NAG 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

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

```
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 $Cy = \lambda y$, nag_zhegst (f08ssc) 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 argument **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.

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

4 References

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

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: **comp_type** – Nag_ComputeType *Input*

On entry: indicates how the standard form is computed.

comp_type $=$ Nag_Compute_1

if **uplo** $=$ Nag_Upper, $C = U^{-\mathrm{H}} A U^{-1}$ when **order** $=$ Nag_ColMajor and $C = U^{-1} A U^{-\mathrm{H}}$ when **order** $=$ Nag_RowMajor;

if **uplo** $=$ Nag_Lower, $C = L^{-1} A L^{-\mathrm{H}}$ when **order** $=$ Nag_ColMajor and $C = L^{-\mathrm{H}} A L^{-1}$ when **order** $=$ Nag_RowMajor.

comp_type $=$ Nag_Compute_2 or Nag_Compute_3

if **uplo** $=$ Nag_Upper, $C = UAU^{\mathrm{H}}$ when **order** $=$ Nag_ColMajor and $C = U^{\mathrm{H}}AU$ when **order** $=$ Nag_RowMajor;

if **uplo** $=$ Nag_Lower, $C = L^{\mathrm{H}}AL$ when **order** $=$ Nag_ColMajor and $C = LAL^{\mathrm{H}}$ when **order** $=$ Nag_RowMajor.

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.

uplo $=$ Nag_Upper
The upper triangular part of A is stored and $B = U^{\mathrm{H}}U$ when **order** $=$ Nag_ColMajor and $B = UU^{\mathrm{H}}$ when **order** $=$ Nag_RowMajor.

uplo $=$ Nag_Lower
The lower triangular part of A is stored and $B = LL^{\mathrm{H}}$ when **order** $=$ Nag_ColMajor and $B = L^{\mathrm{H}}L$ when **order** $=$ Nag_RowMajor.

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

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

On entry: the n by n Hermitian matrix A.

If **order** $=$ Nag_ColMajor, A_{ij} is stored in $\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$.

If **order** $=$ Nag_RowMajor, A_{ij} is stored in $\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1]$.

If **uplo** $=$ Nag_Upper, the upper triangular part of A must be stored and the elements of the array below the diagonal are not referenced.

If **uplo** $=$ Nag_Lower, the lower triangular part 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: **pda** $\geq \max(1, \mathbf{n})$.

7: **b**[dim] – const Complex *Input*

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

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

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

9: **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.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_INT

On entry, **n** = ⟨*value*⟩.
Constraint: **n** ≥ 0.

On entry, **pda** = ⟨*value*⟩.
Constraint: **pda** > 0.

On entry, **pdb** = ⟨*value*⟩.
Constraint: **pdb** > 0.

NE_INT_2

On entry, **pda** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: **pda** $\geq \max(1, \mathbf{n})$.

On entry, **pdb** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: **pdb** $\geq \max(1, \mathbf{n})$.

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.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in the Essential Introduction for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

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 nag_zhegst (f08ssc) 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_zhegst (f08ssc) is not threaded by NAG in any implementation.

nag_zhegst (f08ssc) 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 X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also 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 $4n^3$.

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

10 Example

This example computes 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.

10.1 Program Text

```
/* nag_zhegst (f08ssc) Example Program.
 *
 * Copyright 2014 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          nag_enum_arg[40];
  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);

  printf("nag_zhegst (f08ssc) Example Program Results\n\n");

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%*[^\n] ", &n);
#else
  scanf("%"NAG_IFMT"%*[^\n] ", &n);
#endif
#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)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  /* Read A and B from data file */
#ifdef _WIN32
  scanf_s("%39s%*[^\n] ", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf("%39s%*[^\n] ", nag_enum_arg);
#endif
  /* nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);
  if (uplo == Nag_Upper)
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = i; j <= n; ++j)
```

```
#ifdef _WIN32
            scanf_s(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#else
            scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#endif
        }
#ifdef _WIN32
      scanf_s("%*[^\n] ");
#else
      scanf("%*[^\n] ");
#endif
      for (i = 1; i <= n; ++i)
        {
          for (j = i; j <= n; ++j)
#ifdef _WIN32
            scanf_s(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
#else
            scanf(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
#endif
        }
#ifdef _WIN32
      scanf_s("%*[^\n] ");
#else
      scanf("%*[^\n] ");
#endif
    }
  else
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = 1; j <= i; ++j)
#ifdef _WIN32
            scanf_s(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#else
            scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#endif
        }
#ifdef _WIN32
      scanf_s("%*[^\n] ");
#else
      scanf("%*[^\n] ");
#endif
      for (i = 1; i <= n; ++i)
        {
          for (j = 1; j <= i; ++j)
#ifdef _WIN32
            scanf_s(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
#else
            scanf(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
#endif
        }
#ifdef _WIN32
      scanf_s("%*[^\n] ");
#else
      scanf("%*[^\n] ");
#endif
    }

  /* Compute the Cholesky factorization of B */
  /* nag_zpotrf (f07frc).
   * Cholesky factorization of complex Hermitian
   * positive-definite matrix
   */
  nag_zpotrf(order, uplo, n, b, pdb, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zpotrf (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 */
  /* nag_zhegst (f08ssc).
   * Reduction to standard form of complex Hermitian-definite
   * generalized eigenproblem Ax = lambda Bx, ABx = lambda x
   * or BAx = lambda x, B factorized by nag_zpotrf (f07frc)
   */
  nag_zhegst(order, Nag_Compute_1, uplo, n, a, pda, b, pdb, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zhegst (f08ssc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Reduce C to tridiagonal form T = (Q**T)*C*Q */
  /* nag_zhetrd (f08fsc).
   * Unitary reduction of complex Hermitian matrix to real
   * symmetric tridiagonal form
   */
  nag_zhetrd(order, uplo, n, a, pda, d, e, tau, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zhetrd (f08fsc).\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 = 1; i <= n; ++i)
    printf("%8.4f%s", d[i-1], i%9 == 0?"\n":" ");
  printf("\n");
 END:
  NAG_FREE(a);
  NAG_FREE(b);
  NAG_FREE(d);
  NAG_FREE(e);
  NAG_FREE(tau);

  return exit_status;
}
```

10.2 Program Data

```
nag_zhegst (f08ssc) Example Program Data
  4                                                        :Value of n
  Nag_Lower                                                :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
```

10.3 Program Results

```
nag_zhegst (f08ssc) Example Program Results

Eigenvalues
 -5.9990  -2.9936   0.5047   3.9990
```