NAG Library Function Document

nag_zggev (f08wnc)

1 Purpose

nag_zggev (f08wnc) computes for a pair of n by n complex nonsymmetric matrices (A, B) the generalized eigenvalues and, optionally, the left and/or right generalized eigenvectors using the QZ algorithm. nag_zggev (f08wnc) is marked as *deprecated* by LAPACK; the replacement routine is nag_zggev3 (f08wqc) which makes better use of level 3 BLAS.

2 Specification

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

```
void nag_zggev (Nag_OrderType order, Nag_LeftVecsType jobvl,
     Nag_RightVecsType jobvr, Integer n, Complex a[], Integer pda,
     Complex b[], Integer pdb, Complex alpha[], Complex beta[], Complex vl[],
     Integer pdvl, Complex vr[], Integer pdvr, NagError *fail)
```

3 Description

A generalized eigenvalue for a pair of matrices (A, B) is a scalar λ or a ratio $\alpha/\beta = \lambda$, such that $A - \lambda B$ is singular. It is usually represented as the pair (α, β), as there is a reasonable interpretation for $\beta = 0$, and even for both being zero.

The right generalized eigenvector v_j corresponding to the generalized eigenvalue λ_j of (A, B) satisfies

$$Av_j = \lambda_j Bv_j.$$

The left generalized eigenvector u_j corresponding to the generalized eigenvalue λ_j of (A, B) satisfies

$$u_j^{\mathrm{H}} A = \lambda_j u_j^{\mathrm{H}} B,$$

where u_j^{H} is the conjugate-transpose of u_j.

All the eigenvalues and, if required, all the eigenvectors of the complex generalized eigenproblem $Ax = \lambda Bx$, where A and B are complex, square matrices, are determined using the QZ algorithm. The complex QZ algorithm consists of three stages:

1. A is reduced to upper Hessenberg form (with real, non-negative subdiagonal elements) and at the same time B is reduced to upper triangular form.
2. A is further reduced to triangular form while the triangular form of B is maintained and the diagonal elements of B are made real and non-negative. This is the generalized Schur form of the pair (A, B).

   This function does not actually produce the eigenvalues λ_j, but instead returns α_j and β_j such that

   $$\lambda_j = \alpha_j/\beta_j, \quad j = 1, 2, \ldots, n.$$

   The division by β_j becomes your responsibility, since β_j may be zero, indicating an infinite eigenvalue.
3. If the eigenvectors are required they are obtained from the triangular matrices and then transformed back into the original coordinate system.

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

Golub G H and Van Loan C F (2012) *Matrix Computations* (4th Edition) Johns Hopkins University Press, Baltimore

Wilkinson J H (1979) Kronecker's canonical form and the QZ algorithm *Linear Algebra Appl.* **28** 285–303

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: **jobvl** – Nag_LeftVecsType *Input*

On entry: if **jobvl** = Nag_NotLeftVecs, do not compute the left generalized eigenvectors.

If **jobvl** = Nag_LeftVecs, compute the left generalized eigenvectors.

Constraint: **jobvl** = Nag_NotLeftVecs or Nag_LeftVecs.

3: **jobvr** – Nag_RightVecsType *Input*

On entry: if **jobvr** = Nag_NotRightVecs, do not compute the right generalized eigenvectors.

If **jobvr** = Nag_RightVecs, compute the right generalized eigenvectors.

Constraint: **jobvr** = Nag_NotRightVecs or Nag_RightVecs.

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 $\max(1, \mathbf{pda} \times \mathbf{n})$.

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

a$[(j-1) \times \mathbf{pda} + i - 1]$ when **order** = Nag_ColMajor;
a$[(i-1) \times \mathbf{pda} + j - 1]$ when **order** = Nag_RowMajor.

On entry: the matrix A in the pair (A, B).

On exit: **a** has been overwritten.

6: **pda** – Integer *Input*

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

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

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

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

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

$\mathbf{b}[(j-1) \times \mathbf{pdb} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{b}[(i-1) \times \mathbf{pdb} + j - 1]$ when **order** = Nag_RowMajor.

On entry: the matrix B in the pair (A, B).

