NAG Library Function Document

nag_zhpgvx (f08tpc)

1 Purpose

nag_zhpgvx (f08tpc) computes selected eigenvalues and, optionally, eigenvectors of a complex generalized Hermitian-definite eigenproblem, of the form

$$Az = \lambda Bz, \quad ABz = \lambda z \quad \text{or} \quad BAz = \lambda z,$$

where A and B are Hermitian, stored in packed format, and B is also positive definite. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

2 Specification

```
#include <nag.h>
#include <nagf08.h>
void nag_zhpgvx (Nag_OrderType order, Integer itype, Nag_JobType job,
     Nag_RangeType range, Nag_UploType uplo, Integer n, Complex ap[],
     Complex bp[], double vl, double vu, Integer il, Integer iu,
     double abstol, Integer *m, double w[], Complex z[], Integer pdz,
     Integer jfail[], NagError *fail)
```

3 Description

nag_zhpgvx (f08tpc) first performs a Cholesky factorization of the matrix B as $B = U^{\mathrm{H}}U$, when **uplo** = Nag_Upper or $B = LL^{\mathrm{H}}$, when **uplo** = Nag_Lower. The generalized problem is then reduced to a standard symmetric eigenvalue problem

$$Cx = \lambda x,$$

which is solved for the desired eigenvalues and eigenvectors; the eigenvectors are then backtransformed to give the eigenvectors of the original problem.

For the problem $Az = \lambda Bz$, the eigenvectors are normalized so that the matrix of eigenvectors, Z, satisfies

$$Z^{\mathrm{H}}AZ = \Lambda \quad \text{and} \quad Z^{\mathrm{H}}BZ = I,$$

where Λ is the diagonal matrix whose diagonal elements are the eigenvalues. For the problem $ABz = \lambda z$ we correspondingly have

$$Z^{-1}AZ^{-\mathrm{H}} = \Lambda \quad \text{and} \quad Z^{\mathrm{H}}BZ = I,$$

and for $BAz = \lambda z$ we have

$$Z^{\mathrm{H}}AZ = \Lambda \quad \text{and} \quad Z^{\mathrm{H}}B^{-1}Z = I.$$

4 References

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A and Sorensen D (1999) *LAPACK Users' Guide* (3rd Edition) SIAM, Philadelphia http://www.netlib.org/lapack/lug

Demmel J W and Kahan W (1990) Accurate singular values of bidiagonal matrices *SIAM J. Sci. Statist. Comput.* **11** 873–912

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 2.3.1.3 in How to Use the NAG Library and its Documentation for a more detailed explanation of the use of this argument.

Constraint: **order** = Nag_RowMajor or Nag_ColMajor.

2: **itype** – Integer *Input*

On entry: specifies the problem type to be solved.

itype = 1
$Az = \lambda Bz$.

itype = 2
$ABz = \lambda z$.

itype = 3
$BAz = \lambda z$.

Constraint: **itype** = 1, 2 or 3.

3: **job** – Nag_JobType *Input*

On entry: indicates whether eigenvectors are computed.

job = Nag_EigVals
Only eigenvalues are computed.

job = Nag_DoBoth
Eigenvalues and eigenvectors are computed.

Constraint: **job** = Nag_EigVals or Nag_DoBoth.

4: **range** – Nag_RangeType *Input*

On entry: if **range** = Nag_AllValues, all eigenvalues will be found.

If **range** = Nag_Interval, all eigenvalues in the half-open interval $(\mathbf{vl}, \mathbf{vu}]$ will be found.

If **range** = Nag_Indices, the **il**th to **iu**th eigenvalues will be found.

Constraint: **range** = Nag_AllValues, Nag_Interval or Nag_Indices.

5: **uplo** – Nag_UploType *Input*

On entry: if **uplo** = Nag_Upper, the upper triangles of A and B are stored.

If **uplo** = Nag_Lower, the lower triangles of A and B are stored.

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

6: **n** – Integer *Input*

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

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

7: **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 contents of **ap** are destroyed.

8: **bp**[dim] – Complex *Input/Output*

Note: the dimension, *dim*, of the array **bp** 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 B, packed by rows or columns.

