + 0.00i & -0.23 + 1.11i & -1.18 + 1.37i \\ 1.90 - 0.84i & -0.23 - 1.11i & 4.09 + 0.00i & 2.33 - 0.14i \\ 0.42 - 2.50i & -1.18 - 1.37i & 2.33 + 0.14i & 4.29 + 0.00i \end{pmatrix},$$

using packed storage. Here B is Hermitian positive-definite and must first be factorized by nag_zpptrf (f07grc). The program calls nag_zhpgst (f08tsc) to reduce the problem to the standard form $Cy = \lambda y$; then nag_zhptrd (f08gsc) to reduce C to tridiagonal form, and nag_dsterf (f08jfc) to compute the eigenvalues.

9.1 Program Text

```
/* nag_zhpgst (f08tsc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf07.h>
#include <nagf08.h>

int main(void)
{
```

```

/* Scalars */
Integer i, j, n, ap_len, bp_len, d_len, e_len, tau_len;
Integer exit_status=0;
NagError fail;
Nag_UploType uplo;
Nag_OrderType order;

/* Arrays */
char uplo_char[2];
Complex *ap=0, *bp=0, *tau=0;
double *d=0, *e=0;

#ifdef NAG_COLUMN_MAJOR
#define A_UPPER(I,J) ap[J*(J-1)/2 + I - 1]
#define A_LOWER(I,J) ap[(2*n-J)*(J-1)/2 + I - 1]
#define B_UPPER(I,J) bp[J*(J-1)/2 + I - 1]
#define B_LOWER(I,J) bp[(2*n-J)*(J-1)/2 + I - 1]
    order = Nag_ColMajor;
#else
#define A_LOWER(I,J) ap[I*(I-1)/2 + J - 1]
#define A_UPPER(I,J) ap[(2*n-I)*(I-1)/2 + J - 1]
#define B_LOWER(I,J) bp[I*(I-1)/2 + J - 1]
#define B_UPPER(I,J) bp[(2*n-I)*(I-1)/2 + J - 1]
    order = Nag_RowMajor;
#endif

INIT_FAIL(fail);
Vprintf("f08tsc Example Program Results\n\n");

/* Skip heading in data file */
Vscanf("%*[^\\n] ");
Vscanf("%ld%*[^\\n] ", &n);
ap_len = n * (n + 1) / 2;
bp_len = n * (n + 1) / 2;
d_len = n;
e_len = n - 1;
tau_len = n;

/* Allocate memory */
if ( !(ap = NAG_ALLOC(ap_len, Complex)) ||
    !(bp = NAG_ALLOC(bp_len, Complex)) ||
    !(d = NAG_ALLOC(d_len, double)) ||
    !(e = NAG_ALLOC(e_len, double)) ||
    !(tau = NAG_ALLOC(tau_len, Complex)) )
{
    Vprintf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

/* Read A and B from data file */
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;
}
if (uplo == Nag_Upper)
{
    for (i = 1; i <= n; ++i)
    {
        for (j = i; j <= n; ++j)
        {
            Vscanf(" ( %lf , %lf )", &A_UPPER(i,j).re,
                &A_UPPER(i,j).im);
        }
    }
}

```

```

    }
    Vscanf("%*[^\\n] ");
    for (i = 1; i <= n; ++i)
    {
        for (j = i; j <= n; ++j)
        {
            Vscanf(" ( %lf , %lf )", &B_UPPER(i,j).re,
                    &B_UPPER(i,j).im);
        }
    }
    Vscanf("%*[^\\n] ");
}
else
{
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= i; ++j)
        {
            Vscanf(" ( %lf , %lf )", &A_LOWER(i,j).re,
                    &A_LOWER(i,j).im);
        }
    }
    Vscanf("%*[^\\n] ");
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= i; ++j)
        {
            Vscanf(" ( %lf , %lf )", &B_LOWER(i,j).re,
                    &B_LOWER(i,j).im);
        }
    }
    Vscanf("%*[^\\n] ");
}
/* Compute the Cholesky factorization of B */
f07grc(order, uplo, n, bp, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07gdc.\\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 */
f08tsc(order, Nag_Compute_1, uplo, n, ap, bp, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f08tsc.\\n%s\\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Reduce C to tridiagonal form T = (Q**T)*C*Q */
f08gsc(order, uplo, n, ap, d, e, tau, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f08gsc.\\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 = 1; i <= n; ++i)
    Vprintf("%8.4f%s", d[i-1], i%9==0 || i==n ? "\\n":" ");
Vprintf("\\n");

```

```

END:
  if (ap) NAG_FREE(ap);
  if (bp) NAG_FREE(bp);
  if (d) NAG_FREE(d);
  if (e) NAG_FREE(e);
  if (tau) NAG_FREE(tau);

  return exit_status;
}

```

9.2 Program Data

f08tsc Example Program Data

```

4                                     :Value of N
'L'                                 :Value of UPLO
(-7.36, 0.00)
( 0.77, 0.43) ( 3.49, 0.00)
(-0.64, 0.92) ( 2.19,-4.45) ( 0.12, 0.00)
( 3.01, 6.97) ( 1.90,-3.73) ( 2.88, 3.17) (-2.54, 0.00) :End of matrix A
( 3.23, 0.00)
( 1.51, 1.92) ( 3.58, 0.00)
( 1.90,-0.84) (-0.23,-1.11) ( 4.09, 0.00)
( 0.42,-2.50) (-1.18,-1.37) ( 2.33, 0.14) ( 4.29, 0.00) :End of matrix B

```

9.3 Program Results

f08tsc Example Program Results

```

Eigenvalues
-5.9990  -2.9936   0.5047   3.9990

```
