NAG 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_zpptrf (f07grc), using packed storage.

2 Specification

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

```
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, nag_zhpgst (f08tsc) must be preceded by a call to nag_zpptrf (f07grc) 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.

comp_type	Problem	**uplo**	B	C	z
Nag_Compute_1	$Az = \lambda Bz$	Nag_Upper Nag_Lower	U^HU LL^H	$U^{-H}AU^{-1}$ $L^{-1}AL^{-H}$	$U^{-1}y$ $L^{-H}y$
Nag_Compute_2	$ABz = \lambda z$	Nag_Upper Nag_Lower	U^HU LL^H	UAU^H L^HAL	$U^{-1}y$ $L^{-H}y$
Nag_Compute_3	$BAz = \lambda z$	Nag_Upper Nag_Lower	U^HU LL^H	UAU^H L^HAL	U^Hy Ly

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, $C = U^{-\mathrm{H}}AU^{-1}$;

if **uplo** = Nag_Lower, $C = L^{-1}AL^{-\mathrm{H}}$.

comp_type = Nag_Compute_2 or Nag_Compute_3

if **uplo** = Nag_Upper, $C = UAU^{\mathrm{H}}$;

if **uplo** = Nag_Lower, $C = L^{\mathrm{H}}AL$.

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.

uplo = Nag_Upper
The upper triangular part of A is stored and $B = U^{\mathrm{H}}U$.

uplo = Nag_Lower
The lower triangular part of A is stored and $B = LL^{\mathrm{H}}$.

Constraint: **uplo** = Nag_Upper or Nag_Lower.

4: **n** – Integer *Input*

On entry: n, the order of the matrices A and B.

Constraint: **n** ≥ 0.

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

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

On entry: the upper or lower triangle of the n by n Hermitian matrix A, packed by rows or columns.

The storage of elements A_{ij} depends on the **order** and **uplo** arguments 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 **ap** 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 $\max(1, \mathbf{n} \times (\mathbf{n}+1)/2)$.

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

7: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_BAD_PARAM

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

NE_INT

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

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} if (**comp_type** = Nag_Compute_1) or B (if **comp_type** = Nag_Compute_2 or Nag_Compute_3). When nag_zhpgst (f08tsc) 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_zhpgst (f08tsc) is not threaded by NAG in any implementation.

nag_zhpgst (f08tsc) 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 $4n^3$.

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

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

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

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.

10.1 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          nag_enum_arg[40];
  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);

  printf("nag_zhpgst (f08tsc) Example Program Results\n\n");

  /* Skip heading in data file */
  scanf("%*[^\n] ");
  scanf("%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)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read A and B from data file */
  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);
  if (uplo == Nag_Upper)
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = i; j <= n; ++j)
            {
              scanf(" ( %lf , %lf )", &A_UPPER(i, j).re,
                     &A_UPPER(i, j).im);
            }
        }
      scanf("%*[^\n] ");
      for (i = 1; i <= n; ++i)
        {
          for (j = i; j <= n; ++j)
            {
              scanf(" ( %lf , %lf )", &B_UPPER(i, j).re,
                     &B_UPPER(i, j).im);
            }
        }
      scanf("%*[^\n] ");
    }
  else
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = 1; j <= i; ++j)
            {
              scanf(" ( %lf , %lf )", &A_LOWER(i, j).re,
                     &A_LOWER(i, j).im);
            }
        }
      scanf("%*[^\n] ");
      for (i = 1; i <= n; ++i)
        {
          for (j = 1; j <= i; ++j)
            {
              scanf(" ( %lf , %lf )", &B_LOWER(i, j).re,
                     &B_LOWER(i, j).im);
            }
        }
      scanf("%*[^\n] ");
    }
  /* Compute the Cholesky factorization of B */
  /* nag_zpptrf (f07grc).
   * Cholesky factorization of complex Hermitian
   * positive-definite matrix, packed storage
   */
  nag_zpptrf(order, uplo, n, bp, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dpptrf (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 */
  /* nag_zhpgst (f08tsc).
   * Reduction to standard form of complex Hermitian-definite
   * generalized eigenproblem Ax = lambda Bx, ABx = lambda x
   * or BAx = lambda x, packed storage, B factorized by
   * nag_zpptrf (f07grc)
   */
  nag_zhpgst(order, Nag_Compute_1, uplo, n, ap, bp, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zhpgst (f08tsc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Reduce C to tridiagonal form T = (Q**T)*C*Q */
  /* nag_zhptrd (f08gsc).
   * Unitary reduction of complex Hermitian matrix to real
   * symmetric tridiagonal form, packed storage
   */
  nag_zhptrd(order, uplo, n, ap, d, e, tau, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zhptrd (f08gsc).\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 || i == n?"\n":" ");
  printf("\n");
 END:
  NAG_FREE(ap);
  NAG_FREE(bp);
  NAG_FREE(d);
  NAG_FREE(e);
  NAG_FREE(tau);

  return exit_status;
}
```

10.2 Program Data

```
nag_zhpgst (f08tsc) 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_zhpgst (f08tsc) Example Program Results

Eigenvalues
 -5.9990  -2.9936   0.5047   3.9990
```