The storage of elements B_{ij} depends on the **order** and **uplo** arguments as follows:

if **order** = Nag_ColMajor and **uplo** = Nag_Upper,
B_{ij} is stored in **bp**$[(j-1) \times j/2 + i - 1]$, for $i \leq j$;
if **order** = Nag_ColMajor and **uplo** = Nag_Lower,
B_{ij} is stored in **bp**$[(2n-j) \times (j-1)/2 + i - 1]$, for $i \geq j$;
if **order** = Nag_RowMajor and **uplo** = Nag_Upper,
B_{ij} is stored in **bp**$[(2n-i) \times (i-1)/2 + j - 1]$, for $i \leq j$;
if **order** = Nag_RowMajor and **uplo** = Nag_Lower,
B_{ij} is stored in **bp**$[(i-1) \times i/2 + j - 1]$, for $i \geq j$.

On exit: the triangular factor U or L from the Cholesky factorization $B = U^{\mathrm{H}}U$ or $B = LL^{\mathrm{H}}$, in the same storage format as B.

9: **vl** – double *Input*

10: **vu** – double *Input*

On entry: if **range** = Nag_Interval, the lower and upper bounds of the interval to be searched for eigenvalues.

If **range** = Nag_AllValues or Nag_Indices, **vl** and **vu** are not referenced.

Constraint: if **range** = Nag_Interval, **vl** $<$ **vu**.

11: **il** – Integer *Input*

12: **iu** – Integer *Input*

On entry: if **range** = Nag_Indices, the indices (in ascending order) of the smallest and largest eigenvalues to be returned.

If **range** = Nag_AllValues or Nag_Interval, **il** and **iu** are not referenced.

Constraints:

if **range** = Nag_Indices and **n** = 0, **il** = 1 and **iu** = 0;
if **range** = Nag_Indices and **n** > 0, 1 ≤ **il** ≤ **iu** ≤ **n**.

13: **abstol** – double *Input*

On entry: the absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval $[a, b]$ of width less than or equal to

$$\textbf{abstol} + \epsilon \max(|a|, |b|),$$

where ϵ is the ***machine precision***. If **abstol** is less than or equal to zero, then $\epsilon \|T\|_1$ will be used in its place, where T is the tridiagonal matrix obtained by reducing C to tridiagonal form. Eigenvalues will be computed most accurately when **abstol** is set to twice the underflow threshold 2 × nag_real_safe_small_number( ), not zero. If this function returns with **fail.code** = NE_CONVERGENCE, indicating that some eigenvectors did not converge, try setting **abstol** to 2 × nag_real_safe_small_number( ). See Demmel and Kahan (1990).

14: **m** – Integer * *Output*

On exit: the total number of eigenvalues found. 0 ≤ **m** ≤ **n**.

If **range** = Nag_AllValues, **m** = **n**.

If **range** = Nag_Indices, **m** = **iu** − **il** + 1.

15: **w**[**n**] – double *Output*

On exit: the first **m** elements contain the selected eigenvalues in ascending order.

16: **z**[*dim*] – Complex *Output*

Note: the dimension, *dim*, of the array **z** must be at least

max(1, **pdz** × **n**) when **job** = Nag_DoBoth;
1 otherwise.

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

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

On exit: if **job** = Nag_DoBoth, then

if **fail.code** = NE_NOERROR, the first **m** columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the ith column of Z holding the eigenvector associated with **w**$[i-1]$. The eigenvectors are normalized as follows:

if **itype** = 1 or 2, $Z^{\mathrm{H}} B Z = I$;

if **itype** = 3, $Z^{\mathrm{H}} B^{-1} Z = I$;

if an eigenvector fails to converge (**fail.code** = NE_CONVERGENCE), then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in **jfail**.

If **job** = Nag_EigVals, **z** is not referenced.

17: **pdz** – Integer *Input*

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

Constraints:

if **job** = Nag_DoBoth, **pdz** $\geq \max(1, \mathbf{n})$;
otherwise **pdz** ≥ 1.

18: **jfail**[*dim*] – Integer *Output*

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

On exit: if **job** = Nag_DoBoth, then

if **fail.code** = NE_NOERROR, the first **m** elements of **jfail** are zero;

if **fail.code** = NE_CONVERGENCE, **jfail** contains the indices of the eigenvectors that failed to converge.

