

NAG C Library Function Document

nag_zhegst (f08ssc)

1 Purpose

nag_zhegst (f08ssc) 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_zpotrf (f07frc).

2 Specification

```
void nag_zhegst (Nag_OrderType order, Nag_ComputeType comp_type,
                Nag_UploType uplo, Integer n, Complex a[], Integer pda, const Complex b[],
                Integer pdb, 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 $Cy = \lambda y$, this function must be preceded by a call to nag_zpotrf (f07frc) 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: **a**[*dim*] – Complex *Input/Output*

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

If **order** = **Nag-ColMajor**, the (i, j) th element of the matrix A is stored in **a**[($j - 1$) $\times$ **pda** + $i - 1$] and if **order** = **Nag-RowMajor**, the (i, j) th element of the matrix A is stored in **a**[($i - 1$) $\times$ **pda** + $j - 1$].

On entry: the n by n Hermitian matrix A . If **uplo** = **Nag_Upper**, the upper triangle of A must be stored and the elements of the array below the diagonal are not referenced; if **uplo** = **Nag_Lower**, the lower triangle of A must be stored and the elements of the array above the diagonal are not referenced.

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**.

6: **pda** – Integer *Input*

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

Constraint: $\mathbf{pda} \geq \max(1, \mathbf{n})$.

7: **b**[*dim*] – Complex *Input/Output*

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

If **order** = **Nag-ColMajor**, the (i, j) th element of the matrix B is stored in **b**[($j - 1$) $\times$ **pdb** + $i - 1$] and if **order** = **Nag-RowMajor**, the (i, j) th element of the matrix B is stored in **b**[($i - 1$) $\times$ **pdb** + $j - 1$].

On entry: the Cholesky factor of B as specified by **uplo** and returned by nag_zpotrf (f07frc).

On exit: used as internal workspace prior to being restored and hence is unchanged.

8: **pdb** – Integer *Input*

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

Constraint: $\mathbf{pdb} \geq \max(1, \mathbf{n})$.

9: **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** ≥ 0 .

On entry, **pda** = $\langle value \rangle$.

Constraint: **pda** > 0 .

On entry, **pdb** = $\langle value \rangle$.

Constraint: **pdb** > 0 .

NE_INT_2

On entry, **pda** = $\langle value \rangle$, **n** = $\langle value \rangle$.

Constraint: **pda** $\geq \max(1, \mathbf{n})$.

On entry, **pdb** = $\langle value \rangle$, **n** = $\langle value \rangle$.

Constraint: **pdb** $\geq \max(1, \mathbf{n})$.

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.

8 Further Comments

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

The real analogue of this function is nag_dsyst (f08sec).

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

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

Here B is Hermitian positive-definite and must first be factorized by `nag_zpotrf` (f07frc). The program calls `nag_zhegst` (f08ssc) to reduce the problem to the standard form $Cy = \lambda y$; then `nag_zhetrd` (f08fsc) to reduce C to tridiagonal form, and `nag_dsterf` (f08jfc) to compute the eigenvalues.

9.1 Program Text

```
/* nag_zhegst (f08ssc) 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, pda, pdb, d_len, e_len, tau_len;
    Integer exit_status=0;
    NagError fail;
    Nag_UploType uplo;
    Nag_OrderType order;
    /* Arrays */
    char uplo_char[2];
    double *d=0, *e=0;
    Complex *a=0, *b=0, *tau=0;

#ifdef NAG_COLUMN_MAJOR
#define A(I,J) a[(J-1)*pda + I - 1]
#define B(I,J) b[(J-1)*pdb + I - 1]
    order = Nag_ColMajor;
#else
#define A(I,J) a[(I-1)*pda + J - 1]
#define B(I,J) b[(I-1)*pdb + J - 1]
    order = Nag_RowMajor;
#endif

    INIT_FAIL(fail);
    Vprintf("f08ssc Example Program Results\n\n");

    /* Skip heading in data file */
    Vscanf("%*[^\\n] ");
    Vscanf("%ld%*[^\\n] ", &n);
#ifdef NAG_COLUMN_MAJOR
    pda = n;
    pdb = n;
#else
    pda = n;
    pdb = n;
#endif
    d_len = n;
    e_len = n-1;
    tau_len = n-1;

    /* Allocate memory */
    if ( !(a = NAG_ALLOC(n * n, Complex)) ||
        !(b = NAG_ALLOC(n * n, 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(i,j).re, &A(i,j).im);
    }
    Vscanf("%*[\n] ");
    for (i = 1; i <= n; ++i)
    {
        for (j = i; j <= n; ++j)
            Vscanf(" ( %lf , %lf )", &B(i,j).re, &B(i,j).im);
    }
    Vscanf("%*[\n] ");
}
else
{
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= i; ++j)
            Vscanf(" ( %lf , %lf )", &A(i,j).re, &A(i,j).im);
    }
    Vscanf("%*[\n] ");
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= i; ++j)
            Vscanf(" ( %lf , %lf )", &B(i,j).re, &B(i,j).im);
    }
    Vscanf("%*[\n] ");
}

/* Compute the Cholesky factorization of B */
f07frc(order, uplo, n, b, pdb, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07frc.\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 */
f08ssc(order, Nag_Compute_1, uplo, n, a, pda, b, pdb, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f08ssc.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Reduce C to tridiagonal form T = (Q**T)*C*Q */
f08fsc(order, uplo, n, a, pda, d, e, tau, &fail);
if (fail.code != NE_NOERROR)

```

```

    {
        Vprintf("Error from f08fsc.\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 ? "\n": " ");
    Vprintf("\n");
END:
    if (a) NAG_FREE(a);
    if (b) NAG_FREE(b);
    if (d) NAG_FREE(d);
    if (e) NAG_FREE(e);
    if (tau) NAG_FREE(tau);

    return exit_status;
}

```

9.2 Program Data

f08ssc 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

f08ssc Example Program Results

```

Eigenvalues
-5.9990 -2.9936 0.5047 3.9990

```