On exit: **b** has been overwritten.

8: **pdb** – Integer *Input*

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

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

9: **alpha**[**n**] – Complex *Output*

On exit: see the description of **beta**.

10: **beta**[**n**] – Complex *Output*

On exit: $\mathbf{alpha}[j-1]/\mathbf{beta}[j-1]$, for $j = 1, 2, \ldots, \mathbf{n}$, will be the generalized eigenvalues.

Note: the quotients $\mathbf{alpha}[j-1]/\mathbf{beta}[j-1]$ may easily overflow or underflow, and $\mathbf{beta}[j-1]$ may even be zero. Thus, you should avoid naively computing the ratio α_j/β_j. However, $\max(|\alpha_j|)$ will always be less than and usually comparable with $\|A\|_2$ in magnitude, and $\max(|\beta_j|)$ will always be less than and usually comparable with $\|B\|_2$.

11: **vl**[*dim*] – Complex *Output*

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

$\max(1, \mathbf{pdvl} \times \mathbf{n})$ when **jobvl** = Nag_LeftVecs;
1 otherwise.

The ith element of the jth vector is stored in

$\mathbf{vl}[(j-1) \times \mathbf{pdvl} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{vl}[(i-1) \times \mathbf{pdvl} + j - 1]$ when **order** = Nag_RowMajor.

On exit: if **jobvl** = Nag_LeftVecs, the left generalized eigenvectors u_j are stored one after another in the columns of **vl**, in the same order as the corresponding eigenvalues. Each eigenvector will be scaled so the largest component will have $|\text{real part}| + |\text{imag. part}| = 1$.

If **jobvl** = Nag_NotLeftVecs, **vl** is not referenced.

12: **pdvl** – Integer *Input*

On entry: the stride used in the array **vl**.

Constraints:

if **jobvl** = Nag_LeftVecs, $\mathbf{pdvl} \geq \max(1, \mathbf{n})$;
otherwise $\mathbf{pdvl} \geq 1$.

13: **vr**[*dim*] – Complex *Output*

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

$\max(1, \mathbf{pdvr} \times \mathbf{n})$ when **jobvr** = Nag_RightVecs;
1 otherwise.

The ith element of the jth vector is stored in

$\mathbf{vr}[(j-1) \times \mathbf{pdvr} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{vr}[(i-1) \times \mathbf{pdvr} + j - 1]$ when **order** = Nag_RowMajor.

On exit: if **jobvr** $=$ Nag_RightVecs, the right generalized eigenvectors v_j are stored one after another in the columns of **vr**, in the same order as the corresponding eigenvalues. Each eigenvector will be scaled so the largest component will have $|\text{real part}| + |\text{imag. part}| = 1$.

If **jobvr** $=$ Nag_NotRightVecs, **vr** is not referenced.

14: **pdvr** – Integer *Input*

On entry: the stride used in the array **vr**.

Constraints:

if **jobvr** $=$ Nag_RightVecs, **pdvr** $\geq \max(1, \mathbf{n})$;
otherwise **pdvr** ≥ 1.

15: **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 $\langle value \rangle$ had an illegal value.

NE_EIGENVECTORS

A failure occurred in nag_ztgevc (f08yxc) while computing generalized eigenvectors.

NE_ENUM_INT_2

On entry, **jobvl** $= \langle value \rangle$, **pdvl** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: if **jobvl** $=$ Nag_LeftVecs, **pdvl** $\geq \max(1, \mathbf{n})$;
otherwise **pdvl** ≥ 1.

On entry, **jobvr** $= \langle value \rangle$, **pdvr** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: if **jobvr** $=$ Nag_RightVecs, **pdvr** $\geq \max(1, \mathbf{n})$;
otherwise **pdvr** ≥ 1.

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.

On entry, **pdvl** $= \langle value \rangle$.
Constraint: **pdvl** > 0.

On entry, **pdvr** $= \langle value \rangle$.
Constraint: **pdvr** > 0.

NE_INT_2

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

On entry, **pdb** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
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 2.7.6 in How to Use the NAG Library and its Documentation for further information.