If **job** = Nag_EigVals, **jfail** is not referenced.

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

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

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

NE_CONVERGENCE

The algorithm failed to converge; ⟨*value*⟩ eigenvectors failed to converge.

NE_ENUM_INT_2

On entry, **job** = ⟨*value*⟩, **pdz** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: if **job** = Nag_DoBoth, **pdz** $\geq \max(1, \mathbf{n})$;
otherwise **pdz** ≥ 1.

NE_ENUM_INT_3

On entry, **range** = ⟨*value*⟩, **il** = ⟨*value*⟩, **iu** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: if **range** = Nag_Indices and **n** = 0, **il** = 1 and **iu** = 0;
if **range** = Nag_Indices and **n** > 0, $1 \leq \mathbf{il} \leq \mathbf{iu} \leq \mathbf{n}$.

NE_ENUM_REAL_2

On entry, **range** = ⟨*value*⟩, **vl** = ⟨*value*⟩ and **vu** = ⟨*value*⟩.
Constraint: if **range** = Nag_Interval, **vl** < **vu**.

NE_INT

On entry, **itype** = ⟨*value*⟩.
Constraint: **itype** = 1, 2 or 3.

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

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

An unexpected error has been triggered by this function. Please contact NAG.
See Section 2.7.6 in How to Use the NAG Library and its Documentation for further information.

NE_MAT_NOT_POS_DEF

If **fail.errnum** $= \mathbf{n} + \langle value \rangle$, for $1 \leq \langle value \rangle \leq \mathbf{n}$, then the leading minor of order $\langle value \rangle$ of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

7 Accuracy

If B is ill-conditioned with respect to inversion, then the error bounds for the computed eigenvalues and vectors may be large, although when the diagonal elements of B differ widely in magnitude the eigenvalues and eigenvectors may be less sensitive than the condition of B would suggest. See Section 4.10 of Anderson *et al.* (1999) for details of the error bounds.

8 Parallelism and Performance

nag_zhpgvx (f08tpc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zhpgvx (f08tpc) 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 floating-point operations is proportional to n^3.

The real analogue of this function is nag_dspgvx (f08tbc).

10 Example

This example finds the eigenvalues in the half-open interval $(-3, 3]$, and corresponding eigenvectors, of the generalized Hermitian eigenproblem $Az = \lambda Bz$, where

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

and

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

The example program for nag_zhpgvd (f08tqc) illustrates solving a generalized symmetric eigenproblem of the form $ABz = \lambda z$.