NE_ITERATION_QZ

The QZ iteration failed. No eigenvectors have been calculated but **alpha** and **beta** should be correct from element $\langle value \rangle$.

The QZ iteration failed with an unexpected error, please contact NAG.

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

The computed eigenvalues and eigenvectors are exact for nearby matrices $(A + E)$ and $(B + F)$, where

$$\|(E, F)\|_F = O(\epsilon)\|(A, B)\|_F,$$

and ϵ is the ***machine precision***. See Section 4.11 of Anderson *et al.* (1999) for further details.

Note: interpretation of results obtained with the QZ algorithm often requires a clear understanding of the effects of small changes in the original data. These effects are reviewed in Wilkinson (1979), in relation to the significance of small values of α_j and β_j. It should be noted that if α_j and β_j are **both** small for any j, it may be that no reliance can be placed on **any** of the computed eigenvalues $\lambda_i = \alpha_i / \beta_i$. You are recommended to study Wilkinson (1979) and, if in difficulty, to seek expert advice on determining the sensitivity of the eigenvalues to perturbations in the data.

8 Parallelism and Performance

nag_zggev (f08wnc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zggev (f08wnc) 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_dggev (f08wac).

10 Example

This example finds all the eigenvalues and right eigenvectors of the matrix pair (A, B), where

$$A = \begin{pmatrix} -21.10 - 22.50i & 53.50 - 50.50i & -34.50 + 127.50i & 7.50 + 0.50i \\ -0.46 - 7.78i & -3.50 - 37.50i & -15.50 + 58.50i & -10.50 - 1.50i \\ 4.30 - 5.50i & 39.70 - 17.10i & -68.50 + 12.50i & -7.50 - 3.50i \\ 5.50 + 4.40i & 14.40 + 43.30i & -32.50 - 46.00i & -19.00 - 32.50i \end{pmatrix}$$

and

$$B = \begin{pmatrix} 1.00 - 5.00i & 1.60 + 1.20i & -3.00 + 0.00i & 0.00 - 1.00i \\ 0.80 - 0.60i & 3.00 - 5.00i & -4.00 + 3.00i & -2.40 - 3.20i \\ 1.00 + 0.00i & 2.40 + 1.80i & -4.00 - 5.00i & 0.00 - 3.00i \\ 0.00 + 1.00i & -1.80 + 2.40i & 0.00 - 4.00i & 4.00 - 5.00i \end{pmatrix}.$$

10.1 Program Text

```
/* nag_zggev (f08wnc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <naga02.h>
#include <nagf08.h>
#include <nagm01.h>
#include <nagx02.h>
#include <nagx04.h>

#ifdef __cplusplus
extern "C"
{
#endif
  static Integer NAG_CALL compare(const Nag_Pointer a, const Nag_Pointer b);
  static Integer normalize_vectors(Integer n, Complex v[], Complex e[],
                                   size_t rank[], const char *title);
  static Integer sort_values (Integer n, Complex alpha[],
                             Complex beta[], Complex e[],
                             size_t rank[], double emod[]);
#ifdef __cplusplus
}
#endif

int main(void)
{
  /* Scalars */
  Integer            i, j, n, ninf, pda, pdb, pdvl, pdvr;
  Integer            exit_status = 0;

  /* Arrays */
  Complex            *a = 0, *alpha = 0, *b = 0, *beta = 0, *vl = 0, *vr = 0;
  Complex            *e = 0;
  double             *emod = 0;
  size_t             *rank = 0;
  char               nag_enum_arg[40];

  /* Nag Types */
  NagError           fail;
  Nag_OrderType      order;
  Nag_LeftVecsType   jobvl;
  Nag_RightVecsType  jobvr;
```

```
#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_zggev (f08wnc) Example Program Results\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
   */
  jobvl = (Nag_LeftVecsType) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
  scanf_s(" %39s%*[^\n]", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
  scanf(" %39s%*[^\n]", nag_enum_arg);
#endif
  jobvr = (Nag_RightVecsType) nag_enum_name_to_value(nag_enum_arg);

  pda = n;
  pdb = n;
  pdvl = (jobvl == Nag_LeftVecs ? n : 1);
  pdvr = (jobvr == Nag_RightVecs ? n : 1);

  /* Allocate memory */
  if (!(a = NAG_ALLOC(n * n, Complex)) ||
      !(alpha = NAG_ALLOC(n, Complex)) ||
      !(b = NAG_ALLOC(n * n, Complex)) ||
      !(beta = NAG_ALLOC(n, Complex)) ||
      !(vl = NAG_ALLOC(pdvl * pdvl, Complex)) ||
      !(e = NAG_ALLOC(n, Complex)) ||
      !(emod = NAG_ALLOC(n, double)) ||
      !(rank   = NAG_ALLOC(n, size_t)) ||
      !(vr = NAG_ALLOC(pdvr * pdvr, Complex)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Read in the matrices A and B */
  for (i = 1; i <= n; ++i)
    for (j = 1; 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 = 1; 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

  /* Solve the generalized eigenvalue problem using nag_zggev (f08wnc). */
  nag_zggev(order, jobvl, jobvr, n, a, pda, b, pdb, alpha, beta, vl, pdvl, vr,
            pdvr, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_zggev (f08wnc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  printf("\n     Eigenvalues\n");
  ninf = 0;
  for (i = 0; i < n; ++i) {
    if (beta[i].re == 0.0) {
      ninf = ninf + 1;
      printf(" %4" NAG_IFMT "     Infinite eigenvalue\n", i + 1);
    }
  }
  if (ninf==0) {
    /* Sort values by decreasing modulus and store in e[] */
    exit_status=sort_values (n, alpha, beta, e, rank, emod);
    for (i = 0; i < n; ++i) {
      printf(" %4" NAG_IFMT "     (%7.3f,%7.3f)\n", i + 1, e[i].re, e[i].im);
    }
    if (jobvl == Nag_LeftVecs) {
      exit_status = normalize_vectors(n, vl, e, rank,
                                      "    Left eigenvectors (columns)");
    }
    if (jobvr == Nag_RightVecs && exit_status == 0) {
      exit_status = normalize_vectors(n, vr, e, rank,
                                      "   Right eigenvectors (columns)");
    }
  }

END:
  NAG_FREE(a);
  NAG_FREE(alpha);
  NAG_FREE(b);
  NAG_FREE(beta);
  NAG_FREE(vl);
  NAG_FREE(vr);
  NAG_FREE(e);
  NAG_FREE(emod);
  NAG_FREE(rank);

  return exit_status;
}
static Integer normalize_vectors(Integer n, Complex v[], Complex e[],
                                 size_t rank[], const char *title)
```

```
{

  Complex          scal;
  double           r, rr;
  Integer          errors = 0, i, j, k;
  Nag_OrderType    order;
  NagError         fail;

  INIT_FAIL(fail);

#ifdef NAG_COLUMN_MAJOR
#define V(I, J)  v[(J-1)*n + I - 1]
  order = Nag_ColMajor;
#else
#define V(I, J)  v[(I-1)*n + J - 1]
  order = Nag_RowMajor;
#endif
  /* Re-normalize the eigenvectors, largest absolute element real */
  for (i=1; i<=n; i++) {
    k = 0;
    r = -1.0;
    for (j=1; j<=n; j++) {
      rr = nag_complex_abs(V(j,i));
      if (rr>r) {
        r = rr;
        k = j;
      }
    }
    scal.re = V(k,i).re/(r*r);
    scal.im = -V(k,i).im/(r*r);
    for (j=1; j<=n; j++) {
      V(j,i) = nag_complex_multiply(V(j,i),scal);
    }
    V(k,i).re = 1.0;
    V(k,i).im = 0.0;
  }
  /* Sort eigenvectors according to rank */
  for (i=1; i<=n; i++) {
    for (j=1; j<=n; j++)
      e[j-1] = V(i,j);

    /* Sort eigenvector row i using nag_reorder_vector (m01esc). */
    nag_reorder_vector((Pointer) e, (size_t) n, sizeof(Complex),
                       (ptrdiff_t) sizeof(Complex), rank, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_reorder_vector (m01esc).\n%s\n", fail.message);
      errors = 5;
      goto END;
    }
    for (j=1; j<=n; j++)
      V(i,j) = e[j-1];
  }

  printf("\n");
  /* Print eigenvectors using nag_gen_complx_mat_print (x04dac). */
  fflush(stdout);
  nag_gen_complx_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
                           n, v, n, title, 0, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_complx_mat_print (x04dac).\n%s\n",
           fail.message);
    errors = 6;
  }
#undef V
 END:
  return errors;
}
static Integer sort_values (Integer n, Complex alpha[], Complex beta[],
                            Complex e[], size_t rank[], double emod[])
{
  Integer       i, exit_status = 0;
```

```
  NagError      fail;

  INIT_FAIL(fail);

  for (i = 0; i < n; ++i) {
    /* nag_complex_divide (a02cdc): Quotient of two complex numbers;
     * nag_complex_abs (a02ddc): Moduli of complex number.
     */
    e[i] = nag_complex_divide(alpha[i], beta[i]);
    emod[i] = nag_complex_abs(e[i]);
  }
  /* Rank sort eigenvalues by absolute values using
   * nag_rank_sort (m01dsc).
   */
  nag_rank_sort((Pointer) emod, (size_t) n, (ptrdiff_t) (sizeof(double)),
                compare, Nag_Descending, rank, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_rank_sort (m01dsc).\n%s\n", fail.message);
    exit_status = 10;
    goto END;
  }
  /* Turn ranks into indices using nag_make_indices (m01zac). */
  nag_make_indices(rank, (size_t) n, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_make_indices (m01zac).\n%s\n", fail.message);
    exit_status = 11;
    goto END;
  }
  /* Sort eigenvalues using nag_reorder_vector (m01esc). */
  nag_reorder_vector((Pointer) e, (size_t) n, sizeof(Complex),
                     (ptrdiff_t) sizeof(Complex), rank, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_reorder_vector (m01esc).\n%s\n", fail.message);
    exit_status = 12;
    goto END;
  }
 END:
  return exit_status;
}
static Integer NAG_CALL compare(const Nag_Pointer a, const Nag_Pointer b)
{
  double x = *((const double *) a) - *((const double *) b);
  return (x < 0.0 ? -1 : (x == 0.0 ? 0 : 1));
}
```