10.1 Program Text

```
/* nag_zhpgvx (f08tpc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <stdio.h>
#include <nag.h>
#include <nagx04.h>
#include <nag_stdlib.h>
#include <nagf08.h>
#include <naga02.h>

int main(void)
{
  /* Scalars */
  double abstol, vl, vu;
  Integer i, il = 0, iu = 0, j, m, n, pdz;
  Integer exit_status = 0;

  /* Arrays */
  Complex *ap = 0, *bp = 0, *z = 0;
  double *w = 0;
  Integer *index = 0;
  char nag_enum_arg[40];

  /* Nag Types */
  NagError fail;
  Nag_OrderType order;
  Nag_UploType uplo;

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

  INIT_FAIL(fail);

  printf("nag_zhpgvx (f08tpc) 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
```

```
  if (n < 0) {
    printf("Invalid n\n");
    exit_status = 1;
    goto END;;
  }
#ifdef _WIN32
  scanf_s(" %39s%*[^\n]", nag_enum_arg, (unsigned)_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);

  m = n;
#ifdef NAG_COLUMN_MAJOR
  pdz = n;
#else
  pdz = m;
#endif
  /* Allocate memory */
  if (!(ap = NAG_ALLOC(n * (n + 1) / 2, Complex)) ||
      !(bp = NAG_ALLOC(n * (n + 1) / 2, Complex)) ||
      !(z = NAG_ALLOC(n * m, Complex)) ||
      !(w = NAG_ALLOC(n, double)) || !(index = NAG_ALLOC(n, Integer)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Read the lower and upper bounds of the interval to be searched. */
#ifdef _WIN32
  scanf_s("%lf%lf%*[^\n]", &vl, &vu);
#else
  scanf("%lf%lf%*[^\n]", &vl, &vu);
#endif
  /* Read the triangular parts of the matrices A and B from data file. */
  if (uplo == Nag_Upper) {
    for (i = 1; i <= n; ++i)
      for (j = i; j <= n; ++j)
#ifdef _WIN32
        scanf_s(" ( %lf , %lf )", &A_UPPER(i, j).re, &A_UPPER(i, j).im);
#else
        scanf(" ( %lf , %lf )", &A_UPPER(i, j).re, &A_UPPER(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_UPPER(i, j).re, &B_UPPER(i, j).im);
#else
        scanf(" ( %lf , %lf )", &B_UPPER(i, j).re, &B_UPPER(i, j).im);
#endif
  }
  else if (uplo == Nag_Lower) {
    for (i = 1; i <= n; ++i)
      for (j = 1; j <= i; ++j)
#ifdef _WIN32
        scanf_s(" ( %lf , %lf )", &A_LOWER(i, j).re, &A_LOWER(i, j).im);
#else
        scanf(" ( %lf , %lf )", &A_LOWER(i, j).re, &A_LOWER(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_LOWER(i, j).re, &B_LOWER(i, j).im);
#else
        scanf(" ( %lf , %lf )", &B_LOWER(i, j).re, &B_LOWER(i, j).im);
#endif
  }
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Use the default absolute error tolerance for eigenvalues. */
  abstol = 0.0;

  /* Solve the generalized Hermitian eigenvalue problem
   * A*x = lambda*B*x (itype = 1). using nag_zhpgvx (f08tpc).
   */
  nag_zhpgvx(order, 1, Nag_DoBoth, Nag_Interval, uplo, n, ap, bp, vl, vu,
             il, iu, abstol, &m, w, z, pdz, index, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_zhpgvx (f08tpc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Normalize the eigenvectors */
  for (j = 1; j <= m; j++)
    for (i = n; i >= 1; i--)
      Z(i, j) = nag_complex_divide(Z(i, j), Z(1, j));

  /* Print eigensolution */
  printf("Number of eigenvalues found =%5" NAG_IFMT "\n\n", m);
  printf(" Eigenvalues\n  ");
  for (j = 0; j < m; ++j)
    printf(" %11.4f%s", w[j], j % 8 == 7 ? "\n" : "");
  printf("\n");

  /* Print normalized vectors using nag_gen_complx_mat_print (x04dac). */
  fflush(stdout);
  nag_gen_complx_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, m,
                           z, pdz, "Selected eigenvectors", 0, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_complx_mat_print (x04dac).\n%s\n",
           fail.message);
    exit_status = 1;
  }

END:
  NAG_FREE(ap);
  NAG_FREE(bp);
  NAG_FREE(z);
  NAG_FREE(w);
  NAG_FREE(index);

  return exit_status;
}
```

10.2 Program Data

```
nag_zhpgvx (f08tpc) Example Program Data

   4                                                          : n

   Nag_Upper                                                  : uplo
```

```
  -3.0           3.0                                        : VL and VU

 (-7.36, 0.00) ( 0.77, -0.43) (-0.64, -0.92) ( 3.01, -6.97)
               ( 3.49,  0.00) ( 2.19,  4.45) ( 1.90,  3.73)
                              ( 0.12,  0.00) ( 2.88, -3.17)
                                             (-2.54,  0.00) : matrix A

 ( 3.23, 0.00) ( 1.51, -1.92) ( 1.90,  0.84) ( 0.42,  2.50)
               ( 3.58,  0.00) (-0.23,  1.11) (-1.18,  1.37)
                              ( 4.09,  0.00) ( 2.33, -0.14)
                                             ( 4.29,  0.00) : matrix B
```

10.3 Program Results

```
nag_zhpgvx (f08tpc) Example Program Results

Number of eigenvalues found =    2

 Eigenvalues
       -2.9936      0.5047
 Selected eigenvectors
             1           2
 1      1.0000      1.0000
        0.0000     -0.0000

 2      0.1491      0.1882
        0.0777     -0.7410

 3     -1.2303     -0.2080
       -0.4192     -0.4733

 4      0.5811      0.4524
        1.0051      0.9265
```