10.2 Program Data

```
nag_zggev (f08wnc) Example Program Data

     4                                                             : n

   Nag_NotLeftVecs                                                 : jobvl
   Nag_RightVecs                                                   : jobvr

  (-21.10,-22.50) ( 53.50,-50.50) (-34.50,127.50) (  7.50,  0.50)
  ( -0.46, -7.78) ( -3.50,-37.50) (-15.50, 58.50) (-10.50, -1.50)
  (  4.30, -5.50) ( 39.70,-17.10) (-68.50, 12.50) ( -7.50, -3.50)
  (  5.50,  4.40) ( 14.40, 43.30) (-32.50,-46.00) (-19.00,-32.50) : A

  (  1.00, -5.00) (  1.60,  1.20) ( -3.00,  0.00) (  0.00, -1.00)
  (  0.80, -0.60) (  3.00, -5.00) ( -4.00,  3.00) ( -2.40, -3.20)
  (  1.00,  0.00) (  2.40,  1.80) ( -4.00, -5.00) (  0.00, -3.00)
  (  0.00,  1.00) ( -1.80,  2.40) (  0.00, -4.00) (  4.00, -5.00) : B
```

10.3 Program Results

```
nag_zggev (f08wnc) Example Program Results

     Eigenvalues
    1     (  3.000, -9.000)
    2     (  4.000, -5.000)
    3     (  2.000, -5.000)
    4     (  3.000, -1.000)

    Right eigenvectors (columns)
          1        2        3        4
 1   1.0000   1.0000   1.0000   1.0000
     0.0000   0.0000   0.0000   0.0000

 2   0.1600   0.0089   0.0046   0.1600
    -0.1200  -0.0067  -0.0034  -0.1200

 3   0.1200  -0.0333   0.0629   0.1200
     0.1600  -0.0000   0.0000  -0.1600

 4  -0.1600  -0.0000  -0.0000   0.1600
     0.1200   0.1556   0.0629   0.1200
```